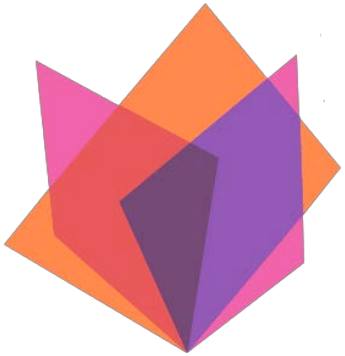
OpenMP®

SC'20 Booth Talk Series



Offloading to GPUs with OpenMP: Case Study with GAMESS

Colleen Bertoni, JaeHyuk Kwack, Argonne National Laboratory, Buu Pham, Iowa State University

ACKNOWLEDGEMENTS

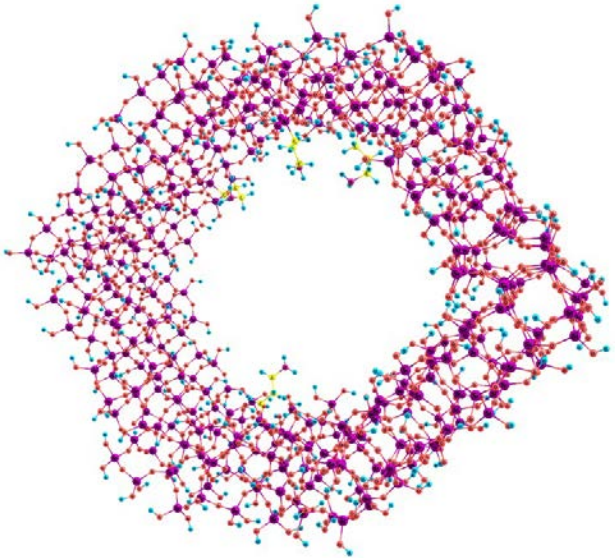
- GAMESS team
  - Buu Pham
  - Melisa Alkan
  - Peng Xu
  - Tosaporn Sattasathuchana
- Argonne collaborators
  - JaeHyuk Kwack

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

INTRO OF GAMESS

- GAMESS is the General Atomic and Molecular Electronic Structure System
  - General-purpose electronic structure code (many methods and capabilities)
  - ~1 million lines of Fortran
  - Optional C/C++ GPU-accelerated libraries/applications in GAMESS
- Scientific problem of interest
  - FMO/RI-MP2 calculations towards accurate simulations of catalysis reactions inside a mesoporous silica nanoparticle



GAMESS ON GPUS

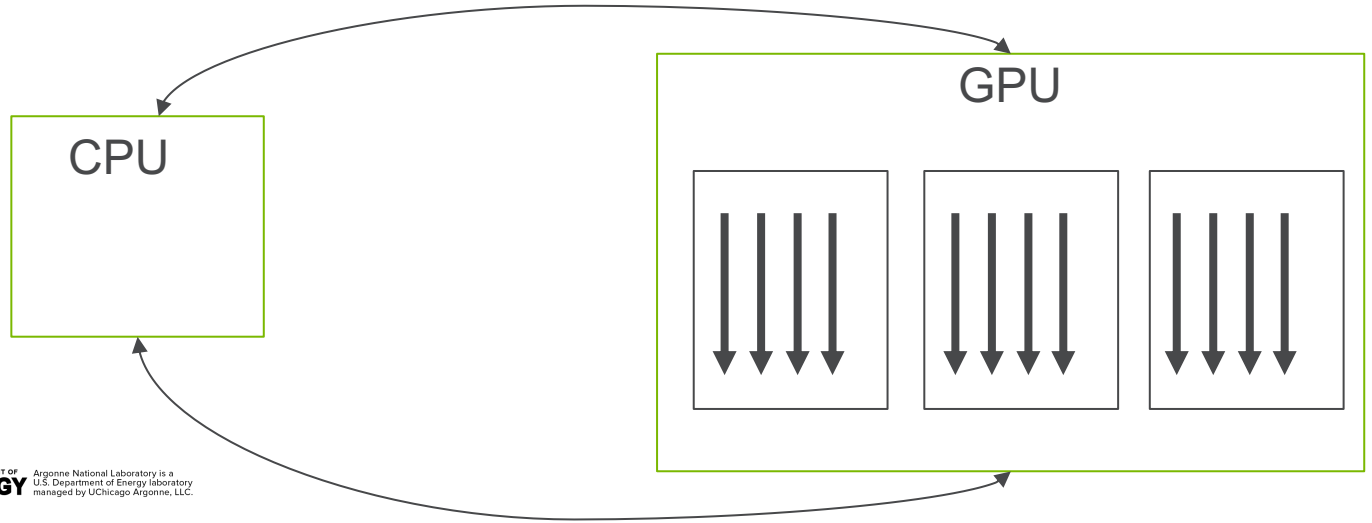
- Several methods ported to GPUs
  - Hartree-Fock
    - Solves a set of non-linear eigenvalue equations iteratively
    - Two main bottlenecks: 1) computation of a very large number of 4-index 2-electron repulsion integrals (4-2ERIs); and 2) forming a N^2 Fock matrix by contracting the N^4 4-2ERI tensor with a density matrix.
  - RI-MP2
    - Correction to Hartree-Fock
    - RI approximation simplifies the integral evaluation from 4-index to 3-index 2-electron repulsion integrals and allows the use of efficient matrix multiplication operations

GPU BACKGROUND

VERY BRIEF GPU HARDWARE INTRO

Nvidia V100 (GPU on Summit)

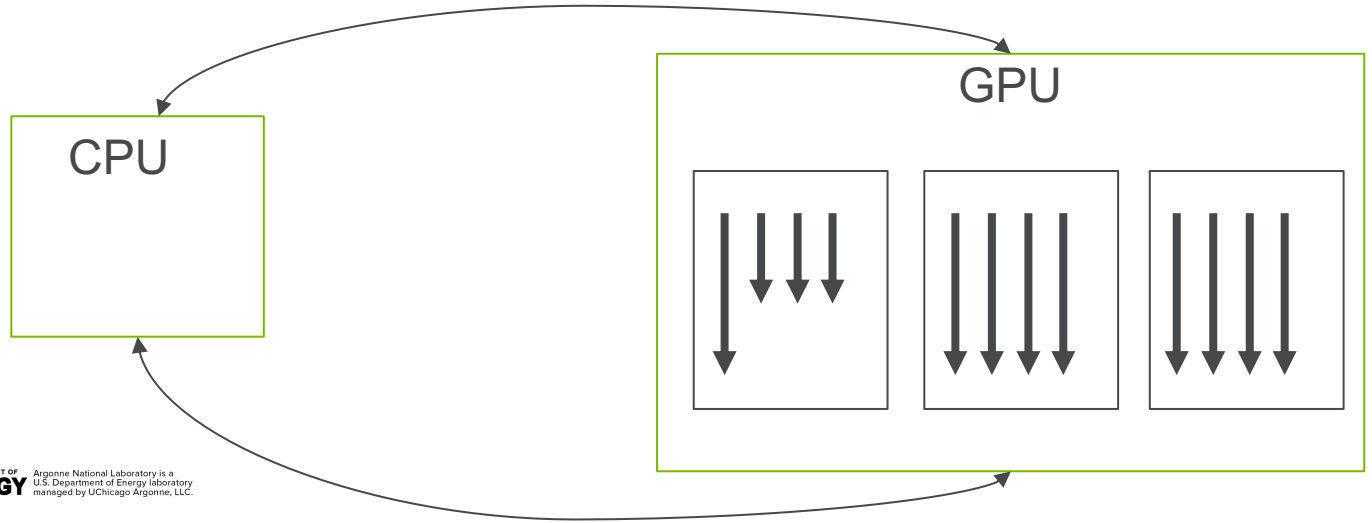
- V100s have hierarchical parallelism (SMs and CUDA cores in an SM)
- Threads execute lock-step in a warp



VERY BRIEF GPU HARDWARE INTRO

Nvidia V100 (GPU on Summit)

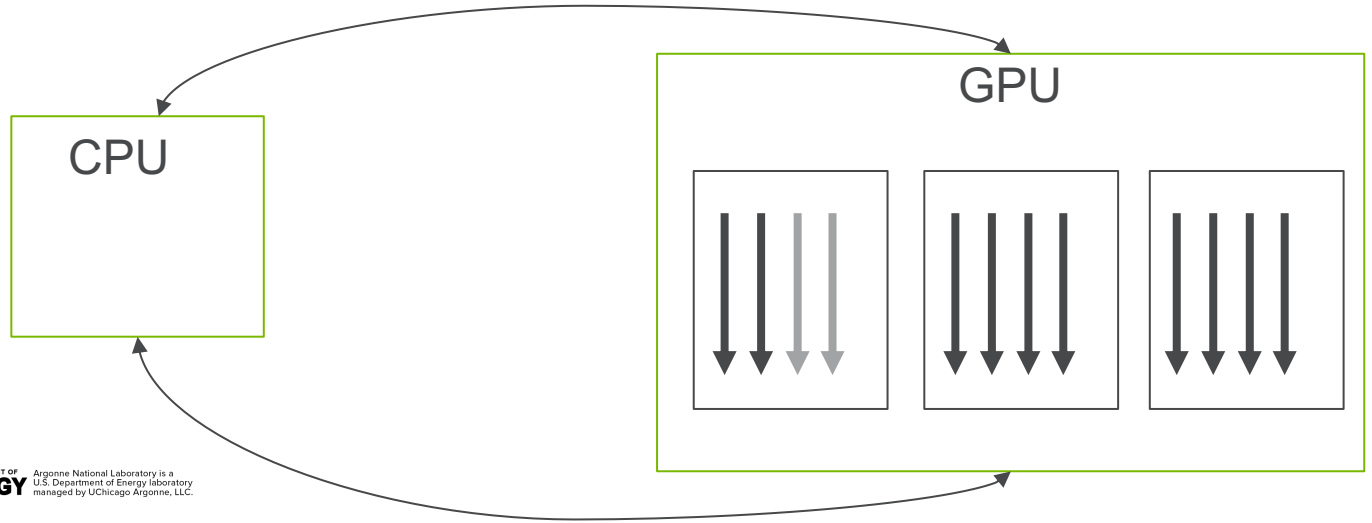
- V100s have hierarchical parallelism (SMs and CUDA cores in an SM)
- Threads execute lock-step in a warp



VERY BRIEF GPU HARDWARE INTRO

Nvidia V100 (GPU on Summit)

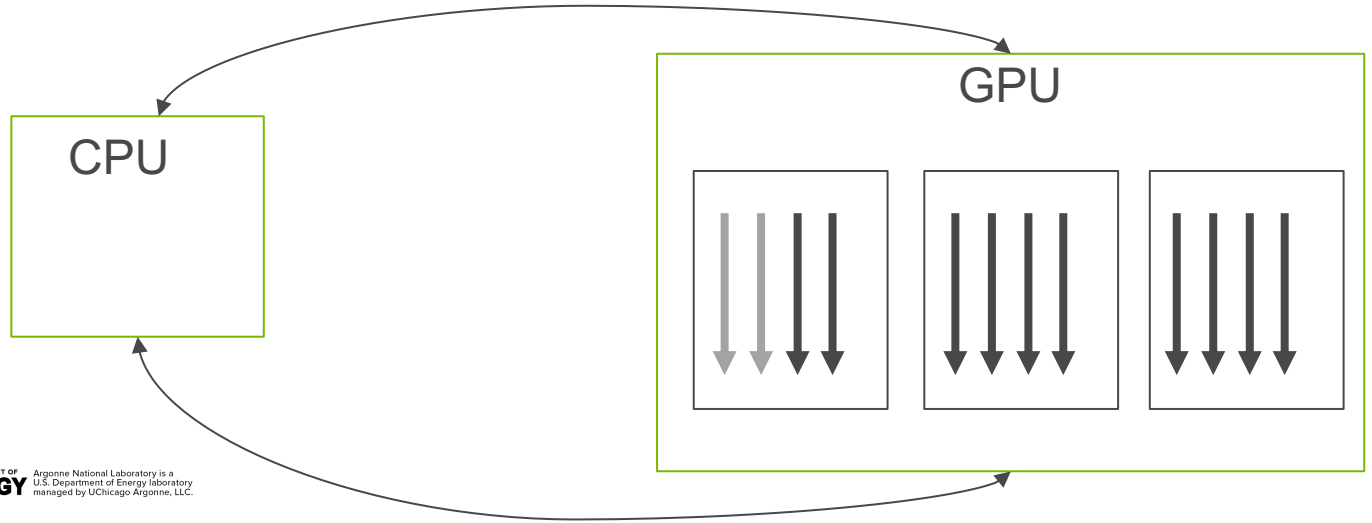
- V100s have hierarchical parallelism (SMs and CUDA cores in an SM)
- Threads execute lock-step in a warp



VERY BRIEF GPU HARDWARE INTRO

Nvidia V100 (GPU on Summit)

- V100s have hierarchical parallelism (SMs and CUDA cores in an SM)
- Threads execute lock-step in a warp



HARTREE-FOCK PORTING

INTRO OF HF: CPU VERSION

```
1  subroutine  compute_integrals ( ... )
2     ! bottleneck loop over N^4 4-2ERIs
3     !$omp parallel do private ( ... )
4     do i=1, N^4
5        ! branching over methods
6        if(method(i) .eq. 1 ) call method_one(i)
7        if(method(i) .eq. 2 ) call method_two(i)
8         ...
9     enddo
10     ...
11 end subroutine compute_integrals
12  ...
13 ! routine for specific integral method
14 subroutine method_one( i )
15     ...
16   ! branch again on integral type
17   if( type(i) .eq. 1 )
18       call int1( i, ...)
19    ...
20 end subroutine method_one
```

- Focused on the 4-2ERI evaluation since they are a main computational bottleneck
- Parallelized previously with MPI+OpenMP CPU threading
  - But multiple levels of conditional statements and load imbalance between threads

INTRO OF HF: CPU VERSION

```
subroutine compute_integrals ( ... )
   ! bottleneck loop over N^4 4-2ERIs
   !$omp parallel do private ( ... )
   do i=1, N^4
      ! branching over methods
      if (method(i) .eq. 1 ) call method_one(i)
      if (method(i) .eq. 2 ) call method_two(i)
       ...
   enddo
    ...
end subroutine compute_integrals
 ...
! routine for specific integral method
subroutine method_one( i )
    ...
  ! branch again on integral type
  if ( type(i) .eq. 1 )
     call int1 ( i, ...)
   ...
end subroutine method_one
```

```
subroutine compute_integrals ( ... )
  do i=1,N^4
     ! create sorted array by method and type
     if (method(i) .eq. 1 && type(i) .eq. 1)
       store i in sorted (method=1,type=1)
     endif
      ...
  enddo
  ! after sorting, integrals of different methods and
  ! types are evaluated separately
  !$ omp target teams distribute parallel do map (...)
  do i in sorted (method=1,type=1)
     call int1 (i ,...)
  enddo
  !$ omp target teams distribute parallel do map (...)
  do i in sorted (method=1,type=2)
     call int2 (i ,...)
  enddo
   ...
end subroutine compute_integrals
```

RESULTS AND ISSUES

N	Wall time CPU (s)	Wall time GPU (s)	Speedup	#QUARTETS
4	0.1	3.0	<0.1	51,519
8	0.1	5.3	<0.1	548,454
16	1.0	7.7	0.1	3,899,510
32	18.0	16.3	1.1	24,078,056
48	102.0	18.0	5.7	65,325,492
64	243.0	26.0	9.3	128,650,354

- Cluster of N water molecules.
- Using 6-31G basis set [SP only for now]

- For GPU calculations: 1 MPI rank, 42 threads, 1 GPU.
- For CPU calculations: 1 MPI rank, 42 threads.
- N is the number of water molecules in water cluster.
- W.t. CPU is the recorded wall time (s) of quartet formation and Fock build for 1 SCF iteration using CPU threaded code.
- W.t. GPU is the recorded wall time (s) of quartet formation and Fock build for 1 SCF iteration using GPU+CPU threaded code.
- Speedup is the ratio of w.t. CPU over w.t. GPU.

RESULTS AND ISSUES

Issues

1. We found that if we had a function call in an offloaded target region and were also using thread-local arrays, our runtime increased with increasing iterations
2. We found that using thread local arrays hurt performance. When we made thread-local arrays ourselves (and indexing by thread ourselves), the issue went away.

Reported to Oak Ridge/IBM support

RI-MP2 PORTING

INTRO OF RI-MP2 MINI-APP

- Computes RI-MP2 perturbative correction to the Hartree-Fock energy

- RI approximation allows the integral evaluation to be simplified and written in terms of matrix multiplication

$$E^{(2)} = \sum_{i \leq j}^{occ} (2 - \delta_{ij}) \sum_{ab}^{vir} \frac{(ia|jb)[2(ja|jb) - (ib|ja)]}{\epsilon_i + \epsilon_j - (\epsilon_a + \epsilon_b)}$$

$$(ia|jb) = \sum_{P}^{aux} B_{ia}^{P} B_{jb}^{P}$$

INTRO OF RI-MP2: CPU VERSION

```
do JACT=1, NACT
    do IACT=1, JACT
        ...
        call  RIMP2_EIJ( B32(:,:,IACT), B32 (:,:, JACT) ,...)
    enddo
enddo
subroutine  RIMP2_EIJ( ...  )
    ...

   call  DGEMM with BI(:,:), BJ (:,:) ,  QVV(:,:)




    do IB=1,NVIR
        do IA=1,NVIR
            compute E2_t  with  QVV(:,:),   ...
        enddo
    enddo



    ...
end  subroutine
```

- Focused on the computation of the perturbative correction, since it is the main bottleneck of the RI-MP2 method
- Majority of the time is spent in a DGEMM (matrix multiply) call

RESTRUCTURING FOR GPU VERSION

```
do JACT=1, NACT
    do IACT=1, JACT
        ...
        call  RIMP2_EIJ( B32(:,:,IACT), B32 (:,:, JACT) ,...)
    enddo
enddo
subroutine  RIMP2_EIJ( ... )
    ...

   call  DGEMM with BI(:,:), BJ (:,:) ,  QVV(:,:)




    do IB=1,NVIR
        do IA=1,NVIR
            compute E2_t  with  QVV(:,:),   ...
        enddo
    enddo



    ...
end  subroutine
```

```
do JACT=1, NACT


        ...
        call  RIMP2_EIJ( B32(:,:, 1:JACT) ,B32 (:,:, JACT) ,...)

enddo
subroutine  RIMP2_EIJ( ... )
    ...
   !$omp target  data  use_device_ptr (BI,BJ,QVV)
   call  DGEMM with BI(:,:,1:JACT),BJ (:,:) ,QVV(:,:,1:JACT)
   !$omp end target  data
   !$omp target  teams  distribute  reduction (+:E2_t)
   do IC=1,JACT
   !$omp parallel  do  reduction (+:E2_t)  collapse (2)
      do IB=1,NVIR
         do IA=1,NVIR
            compute E2_t  with  QVV(:,:,IC) ,  ...
         enddo
      enddo
      !$omp end parallel  do
   enddo
   !$omp end target  teams  distribute
    ...
end  subroutine
```

RESULTS AND ISSUES

Table 10. Walltimes and Speedups of the restructured RI-MP2 kernel (input: c60.kern)

Directives	Math Library	Processors	Wall time (sec)	Speedup
Serial	ESSL	1 core of an IBM Power9	342.697	0.036 x
OpenMP threading	ESSL	2 IBM Power9 (42 threads)	12.231	1 x
OpenMP threading	MKL	2 Intel Xeon 8180M (112 threads)	4.317	2.83 x
OpenMP Offloading	NVBLAS	1 NVIDIA V100	1.734	7.05 x
OpenMP Offloading	cuBLAS	1 NVIDIA V100	1.983	6.17 x
OpenMP Offloading	cuBLASXT	1 NVIDIA V100	1.728	7.08 x
OpenACC Offloading	cuBLAS	1 NVIDIA V100	1.905	6.42 x
OpenACC Offloading	cuBLASXT	1 NVIDIA V100	1.692	7.23 x

J. Kwack, C. Bertoni, B. Pham, J. Larkin, WACCPD '19

RESULTS AND ISSUES

Issues

- Vendor library interoperability with OpenMP

TAKE-AWAYS

TAKE-AWAYS

- Think about GPU architecture and how to write efficient code on it

- OpenMP (IBM 16.1) had all of the features we wanted
  - Some issues: unexpected slow performance for some features and interactions with vendor math libraries
  - But overall, with OpenMP we achieved reasonable speedups over the CPU versions

- Coming in OpenMP 5.1
  - Dispatch construct!

OpenMP®

SC'20 Booth Talk Series

For the OpenMP specification, tutorials, forum, reference guides, and links to other resources, visit **www.openmp.org**