R Benjamin Clay
rbclay@ncsu.edu
HW 4
CSC 548
10/27/09

Problem Description

The NAS Parallel Benchmark standard's Integer Sort (IS) benchmark [1] describes a bucket sort of random integers. The particular implementation we examine is written in C and uses MPI for communication.

Using the mpiP tool [2], we see that the Class B test takes an average of 9.53s to complete, and spends approximately 70% of program execution time performing MPI tasks. The Class C test takes an average of 37.13s to complete, and spends approximately 65% of time performing MPI tasks. The Class D test required too much memory to run on the test systems.

Using the gprof tool [3], we obtain the following partial results:

Class B				Class C			
% time	Duration(s)	Calls	Function	% time	Duration(s)	Calls	Function
39.58	0.67	8388631	randlc()	43.57	3.32	11	rank()
38.10	0.64	11	rank()	33.92	2.59	33554457	randlc()
12.50	0.21	1	full_verify()	10.50	0.80	1	full_verify()

From these results, we see that the functions randlc() and rank() are the two most costly functions in terms of time. Furthermore, randlc() is high in cost because it is called so many times, while rank() takes a very long time to execute once.

Proposed Solution

Of the two most costly functions, the rank() function is best suited for hardware acceleration. It is the core compute kernel for the Integer Sort benchmark, which performs the bucket sort, while randlc() is part of the initialization routine. rank() is called serially 11 times on each MPI node, which could potentially be parallelized using NVIDIA CUDA [4] kernels. Each rank() call would be executed by a single CUDA thread, which would perform MPI synchronization with equivalent CUDA threads on other MPI hosts using the cudaMPI library [5].

One issue with this approach is that the rank() function contains two MPI calls that are not supported by cudaMPI, or DCGN [6], another CUDA-MPI library: MPI_Alltoall() and MPI_Alltoallv(). These function calls occur one after the other roughly in the middle of rank(), presenting the opportunity to split rank() into two CUDA kernels, rank1() and rank2(), and allowing the x86 host to perform these non-supported MPI calls in between the two CUDA kernels. This will add DMA overhead, since values need to be moved from CUDA memory to host and vice versa, but there doesn't appear to be any viable alternative. Even if the IS benchmark was accelerated using Cell, the Cell Messaging Layer [7] also does not support Alltoall() or Alltoallv(), making host execution of these calls a necessity.

R Benjamin Clay
rbclay@ncsu.edu

Timeline

Day 2
- Split rank() function into rank1() and rank2(), performing MPI_Alltoall() calls in main()

Day 8
- Port rank() functions to CUDA kernel code
- Replace rank() MPI calls with corresponding cudaMPI calls

Day 9
- Evaluate array dependencies and effect of executing rank() functions in parallel

Day 12
- Replace serial execution of rank() functions with parallel execution via multiple CUDA threads

Day 13
- Compare performance to original results

Day 14
- Document all coding efforts

Web Page

http://www4.ncsu.edu/~rbclay/

References

[1] NAS Parallel Benchmark. http://www.nas.nasa.gov/Resources/Software/npb.html

[2] mpiP tool. http://mpip.sourceforge.net/

[3] gprof tool. http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html

[4] NVIDIA CUDA. http://www.nvidia.com/object/cuda_home.html

[5] cudaMPI library. http://www.cs.uaf.edu/sw/cudaMPI/

[6] DCGN library. http://jeff.bleugris.com/journal/projects/

[7] Cell Messaging Layer library. http://www.ccs3.lanl.gov/~pakin/software/cellmessaging/