CURRENT STATUS OF THE PROJECT TO ENABLE GAUSSIAN 09 ON GPGPUS

Roberto Gomperts (NVIDIA, Corp.)
Michael Frisch (Gaussian, Inc.)
Giovanni Scalmani (Gaussian, Inc.)
Brent Leback (PGI)
TOPICS

- Gaussian
- Design Guidelines
- Gaussian’s Parallelism and Memory Model
- Implementation Framework
- Current Status
- Early Performance Results
- Closing Remarks
GAUSSIAN

- A Computational Chemistry Package that provides state-of-the-art capabilities for electronic structure modeling
- Gaussian 09 is licensed for a wide variety of computer systems
- All versions of Gaussian 09 contain virtually every scientific/modeling feature, and none imposes any artificial limitations on calculations other than computational resources and time constraints
- Researchers use Gaussian to, among others, study molecules and reactions; predict and interpret spectra; explore thermochemistry, photochemistry and other excited states; include solvent effects, and many more
GPU AS COPROCESSOR FOR CPU PROGRAMS

GPU

Only Critical Functions
Parallelize using CUDA Programming Model

Application Code

Rest of CPU Code

CPU
GOAL AND GUIDELINES FOR THE PROJECT

Establish a Framework for the GPU-enabling of Gaussian

- **Code Maintainability (Code Unification)**
  - Leverage Existing code/algorithms, including Parallelism and Memory Model
  - Simplifies Resolving Problems
  - Simplifies Improvement on existing code
  - Simplifies Adding New Code

- **Accelerate Gaussian for Relevant and Appropriate Theories and Methods**
  - Relevant: many users of Gaussian
  - Appropriate: time consuming and good mapping to GPUs
COMPILER TECHNOLOGY

- **OpenACC**: A set of directive-based extensions to standard Fortran, C and C++ that enable offloading of data and compute-intensive loops and regions of code from a CPU host to an attached GPU or accelerator device.

- **Some Benefits:**
  - It enables an easy path for developers to start reaping the benefits from powerful many-core accelerators like GPUs.
  - It permits a common code base for accelerated and non-accelerator enabled systems.
  - It provides an incremental path for moving legacy applications to accelerators, that may disturb the existing code less than other approaches.
  - It allows programmer tools to focus on supporting a common accelerator standard with their own unique extensions.
OPENACC MEMBERS

- Currently versions available for NVIDIA GPUs, AMD Radeons and Xeon PHIs
Program myscience
...
cpu code...

!$acc kernels if(OnGPU)
do k = 1,n1
do i = 1,n2
... cpu/parallel gpu code...
V(i,k) = ....
endo
do
do
!$acc end kernels
... 
End Program myscience

Portable compiler hints
Compiler parallelizes code
Designed for multicore CPUs & many core GPUs / Accelerators

OpenACC Compiler Directive

Recommended:
S4200 - Advanced Accelerated Computing Using Directives
GAUSSIAN’S PARALLELISM AND MEMORY MODEL

- CPU-side Parallelism
  - Intra-node: OpenMP
  - Inter-nodes: Linda

- Memory Model
  - Pre-define a memory space and use it as a sort of “heap”
  - When a Parallel Region is reached the rest of the memory is partitioned

For example with 4 OpenMP threads:

<table>
<thead>
<tr>
<th>Seq</th>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MAPPING TO GPUS

- For now only inside a node
- Computational Mapping
  - "Assign" an Accelerator to an OpenMP thread
    - If $n_{\text{GPUs}} < n_{\text{Threads}}$ then "free threads" do their regular work (Load Distribution!)
    - If $n_{\text{GPUs}} > n_{\text{Threads}}$ then some GPUs will be idle
- Memory Mapping
  - If $M_{\text{gpu}} < M_{\text{part}}$: Use $M_{\text{gpu}}$ for all threads
  - If $M_{\text{gpu}} > M_{\text{part}}$: Use $M_{\text{part}}$ on the GPU
  - Mapping example with 2 GPUs and 4 threads
ADDITIONAL (POTENTIAL) PARALLELISM WITH GPUS

- CPU/GPU Compute overlap (non-blocking kernels)

- GPU/GPU Compute overlap (GPU “streams”; not used for now)

- CPU->GPU Data Transfers (“Asynchronous” CopyIn)

- Parallelism (& vectorization) inside GPU
  - Resolve parallel dependencies
  - Split loops that contain gather operations (disk I/O) and computational sections
EXPLICIT MEMORY MANAGEMENT

- Arrays passed down a calling tree
- Assign and use GPU if iThread <= nGPUs inside OpenMP region
- Allocate (and initialize) Device arrays once inside OpenMP parallel region
- Deallocate at exit of OpenMP region
IMPLEMENTATION B.O. AND A.O. (BEFORE/AFTER OPENACC)

- Before OpenACC we needed to explicitly allocate the device arrays and pass them through all the calling tree
  - Since the CPU arrays may be needed down the calling tree it was necessary to pass two versions of the arrays.
  - If OnGPU logic, including additional code, is required to decide which array to use

- With OpenACC allocation/mapping/setting happens via directives
  - The same Work routine can now be called. It will have OpenACC directives that will flag whether the device array or the cpu array is to be used.
CURRENT STATUS

- **Direct-HF & 1st Derivatives:** Default paths for closed and open shells
  - Partial implementation using OpenACC and explicit CUDA Fortran
  - There are clear opportunities for further improvement

- **Default DFT calculations with hybrid functionals** benefit from the above

- **(t) corrections for closed and open shell molecules** in default paths for MP4(t) and CCSD(t)
  - Use of NVIDIA’s BLAS library (cuBLAS)

- **Some sections of the iterative CCSD code** (where BLAS is dominant)
  - Investigating performance issues
GAUSSIAN: GENERAL WORK FLOW

Input Processing
"Base" Energies
[Forces/Geometry Optimization]
[Frequencies]
[High Order Energies]
[High Order Derivatives]
Molecular Properties

"Base" Energies (Direct SCF)

... Generate Some ERIs
Form Partial Fock Matrix

... Diagonalize Fock Matrix
Convergence Test

ERIs (Direct SCF)

... Pick/Screen Shell Quartets
Multiply into Temp Arrays

... Form Fock Matrix (Direct SCF)
Gather & Multiply Elements
Scatter into Fock Matrix
EARLY PERFORMANCE RESULTS (DIRECT SCF)

System: 2 Sockets E5-2690 V2 (2x 10 Cores @ 3.0 GHz); 128 GB RAM (DD3-1600); Used 108 GB GPUs: 6 Tesla K40m (15 SMPs @ 875 MHz); 12 GB Global Memory

Valinomycin Force Calculation
Speed Ups Relative to CPU-Only Full Node

<table>
<thead>
<tr>
<th>Method</th>
<th>rB3LYP</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Atoms</td>
<td>168</td>
</tr>
<tr>
<td>Basis Set</td>
<td>6-31G(3df,3p)</td>
</tr>
<tr>
<td>No. of Basis Funcs</td>
<td>3,642</td>
</tr>
<tr>
<td>No. of Cycles</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>l502:</th>
<th>“HF”</th>
<th>XC-Quad</th>
<th>Rest</th>
<th>l703:</th>
<th>“HF”</th>
<th>XC-Quad</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 Threads 0 GPUs</td>
<td>160.3</td>
<td>111.7</td>
<td>124.6</td>
<td>86.6</td>
<td>105.3</td>
<td>66.8</td>
<td>9.1</td>
<td>9.3</td>
</tr>
<tr>
<td>20 Threads 6 GPUs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CCSD(T) WORK FLOW

OpenMP Loop over n CPU Threads
  Loop over IA = 1, nVirtuals (NVA)
  OpenMP Dynamic Load Distribution
  Loop over IB = IA+1, NVA
  Loop over IC = 1, NVA
  SUM(DA) (AB/ID)*A(CJKD) TERM #1
  SUM(LA) -A(ABIL)*(JK/LC) TERM #2
  TRANSPOSE B,I,C,J,K TO ORDER C,I,B,J,K.
  SUM(DB) (ID/AC)*A(JKBD) TERM #4
  SUM(LB) -A(AC/IL)*(JK/LB) TERM #5
  TRANSPOSE C,I,B,J,K TO B,C,I,J,K
  SUM(DB) -(ID/BC)*A(JKAD) TERM #3
  SUM(LB) A(BC/IL)*(JK/LA) TERM #6
  SUM(DA) A(IJAD)*(KD/CB) TERM #7
  SUM(LA) (IJ/LA)*A(LKBC) TERM #9
  TRANSPOSE I.LT.J,B,C,K TO B,I.LT.J,C,K
  SUM(DA) -A(IJBD)*(KD/CA) TERM #8
  SUM(LA) -(IJ/LB)*A(LKAC) TERM #10
End IC-loop
End IB-loop
End IA-loop
End OpenMP-loop

"Base" Energies
Iterative CCSD
(t) Corrections
Molecular Properties

Iterative CCSD
...
Compute ERIs
Form High Order Matrixes
Calculate All
QuadrupleContributions
Convergence Test
...

Out-of-Core logic + Matrix Multiplication

2-index transposition of 4-index matrix
EARLY PERFORMANCE RESULTS (CCSD(T))

Ibuprofen CCSD(t) Calculation
Speed Ups Relative to CPU-Only Full Node

<table>
<thead>
<tr>
<th>Method</th>
<th>CCSD(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Atoms</td>
<td>33</td>
</tr>
<tr>
<td>Basis Set</td>
<td>6-31G(d,p)</td>
</tr>
<tr>
<td>No. of Basis Funcs</td>
<td>315</td>
</tr>
<tr>
<td>No. Occ Orbitals</td>
<td>41</td>
</tr>
<tr>
<td>No. Virt Orbitals</td>
<td>259</td>
</tr>
<tr>
<td>No. of Cycles</td>
<td>15</td>
</tr>
<tr>
<td>No. CCSD iters</td>
<td>16</td>
</tr>
</tbody>
</table>

Labels: Time in hrs

- 24.8
- 3.7
- 24.8
- 3.6
- 22.7
- 1.3
- 2.1
- 2.3

System: 2 Sockets E5-2690 V2 (2x 10 Cores @ 3.0 GHz); 128 GB RAM (DD3-1600); Used 108 GB
GPUs: 6 Tesla K40m (15 SMPs @ 875 MHz); 12 GB Global Memory
CLOSING REMARKS

- Significant progress has been made in creating a framework that keeps an unified code structure for GPU enabled Gaussian
- There is room for performance improvement in the Direct SCF work
- The (t) correction performance looks promising
- Further work: Continue working towards a “product” quality version of Gaussian to be released to customers
  - Continue unification of the code base
  - Tackle non-default paths of the currently enabled code
  - Expand enabling of other Gaussian functionality (2nd Derivatives, XC-quadrature, TDDFT, MP2, etc.)
  - Performance tuning
ACKNOWLEDGEMENTS

The project to enable Gaussian on GPUs is being developed on:

- Hewlett-Packard (HP) Series SL Servers
- NVIDIA® Tesla® GPUs
- PGI Accelerator Compilers with OpenACC