クイックスタート

更新日 2025/11/27

動作要件

QUBO++の構築には以下の環境が使用しています。 互換性を確保するため、記載されているコンポーネントと同等またはそれ以降のバージョンをご使用ください。

- Operating System: Ubuntu 20.04.6 LTS
- glibc: 2.31
- C++ Standard: C++17
- Compiler: g++ 9.4.0
- Boost: 1.81.0
- CUDA: 12.8

> Windows11/WSL2環境でも動作が確認できています。

ABS3(GPUソルバ)は、GPUが必要となります。 サポートしているGPUに関しては下記をご確認ください。

ABS3のサポートGPU

ubuntu 24.04 を推奨します。

ABS3(GPUソルバ)は、GPUが必要となります。 サポートしているGPUに関しては下記をご確認ください。

ABS3のサポートGPU

ubuntu 24.04 を推奨します。

---

2. QUBO++のダウンロード

1. https://hi-qubo.com/download/ よりQUBO++をダウンロードします。

- ダウンロードできるのは、**amd64**(x86_64 architecture)になります。(11/26時点)
- **arm64**(aarch64 architecture)は、QUBO++リリース(最新版) をご確認ください。

2. 必要に応じてライセンスキーを申請します。 取得したライセンスキーを登録することで規模の大きい組み合わせ問題が計算できるようになります。

2.1. Linuxでの利用

UbuntuへQUBO++インストールして利用する場合は、このままインストールへ進んでください。

- QUBO++のインストール

2.2. Windows環境での利用

注意 Micorosoft VisualStudioでは、QUBO++はビルドできません。

QUBO++は、g++ほか、Linux系OSのライブラリ、ツールが必要となります。 **Windows11** で、Windows Subsystem for Linux(以下、WSL)を利用できるPCであれば、 QUBO++を利用することが可能です。 （Ubuntuの仮想環境を作成し、その中でQUBO++を動かすことができる）

セットアップ手順を簡単に紹介いたしますが、 詳しい動作条件、エラーに関しては公式サイトをご確認ください。

- WSL2のインストール要件: https://learn.microsoft.com/ja-jp/windows/wsl/install

以下の手順でWSLの準備を進めてください。

2.3. WSL環境の準備

2.3.1. WSLのインストール

WSLにログインする管理ユーザ名とパスワードをご準備ください。 **linux管理ユーザとして作業するときに必要となります。**

1. アプリより、Windows Power Shellを起動します。 以下のコマンドを入力します。

```
# インストールできる有効なディストリビューションの一覧を表示
 wsl --list --online
```

2. ubuntu24.04をインストールします。 途中、管理ユーザとパスワードを求められますので、入力します。 パスワードは忘れないでください。

```
# Ubuntu 24.04 LTSのインストール
wsl --install Ubuntu-24.04
```

2.3.2. WSL/Ubuntuの起動

1. アプリより、Ubuntuと検索、WSLコンソールを起動します。
2. qubo++の作業用ディレクトリを準備します。 home/ユーザ名/の下に、work_qbppという名前でディレクトリを作成します。 (任意の名前でかまいません)

```
# homeディレクトリに移動
cd ~
# ディレクトリを作成する
mkdir work_qbpp
```

3. ダウンロードしたファイルをLinuxドライブへ移動します。 Widowsエクスプローラーを使う場合：

- Windowsエクスプローラーを立ち上げて、Linuxドライブを確認ください。 （ネットワークの下に、Linuxというドライブがあります）
- home/ユーザ名/に、先ほど作成たディレクトリ(work_qbpp)にダウンロードしたファイルを移動します。

Linuxコマンドでコピーする場合の例: ユーザ名、ダウンロードフォルダ場所などは適宜、自分の環境に合わせて変更して下さい。

```
#windows c:ドライブは、/mnt/c/でアクセスできます
cp /mnt/c/Users/your_name/Downloads/qbpp_amd64_2025.xx.xx.tar.gz ~/work_qbpp/
```

インストールへ進んでください。

QUBO++のインストール

2.3.3. 補足

Linux標準エディタは `vi vim` ですが、Linuxを初めて使う人には少々難しいと思われます。 必要に応じてエディタをインストールしてください。

VScode VScodeは、Windows側でインストールしておきます。 WindowsにVisual Studio Code(VScode)をインストールしている場合、 WSLコンソールから以下のコマンドでVScodeが起動します。

```
code .
```

gedit もうひとつ、`gedit`を載せておきます。

```
# gediotのインストール
sudo apt update
sudo apt install gedit
```

```
# geditでファイルを開く
gedit ファイル名
```

起動後、タスクバーに、アイコンが出てきます。これを表示すれば、メモ帳と同じようにファイルを編集できます。

---

3. QUBO++のインストール＆ビルド

Ubuntuへのインストールとして進めます。

3.1. 依存パッケージの確認、インストール

インストール状況、バージョンが古くないか確認します。

QUBO++の動作要件

```
# g++
g++ -v
# make
make -v
# oneTBB
dpkg -l | grep tbb
# C++ boost lib
dpkg -l | grep boost
```

足らないパッケージがあれば、インストールしておきます。 OSのバージョンによっては、boostの最新版をコンパイルしてインストールする必要があるかもしれません。

```
sudo apt update

# g++, makeツール 一式
sudo apt install -y build-essential
# oneTBB スレディングブロック並列計算用
sudo apt install -y libtbb-dev
# boost lib ライブラリ
sudo apt install -y libboost-all-dev
```

計算結果を描画、画像で確認できるツールをインストールしておきます。

```
# graphviz サンプルコードの画像作成で必要
sudo apt install -y graphviz
# wslviewコマンド、画像表示用
sudo apt install wslu
```

3.2. ビルド

ダウンロードしたqbpp_amd64_2025.xx.xx.tar.gzを任意のディレクトリで展開します。

```
# 解凍
tar xvf qbpp_amd64_2025.xx.xx.tar.gz
```

サンプルコードをビルドします。

```
# sampleコードのディレクトリへ
cd qbpp_xxxx/samples
# sampleのビルド
make
```

しばらく待つとsample/ ディレクトリに実行ファイルが生成されます。

3.3. 実行

セールスマン問題を計算してみます。

```
# セールスマン問題に関するオプションの確認
 ./tsp -h

 # 計算 10地点
 ./tsp -n10 -o output-n10.png
```

output-n10.pngは画像ファイルです。画像を開くと結果が確認できます。

```
 wslview output-n10.png
```

3.4. アンインストール

アンインストールしたい場合は、ディレクトリを削除します。

```
rm -r qbpp_xxxx
```

---

4. ライセンスキー登録とアクティベーション

4.1. 環境変数

申請により得られるQUBO++のライセンスキーを登録します。

~/.bashrcに、環境変数を登録します。以下の行を追記します。

```
export QBPP_LICENSE_KEY=XXXXXX-XXXXXX-XXXXXX-XXXXXX-XXXXXX-XXXXXX
```

```
# 環境変数を有効化
source ~/.bashrc
```

```
# 確認、キーの値が出力されればOK
echo $QBPP_LICENSE_KEY

XXXXXX-XXXXXX-XXXXXX-XXXXXX-XXXXXX-XXXXXX
```

4.2. ライセンスの有効化

アクティベーションを実行します。 ライセンスが有効になり、ライセンスに応じて変数制限が解除されます。

qbpp_xxxxの下にあるbinに移動します。

```
# qbpp_xxxx/binに移動。ディレクトリ名は適宜読みかえてください。
cd ~/qbpp_xxxx/bin
```

qbpp-licenseコマンドで、アクティベーションを実行します。

```
# ライセンス管理のヘルプ
./qbpp-license -h

# アクティベーションの実行
 ./qbpp-license -a

# 出力
QUBO++ (2025.11.23): Professional License(expires at: 2026/03/31 23:59:59)
Variable Limit: 2147483646/2147483646 (CPU/GPU)
Activations: 7/10 (used/allowed)
Deactivations: 22/30 (used/allowed)
```

`Professional`ライセンスの出力例です。

- 有効期限
- CPU/GPUの変数制約
- 利用可能なライセンス数（インストールしたPCでノードロックがかかります）
- デアクティベーションの残り回数（インストールしたPCからライセンスを外すことができます）

別のPCにQUBO＋＋インストールし直したい場合は、一度、ライセンスを外します。

```
# デアクティベーションの実行
 ./qbpp-license -d
```

ノードロックが解除され、Deactivationsカウントが１つ、増えます。 ディアクティベーションは、１ライセンスで3回までとなっています。 この後、別のPCで再度、アクティベーションすることでライセンスが有効になります。

5. 最初の一歩

5.1. Makefileを使ったビルド手順

ここまででサンプルコードのビルドと実行ができました。 さらに自分のコードをコンパイルする手順を説明します。

基本的にはコンパイラのコマンドに必要なオプションを渡せばよいのですが,毎回、指定するのは面倒です。またリンクなどに失敗すると、難しい(?)メッセージが多く表示されます。 最初はなるべく負担を減らすため、サンプルコードと同様に、makeと呼ばれるツールを使うことにします。

5.2. 作業用ディレクトリ作成

Ubuntuを起動し、ダウンロードしたqbpp_xxxxの中に作業用ディレクトリmysrcを作成します。 home/ユーザ名/work_qbpp/qbpp_xxxx/mysrcという階層になります。

```
# home/ユーザ/work_qbpp/qbpp_xxxxに移動
cd ~/work_qbpp/qbpp_xxxx
# ディレクトリを作成する
mkdir mysrc
# mysrcに移動
cd mysrc
```

5.3. Makefileの作成

makeコマンドは、デフォルトでmakefileという名前のファイルを読み込みます。 適当なエディタでファイルを開き、下記の内容を記述します。 makefileという名前で保存します。

makeファイルには空白ではなく、タブを入れるというルールがあります。 VScodeの拡張機能などを利用することで、間違えないようにすることができます。

```
#  2 行目の行頭は空白ではなくタブ
a.out: $(SRC)
(TAB-->) $(CXX) $(CXXFLAGS) $(CPP_INT_FLAGS) -o $@ $^ $(LINK_FLAGS)
```

makefileの構文については、ここでは説明しきれないのですが、

- コンパイラに渡すオプションを宣言部分
- ターゲットを作成するルール に大きく分かれます。

以下の例では、`SRC=factorization.cpp`部分を任意のファイル名に変更することで ビルドすることができます。

```
# 以下をMakefileにコピーしてください
CXX = g++

CXXFLAGS_BASE = -Wall -Wextra -Wpedantic -Wconversion -Wsign-conversion \
                -Wtype-limits -Wshadow -Wnon-virtual-dtor -Woverloaded-virtual -
funroll-loops \
                -std=c++17 -I../include

ifdef DEBUG
  CXXFLAGS = $(CXXFLAGS_BASE) -g
else
  CXXFLAGS = $(CXXFLAGS_BASE) -O3
endif

LD_PATH = ../lib
LDFLAGS = -L$(LD_PATH) -Wl,-rpath,$(LD_PATH)
LIBS = -lqbpp -ltbb -lpthread -lboost_program_options
LINK_FLAGS = $(LDFLAGS) $(LIBS)

ABS3C32_FLAGS = -lqbpp_abs3c32 -DCOEFF_TYPE=int32_t -DENERGY_TYPE=int64_t
ABS3C64_FLAGS = -lqbpp_abs3c64 -DCOEFF_TYPE=int64_t -DENERGY_TYPE=int64_t
CPP_INT_FLAGS = -DCOEFF_TYPE=cpp_int -DENERGY_TYPE=cpp_int

# target source file name ***このファイル名を書き換えます***
SRC=factorization.cpp

# appという実行ファイルを作成: 作成に必要なファイル( 2 行目にルール)
app: $(SRC)
    $(CXX) $(CXXFLAGS) $(CPP_INT_FLAGS) -o $@ $^ $(LINK_FLAGS)

# 実行ファイル、オブジェクトファイルを削除
clean:
    rm -f ./app *.o
```

5.4. ビルド方法

mysrc/makefileがあることを確認してください。 サンプルコードを使ってビルドができるか確認します。

```
# コピー
cp ../samples/factorization.cpp ./

# ビルド
make

# 実行
```

5.5. 実行方法

```
# オプション説明の表示
./app -h

# オプション説明
Factorization using quartic polynomial model:
  -h [ --help ]           Show this help message
  -p [ --p ] arg          Value of p
  -q [ --q ] arg          Value of q
  -t [ --time ] arg (=10) Time limit in seconds
  -e [ --exhaustive ]     Use exhaustive solver instead of easy solver
```

各オプションを入力することで実行することができます。

```
# 実行
./app -p10 -q3
```

以下のコマンドで、app, 不要なオブジェクトファイル(*.o)を削除することができます。

```
# ./app, *.oを削除
make clean
```

---

6. まとめ

1. mysrc/の下に自分のコードを作成します。
2. makefileのSRC=のところを書き換えます。
3. makeでコンパイル、ビルドして、実行ファイルを作成
4. プログラムの実行 といった流れになります。

makefileの中には、mysrc/からの相対位置としてinclude, libのパスが記述してあります。 違うディレクトリで作業する場合、適宜、makefileを修正する必要があります。

7. 参考