Overview of Columbia Physics System(CPS)

Chulwoo Jung
Brookhaven National Laboratory, Upton, USA

August 2, 2013
Lattice 2013(coding session)
U. of Mainz, Germany

Introduction

Why "Legacy" code?

- A lot of existing measurement codes.
- Flexible: always had a collection of hand-optimized code rather than uniformly written one.
- Optimization: RBC always had emphasis on dynamical gauge generation. Code structured to easily optimize gauge/fermion force, etc to match with optimized solvers.

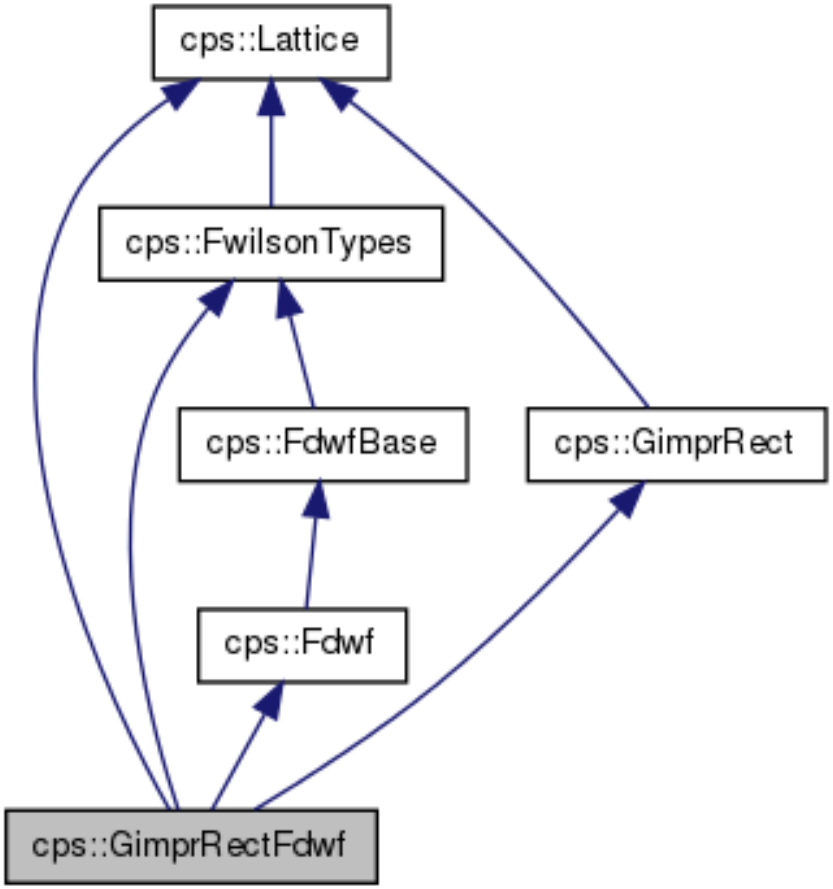
History

CPS started in 1997, when Columbia group wanted to start a new code base to be used for QCDSP, TI DSP + custom designed Chip (Node Gate Array, NGA)
Since then, it has been run in a significant way on QCDOC, Clusters, IBM BG/L, BG/P, BG/Q, GPU...
You can find (slightly out of date) doxygen pages at http://qcdoc.phys.columbia.edu/cps.htmlhttp://qcdoc.phys.columbia.edu

Available routines

- Action
  - Gauge : Wilson, ImprRect(Iwasaki, DBW2, tadpole-improved), Symanzik,
  - Fermion, Wilson/Clover, Staggered/Asqtad, P4, DWF(Shamir,Mobius), Twisted mass(DSDR), G-parity (C. Kelly)
- Algorithms
  - Evolution: Integrators(Leapfrog, Omelyan, Force Gradient (Mawhinney & Yin)), RHMC (M. Clark) , Mass Precondioning
  - Observables: Kaon system, Nucleon, QCD Thermodynamics ....
  - Solvers: CG, BiCGStab, Multimass solver, eigCG, HDCG
  - Evals: Ritz, Implicitly restarted Lanczos, All/Low Mode Averaging (Blum et al.).

Inheritance of DWF+Iwasaki lattice class



- One global instance for SU(3) lattice: Only one Lattice instance can be in scope.
- Gauge action class provides routines for Gauge, forces.
- Fermion class provide Inverter(DiracOp), Fermion Force..
- Mix and match in the integrator class.

Optimization Strategy (Native routines)

Optimization strategy could be quite different for different local volumes & architectures.

- Inverters: Take advantage of existing optimized inverters. Allow different orderings for different fermions and/or architectures. (Analogous to ScIDAC level 3 routines)
- Fermion routines outside inverters:
  Traditionally (Before BG/Q) optimized $\not{D}$ and LinAlg. Mix and match to achieve good performance.
  BG/Q threading model makes this sub-optimal. Overhead for field reordering and thread sycronization too large. Routines such as EigCG re-written to BFM directly. Mostly threaded via OpenMP.

- GF/FF/Smearing: Written with multidimensional parallel-transport, which allows for amortizing overheads from communication and threading, maximize communication bandwidth.

$$
\begin{aligned}
&\text{pt}(X'[], X[], U[], \mu[], N, ...)\ \{ \\
&\text{int i;} \\
&\text{for}(i = 0; i < N; \text{i++})\{ \\
&X'[\mu[i]] = U_{\mu[i]} X[\mu[i]] \\
&..... \\
&\}
\end{aligned}
$$

```
const int N = 4;
Matrix *Units[4];
for(int i = 0;i<N;i++) Units[i] = Unit;
int mu,nu;
{
  int dirs_p[] = {0,2,4,6,0,2,4};
  int dirs_m[] = {1,3,5,7,1,3,5};
  ParTransGauge pt(*this);
    for(nu = 1;nu<4;nu++){
      pt.run(N,tmp1,Units,dirs_m+nu);
      pt.run(N,result,tmp1,dirs_m);
      pt.run(N,tmp1,result,dirs_p+nu);
      for(int i = 0; i<N;i++)
      vaxpy3_m(tmp2[i],&tmp,tmp1[i],tmp2[i],vol*3);
      pt.run(N,tmp1,Units,dirs_p+nu);
      pt.run(N,result,tmp1,dirs_m);
      pt.run(N,tmp1,result,dirs_m+nu);
      for(int i = 0; i<N;i++)
      vaxpy3_m(tmp2[i],&tmp,tmp1[i],tmp2[i],vol*3);
      ForceFlops +=vol*12*N;
    }
    pt.run(N,result,tmp2,dirs_p);
}
```

$$S = \sum_{y\delta L} \sum_{ij} c_{ij} \overline{\psi_i}(y) U_\alpha U_\beta \cdots U_{-\alpha} \psi_j(y+\delta)$$

i,j=spin,rat,flavor
L = different paths connecting y and $y+\delta$.

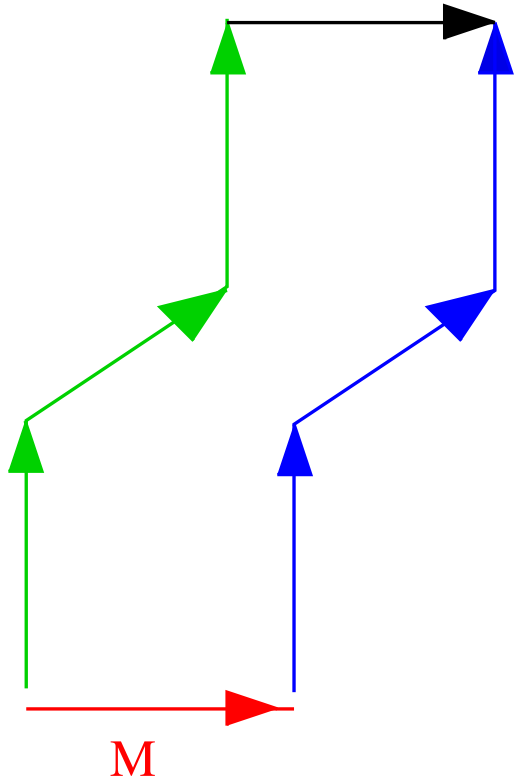
$$\frac{\partial S}{\partial U_\mu(x)} = \sum_{y,\delta,L} \sum_{ij} U_\gamma(x+\mu) \cdots U_{-\alpha} \cdot$$

$$c_{ij} \psi_j(y+\delta) \overline{\psi_i}(y) U_\alpha U_\beta \cdots U_{-\epsilon}(x)$$

$$= \sum_{y,\delta,L} U_\gamma(x+\mu) \cdots U_{-\alpha}$$

$$\sum_{ij} \left[\psi_j(y+\delta) \overline{\psi_i}(y) \right] U_\alpha U_\beta \cdots U_{-\epsilon}(x)$$

$$= \sum_{y,\delta,L} [U_\alpha \cdots U_{-\gamma}(x+\mu)]^\dagger M(y,\delta) U_\alpha U_\beta \cdots U_{-\epsilon}(x)$$



7 staple

$32^3 \times 64 \times 16 (m_s = 0.03)$ BG/P					
	2048×4 core BG/P			4096×4 core BG/P	
m_l	0.008	0.006		0.004	
Local volume	$4^4 \times 16$			$4^4 \times 8$	
Routines	time(sec)		MFlops/s	time(sec)	MFlops/s
MInv	851	950	443	563	374
CG	320	433	464	342	391
GF	34	39	350	45	302
RF	142	162	83	306	23
HF	2	3	30	12	4
Total time(seconds)	1406	1646		1355	
Total flops ($\times 10^{12}$)	4450	5260		5868	

$48^3 \times 64 \times 16$ DWF+I			$32^3 \times 64 \times 32$ DWF+ID		
$m_s = 0.03, 8192 \times 4$ core BG/P			$m_s = 0.045, 2048 \times 4$ core BG/P		
m_l	0.002		0.0042	0.001	
Local volume	$6^3 \times 2 \times 8$		$4^4 \times 32$		
Routines	time(sec)	MFlops/s	time(sec)		MFlops/s
MInv	1672	477	4073	4112	488
CG	1155	517	3438	4839	538
CG(DSDR)			199	196	503
GF	206	408	137	130	344
RF	479	50	643	642	71
HF	29	3	12	12	27
Total time(s)	4050		8888	10500	
Total flops ($\times 10^{12}$)	47857		33017	39439	

$48^3 \times 96 \times 24$ DWF+I			$64^3 \times 128 \times 12$ DWF+I	
$m_s = 0.0362$, 4096 node BG/Q			$m_s = 0.02661$, 8192 node BG/Q	
m_l	0.00078		0.000678	
Local volume	$6^3 \times 12 \times 24$		$16 \times 4 \times 4 \times 12$	
Routines	time(sec)	GFlops/s	time(sec)	GFlops/s
MInv	1538	~ 25	640	~ 50
CG	3366	29	1565	53
GF	56		24	
RF	50		97	
HF	29		20	
Eig	140		140	
Total time(s)	5393		2422	

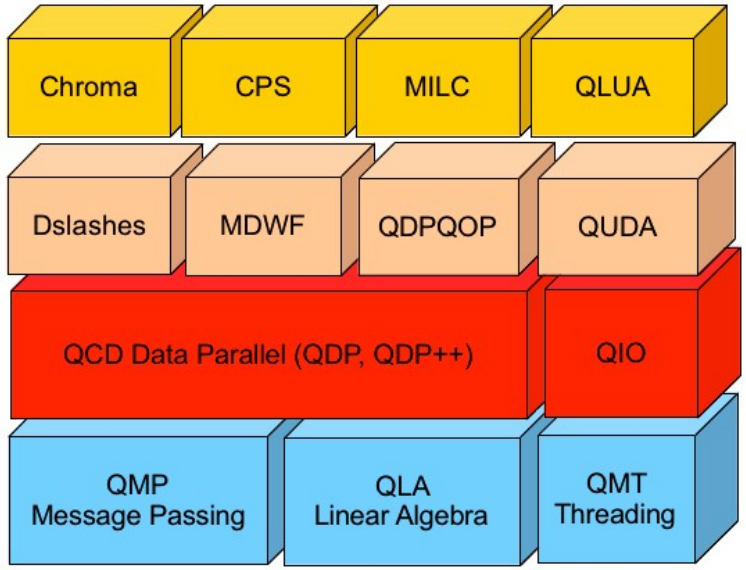
CG+Multimass solver+eigensolver is $\sim 90\%$ of total evolution time.

External routines

- SciDAC routines: API's optimized for latticeQCD, developed by USQCD
  - QMP, QIO, QDP++, Chroma
  - QUDA (M. Clark. Fri. 10G)
  - CG-DWF, MDWF (Optimized Shamir/Mobius package)
- BAGEL Fermion Matrix (BFM) (www2.ph.ed.ac.uk/ paboyle/bagel/)(Not SciDAC!)

SciDAC software

- USQCD
  Collaboration of US lattice community under SciDAC.
  Developed hierachial software suites to enable portability $\rightarrow$ allow optimized code for specific platforms to be shared without much additional effort.
  http://www.usqcd.org/software.html



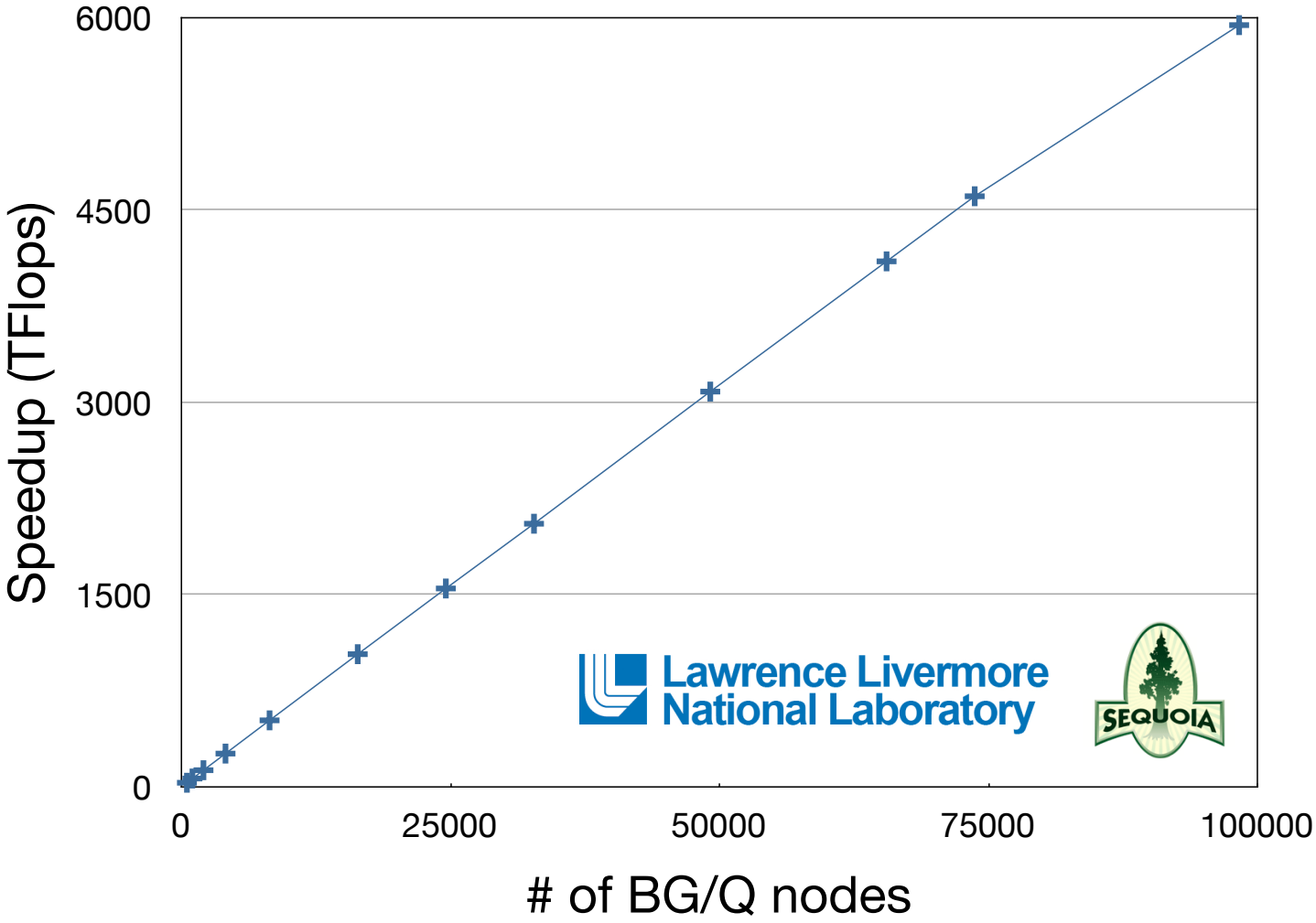
USQCD software stack

BG/Q (BFM) CG performance

http://www2.ph.ed.ac.uk/ paboyle/bagel/Bagel.html
GFlops/node,1 rack, w SPI

Fermion	V	L_s	Double	Single	L_s	Double	Single
Shamir	4^4	8	14	16	16	20	25
	$6^3 \times 8$	8	38	45	16	44	55
	$8^3 \times 8$	8	47	58	16	36	66
	$10^3 \times 8$	8	35	65	16	24	58
	$12^3 \times 12$	8	20	51	16	19	44
Mobius	4^4	8	12	14	16	16	21
	$6^3 \times 8$	8	32	42	16	36	49
	$8^3 \times 8$	8	42	54	16	27	60
	$10^3 \times 8$	8	28	60	16	18	47
	$12^3 \times 12$	8	17	42	16	14	33
WilsonTM	4^4		2.6	2.7			
	$8^3 \times 8$		22	25			
	$10^3 \times 8$		29	35			
	$12^3 \times 12$		24	47			

Weak Scaling for DWF BAGEL CG inverter



Code developed by Peter Boyle at the STFC funded DiRAC facility at Edinburgh

Achieved 5.94PFlops Aug. 2012.

Integration/optimization status

BFM/BAGEL (Peter Boyle, EPCC)

- DWF(5d,4d), Mobius, Wilson, WilsonTM Inverters, with QMP and/or SPI
- Clover (K. Sivalingam, Poster) to be integrated.
- G-parity(C. Kelly Thur. 8C)
- QDP++ interface included. In use with UKHADRON(Chroma)
- So far only used with 1 MPI per node
  - Native (contiguous) mapping (logical nearest nodes should be also physically nearest)
  - QMP w/SPI allows flexible mapping.

QUDA in CPS

From H. Kim, GTC 2013

CG performance on QUDA GFLOPS for all GPUs, Single-Double mixe

↓Lattice Nodes→	4	8	16	32
24^3x64x16	892 (1,1,1,4)	1110 (1,1,1,8)	1310 (1,2,2,4)	1692 (1,1,4,8)
32^3x64x16	N/A	1110 (1,1,2,4)	1442 (1,2,2,4)	1743 (2,2,2,4)

※Scalability on virtual size of lattices (= # node / 1 node)

1 (24^3x16x16)	4 (24^3x64x16)	8 (24^3x128x16)	16 (24^3x256x16)
248.0 (1.00)	890 (3.59)	1691 (6.82)	3367 (13.6)

※Surface / Volume ratio

Nodes→ ↓Lattice	4 (1,1,1,4)	8 (1,1,1,8)	16 (1,1,2,8)	32 (1,1,4,8)
24^3x64x16	0.067	0.143	0.263	0.412
32^3x64x16	N/A	0.143	0.231	0.333

However...

Vol	Geometry	GFlops
$48^3 \times 96$	(2,3,3,4)	3547
$64^3 \times 128$	(2,2,4,8)	2376

DWF on larger lattices forces users to use many GPUs, and surface/volume deteriorates quickly..

QUDA in CPS (cont.)

We(Mostly H-J. Kim) have taken steps to take advantage of implementation/hardware/algorithm improvement from QUDA and are also contributing to it.

- DWF(Shamir) inverter interface. Others being considered. (SWME has been running staggered inverter on GPU)
- QUDA implementation of 4D/Mobius done (H. Kim, Fri 10G)
- Implicitly restarted Lanczos, EigCG implmentation being explored.
- DSL/QDP-JIT?

AMA in CPS

In QPropW class.
Eigenvectors generation,storage,etc mostly hidden from users.
Allows for fast deployment.
Being used for WME, Nucleon, Static HQ, Nucleon decay, $\cdots$
On Clusters: Exact eigenvectors generated in batches and (temporarily) saved. Can be shared with others in principle.
On BG/Q: Flops/IO bandwith much worse. Organize jobs to generate and consume eigenvectors in one job. Good strong scaling of BG/Q ($\sim$ 35 GFlops/node on 32K partition) allows to fit all (Bk, kl3, k$\rightarrow \pi\pi\cdots$) jobs within time limits ($\sim$ 5.3 hours)

Build system, Parameter passing, etc

- Parameter passing: Virtual Markup Lanaguage(P. Boyle)
  Derived from rpcgen. Processes C++ class definition-like input file and generate parser. Every entry has to be present, or fails.
- Development environment: CVS $\rightarrow$ git
  CPS has more than one active developer at any time. Have used CVS with access control list patch to allow multiple developers. Recently moved to use GitLab(gitlab.org, https://github.com/gitlabhq/gitlabhq) a Github-like program with hosting capability.
  http://cuths01.phys.columbia.edu:8080/
- Build system: GNU autoconf, build out of source tree?? possible. Hierarchical directory structure: directories for most of classes, machine depenent files.

```
$ ls src/util/dirac_op/d_op_mobius/sse/
blas-subs.h                       Makefile
dwf_dslash.C                      READEME-4deo.txt
dwf_dslash_5_plus.C               sse-blk-dag0.h
dwf_dslash_5_plus-nonowait.C      sse-blk-dag1.h
dwf_end_sse.C                     sse-bnd-dag0.h
dwf_init_sse.C                    sse-bnd-dag1.h
dwf_m_4dpc.C                      sse-dwf_dslash_4.C
dwf_mdag_4dpc.C                   sse-macros.h
dwf_mdagm.C                       sse-macros-dag0.h
dwf_mdagm_4dpc.C                  sse-macros-dag1.h
dwf_mdagm_5dpc.C                  sse-subs.h
fake_omp.h                        sse-subs-axpys.h
m5inv.C
```

Lessons/suggestions from a "legacy" code

- Stay flexible inside to be portable and inviting to motivated developers.
- Threading strategy choice becoming more pervasive.
- (If you expect to use your code on new machines much) pay attention to propfiling. Make sure you know where your cycles are going.
- Simple parallel transport can go a long way in optimizing routines outside Inverters.

Thank you!

BG/Q optimization from EigCG

- Dense matrix (H) inversion: $\sim N_{eig}^4$, factor of ~ 50 speed up from (single threaded) Engineering and Scientific Subroutine Library (ESSL) http://www-03.ibm.com/systems/software/essl/
- $V \leftarrow V(QZ)$: $(L \times m) \times (m \times n)$, Consuming comparable time to $\not{D}$ BG/Q, can be written with vaxpy, but memory bandwith bounded (single $\sim$ 10GFlops/node). Use intrinsic (vector4double), rearrange the loop to have V_{ij} read only once per call and use QPX, to get $\sim$ 30 GFlops single (& double) precision.
- Programmable prefetch (list engine)?

Considerations for optimizing for BG/Q

- SIMD obviously important.
- Factor of 64 is big! trivial routines like field import/export can be significant. Profiling is useful.
- typical operations needed for evolution (SU(3) Matrix multiply): 108 (m+a) operations / 2*8*18 Bytes $\sim$ 6 flops/8 bytes. Compared to 209(GFlops/s)/40(GB/s) $\rightarrow$ DDR bandwidth is bottleneck, reusing cache effectively is important. Combining local (no shift) operations likely very important. Inverter is likely to be more efficient for relatively small volume.
- Thread creation/destruction overhead
  CG/MInv $\sim$ 10000 times, Optimized DWF/Wilson keeps threads across different iterations. GF/FF/smearing: order of $1 \sim 100$
- Multi directional communication hardware: decrease overhead by doing comms multiple directions simnulateously

CPS

- Variants of BFM (DWF, Mobius, $\cdots$) handled by a dedicated fermions class (Fbfm)(Hantao Yin).
- Mixed precision solver now in BFM
- Rest of CPS?? ((R)HMC force term, contractions...)
  Relevant part for DWF evolution threaded with openMP, getting 5-10GFlops in double precision.
  So far only with openMP and GCC, no QPX, Further improvement possible with XLC or BAGEL. A lot of routines in Lattice QCD are memory bandwith bounded (flops/Byte < 1) $\rightarrow$ should be able to get reasonably close to "theoretical" peak by organizing the code to minimize the bandwidth, with better compiler(XLC..)