Progress on the QUDA code suite

Ron Babich, Richard Brower
Center for Computational Science, Boston University, Boston MA 02215, USA
E-mail: `rbabich@bu.edu`, `brower@bu.edu`

Mike Clark
Harvard-Smithsonian Center for Astrophysics, Harvard University, Cambridge, MA 02143, USA
E-mail: `mikec@seas.harvard.edu`

Steven Gottlieb[*]
Physics Department, Indiana University, Bloomington, IN 47405, USA
E-mail: `sg@indiana.edu`

Balint Joó
Thomas Jefferson National Accelerator Facility,
Newport News, VA 23606, USA
E-mail: `bjoo@jlab.org`

Guochun Shi
National Center for Supercomputing Applications, University of Illinois, Urbana, IL 61801, USA
E-mail: `gshi@ncsa.uiuc.edu`

At the time of Lattice 2010, we were about to announce a distribution of the code (QUDA 0.3) that supported both Wilson/clover and improved staggered quarks for computation on a single GPU. Multi-GPU code was running for both solvers, but with the restriction of grid partitioning in only the time dimension. In the past year, we developed code that allows us to cut the lattice in all four dimensions. This allows us to scale computations to order 100 GPUs yielding multi-teraflop performance. We will present results for both types of solvers on GPU clusters and for other kernels important for physics projects. We also compare performance and cost-effectiveness of full application codes running on CPUs with our GPU accelerated code.

The XXIX International Symposium on Lattice Field Theory - Lattice 2011
July 10-16, 2011
Squaw Valley, Lake Tahoe, California

[*]Speaker.

1. Introduction

As the performance of Graphics Processing Units (GPUs) has rapidly increased in recent years and the ease of programming them has improved, they have become increasingly popular in our field. We have been developing a code suite for lattice QCD called QUDA based on NVIDIA's C for CUDA extensions to the C language [1, 2, 3]. We have been making steady progress on improving the capabilities of our code, most notably by including support for multi-GPU running [4].

We describe the current state of the QUDA code distribution and the challenges of dealing with an architecture whose balance is not ideal for lattice QCD. We also present benchmarks for our inverters and evaluate the cost-effectiveness of one production code by comparing running with GPUs and without them.

2. Current distribution

QUDA is now available through the github service at lattice.github.com/quda. Version 0.3.2 was released on January 18, 2011 and is the most recent stable release. One can also download the development version of the code from the source code repository. However, as the code is being actively developed, there is no guarantee of the correctness of the development version.

The current release includes solvers for a number of lattice quark formulations. These include Wilson/Clover, twisted mass, improved staggered (both asqtad and HISQ), and domain wall quarks. There are mixed-precision implementations of the solvers for both the conjugate gradient algorithm and for BiCGstab. Double precision, single precision and half precision (16-bit fixed-point) are supported. In addition to the solvers, the staggered implementation includes code for asqtad link fattening, as well as force terms for the one-loop improved Symanzik gauge action and asqtad fermions. There is also a multi-shift CG solver.

From github, one can download the distribution, sign up for a mailing list quda-announce, or request help.

3. Unbalanced architecture

The GPU has tremendous peak floating point performance compared with the typical CPUs in todays computers; however, the memory bandwidth of about 150 GB/s, while much greater than the memory bandwidth of the host system, is insufficient to keep the floating point units busy for applications such as lattice QCD. The PCI bus is also a crucial performance bottleneck. The bus speed of approximately 5 GB/s limits transfers of data from host memory to and from the GPU and is also a factor in passing messages from one GPU to another in multi-GPU runs.

In order to pass a message from a GPU on one node to a GPU on another node, the data must be moved five times. In the first stage, the data is copied from the GPU memory to the host. However, the data will then reside in what is called pinned memory dedicated to the GPU. There is then an extra copy to pinned memory for the Infiniband card. The data is then copied over the network to the other node, where it must be transferred from the pinned memory for the infiniband card to pinned memory for the GPU and finally into the GPU. NVIDIA and Mellanox have announced

Model (Architecture)	Cores	BW (GB/s)	SP (GF/s)	DP (GF/s)	RAM (GB)
GeForce GTX 280 (GT200)	240	142	933	78	1.0
GeForce GTX 285 (GT200b)	240	159	1062	88	1–2
Tesla C1060 (GT200)	240	102	933	78	4.0
Tesla S1070 (GT200)	four copies of above				
GeForce GTX 480 (Fermi)	480	177	1345	168	1.5
Tesla C2050 (Fermi)	448	148	1030	515	3.0
Tesla C2070 (Fermi)	448	148	1030	515	6.0

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.

a GPU Direct protocol that should eliminate the copy between the two types of pinned memory. Details of the timing without GPU Direct appear in Ref. [5].

Table 1 presents a comparison of several types of GPU hardware to which we have access. The GTX models are primarily designed for consumer graphics, while the Tesla and Fermi models with four digit designations are designed for high performance servers. The second column gives the number of compute cores in the GPU, the third column contains the peak bandwidth of the GPU memory, and the fourth and fifth columns give the peak single and double precision performance, respectively. The last column in the table has the GPU memory capacity. Note that only the Fermi C2050 and C2070 have a peak double precision speed that is half that for single precision. The other models all have much lower double precision peak performance (1/12 or 1/8 of single precision). The Tesla S1070 packages four GPUs together in such a way that pairs of GPUs share a PCI bus. Finally, only the C2050 and C2070 support error correction.

4. Grid partitioning

The production release of QUDA 0.3.2 (1/18/11) only supports partitioning the lattice in the time direction. However, the development version of the code supports cutting all four dimensions for the solvers. Benchmarks can be found in the next section. In addition, some other routines such as the asqtad fat link calculation support partitioning in all four dimensions.

5. Benchmarks

We will first present our newest results for partitioning the lattice in up to four dimensions. We also have some older results for one-dimensional partitioning on production jobs, including full application performance where some of the application is running on the CPU. In this paper, we will emphasize results for staggered quarks. A companion contribution will emphasize Wilson/Clover results [6].

Figure 1 displays total performance of the asqtad solver [4] on a $64^3 \times 192$ lattice, which is the largest grid among the MILC ensembles [7]. A mixed precision multi-mass solver was used for

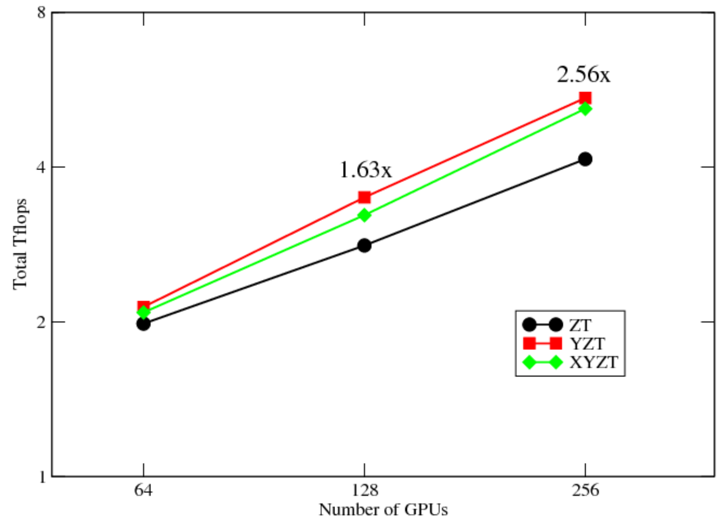

Figure 1: Performance of the asqtad conjugate gradient solver on the Edge computer vs. the number of GPUs. The problem size is fixed at $64^3 \times 192$.

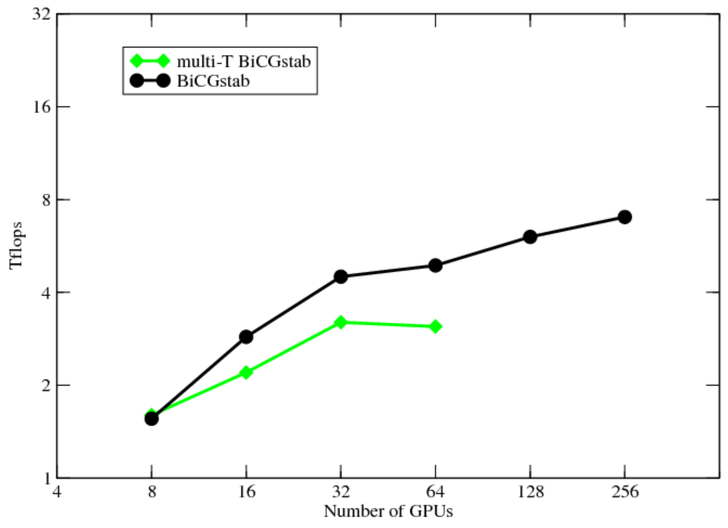

Figure 2: Performance of the clover BiCGstab solver on the Edge computer vs. the number of GPUs. The problem size is fixed at $32^3 \times 256$.

computer	size	1	2	4	4×1	8	16
J/Psi	$20^3\times64$	39.3	34.7	23.7	30.3	10.7	
J/Psi	$28^3\times96$	—	41.4	37.5	37.5	17.7	
Dirac	$28^3\times96$	—	—	—	49.5	42.6	27.6

Table 2: Results on the multimass, multiprecision asqtad inverter on Fermilab and Dirac GPU nodes. All results in GFs/GPU. Dirac has one GPU per node. At Fermilab there are two GPUs per node. The column labeled 4×1 contains results using one GPU per node.

from 64 to 256 GPUs on the Edge GPU cluster at Livermore. A speed of 5.5 TFlops was achieved with 256 GPUs. Results are shown for partitioning the lattice in two (Z,T), three (Y,Z,T) or four (X,Y,Z,T) dimensions. The best performance is achieved using the three dimensional partition. Performance here is comparable to what has been seen on several thousand CPU cores.

In Fig. 2, we show solver performance [4] on a $32^3\times256$ anisotropic Clover configuration. A mixed precision BiCGstab solver was used [8]. The performance of 8 Tflops is comparable to what has been seen on 16K cores of Jaguar (Cray XT5) or Intrepid (BlueGene/P). More details, including results for an additional algorithm with improved scaling can be found in the companion paper [6].

We have also run benchmarks on much more modest grid sizes. In Table 2, we show benchmarks for the multimass, multiprecision staggered solver on 20^3 and 28^3 lattices. The first two lines come from runs at Fermilab using S1070 GPUs servers. Each compute node is attached to two of the GPUs in the S1070. These two GPUs share the PCI bus, so we ran four GPU benchmarks using two different setups. In the column marked 4, we use two nodes with two GPUs in each node. In the column marked 4×1, we used four nodes with just one GPU per node. By comparing results in these two columns we can see whether sharing the PCI bus leads to reduced performance. We note that for the smaller grid there is a big difference: 23.7 GF/GPU *vs*. 30.3 GF/GPU. However for the larger grid, performance is identical. For the smaller grid with four GPUs, the local volume is $20^3\times16$, whereas for the larger grid it is $28^3\times24$. In the former case, there is not enough local work to allow overlap of communication with computation when the communication is slowed by the shared bus, but in the latter case, the message passing can be overlapped with computation. The Dirac cluster at NERSC has two quad-core 2.4 GHz Nehalem CPUs and one C2050 GPU per compute node. The C2050 has 3 GB of memory per GPU and the larger lattice requires a minimum of four GPUs. We see quite reasonable performance on up to 8 GPUs. For the runs with 8 GPUs, the lattice was cut in the Z and T dimensions. With 16 GPUs, it was also cut in the Y dimension.

We have also done some tests of asqtad link fattening. These tests were run on Longhorn, a GPU cluster at the Texas Advanced Computing Center. Each node has two quad-core 2.53 GHz Nehalem processors and two FX5800 GPUs. The link-fattening code is run in double precision. In Table 3, we show results for a $64^3\times192$ lattice on Longhorn and a $48^3\times144$ lattice on Dirac. In each case, we compare the speed of running on all the CPU cores with running on the GPUs. On Longhorn the GPU code is about 3.2 times faster than the CPU code, and on Dirac the speedup is about 3.5.

nodes	**cores**	**sublattice**	**GF/core**	**GPUs**	**sublattice**	**GF/GPU**
Longhorn Quadro FX5800 $64^3 \times 192$						
24	192	$16^2 \times 32^2$	0.72	48	$64 \times 32^2 \times 16$	9.2
Dirac C2050 $48^3 \times 144$						
32	256	$12^2 \times 24 \times 18$	0.54	32	$48 \times 24^2 \times 18$	15.3

Table 3: Performance of the asqtad link fattening code on Longhorn at TACC and Dirac at NERSC. Performance is either per core or per GPU. Longhorn has two GPUs per node and Dirac has one. Details of the CPUs may be found in the text.

6. Cost-effectiveness

We wanted to compare the cost-effectiveness of the CPU and GPU codes. The FNAL J/Psi cluster is a computer where we know the cost of the nodes with or without the GPUs. This cluster was purchased on October, 2008 and March, 2009. Don Holmgren supplied us with the cost information. The base price of an 8-core node is \$2213. A GPU node has extra memory and 2 GPUs at a cost of \$672 and \$3222, respectively. Thus, the total cost of the GPU node is \$6107. Assuming 8,700 hours of operation per year and 3 years of running, the capital cost is 0.0106 \$/(core-hr) for CPU running and 0.117 \$/(GPU-hr) for GPU running.

In Table 4, we compare the run time of a production job studying electromagnetic corrections on a $28^3 \times 96$ lattice. The first column of the table gives the number of cores or GPUs. the second column is the time for the job, and the third is either core-hr or GPU-hr to run the job. The fourth column is the cost of the job. In each section of the table, rows correspond to running on 1, 2, 4 and 8 nodes. We see that the runs with GPUs are more cost-effective. We also note that one or two node runs are best for the GPU runs and that the cost-effectiveness on the CPU runs is not increasing as steeply when we run on more nodes. For one or two nodes, the GPU runs are about 3.75 times more cost-effective than the CPU runs, and they are about ten times faster.

7. Conclusions

We find that for large lattices we can run jobs using up to 256 GPUs when our multi-dimensional partitioning is used. Multi-dimenional partitioning is essential for some of the large spatial volume ensembles we have archived. We are able to achieve multi-teraflop speeds with both Wilson/clover and staggered quarks.

We have also seen that GPU clusters can be several times more cost-effective than CPU only clusters. This was seen for the J/Psi cluster at Fermilab which is part of the USQCD Lattice Computing Project for which we have cost data. As we gain experience with more production jobs on various hardware, we will be better able to optimize the design of our GPU clusters. The bottleneck of the PCI bus can be significant and it is important not to overload that bus by having too many GPUs share it.

There is much opportunity for additional code development, performance tuning, and performance modeling. The QUDA suite is open source, and we encourage you to join in the fun (and effort) of enhancing it and using it.

CPU code			
cores	seconds	core-hr	cost ($)
8	27091	60.20	0.64
16	13954	62.02	0.66
32	8463	75.23	0.80
64	4236	75.3l	0.80
GPU code			
GPUs	seconds	GPU-hr	cost ($)
2	2566	1.43	0.17
4	1411	1.57	0.18
8	1105	2.46	0.29
16	985	4.38	0.51

Table 4: Comparison of cost to run Fermilab's J/PSI cluster either using the CPU alone or the GPU for the solver and the CPU for the rest of the application. The number of cores or GPUs is in the first column. The second column has the run time in seconds, and the third column converts this to core-hr or GPU-hr. The final column has the cost to run the job, assuming a three year lifetime for the computer.

8. Acknowledgements

We are grateful to LLNL (Edge), NERSC (Dirac), FNAL (J/Psi) and TACC (Longhorn) where we ran benchmarks. This work was supported in part by NSF grants OCI-0946441, OCI-1060012, OCI-1060067, and PHY-0555234, as well as DOE grants DE-FC02-06ER41439, DE-FC02-06ER41440, DE-FC02-06ER41443, DE-FG02-91ER40661, and DE-FG02-91ER40676. BJ additionally acknowledges support under DOE grant DE-AC05-06OR23177, under which Jefferson Science Associates LLC manages and operates Jefferson Lab. GS is funded through the Institute for Advanced Computing Applications and Technologies (IACAT) at the University of Illinois at Urbana-Champaign.

References

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

[2] S. Gottlieb, G. Shi, A. Torok and V. Kindratenko, PoS(LATTICE 2010)026.

[3] R. Babich, M. A. Clark and B. Joo, arXiv:1011.0024 [hep-lat].

[4] R. Babich, M. A. Clark, B. Joo, G. Shi, R. C. Brower and S. Gottlieb, arXiv:1109.2935 [hep-lat].

[5] S. Shi, S. Gottlieb, A. Torok, V.V. Kindratenko, Proc. IEEE Intl. Parallel & Distributed Processing Symposium; 2011, IEEE Press.

[6] M. A. Clark, R. Babich, R. Brower, S. Gottlieb, B. Joó, G. Shi, PoS(LATTICE 2011)029.

[7] A. Bazavov *et al.*, Rev. Mod. Phys. **82**, 1349 (2010) [arXiv:0903.3598 [hep-lat]].

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