INTERFACING ATOMIC AND MOLECULAR COMPUTER CODES WITH A GRAPHICAL PROCESSING UNIT

Morgan Leider, Dr. Christine Morales, Dr. Fred King

Department of Chemistry – University of Wisconsin - Eau Claire

Introduction

- The primary goal of this research was to modify existing scientific computational chemistry code to use GPUs in order to greatly enhance research speed and to provide a framework for computational research on campus to more quickly and fully exploit this technology.
  - By working on far larger sets of data we can expect to see an overall performance gain in scientific computing of 10x - 35x on calculations that can be done in parallel with current generation GPUs.
  - By leveraging this new technology we can accomplish in months what would previously have taken decades.

Technical Terms

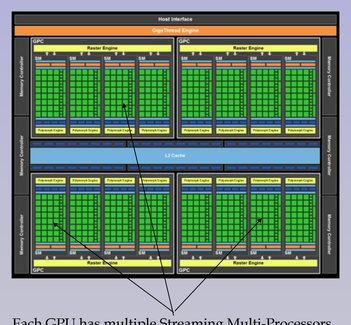
- GPUs - Graphical Processing Units are often referred to as video cards. Traditionally video cards were only useful for improving the visual effects in computer games. Now they are programmable for uses other than gaming such as scientific computing.

Nvidia's Tesla GPU

- Cuda - (Compute Unified Device Architecture) A set of programming commands added to existing program ming languages. Cuda allows code to be run on the GPU instead of a computer's CPU.
- Thread – an independent unit of information containing values, which can then be used in a series of mathematical operations.
- Outlier – A thread that takes significantly longer to complete its calculations than the average thread.
- Parallel Processing – The ability to process or work on multiple threads concurrently.
- Thread Block – A group of threads which all have the same mathematical operations performed on them.
- CPU – Central Processing Unit, the traditional "brain" of a computer.
- Thread Block Size – The number of threads in a thread block.
- Streaming Multi-Processor – The equivalent of a CPU in a GPU which has multiple cores of its own. Multi-Processors are specifically designed to process an entire thread block at once using parallel processing.

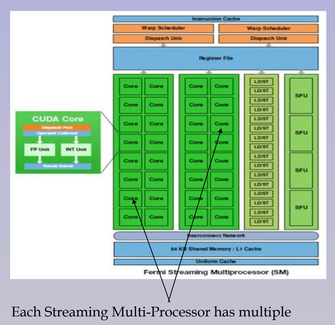
Why Use a GPU Instead of a CPU?

GPU



Each GPU has multiple Streaming Multi-Processors.

Streaming Multi-Processor



Each Streaming Multi-Processor has multiple Cores.

In this example the GPU has 16 Streaming Multi-Processors and each Streaming Multi-Processor has 32 Cores for a total of 512 Cores.

Methods

- Rewrite existing scientific code to allow for parallel processing on a multi-core CPU.
- Take the rewritten parallel code and implement it for GPUs using Cuda Fortran.
- Fine tune the number of threads sent to the GPU and the thread block size for maximum performance.
- The results for this poster were generated using the example code shown to the right.
  - Testing for outliers was done by adding a check to testFunc2 which increased the number of loops from 10,000 to 25,000 if the thread number was 71.

Code Comparison Between Serial, Parallel, and GPU Computing

Serial Programming

```
program test
 double precision a, b,testFunc1
 real t0, t1, tdiff




 call cpu_time(t0)
 a = 1.0000000d0
 b = testFunc1(a)




 call cpu_time(t1)


 tdiff = t1 - t0
 write(6,*) a, b, tdiff
 call exit
end



double precision function testFunc1(a)
 double precision a, testFunc2, tot
 integer i
 tot = 0.d0


do i = 1, 1024
   a = testFunc2(a)
   tot = tot + a
 enddo



 testFunc1 = tot
 return
end

double precision function testFunc2(a)
     double precision a, testFunc3
     integer j
      a = (a * 3.d0) / 3.d0
      do j = 1, 10000
        a = testFunc3(a)
      enddo
      testFunc2 = a
      return
end

double precision function testFunc3(a)
 double precision a
 integer i
 do i = 1, 10000
  a = (sqrt(a * a) * 3.d0) / 3.d0
 enddo
 testFunc3 = a
 return
end
```

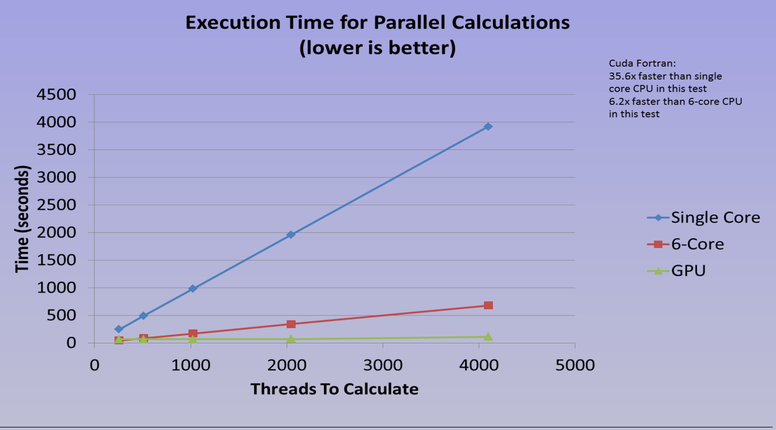
Parallel Programming

```
program testOpenMP
 double precision a, b,testFunc1
 real t0, t1, tdiff




 call cpu_time(t0)
 a = 1.0000000d0
 b = testFunc1(a)




 call cpu_time(t1)


 tdiff = t1 - t0
 write(6,*) a, b, tdiff
 call exit
end



double precision function testFunc1(a)
 double precision a, testFunc2, temp, tot
 integer i
 tot = 0.d0
!$OMP PARALLEL SHARED(tot) PRIVATE(temp)
!$OMP DO
  do i = 1, 1024
    temp = testFunc2(a)
!$OMP ATOMIC
    tot = tot + temp
  enddo
!$OMP END DO
!$OMP END PARALLEL
 testFunc1 = tot
 return
end

double precision function testFunc2(a)
 double precision a, testFunc3
 integer j
 a = (a * 3.d0) / 3.d0
 do j = 1, 10000
   a = testFunc3(a)
 enddo
 testFunc2 = a
 return
end

double precision function testFunc3(a)
 double precision a
 integer i
 do i = 1, 10000
  a = (sqrt(a * a) * 3.d0) / 3.d0
 enddo
 testFunc3 = a
 return
end
```

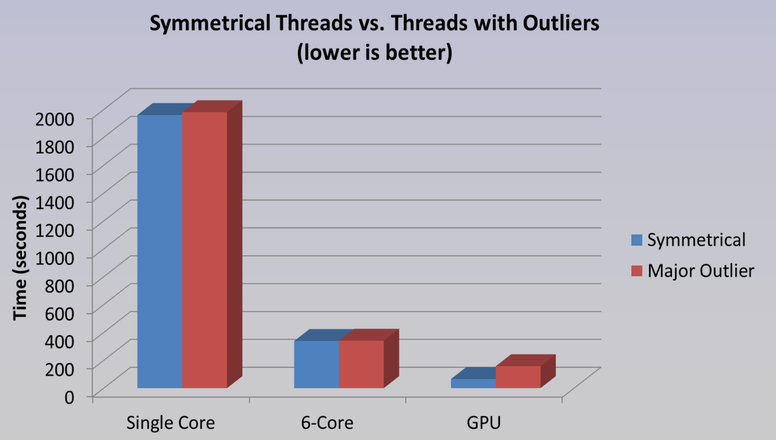
GPU Programming

```
program testCuda
 use cudafor
 use cudaFuncs
 integer, parameter :: NSIZE=1024
 integer, parameter :: BSIZE=256
 double precision b
 double precision, dimension(NSIZE)        :: a
 double precision, dimension(NSIZE), device :: aDev
 type(dim3) :: grid, block
 real t0, t1, tdiff
 integer i

 block = dim3(BSIZE,1,1)
 grid = dim3((NSIZE+BSIZE-1)/BSIZE,1,1)
 call cpu_time(t0)
 aDev = 1.0000000d0
 call testFunc1<<<grid,block>>>(aDev,NSIZE)
 a=aDev
 do i = 1, NSIZE
   b = b + a(i)
 enddo
 call cpu_time(t1)


 tdiff = t1 - t0
 write(6,*) a(1), b, tdiff
 call exit
end

c    % cat testFuncs.cuf
module cudaFuncs contains

 attributes(global) subroutine testFunc1(a,nsize)
  double precision, dimension(:) :: a
  integer, value                 :: nsize
  integer i
  i = (blockIdx%x-1)*blockDim%x + threadIdx%x

  if (i .le. nsize) then
   a(i) = testFunc2(a(i))
  end if



 end

double precision attributes(device) function testFunc2(a)
  double precision a
  integer j
   a = (a * 3.d0) / 3.d0
   do j = 1, 10000
    a = testFunc3(a)
   enddo
   testFunc2 = a
   return
end

double precision attributes(device) function testFunc3(a)
 double precision a
 integer i
 do i = 1, 10000
  a = (sqrt(a * a) * 3.d0) / 3.d0
 enddo
 testFunc3 = a
 return
end
end module cudaFuncs
```

Results



- Outliers have a more significant impact on time in GPUs than other processing methods.



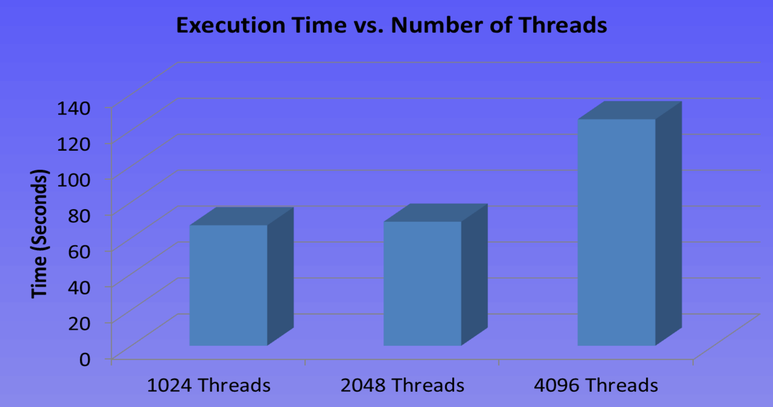
In GPUs processing time is approximately doubled, while it is essentially unaffected in the other 2.

(2048 threads with thread 71 taking 250% of typical execution time. Cuda executing with a block size of 32.)

Maximizing Performance in Cuda

- Two methods for maximizing performance.
  - Change the number of threads.
    - Increasing the overall number of threads helps minimize idle cores.
  - Change the thread block size.
    - Decreasing block size minimizes the number of streaming multi-processors used by a block, which reduces the number of idle cores when an outlier is encountered.

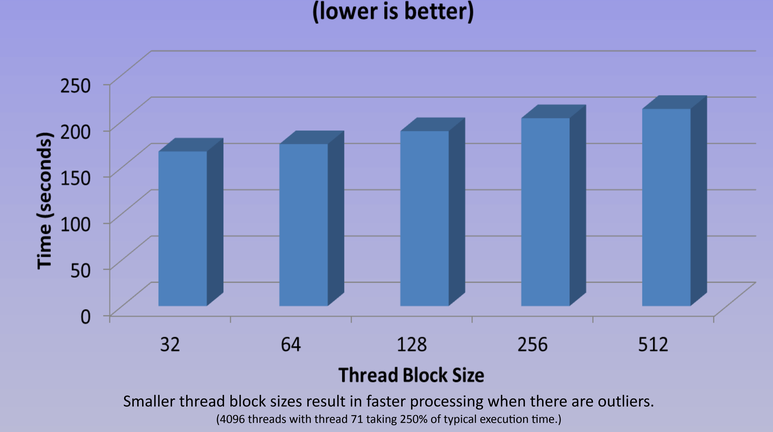
- GPUs need a large number of threads to maximize performance.



No significant change in time going from 1024 to 2048 threads, which is double the work for the same time. Therefore a multiple of 2048 threads should always be used with current GPUs. This is because the GPU has enough cores that it can process 1024 to 2048 threads concurrently. Once you exceed 2048 threads the work is done in 2 batches which doubles the time it takes to complete.

(Thread block size of 256)

- Shrinking the block size can affect performance in GPUs by up to 33% when outliers are present.



Smaller thread block sizes result in faster processing when there are outliers.

(4096 threads with thread 71 taking 250% of typical execution time.)

Conclusions

- When you have calculations that are not perfectly parallel, Cuda can suffer a significant slowdown due to the way the GPU works.
  - This slowdown can be mitigated by increasing the number of threads and reducing thread block size to reduce idle cores on the GPU.
- Single and multi-core microprocessors are better able to handle outliers than GPUs due to the greater independence of CPU cores.
- Current code can run up to 35 times faster if modified to run in parallel on a GPU.

Future

- Continue to rewrite existing computational research code to take advantage of GPUs to improve the speed, scope and accuracy of research.
- Write new computational research code to do things which weren't possible before with the amount of computational power prior to GPU programming.

Acknowledgements

- UWEC Office of Research and Sponsored Programs
- Summer Research for Undergraduates Program
- Differential Tuition Funding for Student Research Day Poster Printing