Oakbridge-CX への計算物質科学ソフトウェアのインストール ー CP2K・LAMMPS

芝　隼人

東京大学情報基盤センター

１．はじめに

現在、東京大学情報基盤センター（以下、当センター）では Reedbush-H/L, Oakforest-PACS, および Oakbridge-CX の３つのスーパーコンピューター(スパコン)システムが運用されている。これらの中で比較すると Oakbridge-CX （OBCX） がもっとも最近になって調達されたシステムであるが、以前のシステム上で利用されたプログラムからの高い移植性および汎用性を重視した設計となっており、幅広い研究分野における利用が可能な構成となっている。

私自身は 2020 年 2 月に当センターに着任して研究業務および運用業務にあたるようになったが、もとは物質科学・材料科学分野を出自とするアプリケーション屋である。この中で、OBCX において物質科学系のプログラムに関する相談案件が多いことが特に目に止まったため、将来導入するシステム構成検討も兼ねて OBCX システムにおける物質・材料系アプリケーション構成の見直しを行った。物質・材料系のソフトウェアは（工学・ものづくり分野のものとの比較においては）オープンソースライセンスによるものが多い。したがってセンターの立場からすると、サポートへの障壁も低い。物質・材料系分野の研究者による利用が多いアプリケーションとして CP2K および LAMMPS を選び、OBCX 上でビルドし、提供することにした。本稿は、利用者に資するべく、これらソフトウェアのビルドの手順を紹介するものである。

２．CP2K

CP2K は分子軌道法および第一原理密度汎関数法による分子の電子状態計算および各種アンサンブルによる分子力学計算を行うことができるオープンソースソフトウェアである。利用者の量子化学理論の背景知識を前提として、第一原理（ab initio）分子動力学法や QM/MM 法の大規模並列計算を可能にする。2000 年前後にプロトタイプコードの開発が開始、以降開発が現在まで続けられている。本稿執筆(2020 年 11 月末)時点での最新安定版は v7.1 であるが、今回は開発最新版である Trunk（v8.0, development）のインストールを行った。

以下、 OBCX における v8.0 のビルド方法を示す。なお、今後最新安定版として v8.1 がリリースされ次第、改めてインストールする予定である。

2.1　ソフトウェアのダウンロード

最初に、OBCX システムのログインノードにて、ソフトウェアをインストールする /work 以下の自身のディレクトリに移動するとともに、デフォルトコンパイラ Intel Parallel Studio XE 2019（19.0.5.281）がロードされていることを確認する。

```
$ module list
>   1) impi/2019.5.281    2) intel/2019.5.281
```

続いてソフトウェアを GitHub リポジトリからダウンロードする。

```
$ git clone --recursive https://github.com/cp2k/cp2k.git cp2k
```

作業ディレクトリ内に cp2k の名称のディレクトリが作られる（以下、${WORKDIR}と表記する）。その内部にソースコード、ライブラリ構築のための toolchain など、必要なファイルが配置される。

2.2 toolchain を用いたライブラリの構築

CP2K のビルドに際して、必要な追加ライブラリのビルド作業を実施する。不足しているライブラリは CP2K に付属して提供される toolchain を用いて導入することができる。まずは、toolchain の配置されるディレクトリに移動するとともに、使用コンパイラの情報を環境変数により設定する。

```
$ cd ${WORKDIR}/tools/toolchain
$ export CC=icc
$ export CXX=icpc
$ export FC=ifort
$ export MPICC=mpiicc
$ export MPICXX=mpiicpc
$ export MPIFC=mpiifort
```

toolchain を使用して、OBCX システムに備わっていない一連のライブラリの構築を実施する。toolchain においては tools/toolchain/scripts ディレクトリ内にそれぞれのインストールスクリプトがあり、以下の書き換えを実施する（以下、行番号は開発版のアップデートによりずれることがあり得る）。

- tools/toolchain/scripts/install_libint.sh: 74 行目付近に 1 行追加する
  - ✓ 自然科学研究機構（分子研）スパコンサイト[1]にある CP2K v7.1 のビルド方法の記載と同様、intel compiler でビルドが通らないパッケージを強制的に外している。

```
 L72: #cmake --build . > cmake.log 2>&1
 L73: #cmake --build . --target install > install.log 2>&1
+L74: sed -i -e "s/fortran_example check_test/libint_f.o \
       check_test/" fortran/Makefile.in
 L75: ./configure --prefix=${pkg_install_dir}  --with-cxx="$CXX \
        $LIBINT_CXXFLAGS"  --with-cxx-optflags="$LIBINT_CXXFLAGS \
       ……     （以下略）
```

[1] https://ccportal.ims.ac.jp/node/2639

- tools/toolchain/scripts/install_elpa.sh
  - ✓ Intel MKL をリンクさせるため。

```
L101: FCFLAGS="-mkl=cluster ${FCFLAGS} ${MATH_CFLAGS} …"
```

fftw, mpi-fftw, gsl, hdf5, superlu は、OBCX システムにプリインストールされたものを Environmental Module を呼び出すことで利用する[2]。

```
$ module load fftw mpi-fftw
$ module load gsl
$ module load hdf5
$ module load superlu
```

以上の準備ができたら、toolchain スクリプトを、必要なオプション指定を行った上で走らせる（cmake 3 系および python 3 系を要求されるので、こちらの Environmental module もロードしておく）。 CP2K v8.0 以降に附属の toolchain では、Intel MPI を使用したビルドのオプションが利用可能となって（--mpi-mode=intelmpi）、OBCX 環境でのビルドが容易となっている。なお現在のところ、SIRIUS および COSMA の両ライブラリの OBCX 上でのビルドについては、成功していない。

```
$ cd ..
$ module load cmake/3.14.5
$ module load python/3.7.3
$ ./install_cp2k_toolchain.sh --math-mode=mkl --mpi-mode=intelmpi \
--with-cmake=system --with-libxsmm=install --with-mkl=system \
--with-fftw=system --with-reflapack=no --with-scalapack=no \
--with-sirius=no  --with-cosma=no --with-plumed --with-gsl=system \
--with-hdf5=system --with-superlu=system
```

以上により、CP2K が利用するライブラリの構築が完了となる。

---

[2] システム側で FFTW3 のプリインストールがない場合には、toolchain を用いてインストールが可能である。この場合、Intel Compiler 使用時は次のような形で書き換える。toolchain インストールスクリプトの実行オプションでは--with-fftw=install とする。

- tools/toolchain/scripts/install_fftw.sh

```
 L49: grep '\bavx512f\b' /proc/cpuinfo 1>/dev/null && \
      FFTW_FLAGS="${FFTW_FLAGS} --enable-avx512"
+L50: sed -i -e "s/-no-gcc//g" configure
 L51: ./configure  --prefix=${pkg_install_dir} \
   --libdir="${pkg_install_dir}/lib" ${FFTW_FLAGS} > configure.log
```

2.3 ソフトウェアビルド

以上の準備が完了したら、ビルドに入ることが可能となる。OBCX システムではノード内に 28 コアの Intel CascadeLake プロセッサ 2 基が配され、ノードあたり合計 192 GiB のメモリを搭載している。CP2K はメモリ量に対する要求が非常に強いアプリケーションであるため、メモリ使用量の逓減のために CPU ソケット内部でのスレッド並列とするハイブリッド並列を行うことが合理的な選択となる。この理由から、今回は、psmp 版（MPI + OpenMP ハイブリッドにより並列化されたもの）をビルドすることとした。

CP2K パッケージの arch ディレクトリ内には、いくつかのアーキテクチャ向けのビルドオプション、コンパイルリンクを指定した arch ファイルが配置されている。ここでは、そのうちの Linux-x86-86-intel-regtest.psmp をリネームし、obcx.psmp として使用する。

```
$ cp Linux-x86-86-intel-regtest.psmp obcx.psmp
```

obcx.psmp の冒頭部分でコンパイラオプションおよびライブラリパスを次のように設定する。なお太字で示したのが変更部分であり、${WORKDIR}は CP2K のインストールディレクトリを示す。なお、Intel MKL のライブラリはシーケンシャル・モードを使用する。

obcx.psmp

```
CC          = mpiicc -std=c1x
FC          = mpiifort
LD          = mpiifort
AR          = ar -r

MPI_PATH    = ${WORKDIR}/tools/toolchain/install
INTEL_PATH = ${WORKDIR}/tools/toolchain/install

include  \
    $(MPI_PATH)/plumed-2.6.1/lib/plumed/src/lib/Plumed.inc.static

ELPA_VER    = 2020.05.001
ELPA_INC \
    = (MPI_PATH)/elpa-2020.05.001/include/elpa_openmp-$(ELPA_VER)
ELPA_LIB    = $(MPI_PATH)/elpa-2020.05.001/lib

LIBINT_INC  = $(INTEL_PATH)/libint-v2.6.0-cp2k-lmax-5/include
LIBINT_LIB  = $(INTEL_PATH)/libint-v2.6.0-cp2k-lmax-5/lib

LIBXC_INC   = $(INTEL_PATH)/libxc-4.3.4/include
LIBXC_LIB   = $(INTEL_PATH)/libxc-4.3.4/lib
```

（次頁へ続く）

```
LIBXSMM_INC = $(INTEL_PATH)/libxsmm-1.16.1/include
LIBXSMM_LIB = $(INTEL_PATH)/libxsmm-1.16.1/lib

SPGLIB_INC  = $(INTEL_PATH)/spglib-1.15.1/include
SPGLIB_LIB  = $(INTEL_PATH)/spglib-1.15.1/lib
```

また、同じ obcx.psmp ファイル中、次のように書き換えを実施する。

```
L58:  LDFLAGS = $(FCFLAGS) -static-intel -static_mpi -lstdc++
L63:  LIBS = $PLUMED_DEPENDENCIES -lgsl -lgslcblas
                                  (なお、libz.a のリンクは外す)
L76:  [LIBS += $(GCC_LIBRARY_DIR)/libstdc++.a] → 削除
```

以上の書き換えが終わったら、インストールディレクトリ中で次のコマンドを実行し、ビルドを行う。

```
$ make -j ${PARALLEL} ARCH=obcx VERSION=psmp
```

これにより実行バイナリ（obcx.psmp）が生成され、CP2K が実行できるようになる。ビルドは OBCX のログインノード上で 1 時間以内に終了する。また、CP2K をライブラリとして使用する場合には、次のコマンドにより libcp2k.a を./lib ディレクトリ内に作ることができる。

```
$ make -j ${PARALLEL} ARCH=obcx VERSION=psmp libcp2k
```

以上で CP2K のインストールは完了である。

今回のビルドでは FFTW3 はシステム側のものを利用する設定としたので、実行する際には

```
$ module load fftw mpi-fftw
```

として該当の Environmental Modules を呼び出されたい。また、使用するライブラリによっては hdf5, superlu も呼び出す必要がある。

また、Intel MKL を使用して MPI-OpenMP ハイブリッドジョブを実行する場合、プロセスあたり OpenMP スレッド数が 2 以上だとしばしばスタックサイズが上限を上回るため、プログラムがエラーメッセージとともに停止する。これを回避するためには環境変数を

```
export OMP_STACKSIZE = 100m
```

などと設定して、スタックサイズを拡張しておく必要がある。今回 OBCX システムに導入した Environmental Module（cp2k/v8.0dev)をロードすると、デフォルトでこの設定が反映される。

また、プロセスあたりスレッド数を 1、すなわち flat MPI とする場合であっても、明示的にスレッド数の指定（例えば export OMP_NUM_THREADS=1）が必要である。基底データファイルを読み出す先のディレクトリは CP2K_DATA_DIR 環境変数によって指定できる。

３．LAMMPS

LAMMPS（Large-scale Atomic/Molecular Massive Parallel Simulator）は、サンディア国立研究所の Steve Plimpton 博士らのグループが中心となって開発した並列古典分子動力学シミュレーション用のオープンソース汎用ソフトウェアである。LAMMPS においては、力場やアンサンブルをはじめとする各種の設定は、モジュール化されたプログラムを呼び出すことで利用する設計となっており、また逆に利用者が必要とする機能や最適化などをモジュールとして実装することも可能である（GPL ライセンスに依拠）。この特徴により、古典分子動力学法が関係する多彩なモデリングを広範にカバーする一大ソフトウェアに成長し、また利用にあたって必要なドキュメンテーションも充実している。

CP2K よりもビルドは容易で、ドキュメンテーションに従っていけば自然とインストールできる。下記に OBCX 上での一通りのセットアップ手順を記す。

3.1 ソフトウェアのダウンロード

最初に、OBCX システムのログインノードにて、ソフトウェアをインストールする /work 以下の自身のディレクトリに移動するとともに、デフォルトコンパイラ Intel Parallel Studio XE 2019（19.0.5.281）がロードされていることを確認する。

```
$ module list
>  1) impi/2019.5.281    2) intel/2019.5.281
```

LAMMPS ソフトウェアを GitHub リポジトリからダウンロード、ビルドを行う作業ディレクトリである src に移動する。

```
$ git clone -b stable https://github.com/lammps/lammps.git mylammps
$ cd  ./mylammps/src
```

“mylammps” はインストールディレクトリ名であり、以下では${INST_DIR}と表記する。これによりダウンロードされるの、2020 年 9 月時点では ver. 3Mar2020 であったが、本稿執筆（2020 年 11 月）の直前に最新安定版が ver. 29Oct2020 アップデートされた。このアップデートに伴う最大の変更点は、ソースコードが C++11 仕様に統一されたことであり、ビルドの Makefile において -std=c++11 のオプションを明示することが必要となった。また、パッケージ構成に一部変更があった。本稿では ver. 29Oct2020 のビルド方法を示す。

3.2 パッケージの選択

LAMMPS のソースコードは、機能に従って分類された複数のパッケージから構成され、パッケージごとにインストールを行うか否かを指定することができる。現在のインストールを指定しているパッケージ状況は、src ディレクトリにて次を実行することで確認できる。

```
$ make package-status
     Installed  NO: package ASPHERE
     Installed  NO: package BODY
     Installed  NO: package CLASS2
     Installed  NO: package COLLOID
                :
```

インストールしたいパッケージについて、次のコマンドを実行する。

```
$ make yes-${PACKAGENAME}
```

${PACKAGENAME}にはパッケージ名を入れるが、LAMMPS にあっては全て大文字で記述する。今回 OBCX システムの Environmental module (lammps/3mar2020)においては、外部ライブラリへの依存性の強い GPU, KIM, KOKKOS, USER-ADIOS, USER-QUIP, USER-SCAFACOS, USER-VTK を除く全てのパッケージをインストールした(この一部にはインストール可能とみられるものもあるが、私自身の時間の制約上、導入を見送った)。

また、LAMMPS Ver. 29Oct2020 では機械学習原子間相互作用ポテンシャルの計算を行う MLIAP パッケージが正規のパッケージとして追加されたが、OBCX における lammps/3mar2020 では提供していない。

3.3 ライブラリのビルド

同じ src ディレクトリ上から、ライブラリ(実体は ${INST_DIR}/lib 上にある)をビルドできる。本節では少し長くなるが、traditional make による各インストール方法を紹介する。HDF5, NetCDF, GSL は OBCX システム上のものを利用できるため、改めてインストールの必要はない。

・LATTE

まず、Intel MKL を利用するようにライブラリビルド用の makefile の設定を実施する。OBCX 上では、現状マニュアル通りのインストール方法のままではビルドが通らないため、ややアドホックではあるが下記のように対処した[3]。

[3] 次のウェブサイトの記述を参考にした:
Compile LATTE package for LAMMPS using Intel Compilers
https://thelinuxcluster.com/2019/06/10/compile-latte-package-for-lammps-using-intel-compilers/

```
$ cd ${INST_DIR}/src
$ make lib-latte args="-b -v 1.2.2"
    (-b はパッケージのダウンロードおよびインストールを指示するオプションでは
     あるが、一発ではビルドは通らない)

$ vi ${INST_DIR}/lib/latte/LATTE-1.2.2/makefile.CHOICES
    L46: #For intel compiler
    L47: FC = ifort
    L48: FCL = $(FC)
    L49: FFLAGS = -O3 -fpp -qopenmp
    L50: LINKFLAG = -qopenmp

    L52: #GNU BLAS/LAPACK libraries:
    L53: #LIB = -llapack -lblas      (コメントアウト)
    L54: #Intel MKL BLAS/LAPACK libraries:
    L55: LIB = -Wl,--no-as-needed -L${MKLROOT}/lib/intel64 \
         -lmkl_lapack95_lp64 -lmkl_gf_lp64 -lmkl_gnu_thread \
         -lmkl_core -lmkl_gnu_thread -lmkl_core -ldl -lpthread -lm

$ cd ${INST_DIR}/lib/latte
$ mv Makefile.lammps Makefile.lammps.past
$ cp Makefile.lammps.ifort Makefile.lammps

$ vi ${INST_DIR}/lib/latte/Makefile.lammps
     latte_SYSINC =
     latte_SYSLIB = ../../lib/latte/filelink.o \
          -llatte -lifport -lifcore -lsvml \ #-lompstub -limf 【削除】
          -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core \
          -lmkl_intel_thread -lpthread -qopenmp  【書換】
     latte_SYSPATH = -qopenmp -L${MKLROOT}/lib/intel64 \    【書換】
                     -lmkl_lapack95_lp64
$ cd ${INST_DIR}/lib/latte
```

・MESSAGE

LAMMPS の現バージョン（ver. 29Oct2020）では、言語仕様として C++11 のみをサポートとなったため、次の通り Makefile を書き換えることが必要となる（以下、他のパッケージでも同様の書き換えを行なっている）

```
$ vi ${INST_DIR}/lib/message/cslib/src/Makefile
   L43: #CC=mpigcc     (次の書き換えを実施する)
  +L43: CC=mpigxx -std=c++11
$ cd ${INST_DIR}/src
$ make lib-message args="-m"
```

・MSCG

```
$ module load gsl
$ vi ${INST_DIR}/lib/mscg/Makefile.mpi
   L26: #CC=mpicc   (次の書き換えを実施する)
  +L27: CC=mpicc -std=c++11
$ cd ${INST_DIR}/src
$ make lib-message args="-b -m mpi"
```

・POEMS

```
$ cd ${INST_DIR}/lib/poems
$ vi Makefile.mpi   (次の書き換えを実施する)
  -L70: #CC =  mpicxx
  -L71: #CCFLAGS =   -O3 -g -fPIC  -Wall
  +L72: CC = mpiicpc -std=c++11
  +L73: CCFLAGS = -O2 -g -fPIC  -Wall
$ cd ${INST_DIR}/src
$ make lib-poems args="-m mpi"
```

・VORONOI

```
$ cd ${INST_DIR}/src
$ make lib-voronoi args="-b"
```

・USER-ATC

```
$ cd ${INST_DIR}/lib/atc
$ vi Makefile.mpi    (次の書き換えを実施する)
  -L72: #CC =  mpicxx
  -L73: #CCFLAGS =      -O3 -Wall -g -fPIC
  +L72: CC = mpiicpc -std=c++11
  +L73: CCFLAGS = -O2 -fp-model fast=2 -no-prec-div \
                  -qoverride-limits -qopt-zmm-usage=high
$ cd ${INST_DIR}/src
$ make lib-atc args="-m mpi"     (← ATC をコンパイル)
```

・USER-AWPMD

```
$ cd ${INST_DIR}/lib/awpmd
$ vi Makefile.mpi    （次の書き換えを実施する）
  -L39: #CC = mpicxx
  -L40: #CCFLAGS =  -O3 -fPIC -Isystems …
  +L39: CC = mpiicpc
  +L40: CCFLAGS = -O2 -g -fPIC -Isystems …（以下そのまま）
$ cd ${INST_DIR}/lib/linalg
$ make -f Makefile.mpi
$ vi Makefile.mpi    （次の書き換えを実施する）
  -L21: #CC = mpifort
  -L21: #CCFLAGS =  -O3 -fPIC
  +L21: CC = mpiifort
  +L21: CCFLAGS = -O2 -fPIC
$ cd ${INST_DIR}/src
$ make lib-linalg args="-m mpi"
$ make lib-awpmd args="-m mpi"
```

・USER-COLVARS

```
$ cd ${INST_DIR}/lib/colvars
$ vi Makefile.mpi    （次の書き換えを実施する）
  -L8 : #CXX =  mpicxx
  -L9 : #CXXFLAGS =  -O3 -g -Wall -fPIC -funroll-loops
  +L10: CXX = mpiicpc -std=c++11
  +L11: CXXFLAGS = -O2 -fp-model fast=2 -no-prec-div \
$ cd ${INST_DIR}/src
$ make lib-colvars args="-m mpi"     (← COLVARS をコンパイル)
```

・USER-H5MD

```
$ cd ${INST_DIR}/src
$ make lib-h5md args="-m h5cc"    (← H5MD をコンパイル)
```

・USER-MESONT

```
$ cd ${INST_DIR}/src
$ make lib-mesont args="-m ifort"   (←MESONT をコンパイル)
```

・USER-PLUMED

```
$ cd ${INST_DIR}/src
$ make lib-plumed args="-b"       (← PLUMED をダウンロード、コンパイル)
```

・USER-QMMM

```
$ cd ${INST_DIR}/src
$ make lib-qmmm args="-m mpi"     (← Quantum Espresso をコンパイル)
```

・USER-SMD

```
$ cd ${INST_DIR}/src
$ make lib-smd args="-b"          (← Eigen をダウンロード)
```

3.4 ソフトウェアビルド

必要なライブラリのインストールができたら、最後にソフトウェアをビルドする。LAMMPS では src/MAKE 以下のディレクトリに、各種のアーキテクチャに対応した Makefile が用意されている。src/MAKE/Makefile.mpi が標準の MPI 並列用の Makefile である。ここでは、OBCX システムにおいて最適な Intel MKL + Intel MPI ライブラリを利用するもので上書きしておく。また、一部パッケージにおいて使用するシステム側のライブラリの Environmental modules をロードする。

```
$ cd MAKE
$ mv Makefile.mpi  Makefile.mpi.past
$ cp ./OPTIONS/Makefile.intel_cpu_intelmpi ./Makefile.mpi
$ module load hdf5/1.10.5  netcfg/4.7.0  gsl/2.5
$ cd ..
```

こうして作成した Makefile.mpi によって、バイナリ名 lmp_mpi という Intel プロセッサ向け最適化済みのバイナリを作成することができる。ビルドは次のコマンドで実行できる。

```
$ make -j ${PARALLEL} mpi
```

また、静的ライブラリオブジェクト liblammps_mpi.a が必要であれば、次のコマンドで生成できる。

```
$ make mode=static mpi
```

ソフトウェアの実行については、src ディレクトリに生成されたバイナリ lmp_mpi を実行することとなるので、src ディレクトリに対して PATH を通しておく。

一部機能（USER-OMP, USER-INTEL パッケージ内）においては MPI-OpenMP ハイブリッド並列を利用できる場合がある。OBCX システムにおけるハイブリッド並列の際のコアバインディングのデフォルト指定は KMP_AFFINITY=balanced となっているが、これは LAMMPS におけるハイブリッド並列では非推奨である。代替として KMP_AFFINITY=scattered などを使用いただきたい。

より具体的な実行インプットファイルの作成方法については、LAMMPS 開発者によって更新されている公式マニュアル（ https://lammps.sandia.gov/doc/Manual.html ）を参照いただきたい。

４．OBCX システムにおけるビルド済 CP2K, LAMMPS の利用方法

OBCX 上にインストールした CP2K および LAMMPS は、OBCX システム上 Environmental Module を呼び出すことにより利用することが可能である。

```
$ module load cp2k/v8.0dev        (CP2K v8.0 development)
$ module load lammps/3mar2020     (LAMMPS ver. 3Mar2020)
```

これらの Environmental Modules の簡単な実行方法については

```
$ module help cp2k/v8.0dev
$ module help lammps/3mar2020
```

Environental Modules の設定内容については

```
$ module show cp2k/v8.0dev
$ module show lammps/3mar2020
```

によりコマンドライン上に表示できるので、参照されたい。

５．結語にかえて

物質・材料研究分野のソフトウェアは単一のランの内部で呼び出すワークロードが比較的多岐に及び、ユーザーが汎用性の高い CPU を搭載したシステムとして OBCX を積極的に選んでいると考えられる。OBCX はこの要件を満たす国内指折りのスパコンシステムであり、物質・材料系のワークロードは当面高い割合を示し続けると予想している。量子化学計算と古典分子動力学計算という物質・材料系を代表する２手法をカバーする代表的ソフトウェアである CP2K, LAMMPS の利用が可能になったことで、これら周辺分野を含めた研究が活性化することを願っている。

CP2K, LAMMPS とも一部機能は GPU による計算が可能である。2021 年 5 月に稼働開始予定の Wisteria/BDEC-01 システムにおいても、CPU 演算ノード群である Odyssey、GPU 搭載ノード群である Aquarius の双方でインストールされる予定である。利用いただければ幸いである。

最後になって付け加えるが、もう一つのソフトウェアとして VASP (The Vienna Ab initio Simulation Package) に言及したい。VASP は PAW 擬ポテンシャル法による平面波基底第一原理計算を行うことができ、第一原理電子状態計算のソフトウェアとして最も普及度が高いものであるが、実は OBCX において最も顕著に多く利用されているソフトウェアであることがわかってきた。2020 年 6 月以来、利用動向調査を OBCX 納入ベンダーである富士通株式会社システムエンジニア各位のご協力をいただき実施継続しているが、ここ半年では、全ジョブのトークン消費量中、

約 15% が VASP 利用と見られる。VASP は利用者グループが直接ライセンスを購入する必要がある有償ソフトウェアであり、センターから積極的なサポートすることは予算面の制約から容易ではないが、引き続き利用動向を注視していきたい。

謝辞

CP2K のビルドに際しては、石井良樹博士（兵庫県立大学大学院シミュレーション学研究科）、大戸達彦博士（大阪大学大学院基礎工学研究科）に助言、意見、およびインプットスクリプトの提供をいただいた。Bo Thomsen 博士（日本原子力研究開発機構システム計算科学センター）には、CP2K の Environmental Module に対して有益なフィードバック情報を戴いた。また、これらソフトウェアを OBCX における Environmental Modules として提供する際に富士通株式会社のエンジニアの方々にお世話になった。ここにこれらの方々に謝意を表する。