キャッシュブロッキングを利用した効率的な量子計算機シミュレーションの並列化

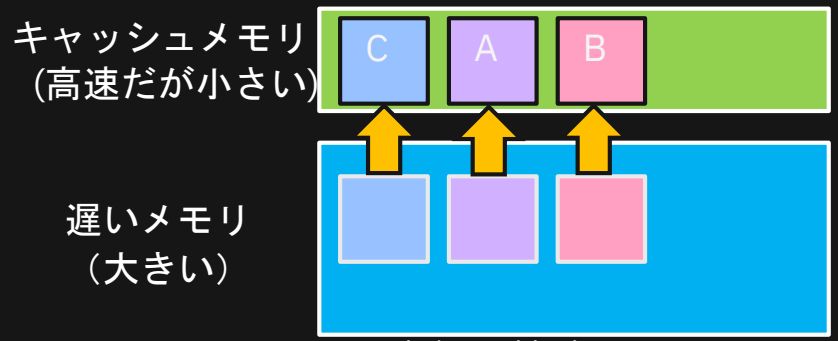
土井　淳,　堀井　洋

IBM Quantum, IBM Research – Tokyo

doichan@jp.ibm.com, horii@jp.ibm.com

キャッシュブロッキングとは？

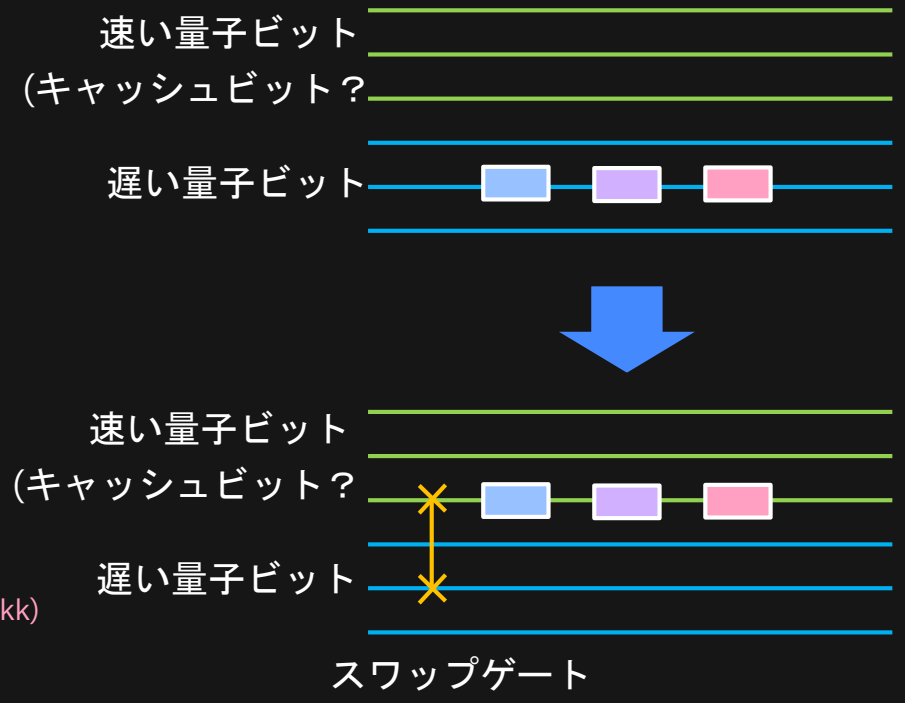
古典計算機におけるキャッシュブロッキング




```
do i=1,N
 do j=1,N
  do k=1,N
   c(j,i) += a(k,i)*b(j,k)
  enddo
 enddo
enddo
```

```
do i=1,N,block
 do j=1,N,block
  do k=1,N,block
   do ii=1,block
    do jj=1,block
     do kk=1,block
      c(j+jj,i+ii) += a(k+kk,i+ii)*b(j+jj,k+kk)
     enddo
    enddo
   enddo
  enddo
 enddo
enddo
```

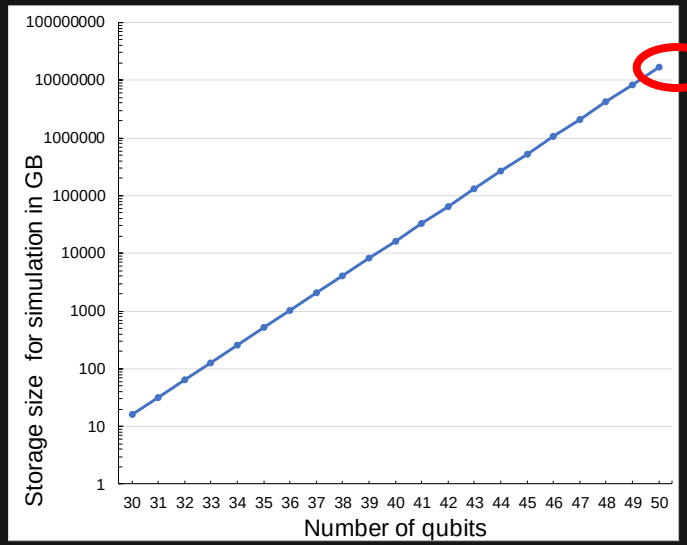
量子計算機におけるキャッシュブロッキング（？）



背景

N量子ビットの確率振幅 $|\varphi\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle$

古典計算機上の量子状態配列



*16*2^n* バイト(倍精度浮動小数点数)

16 PB (50量子ビット)

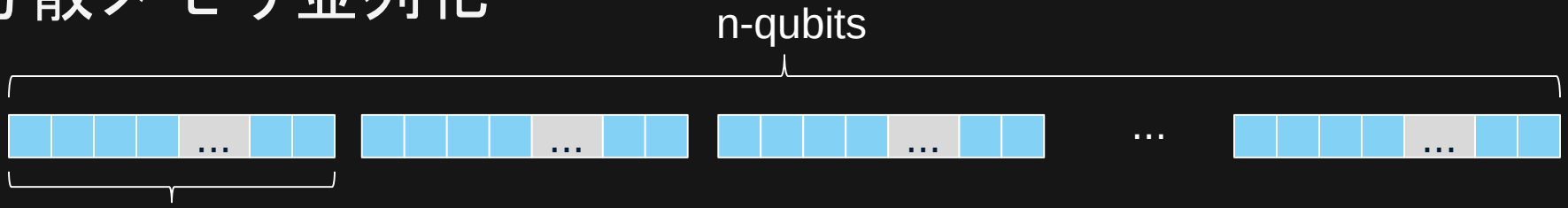
+1 量子ビット = x2 メモリ量
+ x2 計算量

アドレス	複素数配列
0000	r_0, i_0
0001	r_1, i_1
0010	r_2, i_2
0011	r_3, i_3
0100	r_4, i_4
0101	r_5, i_5
0110	r_6, i_6
0111	r_7, i_7
1000	r_8, i_8
1001	r_9, i_9
1010	r_{10}, i_{10}
1011	r_{11}, i_{11}
1100	r_{12}, i_{12}
1101	r_{13}, i_{13}
1110	r_{14}, i_{14}
1111	r_{15}, i_{15}

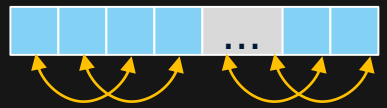
4 qubits probability amplitudes = 2^4

大きな量子計算機シミュレーションには分散メモリ並列化が必須

量子状態ベクトルの分散メモリ並列化



量子状態ベクトルを$np=2^p$ 個の分散メモリ空間に分割する



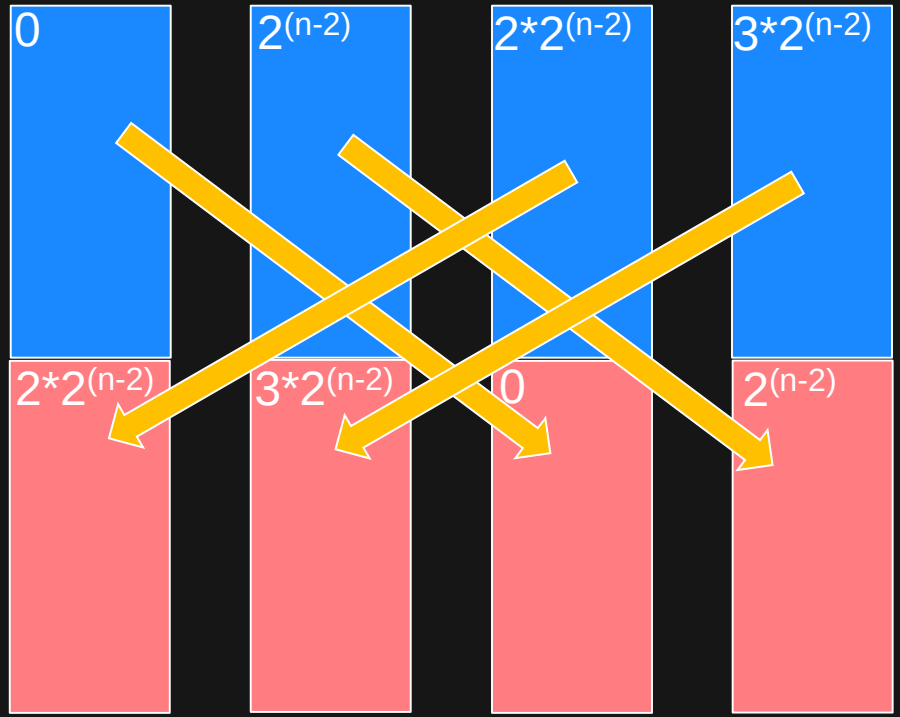
量子ゲートのビット < n-p であれば同じメモリ空間上で計算可能

(gate is on qubit 1)

量子ゲートのビット >= n-p の場合別の分散メモリ空間上の確率振幅を参照する必要あり

分散メモリ空間間の量子状態の参照

ナイーブな実装例



例

np = 4 (=2^2),
量子ゲートのビット = $2^{(n-1)}$ の場合

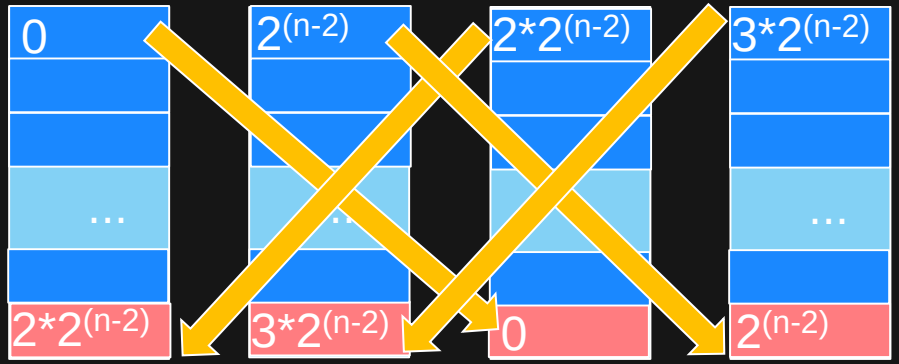
分散された量子状態ベクトル

量子状態を受け取るためのバッファ

この方法だと倍のメモリ量が必要で非効率

チャンクを利用した分散メモリ並列化

量子状態ベクトルをさらに小さい 2^{nc} の長さの配列（チャンク）に分割



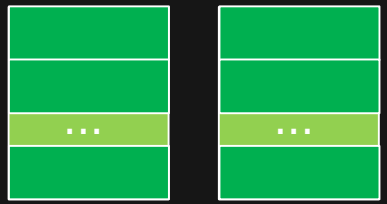
例

np = 4 (=2^2),
量子ゲートのビット = $2^{(n-1)}$ の場合

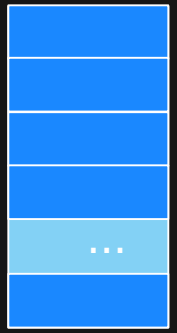
チャンクに分割した量子状態ベクトル

チャンクを受け取るためのバッファ

メモリ使用量を節約するだけでなくフレキシブルな並列化が可能



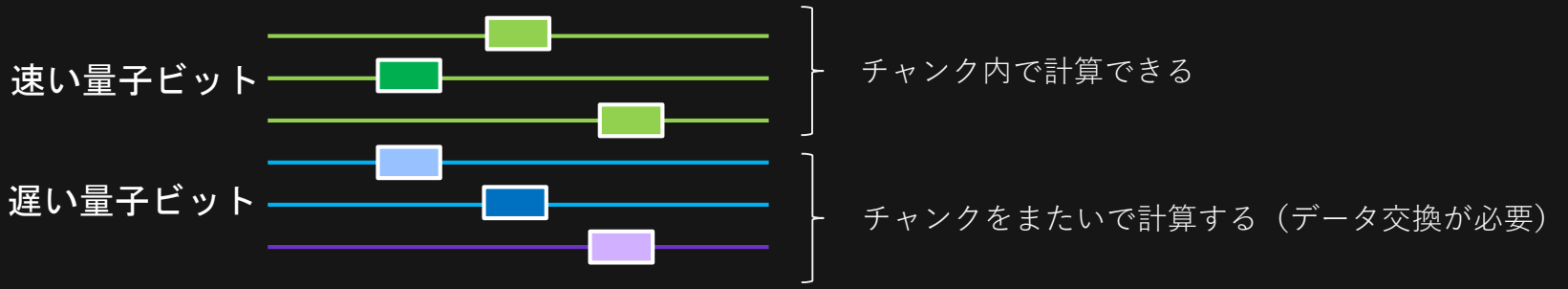
メモリの小さなGPUには
少ないチャンク数を保存

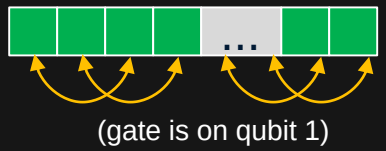


メモリの大きなCPUに
は多くのチャンクを保存

2のべきの制限も無い

チャンクを利用した量子計算機シミュレーション

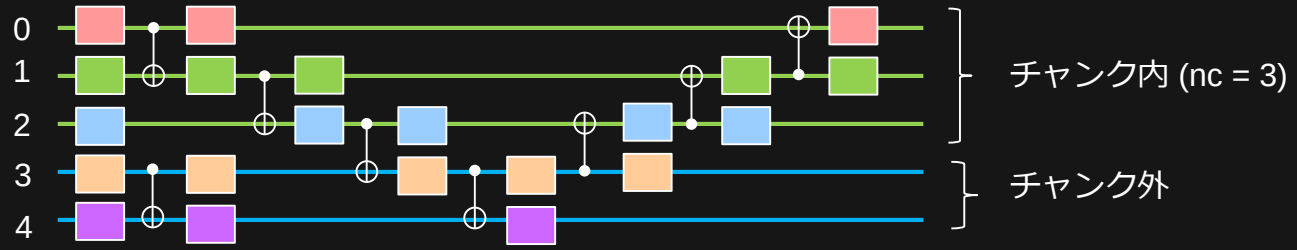




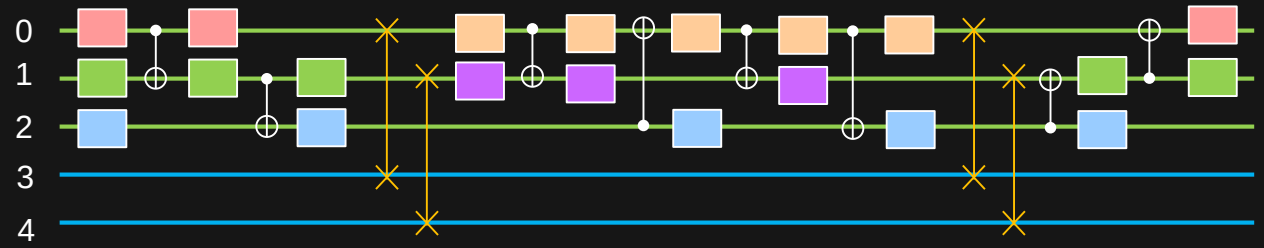
量子ゲートのビット＜nc　の場合チャンク内で計算が完結
それ以外の場合，チャンク間でデータ交換が必要

“速い量子ビット”を利用したキャッシュブロッキングで効率よく並列化

前処理による
量子回路のキャッシュブロッキング



ノイズ無しのスワップゲートを挿入し
すべての量子ゲートをチャンク内に移動する



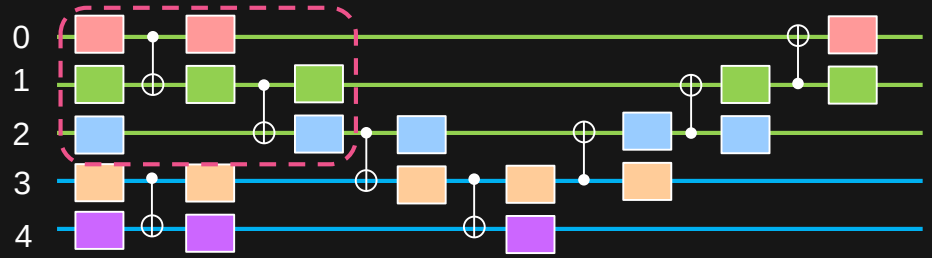
スワップゲート以外はチャンク間データ交換が生じない！

ヒューリスティックな キャッシュブロッキング

- あくまでもシミュレーションの前処理なので
  - 前処理自体に処理時間がかかってしまっては本末転倒
  - 最適解は必要ない
- 次を満たすようなそれらしい出力を得られれば十分
  - 全ての量子ゲートがチャンク内に収まる
  - なるべくスワップゲートは少なく
  - チャンク内で連続して実行できる量子ゲート列をなるべく長く
    - （古典計算機におけるキャッシュブロッキングを効かせるため）

キャッシュブロッキングの例

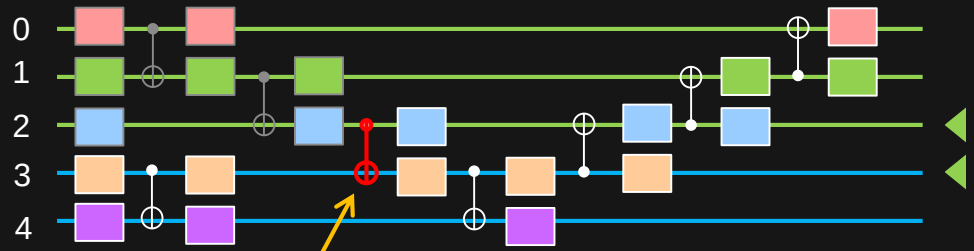
量子ビット 0-2 にある量子ゲートはそのまま出力する



それ以外はキューに積んでおく

入力となる量子ゲート列を先頭からスキャンする

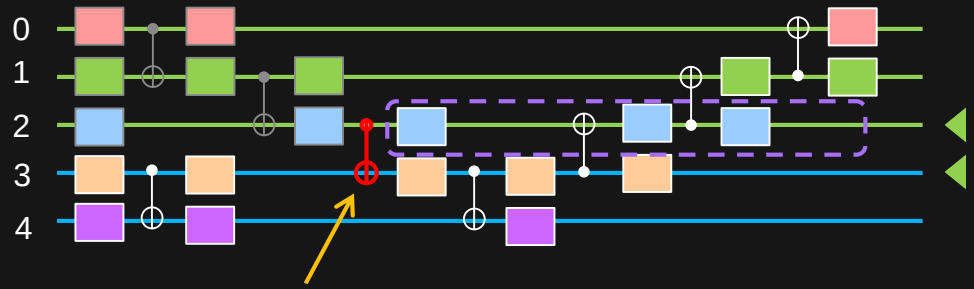
チャンクをまたぐ量子ゲート



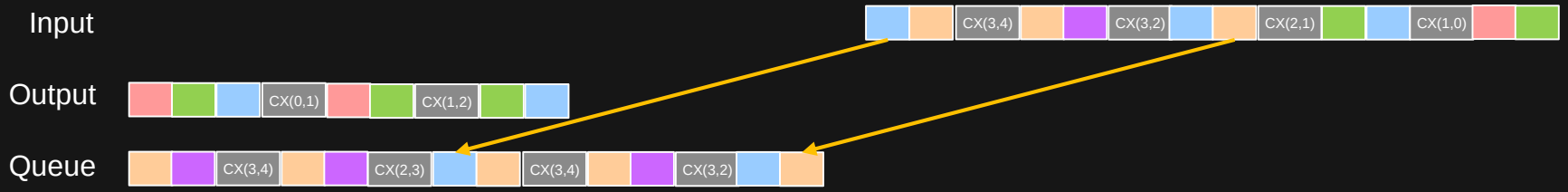
ビット2と3は次回チャンク内に入れる候補にする

CXゲートがチャンクをまたぐので，実行できない

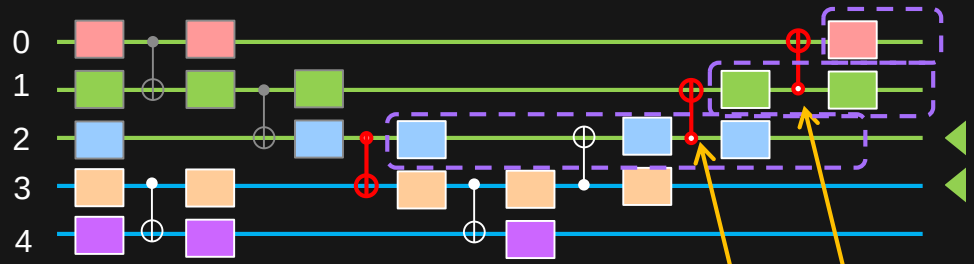
実行を阻害する量子ゲート



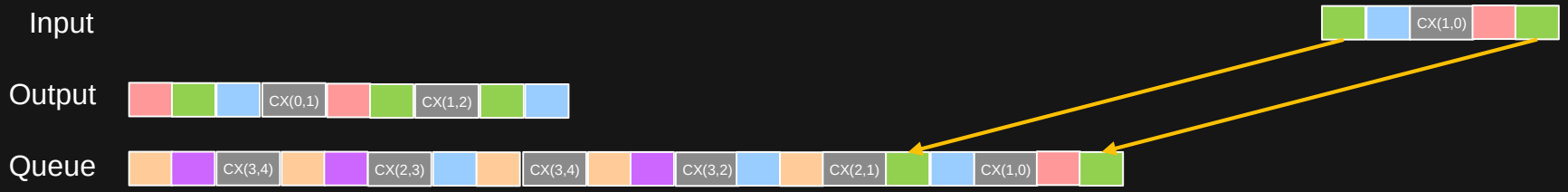
これ以降ビット2のゲートは実行できないのでキューに積む



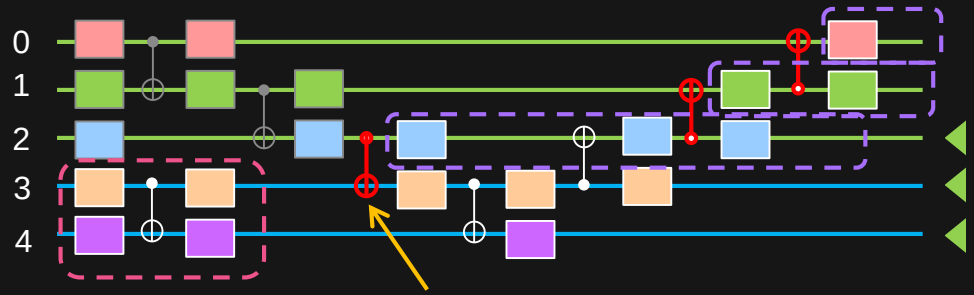
同様に最後までスキャン



これらのCXゲートも実行できないので以降ビット0と1も実行不可

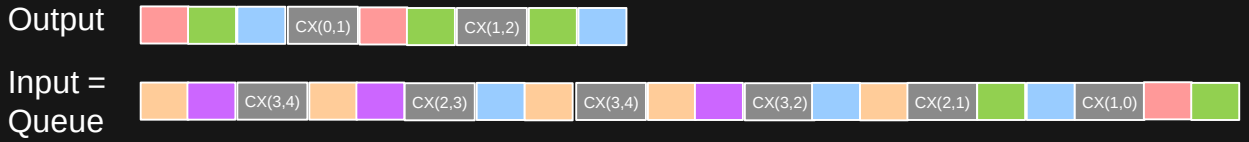


チャンク内に入れるビットを決める



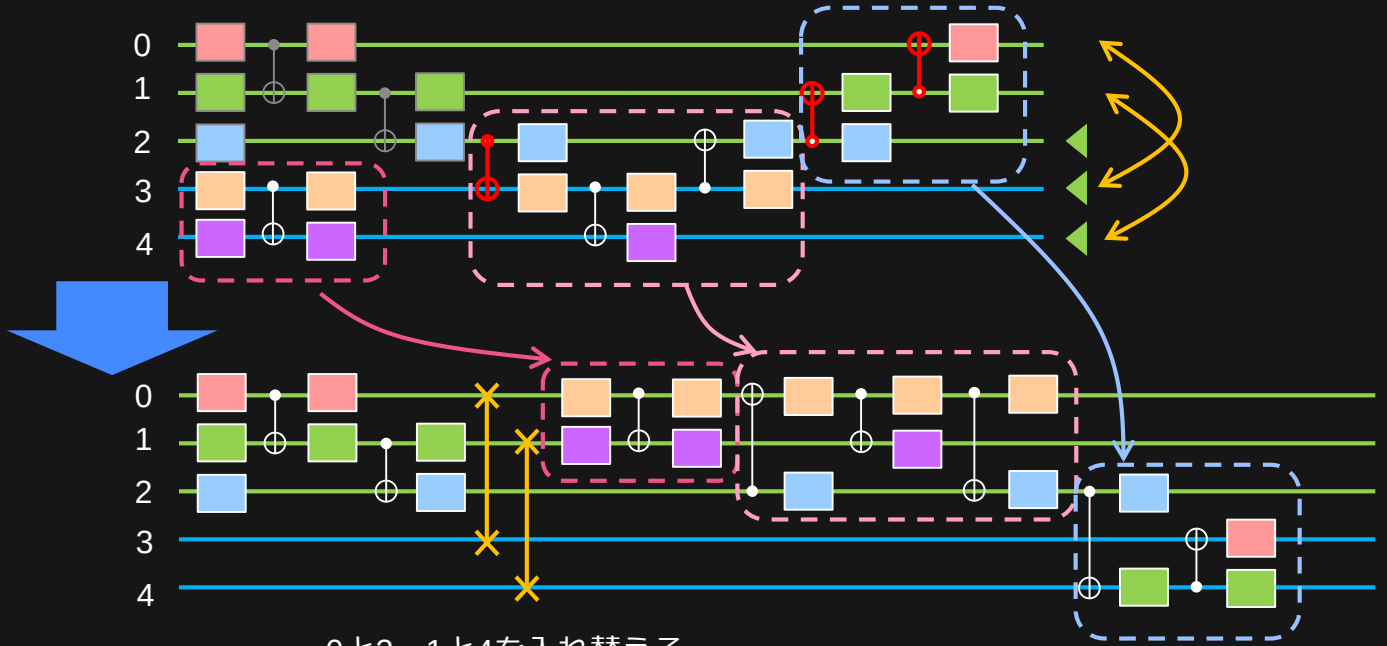
ビット3，4，5をチャンク内に入れる

このCXゲートを実行する前にビット3と4のゲートを実行する必要がある

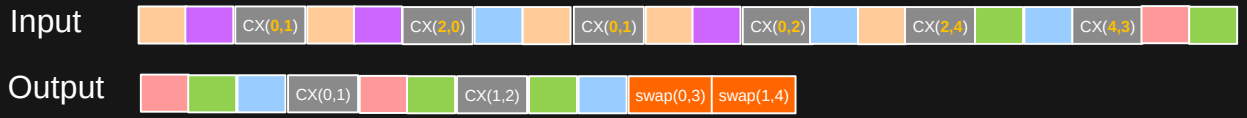


スワップゲートの挿入

IBM **Quantum**

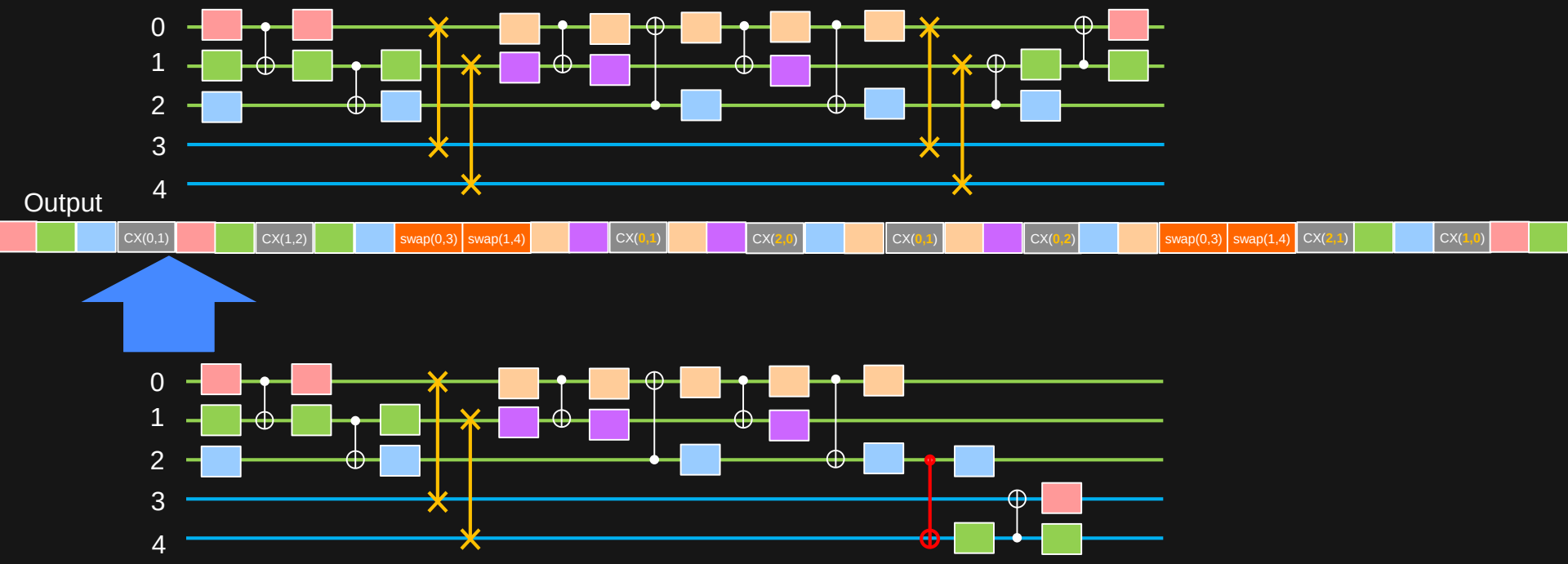


0と3，1と4を入れ替える



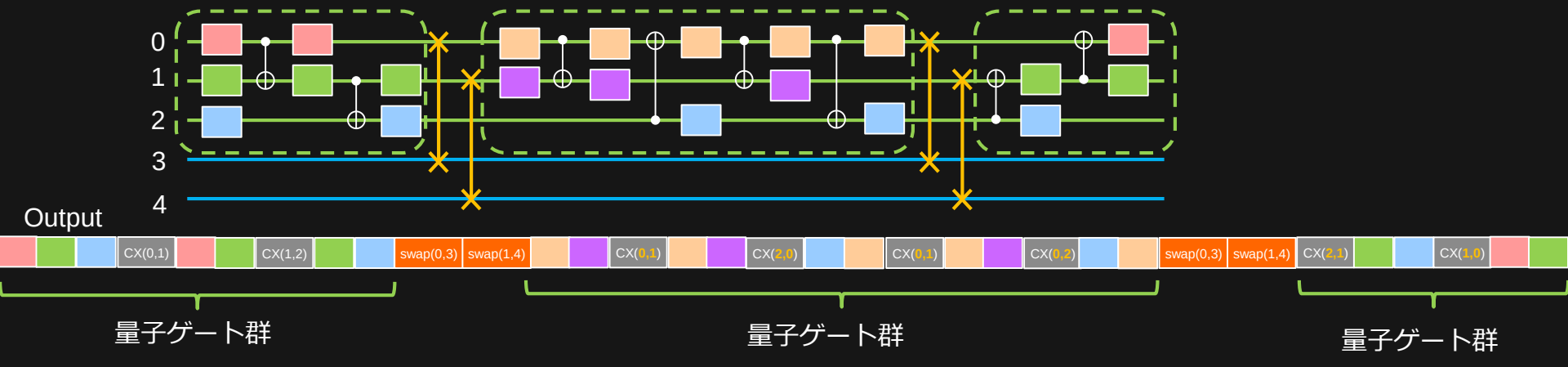
キャッシュブロッキングされた量子回路の出力

IBM Quantum



古典計算機におけるキャッシュブロッキング

IBM **Quantum**




```
for g = gates in circuit sequence
  for c = all chunks
    apply gate g to chunk c
```

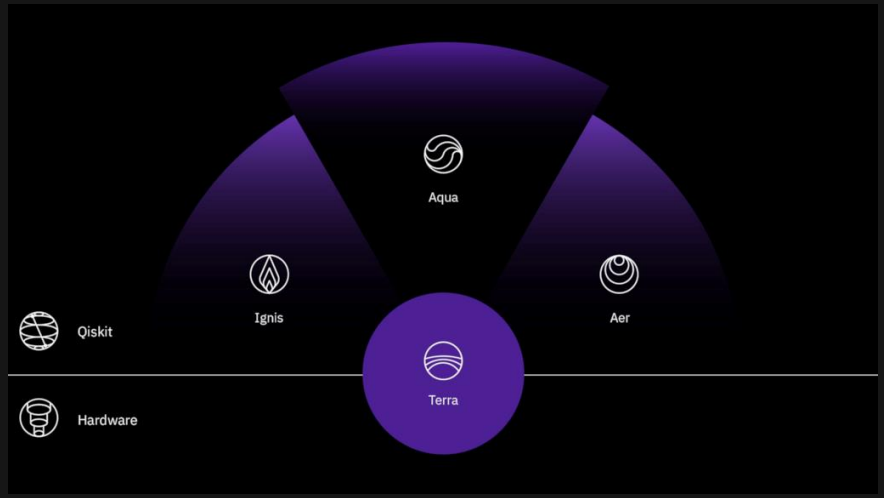
ループスワップ

```
for b = block of gates in circuit sequence
  for c = all chunks
    for g = gates in block b
      apply gate g to chunk c
```

チャンクをキャッシュに入れて
量子ゲート群をまとめて計算することで高速化できる

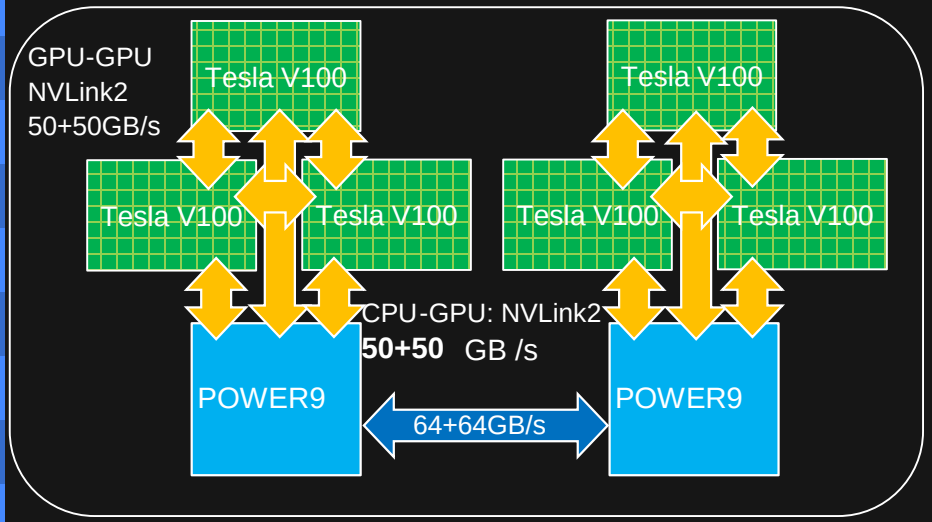
量子計算機シミュレーションの実装

- オープンソースシミュレーターQiskit Aerに実装
  - C++で書かれたシミュレーター
  - GPUを用いた高速化
  - MPIによる分散メモリ並列化
- オープンソースに適したコーディング
  - 可読性，保守性の高いコーディングが必要
  - CUDAのような低レベルな記述は好ましくない
  - Thrustライブラリを利用してコーディング
    - STLのようにGPUを扱える



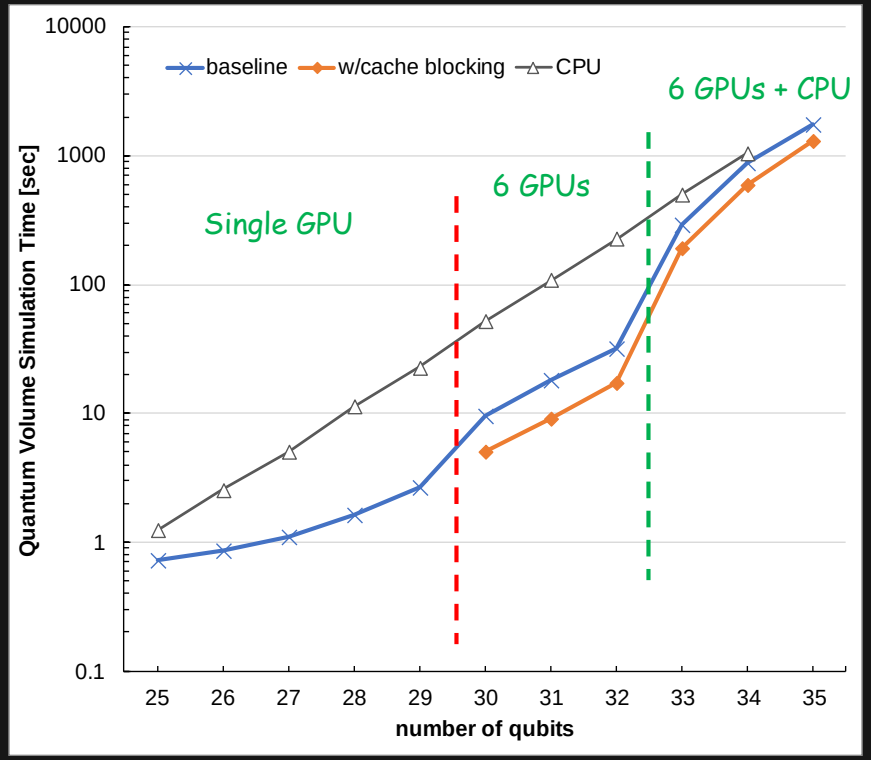
計算機クラスターによる性能評価

計算機ノード	IBM Power System AC922
CPU	POWER9
ノードあたりCPU数	2
CPUあたりコア数	21
CPUメモリ	512 GB
GPU	NVIDIA Tesla V100
ノードあたりGPU数	6
CPU - GPUインターコネクト	NVLink2
ノード間インターコネクト	Infiniband EDR
OS	Red Hat Enterprise Linux 7.6
CUDA Toolkit	10.1
MPI	IBM Spectrum MPI 10.3.1

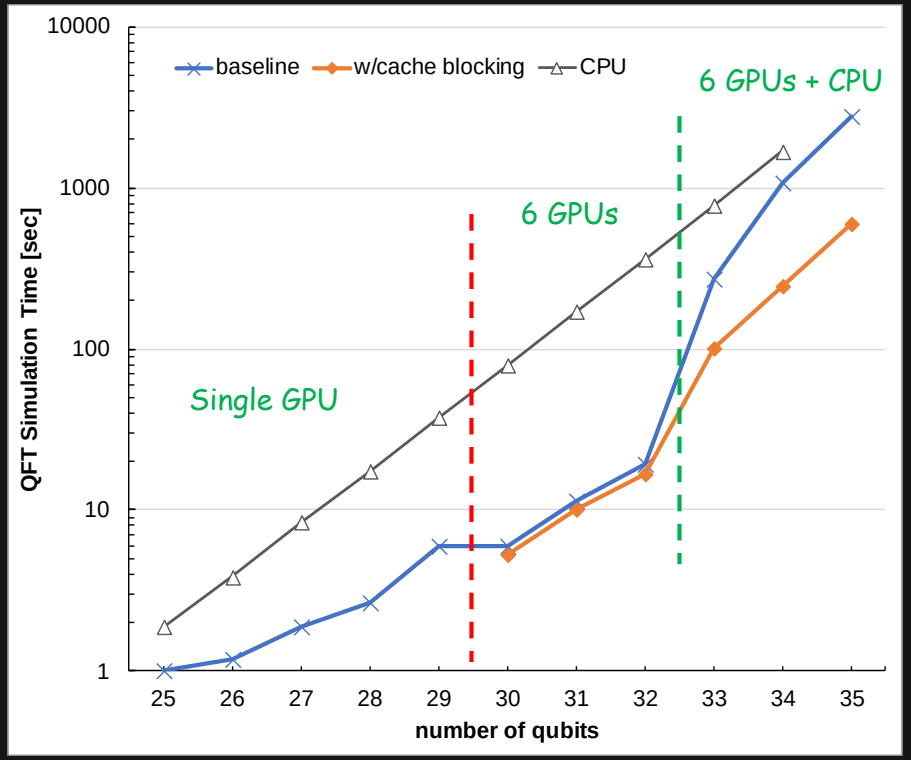


単一ノードでの評価

Quantum Volume (ランダム回路)



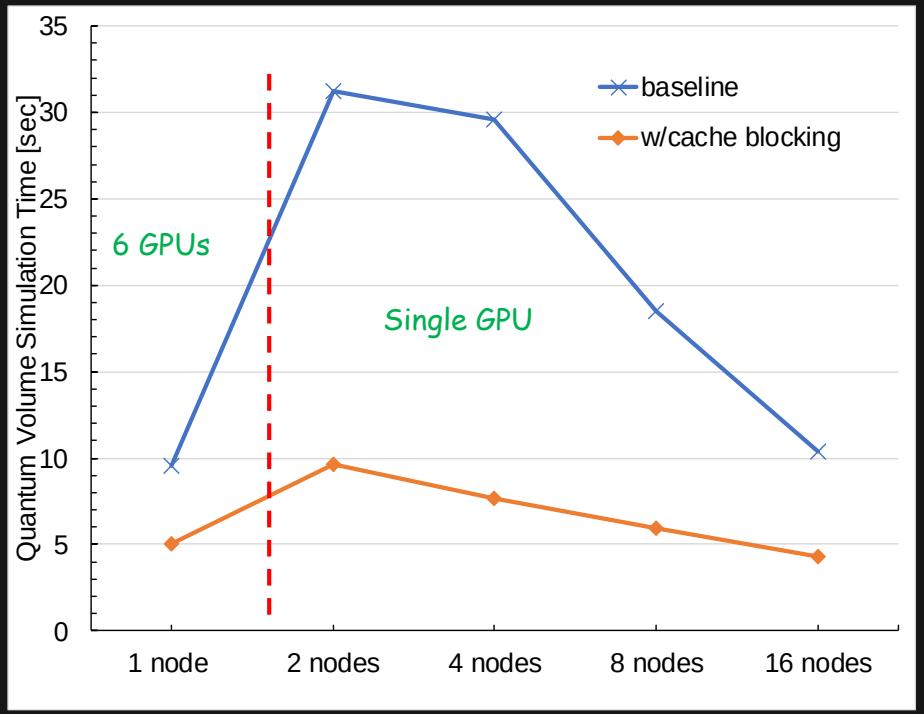
Quantum Fourier Transform (QFT)



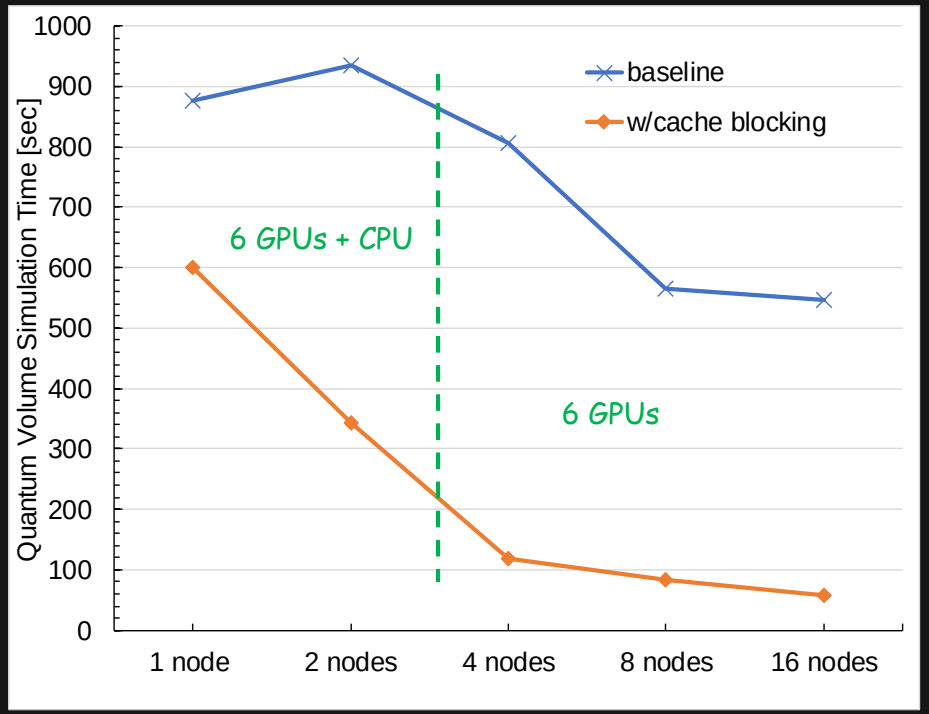
IBM Power System AC922 (6x NVIDIA Tesla V100)

スケーラビリティの評価（強スケーリング）

Quantum Volume, Strong Scaling (30 qubits)



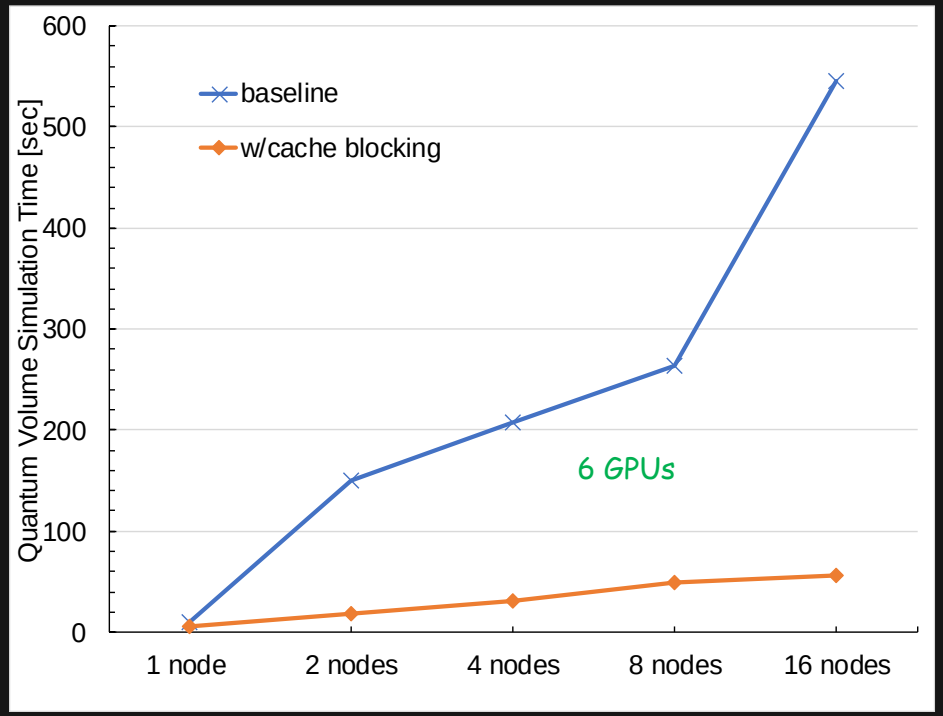
Quantum Volume, Strong Scaling (34 qubits)



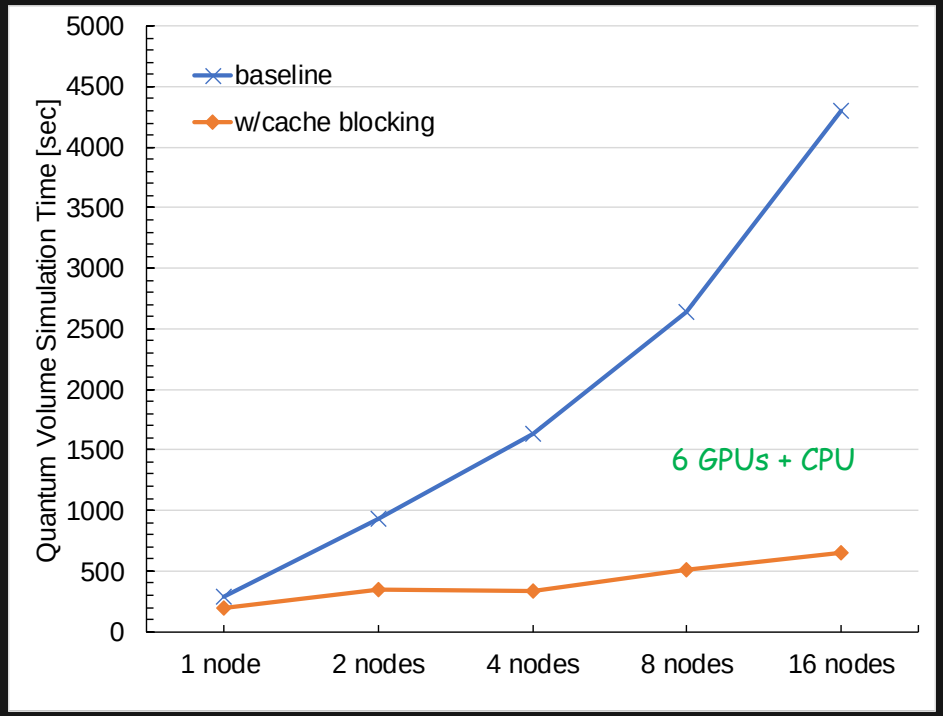
IBM Power System AC922 (6x NVIDIA Tesla V100 per node)

スケーラビリティの評価（弱スケーリング）

Quantum Volume Weak Scaling (2^{30} amplitudes / node)



Quantum Volume Weak Scaling (2^{33} amplitudes / node)



IBM Power System AC922 (6x NVIDIA Tesla V100 per node)

まとめ

- 量子キャッシュブロッキング
  - 速い量子ビットで実行
  - スワップゲートの挿入による量子回路の変換
- 古典キャッシュブロッキング
  - 分散メモリ間のデータ交換を最小化
  - 古典的キャッシュブロッキングによる高速化
- 今後の展望
  - Qiskit Aerへの正式実装（まもなく）
  - ファイルI/Oへのキャッシュブロッキングの適用（さらに大きな量子ビットの回路のシミュレーション）