PROCEEDINGS OF SCIENCE

QUDA programming for staggered quarks

Steven Gottlieb*
National Center for Supercomputing Applications, University of Illinois and Indiana University, Bloomington, IN 47405, USA
E-mail: sg@indiana.edu

Guochun Shi
National Center for Supercomputing Applications, University of Illinois
E-mail: gshi@ncsa.illinois.edu

Aaron Torok
Indiana University, Bloomington, IN 47405, USA
E-mail: amtorok@indiana.edu

Volodymyr Kindratenko
National Center for Supercomputing Applications, University of Illinois
E-mail: kindr@ncsa.illinois.edu

We have been extending the QUDA GPU code developed at Boston University to include the case of improved staggered quarks. Improved staggered quarks such as asqtad and HISQ require both first and third nearest neighbor terms in the Dirac operator. We call the corresponding links fatlinks and longlinks. The fatlinks are not unitary, and staggered phases are included in the links, so link reconstruction techniques may either be inapplicable or require modification. A single precision inverter using compressed storage for the longlinks achieves a speed of 100 GF/s on an NVIDIA GTX 280 GPU on a $24^3 \times 32$ lattice.

In addition to the inverter code, we have code for fatlink computation, gauge force and fermion force. They run at 170, 186 and 107 GF/s, respectively, for similar conditions to the solver speed above. The single GPU code is currently in production on NCSA's AC cluster for the study of electromagnetic effects. The double precision multimass solver is running at 20 GF/s, about 80% of the speed of an 8-node or 64-core job on Fermilab's jpsi cluster. The AC cluster has C1060 Tesla boards with lower memory bandwidth than the GTX 280, where the DP inverter runs at 33 GF/s. Multi-GPU code is in development.

The XXVIII International Symposium on Lattice Field Theory, Lattice2010
June 14-19, 2010
Villasimius, Italy

*Speaker.

PoS(Lattice 2010)026

1. Introduction

The MILC Collaboration has had a very long relationship with the National Center for Supercomputing Applications (NCSA) dating to the early 1990s. In addition to providing cycles, NCSA has been helpful with code development. For example, NCSA's Innovative Systems Laboratory was responsible for porting the MILC code to IBM's Cell Broadband Engine [1]. As interest in the Cell/B.E. waned and interest in GPU computing increased, efforts to port MILC to the latter began. Initial efforts at long distance collaboration mostly using student labor met with very limited success [2]. However, a sabbatical provided an opportunity starting in August, 2009 for all of us to be in the same place and for much more rapid progress.

The Boston University group has been developing GPU code for some time [3] for a Wilson/Clover inverter. Their approach is known as QUDA. A workshop at Jefferson Laboratory provided an opportunity for two of us to meet with a number of other developers. As most of the MILC work has been using improved staggered or asqtad quarks, it was natural to extend QUDA to include support for staggered quarks.

2. New staggered code

Our initial effort was directed at writing a Dslash operator for a single GPU. Once that was running, a conjugate gradient solver was written and then extended to a multimass solver. The next piece of code to be tackled was the fat link computation. After the fat link code was completed, the gauge and fermion force routines were ported to the GPU. Finally, wrappers were written to allow the MILC code to call the GPU routines rather than doing the computation on the CPU for each of the above phases of the code. These wrappers were designed to take care of all the work of tranforming MILC's data structures to those required for the GPU, sending the input information to the GPU, retrieving the results and placing them into the normal MILC format.

This code development at NCSA was done independently of ongoing developments at BU. There were changes to the QUDA code that required some effort to merge the Wilson and staggered codes. A private version of the merged code was available to the developers for some time; however, public release of QUDA version 0.3 (the version that includes support for both Wilson and staggered quarks on a single GPU) did not occur until October 1, 2010.

Given the size of the configurations MILC has been generating and the limited memory on a single GPU, it is important to develop multi-GPU code. This had already been done for Wilson type quarks [4]. Multi-GPU code for staggered quarks is working, and we report here on benchmark results. So far, the lattice is only cut in the time direction. Communication and computation are overlapped by employing both interior and exterior kernels. If time slices 0, 1, ..., $T_l - 1$, are assigned to the node, the interior kernel completes contributions from all directions on time slices 3, 4, ..., $T_l - 4$. Because of the Naik term which extends three links in the time direction, the smallest three and largest three times on each node have off-node neighbors in the time direction. For these time slices, the interior kernel also computes all the terms with spatial links. Once all the required off-node spinors have arrived, the exterior kernel computes all the contributions that depend on those spinors. The multi-GPU code is not in QUDA 0.3.

Type	Cores	BW (GB/s)	SP (GF/s)	DP (GF/s)	RAM (GB)
GTX 280	240	142	933	78	1.0
GTX 285	240	159	1062	88	1–2
Tesla C1060	240	102	933	78	4.0
Tesla S1070	four copies of above				
Fermi GTX 480	480	177	1345	168	1.5
Fermi C2050	448	148	1030	515	3.0

Table 1: Characteristics of systems studied, including model type, number of cores, peak bandwidth of GPU memory, peak floating point speed in single and double precision, and total GPU memory.

There are other important differences between the Wilson and staggered codes. We have already mentioned that the Naik term requires more planes of spinors from neighbors. The Naik term, or course, requires additional storage for the long links. Each long link is the product of three $SU(3)$ matrices, so it is an element of $SU(3)$. This means that the reconstruction methods implemented in QUDA can be applied to the long links. In contrast, the fat links that are part of improved staggered actions are not elements of $SU(3)$, so they are not compressed in the GPU. The reconstruction methods reduce the 18 operands required for a complex 3×3 matrix to either 12 or 8 operands [3]. In the former case, unitarity is used to compute the third row as a cross product of the first two rows. In the latter case, the computation of the entire matrix from only 8 parameters requires more floating point operations. To summarize, for Wilson quarks each of the 4 links is stored as 18, 12 or 8 operands. In the staggered case, the fat links are stored as 18 operands, and the long links are stored as 18, 12 or 8 operands.

3. Benchmarks

Several different models of NVIDIA GPUs are available for running benchmarks or for production running. At NCSA, there are systems with GTX 280, Tesla S1070 and Fermi GTX 480 GPUs. At Jefferson Lab, the GTX 285, Tesla C1060, S1070 and Fermi GTX 480 are available. Fermilab has systems with the Tesla S1070. At NERSC, there are nodes with Tesla C1060 and Fermi C2050 GPUs. The Fermi GPUs are the most recent, and only the C2050 supports error correction. Table 1 details the important characteristics of each of the six GPU models that we have run on.

In Table 2, we give the performance of the single mass and multimass conjugate gradient solvers. Four masses are used for the multimass case. This table contains results for double precision, single precision and half precision solvers [5]. The first column indicates how the long links are reconstructed, with 18 denoting no reconstruction, *i.e.*, all 18 operands of each link matrix are stored on the GPU. Note that the fat links are always stored as 18 operands and that in single precison and half precision compressed storage helps, but that for double precsion, reconstruction reduces performance. The peak double precision floating point speed on the GTX 280 is only a small fraction of the single precision floating point speed, so there are not as many spare flops

precision	reconstruct	CG(GF/s)	multimass CG (GF/s)
DP	12	31	31
	8	15	16
	18	33	34
SP	12	98	92
	8	108	96
	18	83	80
HP	12	123	106
	8	128	113
	18	108	98

Table 2: Performance of single and multimass CG solvers for a $24^3 \times 32$ lattice on a GTX 280. Different precisions and reconstruction techniques for the long links are shown.

to use for the reconstruction. Also, this table does not reflect additional iterations that might be required for the lower precision solvers.

For gauge configuration generation, additional routines such as the fat link, gauge force and fermion force computation are necessary. Table 3 contains results for single mass and multimass CG solvers, as well as the fat link, gauge force and fermion force. In each case, we have the speed of the computation without the overhead of copying the initial data to the GPU and copying the result back to the CPU, the speed we could expect to get if we can achieve 100 GB/s GPU memory bandwidth (about 2/3 the peak) and the performance including the overhead of copying data to and from the GPU. That overhead can be substantial for the fat link and gauge force computations, so it is advantageous to arrange a production job so that the gauge links can remain resident in the GPU throughout the job, and they are only copied back to the GPU when needed there. All benchmarks here are for single precision on a $24^3 \times 32$ lattice on a GTX 280. The CG solver used 500 iterations and 12-reconstruct was used when possible. Table 4 compares CG speeds using varions precisions and reconstruction methods on different hardware and shows the cost of taking advantage of error correction on the C2050 processor.

We also have some weak scaling results for multiGPU benchmarks on the AC cluster at NCSA (Table 5) that uses S1070 GPU servers and the Dirac cluster at NERSC (Table 6) using C2050 GPU cards. These are all done with a $24^3 \times 32$ local volume. Message passing performance is important, so we have measured the time for each phase. For example, with the GTX 280, we found that for single precision, it takes 0.29 ms $=$ 3.3 GB/s to pack the GPU data and copy to host; 0.16 ms $=$ 6.14 GB/s to complete the MPI transfer; and 0.2 ms $=$ 4.8 GB/s to transfer the data from the host to the GPU. For multiGPU running, we find that the design of the node can be very important. The AC cluster is a few years old, and it is designed with one core per GPU (four GPUs per node). Also, the S1070 is designed so that a pair of GPUs share a single PCI connection to the node, so this is an obvious point of contention. On the other hand, the Dirac cluster uses the C2050 card which has about 50% more bandwidth to GPU memory than the S1070 and no contention for the PCI bus. Further, there is only one GPU per node, two Intel 5530 quad core CPUs capable of 5.86 GT/s

	GF/s		
	standalone	goal: assuming 100GB/s	with PICe overhead
CG	98	100	71
MM-CG	92	100	71
fat link	178	168	62
gauge force	208	349	112
fermion force	111	128	94

Table 3: Performance in GF/s of single and multimass CG solvers, fat link, gauge force and fermion force computations for a $24^3 \times 32$ lattice on a GTX 280. All results are for single precion and 12-reconstruct is used when possible.

		GF/s			
	reconstruction	GTX 280	GTX 480	C2050 ECC	C2050 No ECC
DP	12	29	31	20	24
	8	15	16	11	13
	18	32	50	30	41
SP	12	92	116	66	96
	8	99	126	72	100
	18	79	104	57	86
HP	12	77	154	97	122
	8	74	157	101	123
	18	76	131	84	104

Table 4: Comparison of results for the CG solver on a $24^3 \times 32$ lattice on the GTX 280 (Tesla) and Fermi architectures. For the latter case, we have results on the GTX 480 (consumer card) and C2050 both with and without error correction. All results are in GF/s.

and 24 GB of DDR3 1066 memory. (Newer motherboards and CPUs can use DDR3 1333 memory and are capable of 6.4 GT/s.) In Table 7, we compare details of the time for internal and external kernels and the communication time. These results are all for single precision using 8 reconstruct. Note the difference in communication time between one and four GPUs on the AC cluster. This results from contention between the different messages that need to be passed at the same time. The contention is probably on the PCI bus. In the S1070 two GPUs share a common PCI bus. Note that the increased time on AC when using all four GPUs results in the communication taking so long that the GPU is stalled after completion of the interior kernel. All times are decreased on Dirac which uses the C2050 GPU and has only one GPU per node.

4. Production experience

We have been using GPUs for calculating electromagnetic effects, *i.e.*, for $SU(3) \times U(1)$. So

		# of GPUs						
	reconstruct	1	2	4	8	12	16	20
DP	12	22	22	22	21	18	17	16
	8	13	13	13	12	11	11	10
	18	23	23	23	21	18	18	17
SP	12	58	56	43	40	32	31	31
	8	65	56	40	39	32	33	32
	18	50	50	43	41	35	34	31
HP	12	61	60	40	40	33	33	31
	8	60	59	41	39	36	31	31
	18	61	59	40	40	36	29	32

Table 5: Weak scaling in study on the AC cluster at NCSA using a local volume of $24^3 \times 32$. All results are in GF/s.

		# of GPUs		
	reconstruct	1	2	4
DP	12	24	23	23
	8	13	12	12
	18	41	41	41
SP	12	96	94	93
	8	100	100	100
	18	86	83	83
HP	12	122	120	116
	8	123	120	119
	18	104	101	101

Table 6: Weak scaling study of CG performance on the Dirac cluster at NERSC using a local volume of $24^3 \times 32$. Dirac has one C2050 GPU per node and error correction was not used in this study. All results are in GF/s.

	AC: 1 GPU	AC: 4 GPUs	Dirac: 4 GPUs
interior kernel (ms)	3.15	3.13	1.92
exterior kernel (ms)	0.30	0.32	0.17
message time (ms)	1.79	3.94	1.19
Dslash time (ms)	3.47	4.34	2.10

Table 7: Details of Dslash timing including the times for the interior and exterior kernels, the message passing time and total time. Three conditions are contrasted, AC using one GPU, AC using all four GPUs on the node and Dirac using four nodes, each of which has one GPU.

far, we have only been using ensembles that fit in a single GPU. About 4,000 configurations have been analyzed of size $20^3 \times 64$ or $28^3 \times 96$. The physics results from this analysis are presented in a talk by Aaron Torok [6]. Production runs have been done on the AC cluster at NCSA, the Dirac cluster at NERSC, and at Fermilab. As an example, running on CPUs only, a job takes 6.04 node-hrs or 48.2 core-hrs, whereas running on a GPU 1.49 node-hr are required. Since the other cores are idle, this is 11.9 core-hrs on the same cluster.

5. Where to get the code

Version 0.3 of QUDA which integrates staggered and Wilson/clover codes was released on October 1, 2010. It can be downloaded from http://lattice.bu.edu/quda. CUDA version 3.0.14 is required. We plan to release multiGPU code later, so if you are interested in staggered code, you will need to contact one of authors of this paper. We also expect to have an svn repository in the future.

6. Future

To complete the study of electromagnetic effects, we will need to run multiGPU production jobs for some of the larger lattices. For those with spatial size 56^3 or 64^3, we are likely to need code that cuts the lattice in both space and time dimensions. Otherwise, these ensembles will need to be analyzed on clusters or supercomputers. We also are interested in trying to do analysis of heavy-light mesons with GPUs. In this case, we will need code that can treat both clover and staggered quarks for multiGPU jobs. Thus, this is a high priority for the next version of QUDA. Also, the MILC lattice generation program for asqtad is essentially over, and modifications to the code to accommodate HISQ quarks need to be made. So, there is quite a bit of code development that remains to be done. Beyond the coding, we also need to investigate strong scaling as supercomputers are now reaching for petaflop/s performance. Can we efficiently run GPU jobs that make use of 100s of GPUs, not just a few, as we are currently doing? It is essential to decide what other parts of our production running can profitably be shifted to GPUs.

References

[1] G. Shi, V. Kindratenko and S. Gottlieb, PoS(LATTICE 2008)026, [arXiv:0910.0262 [hep-lat]].

[2] D. Roeh, J. Troup, G. Shi, V. Kindratenko, "Porting MILC to GPU: Lessons learned", Workshop on using GPUs for LQCD, August 19-21 2009, Thomas Jefferson NAF, Newport News, Virginia, http://www.ncsa.illinois.edu/~kindr/projects/hpca/files/jlab_QCD_on_GPU_presentation.pdf .

[3] K. Barros, R. Babich, R. Brower, M. A. Clark and C. Rebbi, PoS(LATTICE 2008)045 [arXiv:0810.5365 [hep-lat]].

[4] R. Babich, M. A. Clark and B. Joo, Proc. Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (Supercomputing 2010), New Orleans, LA, Nov. 2010.

[5] M. A. Clark, R. Babich, K. Barros, R. C. Brower and C. Rebbi, Comput. Phys. Commun. **181**, 1517 (2010) [arXiv:0911.3191 [hep-lat]].

[6] A. Torok *et al.*, PoS(LATTICE 2010)127.