The Status of User Software on QCDOC

Chulwoo Jung

(Brookhaven National Laboratory/Columbia University)

For QCDOC collaboration

June 23th 2004, Fermilab

- Overview of SciDAC softwar activities
- Performance of Level 3 routines
- Programming Environment
- Future Activities

QCDOC Collaboration

Columbia University:	N. Christ, R. Mawhinney, A. Yamaguchi, H. W. Lin, G. F. Liu, X. Liao, C. Kim, L. Levkova, S. Cohen, S. Li
UKQCD:	P. Boyle, B. Joo, M. Clark
BNL:	C. Jung, K. Petrov
RBRC:	T. Wettig(Regensburg), S. Ohta (RIKEN)
IBM Yorktown:	A. Gara, D. Chen

Hardware overview of QCDOC

(Planery talk by P. Boyle)

QCDOC should be able to sustain performance when the number of processors increases to several thousands.

- Use On-Chip Technology to combine computation and communication units in one chip (QCDOC = QCD On a Chip)
- ASIC (Application-Specific Integrated Circuit) combines existing IBM components and QCDOC-specific, custom-designed logic:
  - 500 MHz PowerPC 440 processor core with 64-bit, 1 GFlops FPU (32+32kB L1 cache)
  - 4 MB on-chip memory (embedded DRAM), accessed through custom-designed prefetching eDRAM controller (PEC)
  - 6-dimensional nearest-neighbor, low latency serial communications unit (SCU) with aggregate bandwidth of 12 Gbit/s with software partitioning capability for physics communications.
- Auxiliary fast ethernet network for booting, IO, and diagnostics.
- $\sim$ 1 US-$ per sustained MFlops, Scalable to several 10 TFlops

SciDAC Software Program

Software objective: Create a unified programming environment that will enable US lattice community to achieve high efficiency on diverse multi-teraflop scale hardware.

Existing codes for lattice QCD (CPS, MILC, Chroma/SZIN, ...) can be run on various hardwares by adopting to programming environment provided by SciDAC Software effort.

Participant list

Arizona	Doug Toussaint	Illinois	Celso Mendez
Arizona	Eric Gregory	Illinois	Daniel Reed
Boston U.	Richard Brower	Jlab	Robert Edwards
Boston U.	Hartmut Neff	Jlab	Chip Watson
BNL	Chulwoo Jung	Jlab	Walt Akers
BNL	Konstantin Petrov	Jlab	Jie Chen
Columbia	Bob Mawhinney	MIT	John Negele
Fermilab	Don Holmgren	MIT	Andrew Pochinsky
Fermilab	Jim Simone	Utah	Carleton DeTar
Fermilab	Amitoj Sing	Utah	James Osborn

QCD-API on QCDOC

- Level 3: Highly optimized routines for Lattice QCD (QOP)

  Currently available in Columbia Physics System (CPS):

  - Dirac matrix inverter
    - Wilson, Clover, DWF (P. Boyle)
    - Staggered (C. Cristian)
    - Asqtad (Improved staggered) (CJ, H. Lin)
  - Asqtad force term (CJ, Z. Sroczynski)

  Asqtad inverter has beend tested with MILC code (CJ, E. Gregory).

  Level 3 interface is being implemented for Asqtad inverter/force term (CJ).

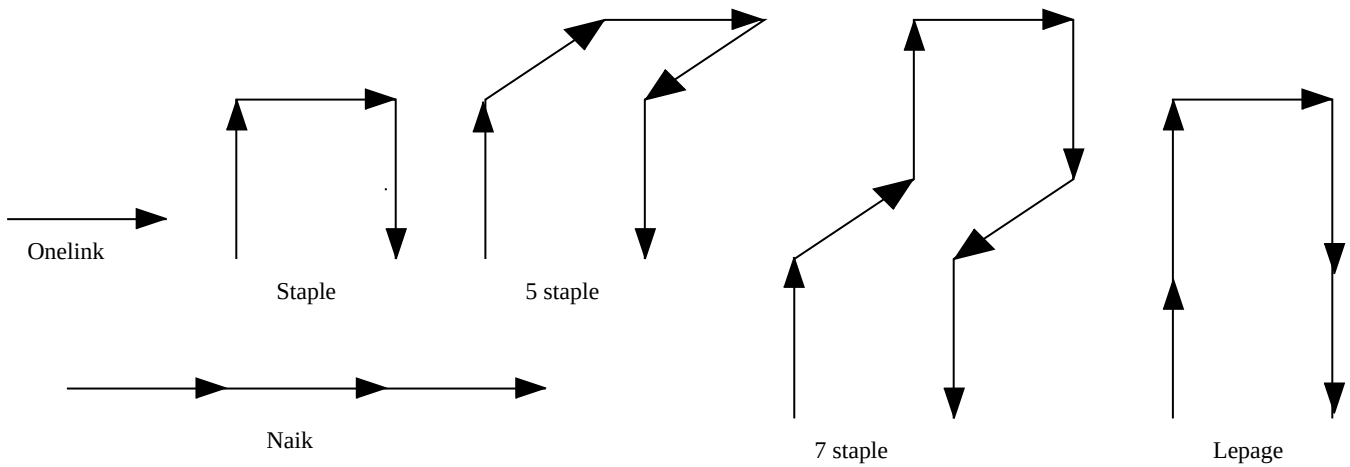
  Asqtad HMC is being tested and also being used for hardware testing.

  Wilson Dslash is also integrated into Chroma/QDP++(P.Boyle, B. Joo).

- Level 2:
  - QCD Data Parallel QDP/QDP++: Lattice-wide operations with both communication and computation (J. Osborn, B. Joo, R. Edwards)
    example: parallel transport $\chi'(x) = U_\mu(x)\chi(x+\hat{\mu})$
    Both QDP(C) and QDP++(C++) has been tested with QMP.
  - QIO: Lattice data IO (B. Joo, K. Petrov)
- Level 1:
  - QMP: Internode communication (CJ)
  - QLA: Single node linear Algebra (J. Osborn).

Improved Staggered fermion action (Asqtad)

Asqtad action is one of the improved staggered fermion action, with smaller lattice spacing error and flavor mixing.



Computation counts for various terms needed for HMC (per site):

Dirac inteverter: $\sim (1150 \times (n \sim 1000))$

Fermion Force term: $\sim$ 434,000 for 2+1 flavor. Calculation of the force term mostly consists of parallel transport and SU(3) vector outer products.

Parallel transport: $\chi'(x) = U_\mu(x)\chi(x+\hat{\mu})$

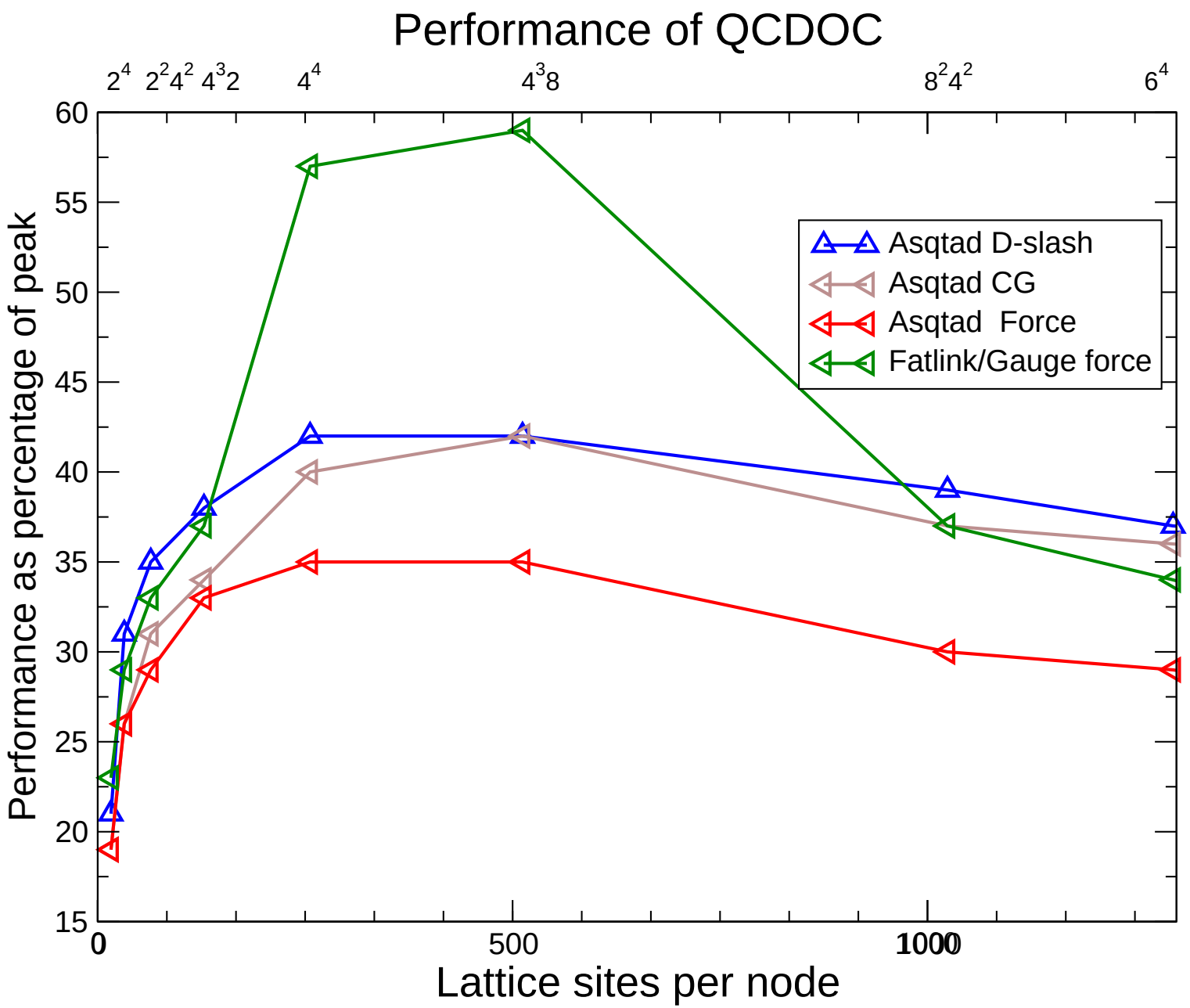
Vector outer product: $P(x) = \chi(x)\chi'^{\dagger}(x)$

Fatlink : $\sim 60,000$ Gauge Force: $\sim 150,000$

Performance of QCDOC

	Sites/node	Optimized			MILC	MILC/QDP
Precision		Double			Single	
Application		D	D	CG	CG	
Machine size		Sim	128	128	128	16
Asqtad Dirac/CG	2^4		22%	19%		2%
	4^4	43%	42%	40%	9.5%	15%
	6^4		37%	36%	13.6%	19%
	8^4		29%	28%	12.3%	
Machine size		128			128	
Asqtad Force	2^4	19%				
	4^4	35%			10.0%	
	6^4	29%			10.8%	
	8^4	20%			8.2%	
Asqtad fatlink/ Gauge Force*	2^4	23%				
	4^4	57%			7.2%	
	6^4	34%			8.1%	
	8^4	27%			8.5%	

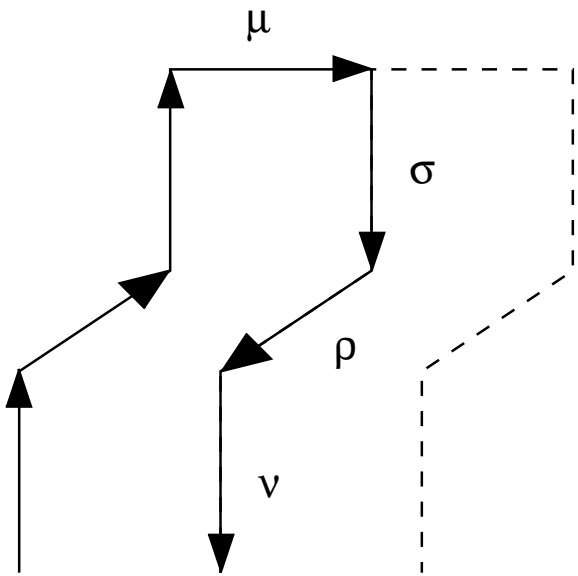
(*):multi-dimensional SU(3) parallel transport


Performance of QCDOC
2^4 $2^2 4^2$ $4^3 2$ 4^4 $4^3 8$ $8^2 4^2$ 6^4
Performance as percentage of peak
60
55
50
45
40
35
30
25
20
15
0
500
1000
Lattice sites per node
Asqtad D-slash
Asqtad CG
Asqtad Force
Fatlink/Gauge force

Optimization strategy for QCDOC

- For most computation routines, memory bandwidth is bottleneck (EDRAM/DDR $\rightarrow$ L1 cache $\rightarrow$ CPU). Changing to single precision only improves memory $\rightarrow$ L1 bandwidth.
- Sequential read access maximizes memory bandwidth. (Scatter don't gather!)
- Assembly routines are written by C++ progams (CJ, P. Boyle(BAGEL))
- For asqtad dirac, computation is divided into local/non-local. (Usual forward-backword split increases the amount of fermion fields communication)
  - Start communication
  - Local computations
  - Wait for communication
  - Non-local computations

For the force term and fatlink generation, multi-dimensional parallel transport is used to improves communication bandwidth. This should be balanced with memory usage.



- For example, 7 plaquette with $+x, -y, -t, +z$ can be calculated simultaneousely with $+y, -z, -x, +t$, $-x, +y, +t, -z$, etc. Also, you can save on number of computations by calulating $\pm\mu$ simultaneously.
- Using multi-directional parallel transport whenever possible is a fast way to generate reasonably optimized code for small local volume. (Strategy for QDP?)

Programming Environment

- QOS - Custom-designed Operating system (P. Boyle)
  - Boot (Ethernet/JTAG)
  - Load/Run Apps (Ethernet)
  - Hardware diagnostics
  - Command line argument passing
  - File I/O - NFS heavily tested (currently to the front end)
  - Shell support (qcsh : tcsh + QCDOC-specific commands) (K. Petrov/ P. Boyle)
  - Profiling, Source level debugging
- Standard GNU tools (Compiler, Linker, Binutils...) at the front end
- Commercial PowerPC tools
  - High-performance compilers - XLC
  - Multi-processor debugger - Riscwatch

Brief description of QMP

The goal of QMP is to provide portable, low-latency, high-bandwidth communication routines suitable for Lattice QCD.

- Point-to-point communication
  - Nonblocking
  - Simultaneous, multidirectional transfer capability
  - Chained block/strided transfer capability
  - Separate routines for the initialization and commencement of transfers $\rightarrow$ opened communication channels can be reused, minimize overhead for repeated transfers.
- Global operation
  - Global reduction
    {Sum, Max, Min} {Single precision, Double precision} {Scalar, Vector}
    General binary reduction available
  - Broadcast
  - Barrier

Implementation of QMP on QCDOC

- Point-to-point communications
  For nearest neighbor, the requirements for QMP are well satisfied by QCDOC physics network
  $\rightarrow$ Small software overhead.
  Both C/C++ Binding implemented over QCDOC system calls.
- Global operations
  Store-and-forward capability of QCDOC is used.
  Both C/C++ Binding implemented over QCDOC system calls.
  Communication time grows as $\sim L \times N_d$
  $L =$ Number of sites $N_d =$ Number of Dimension.
- Implementation complete except non-nearest communication.

Non-Nearest Neighbor Communication(NNC) on QCDOC

While repeated communications are often to/from nearest-neighbor, NNC gives more flexibility and portability.

Desired features:

- Minimum number of copying
- Minimal overhead when destination is nearest
- Should be able to co-exist with nearest neighbor communication without introducing excessive software overhead to nearest communication.
- Consideration for MPI implementation (R. Bennet, BNL).

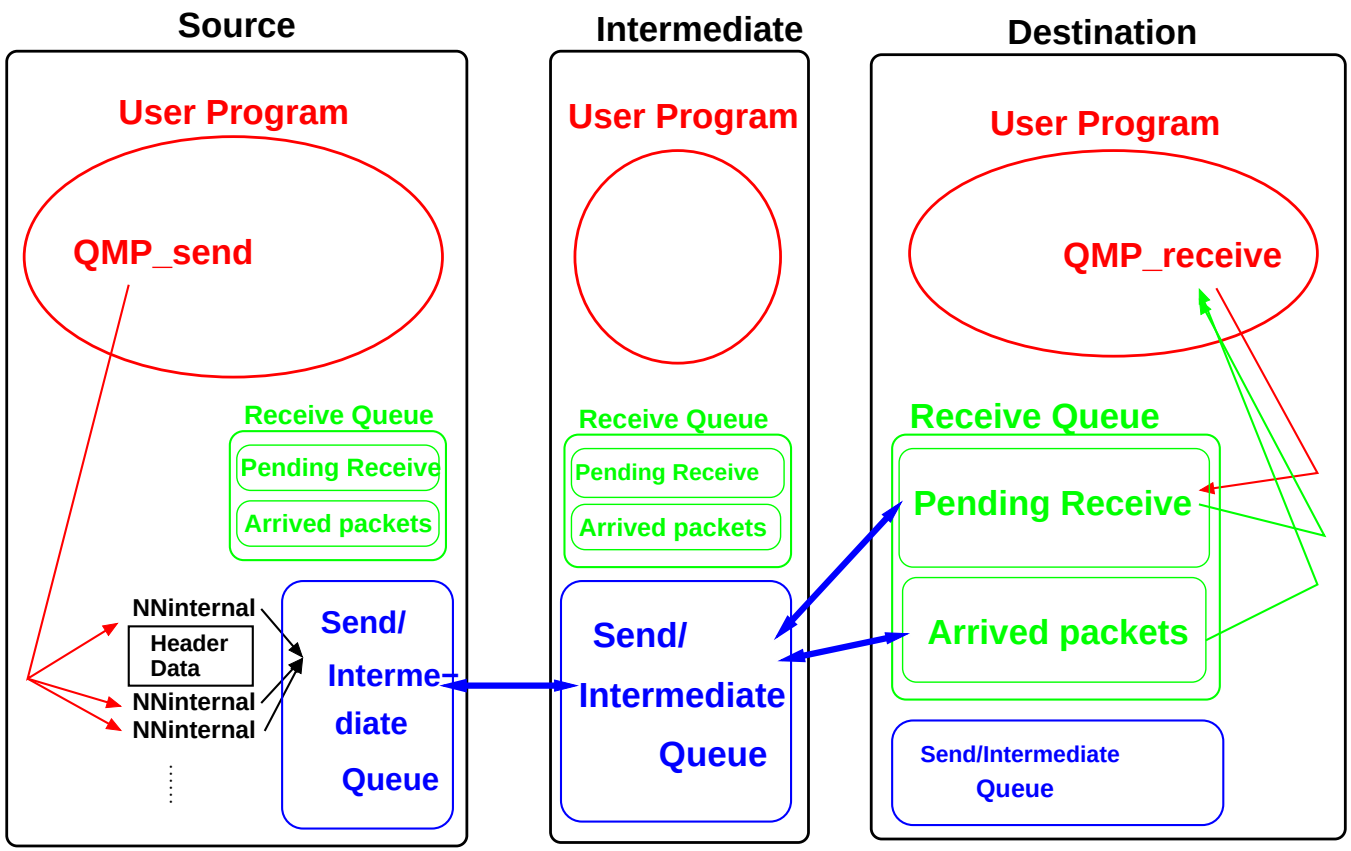
Implementation strategy:

- Interrupting communication channel (Supervisor) used for the header transfer
- Non-interrupting physics network used for user data transfer

- Static, Manhattan-style routing to prevent lock-ups/long delays
- Consideration for MPI: Many of the necessary functionalities needed for MPI such as
  - Message routing
  - Buffer management
  - Packetization
  - Queueing

  will be implemented for QMP. MPI implementation will use QMP whenever it is efficient.

Diagram of NNC



Planned activities

- Implmentation & Revision of QMP
- Level 3 interfaces
- Batch system(PBS)
- Parallel file system
- Runtime environment
- Linpack(D. Krohn, P. Boyle), etc..