PROCEEDINGS OF SCIENCE

Overview of Columbia Physics System

Chulwoo Jung* **for the RBC and UKQCD collaborations**
Brookhaven National Laboratory, Upton, USA
E-mail: chulwoo@bnl.gov

An introduction of basic structure and features of Columbia Physics System(CPS) is given.

31st International Symposium on Lattice Field Theory - LATTICE 2013
July 29 - August 3, 2013
Mainz, Germany

*Speaker.

PoS(LATTICE 2013)417

1. Introduction

CPS started in 1997, when Columbia group wanted to start a new C++ code base for QCDSP computer[1]. It has been and is continuing to be used for all of ensemble generation and majority of measurements done by RBC/UKQCD collaborations. This requires CPS to support vastly different measurement programs for both Wilson type (Domain Wall Fermion(DWF) included) and staggered type fermions, on diverse hardware platforms. Some design choices to meet these requirements is presented.

A list of the available routines in CPS will be followed by the description overall structure in section 3 and description of overall optimization strategy and profiling and testing features in sections 4 and 5.

2. Available routines

Here are a partial list of supported hardware and software routines. There are a number of additional routines not listed , as they are not yet merged to the main trunk or not actively maintained.
Hardware:

- Legacy: QCDSP, QCDOC
- Supported : IBM Blue Gene (L, P, Q(with BFM[2])), CPU Clusters, GPU(QUDA)[3, 4]

Action:

- Gauge : Wilson, Plaquette+Rectangle(Iwasaki, DBW2), 1-loop Symanzik
- Fermion: Wilson, Clover, Staggered, Asqtad, P4, DWF/Mobius, Twisted mass(DSDR)[5], G-parity[6].

Algorithms:

- Evolution: Heatbath, Leapfrog, Omelyan, Force Gradient[7, 8], RHMC[9], Quotient[10]
- Measurements: Weak decay measurements[11], Nucleon matrix elements[12], QCD Thermodynamics[13].
- Solvers: CG, BiCGStab, Multimass solver, Mobius-accelerated DWF(MADWF)[8], eigCG[14], HDCG[15]
- Eigenvalue/eigenvector calculation: Ritz, Implicitly restarted Lanczos

3. Basics

Classes in CPS belongs to 1 of 2 categories: Algorithm classes which defines hardware independent operations such as physics measurements and integrators for ensemble generation, and utility classes which implements lower level operations such as Lattice classes(fermion and gauge actions), Dirac operators and data structures used in algorithm classes. These often has different implementations for different hardwares. While most fermion classes has direct 1-to-1 relation to the corresponding Dirac operators, formal separation between the two allows easier integration of external packages or algorithmic developments such as Mobius-accelerated DWF[8]. There are often intermediate layers of inheritance in lattice and Dirac operator classes to avoid unnecessary duplication of codes, as shown in Fig. 1

3.1 Internode communication and I/O

CPS assumes torus geometry and has only nearest neighbor communication and global operation such as global sums. Currently only QMP implementation is supported for CPS-native internode communications (outside external packages). Hardware specific optimization such as System Programming Interface(SPI) for IBM Blue Gene machines are accessible via QMP.

For the lattice I/O, CPS has own routines for NERSC format lattices. It has 'serial' modes where all the data is transferred to node 0 via barrel-shifting, and 'parallel' mode where every nodes open the same file. To lessen the stress on the file system, CPS has a concurrency control where user specifies the number of nodes writing concurrently in parallel mode. Availability of these often made it possible for reliable physics production even when the file system was not robust or fast enough for QIO routines. For I/O of other data such as propagators or eigenvectors, CPS uses SciDAC QIO package.

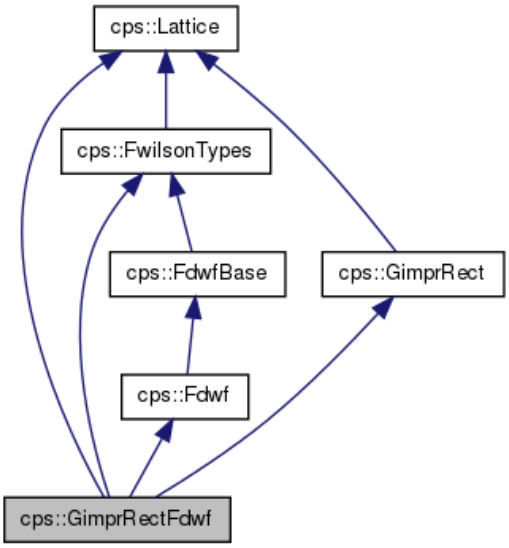

Figure 1: Inheritance of the lattice action class for Domain Wall Fermion with Iwasaki gauge action

```
class ActionRationalQuotientArg rat_quo_arg = {
class ActionBilinearArg bi_arg = {
FclassType fermion = F_CLASS_DWF
Array bilinears[1] = {
class BilinearDescr bilinears[0] = {
double mass =   0.0000000000000000e+00
int max_num_iter = 5000
}
}
class ActionArg action_arg = {
ForceMeasure force_measure = FORCE_MEASURE_YES
string force_label = "RationalQuotient"
}
}
double spread =   0.0000000000000000e+00
int remez_generate = 0
string rat_poles_file = ""
Array bsn_mass[1] = {
double bsn_mass[0] =   1.0
}
Array frm_mass[1] = {
double frm_mass[0] =   0.04
}
Array bosons[1] = {
```

Figure 2: A portion of VML input files for a 1-flavor DWF evolution with rational approximation

3.2 Random number generator(RNG)

RNG in CPS uses Lagged Fibonacci generator. Originally one RNG per node was instantiated with different initial seeds irrespective of machine topology. To ensure that results from CPS is independent of the local lattice sizes modulo roundoffs coming from inevitable change in the order of sums, even when we have having more than 1 nodes for the 5th direction, it has been modified to instantiate one RNG per 1 2^5 lattice sites, and a separate one per 2^4 sites. The ability of CPS to run with more than one node in the 5th direction has been crucial in mapping relatively small lattices onto massively parallel machines with predetermined machine topology, such as Blue Gene P installations at Argonne Leadership Computing Facility(ALCF).

3.3 Build system, Parameter passing, etc

CPS build system uses GNU autoconf to deal with multiple architectures and different options. As it is often necessary to manually edit makefile's while developing for new architecture or new software package, it is desirable to keep the number of configured makefiles at minimum. To achieve this, configure script for CPS changes only makefiles at the top directory with compile and link options, and the makefiles in subdirectories refers to one in the top directory.

Increasing sophistication of both evolution and measurement techniques has made it necessary to manage the parameters often on the order of hundreds efficiently, while remaining human readable to allow for easy manual change when necessary. In CPS, this is achieved by creating classes for parameters for each algorithm class, and process it via rpcgenvml, a modified version of rpcgen originally written for QCDOC[1] operating system, to generate parsers for the input files of the parameter classes in a format called Virtual Markup Language(VML). While it is more restrictive than other markup languages such as XML in that VML is not extensible(variable length array is supported) and every members of the class has to be assigned in order of declaration, it is highly human readable and easy to edit manually. Fig. 2 shows an example of VML input file for a fermion with rational approximation. Further organization of these classes such as evolution with nested integrators and different measurements are often handled in the C++ main program itself rather than nesting the existing structures.

4. Optimization Strategy

Most of the algorithms which involves fermion are built on optimized Dirac operators with order-neutral linear algebra routines. From the very beginning, CPS has had optimized and mostly standalone Dirac operators of both Wilson and staggered type of fermions with different data ordering. This made it natural to organize CPS suitable for external packages, well before SciDAC level 3 packages came into existence. All the input and output gauge and pseudofermion fields to inverters are contiguous blocks of numbers, and there is explicit reordering of the these fields as a specific Dirac operator class is created and destroyed. Later, interfaces for external inverter packages such as BFM[2], cg-dwf and MDWF[16] has been written for a significant portion of gauge evolution and measurements done in CPS.

To maintain overall code efficiency, it is important to have code structured for easy optimization of routines outside solvers, especially ones used in ensemble generation. To achieve this, CPS has a small set of multi-directional parallel transport operations:

$$pt(X'[],X[],U[],\mu[],N) = \quad \{\text{for (int i=0;i<N;i++)}\{X'[i][x] = U_{\mu[i]}[x]X[i][x+\hat{\mu}[i]];\} \quad (4.1)$$

where $X[i]$ is 1- or 3- column color vector, x is the site and $\mu[i](i = 0\cdots N-1)$, are the directions of the parallel transports. Combining logically independent operations makes it possible to amortize communication overhead while maximize communication bandwidth, especially when hardware supports multidimensional communication. and threading with maximum vector length compared to calling routines for different directions independently. To ensure efficient use of the multidimensional communication hardware, $\mu[i]$'s are required to be different from each other.

```
const int N = 4;
Matrix *tmp1[N];
Matrix *tmp2[N];
Matrix *result[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};
// 0=+x, 2=+y, 4=+z, 6=+t
  int dirs_m[] = {1,3,5,7,1,3,5};
// 1=-x, 3=-y, 5=-z, 7=-t
  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);
    }
    pt.run(N,result,tmp2,dirs_p);
}
```

In practice this usually does not pose any additional limit on the number of parallel transports run concurrently, as 4 or sometimes 8 independent operations can be easily run by simply permutation of the directions ($x \to y \to z \to t \to x$). A portion of the gauge force calculation for Wilson action, which calculates staples for links in 4 different directions concurrently is shown on the left.

Force terms for the various fermion actions can be implemented with the same parallel transport quite efficiently. All the fermion actions in use can be written as:

$$S_F = \sum_{y,\delta,L} \sum_{ij} c_{ij} \overline{\psi'}_i(y) U_\alpha U_\beta \cdots U_\gamma \psi_j(y+\delta) \tag{4.2}$$

Where i, j are a combined index for spin(for Wilson type fermions), flavor, 5th dimension (for DWF type fermions) and rational approximation, and L ={ $\alpha, \beta, \cdots \gamma$} are paths connecting y and $y+\delta$ for a given displacement δ. The derivative of the action with respect to a particular link $U_\mu(x)$ can be calculated by

$$\begin{aligned}
\frac{\partial S_F}{\partial U_\mu(x)} &= \sum_{y,\delta,L} \sum_{ij} U_\nu(x+\hat{\mu}+\hat{\nu}) \cdots U_\gamma \cdot c_{ij} \psi_j(y+\delta) \overline{\psi_i}(y) U_\alpha U_\beta \cdots U_\varepsilon(x-\hat{\varepsilon}) \\
&= \sum_{y,\delta,L} U_\nu(x+\hat{\mu}) \cdots U_\gamma \sum_{ij} \left[c_{ij} \psi_j(y+\delta) \overline{\psi'}_i(y) \right] U_\alpha U_\beta \cdots U_\varepsilon(x) \\
&= \sum_{y,\delta,L} \left[U_{-\gamma} \cdots U_{-\nu}(x+\hat{\mu}) \right]^\dagger M(y,\delta) U_\alpha U_\beta \cdots U_\varepsilon(x) = \sum_{y,\delta,L} \left[X_{-\gamma \cdots -\nu}(x+\hat{\mu}) \right]^\dagger Y_{\delta,\alpha' \cdots \varepsilon}(x)
\end{aligned} \tag{4.3}$$

where $X_{\alpha \cdots -\gamma}$ is the path ordered product built with only gauge fields, and $Y_{\delta,\alpha' \cdots -\varepsilon}$ is the path ordered product started with $M(y,\delta) = \sum_{ij} \left[c_{ij} \psi_j(y+\delta) \overline{\psi'}_i(y) \right]$. Forces for actions with smeared links U' instead of regular links can be calculated with the same routine(4.1) with smeared links multiplied with derivatives $\partial U'/\partial U$ to satisfy the chain rule. CPS has routines for $M(y,\delta)$ for Asqtad/HISQ and P4.

The optimization strategy based on optimized Dirac operators and parallel transports has been effective. However, in multithreaded architecture this often makes it necessary to at least synchronize or even stop and restart different threads in each dslash or parallel transport routine, which often introduce significant overhead depending on the hardware and software implementation of multithreading. Reorganization of the CPS routines to be explicitly for each threads, similar to the routines in BFM, to avoid logically unnecessary synchronization is being explored.

As for measurements, RBC/UKQCD recently adopted All-mode averaging(AMA)[17], which

allows efficient measurement of multiple source points per configuration, especially the measurements involving light quarks propagators. AMA can be run with different deflation schemes such eigCG[14], exact deflation with Lanczos, or HDCG[15]. Details of inversion and deflation algorithm is visible to users only via the choice of input flags(CgArg::InverterType) for each algorithms, which minimizes effort necessary for porting and regression testing.

5. Error checking/reproducibility testing, Profiling

CPS has 2 kinds of built-in reproducibility testing. One is in the gauge evolution where a full trajectory is run twice from the original lattice, starting momenta and pseudofermion fields, and the checksums of the gauge field after 2 runs are compared at the end. If it disagrees, it reports the error and retries until the maximum number of retries(typically 1) set in input VML file is reached. All the evolution done by RBC is typically run with 10% (1 traj. every 10) reproduced in this manner. CG implementation in CPS can also run CG twice and compare the residual at each step, with an adjustable frequency.

It has been our experience that the reproducibility error often comes from a relatively small number of nodes more vulnerable to bit errors than the others. To identify these nodes, a different kind of testing was implemented. In the testing mode, RNG on each node is initialized with the same seed, which makes each node run with exactly same set of numbers. Wherever a global sum is called, the numbers from neighboring nodes are compared, and any discrepancy between neighboring nodes is reported. A CPS test derived from our DWF evolution in this manner was able to locate the erratic nodes on IBM BG/P and BG/Q installations even after they passed IBM's testing suite.

While mature runtime environment usually provides sophisticated profiling and error checking capabilities, these are often not available at the time of the deployment. CPS has built-in flop counter where the number of total flops on each node is updated within the Dirac operator, and also simple timing routine built on gettimeofday() which allows for an accurate accounting of performance for each routines in the evolution. Having built-in self-checking and profiling routines provides a significant help in CPS application codes to avoid suffering from Amdhal's law and ensure the generated output is scientifically valid even in the testing stage of hardware installations.

6. Availability and Version control

CPS has had multiple active developer throughout its lifetime, and it often contained codes which were restricted from public access, To satisfy these needs while providing access to multiple developers, RBC/UKQCD have used CVS with access control list patch until recently. Now the CVS repository has been converted to a Git repository with all the branches and tags, and is currently managed by GitLab(http://gitlab.org/).

The CPS GitLab site (http://cuths01.phys.columbia.edu:8080/) will provide a public version of the CPS repository shortly. Meanwhile, tarballs of older versions of CPS as well as doxygen-generated user manual and reference manual can be still found at http://qcdoc.phys.columbia.edu/cps.html.

7. Summary

CPS is a C++-based code suite for lattice QCD, developed mainly be past and present members of RBC/UKQCD collaboration. Despite its relatively long history, its organization has allowed for efficient implementation or quick development of interfaces for optimized packages of various routines essential for ensemble generation and various lattice QCD measurements on diverse hardware platforms from QCDSP to IBM BG/Q. This was made possible by being able to adapt to the changing hardware and software environment and accessible enough to facilitate code development by new members.

Acknowledgments

We would like to thank many members of RBC/UKQCD collaborations for their valuable contribution to CPS. Only a small portion of the members are represented in the references below.

This talk was a part of a coding session sponsored partially by the PRACE-2IP project, as part of the "Community Codes Development" Work Package 8. PRACE-2IP is a 7th Framework EU funded project (http://www.prace?ri.eu/, grant agreement number: RI-283493). CJ was supported by the US DOE under contract DE-AC02-98CH10886.

References

[1] Review of the QCDSP and QCDOC computers, P. A. Boyle et al, IBM research journal and Development, Vol. 49, No. 2.3, March 2005.

[2] P. A. Boyle *Comput.Phys.Commun.* **180** (2009) 2739–2748.

[3] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi arXiv:0911.3191 [hep-lat].

[4] M. Clark, *PoS* **LATTICE2013** (2013) 420, H.J. Kim, *PoS* **LATTICE2013** (2013) 033.

[5] D. Renfrew, T. Blum, N. Christ, R. Mawhinney, and P. Vranas *PoS* **LATTICE2008** (2009) 048, arXiv:0902.2587 [hep-lat].

[6] C. Kelly, *PoS* **LATTICE2013** (2013) 401.

[7] A. D. Kennedy, M. A. Clark, and P. J. Silva *PoS* **LAT2009** (2009) 021, arXiv:0910.2950 [hep-lat].

[8] H. Yin and R. D. Mawhinney *PoS* **LATTICE2011** (2011) 051, arXiv:1111.5059 [hep-lat].

[9] M. Clark and A. Kennedy *Nucl.Phys.Proc.Suppl.* **129** (2004) 850–852, arXiv:hep-lat/0309084 [hep-lat].

[10] M. Hasenbusch *Phys. Lett.* **B519** (2001) 177–182, hep-lat/0107019.

[11] R.D. Mawhinney, *PoS* **LATTICE2013** (2013) 404. T. Janowski, *PoS* **LATTICE2013** (2013) 402, J. Frison, *PoS* **LATTICE2013** (2013) 460, A. Jüttner,, *PoS* **LATTICE2013** (2013) 396, for the **RBC-UKQCD collaborations.**

[12] S. Ohta, arXiv:1309.7942 [hep-lat]. M. Lin *PoS* **LATTICE2013** (2013) 275. for the **RBC-UKQCD collaborations.**

[13] C. Schroeder, *PoS* **LATTICE2013** (2013) 160.

[14] A. Stathopoulos and K. Orginos *SIAM J.Sci.Comput.* **32** (2010) 439–462, arXiv:0707.0131 [hep-lat].

[15] P.A. Boyle, *PoS* **LATTICE2013** (2013) 029.

[16] https://usqcd.lns.mit.edu/redmine/projects/mdwf

[17] T. Blum, T. Izubuchi, and E. Shintani arXiv:1208.4349 [hep-lat].