スパコン・GPUでJuliaを使う：cTPQ状態法を舞台に

大阪大学基礎工学研究科　山田昌彦 2021/9/3　（共同研究者：藤本聡）

今日話すこと
MPI.jlとCUDA.jlの紹介

- 今まで：high-performance computingというと敷居が高いイメージ
  - イメージ：ローレベルAPIを叩く必要がある。C/C++/CUDAやFortranで書かれている。職人技が必要。カリカリにチューニングしなきゃいないんじゃない？　　→全部（ほぼ）不要です！
- Julia+MPI.jl/CUDA.jlでほとんどのことができます！
- 時間がないので、物理の話よりも数値計算の話がメイン。
  - MPI.jlとCUDA.jlを使って何ができるか解説。

スパコンでJuliaを使う

- Julia標準のスレッド並列・プロセス並列なども存在しているが、これらの機能はいずれもまだスパコンでのHPCに最適化されていないし、スパコンがノード間の通信プロトコルとしてMPI以外許容しない場合も多い。スパコンで複数ノード動かして大規模並列計算をするには事実上MPI.jl一択である。
- もちろん、ハイブリッド並列にすることもできる。その場合はJuliaのスレッド並列orプロセス並列or両方と組み合わせることになる（今回は扱わない）。
- MPIを補助するツールも存在するが、今回はMPI.jlの標準機能のみを用いる。

cTPQ法（カノニカル量子純粋状態法）

簡単にいうとトレースを近似する方法 *有限系の場合

- 一言で言うとHaarランダムベクトルを虚時間発展させ期待値を求める。

$$\mathrm{Tr}Ae^{-\beta H} \sim \langle 0 | e^{-\frac{\beta H}{2}} A e^{-\frac{\beta H}{2}} | 0 \rangle$$

- トレースの近似はHilbert空間の次元Dが大きくなるほど精度が良くなる。

- SU(4)ハイゼンベルク模型の場合D=4^Nなので特に有利。また、エントロピー大（＝フラストレーション大）ほど有利。

- 実際の計算はHのべきで展開した方が効率的なので、Hを繰り返しかける計算を考える。

$$|k\rangle = (l - H)^k |0\rangle$$

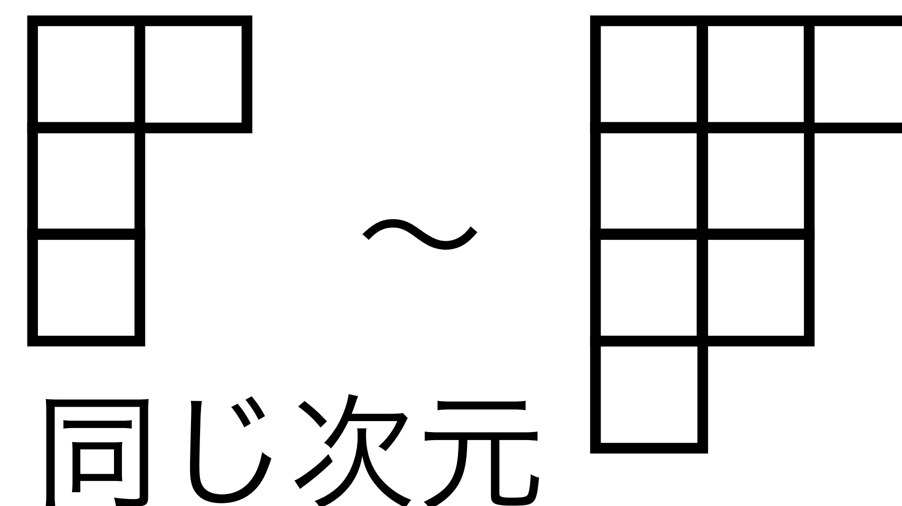
A. Hams and H. De Raedt, PRE **62**, 4365 (2000).
S. Sugiura and A. Shimizu, PRL **111**, 010401 (2013).

Hilbert空間の分解

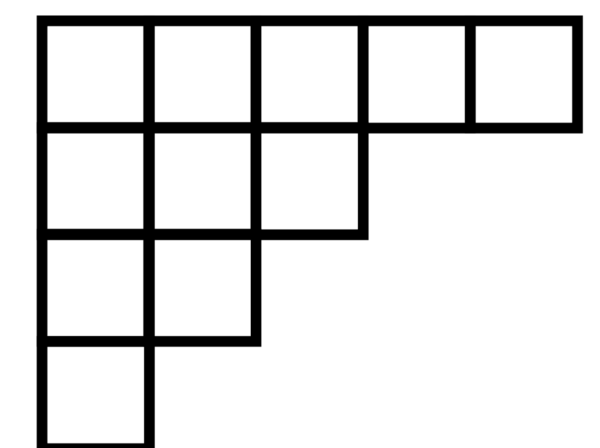
保存量による分解

- そのままだと$D=4^N$次元のベクトルを格納しなければならないので、サイズが限られる。Hilbert空間を直和分解して分割統治する。
- 空間群の対称性など様々な対称性で分割できるが、最も効率よく分割できるのはSU(4)の表現による分割である。SU(4)の既約表現の各セクターごとに分解してcTPQ法の計算を行い、最後に足し上げる。
- SU(4)の既約表現は4行までのヤング図形によって表現される。

例：随伴表現



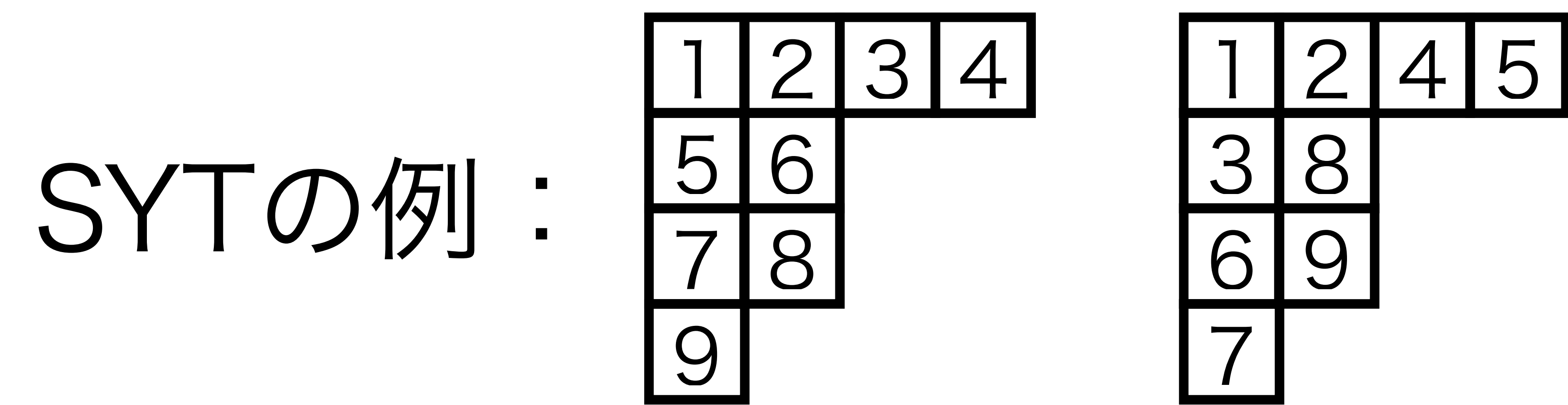
適当な表現



SYTとτ行列

Hilbert空間の各基底はstandard Young tableauと一対一対応

- 各既約表現に属するHilbert空間の各基底はstandard Young tableaux (SYT)でラベルされる。
- SYTの定義：1から順番に整数を
    - 右に行ったら大きくなり、
    - 下に行ったら大きくなるように埋めていく。



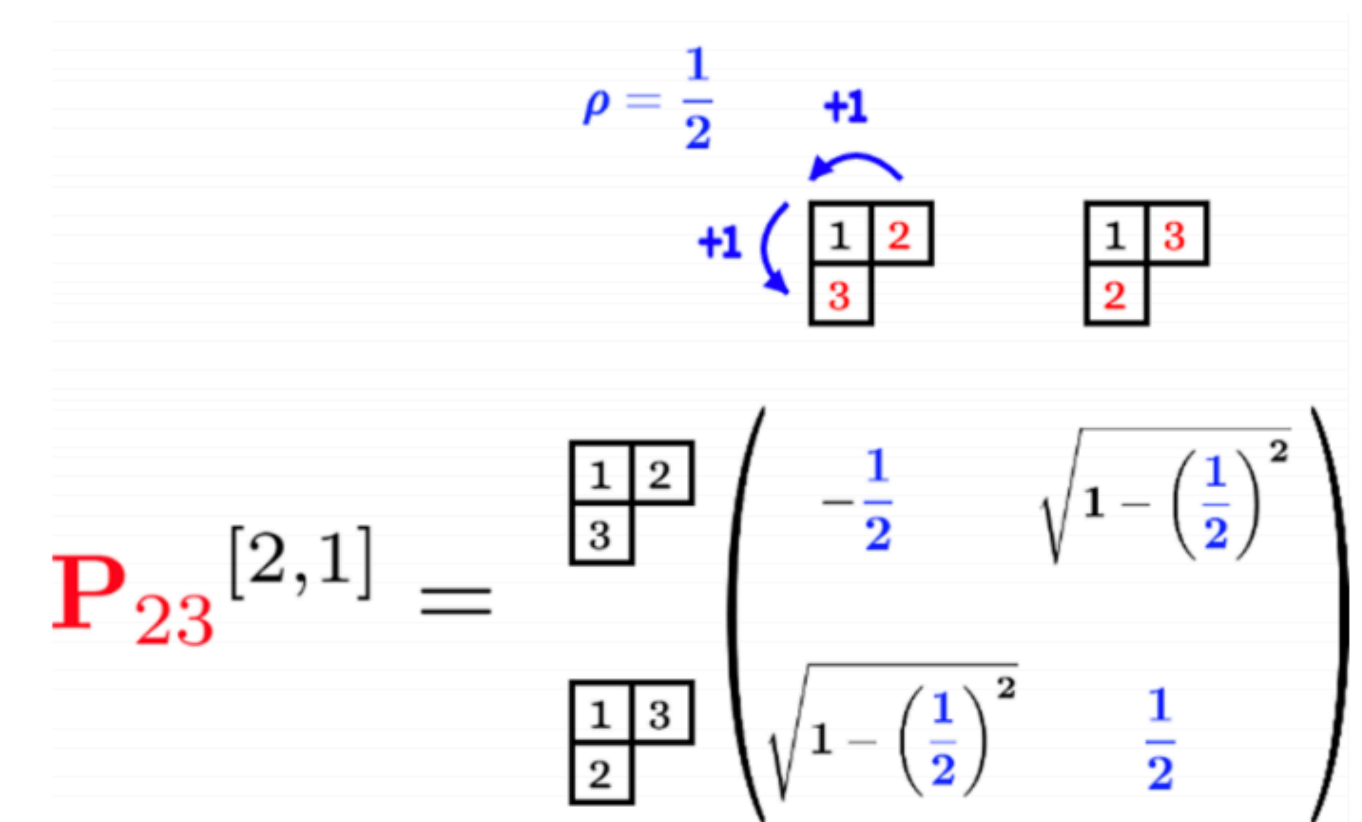
P. Nataf and F. Mila, PRL **113**, 127204 (2014).

- ハミルトニアンは次に述べるτ行列という疎行列だけから再構成できる。
- （発展）SYTの順引き/逆引きが問題となる→Wilf-Rao-Shanker法

非対角項

τ 行列の非対角項は非常に少ない

- τ 行列（隣あう番号のサイトのハイゼンベルク項）の非対格要素はSYT tに対して、SYT uのみ。
  - SYT uは τ_{ij} ならばSYT tのiとjを入れ替えて得られる。



P. Nataf and F. Mila, PRB **97**, 134420 (2018).

- 非常にスパースなので、事前に相方uと相方uとの“axial distance”だけ計算して保存しておけば十分。

- axial distanceは8bit intに収まる→メモリが減らせる。

Message-passing-based ED/cTPQ

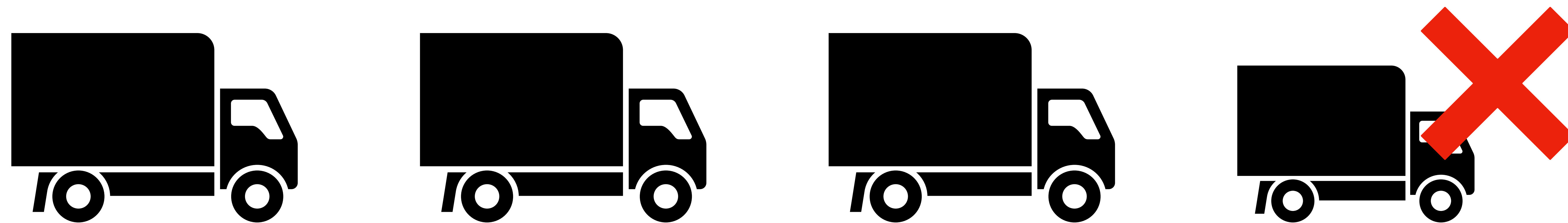
MPI Alltoallvの利用

- 行列の掛け算を安直に実装すると、MPIのAllgathervがメモリ不足で落ちる。
- 各プロセス毎にtに対するuのテーブルを事前に生成すれば、τ 行列を掛け算したときの各要素の行き先は事前にわかる。→行き先を指定してAlltoallv
- これでMPI並列数分だけ手元に置いておくメモリ量が減らせる。
    - 18,432並列の時、7,228,208,988次元ベクトル→392,156次元ベクトル
    - アドレスが32bit intに収まる→さらにメモリが減らせる。

発展：ロードバランスについて

MPIの宿命

- MPIに限らず並列化に宿命的な問題はロードバランシングである。



- 積荷の量（計算量）が事前にわかる場合 → Monte Carlo load balancer
  - 計算量の最大値$E = \max_i \{load_i\}$をコスト関数（エネルギー）としてEを小さくするように古典モンテカルロ法で低温に持っていき、最適化する。
  - 例えば二人でsvdの計算を分け合う場合 $E = \max \left\{ \sum_{\sigma_i=\uparrow} m_i^3, \sum_{\sigma_j=\downarrow} m_j^3 \right\}$を小さくすれば良い。
- これが一般論だが、今回はほぼ自動的にロードバランスが最適になるので不要。

CUDA.jlの利用

GPUはむしろMPIより簡単！

注意：忘れない

- **配列を初期化する関数にCUDA.をつける。**
  - zeros(m, n) → CUDA.zeros(Float64, m, n)
- 時々CPUとGPUのストリームを同期する関数CUDA.synchronize()を呼ぶ必要がある。
- **GPU上で動かす関数は@cuda付きで呼ぶ。** 厳密にはそれだけではダメで少し書き換えが必要
- BLASをCUBLASに置き換える。
- LAPACKをcuSOLVERもしくはMAGMAに置き換える。
- **メモリがCPUとGPUのどちらにあるか意識する。**

CPUの型	GPUの型
Array	CuArray
Vector	CuVector
Matrix	CuMatrix

CPUの関数	GPUの関数
BLAS.syrk!	CUBLAS.syrk!
svd!	magma_gesdd!等

MAGMA.jlをロードする必要あり！

発展：@cudaで呼ぶ関数の例

- たくさんの線形代数演算がCUDA.jlで標準装備されているので、ほとんど関数はそのまま動かせる。
- 自分で作った関数をGPUで動かしたい時は、CUDAのlow-levelの仕組みをある程度学ぶ必要あり。
- grid -> block -> thread
- 基本的に機械的な書き換えで行けるので省略。

```
24  function τ!(y, x, k1::Int64, u1listdata, axialdata)
25      index = (blockIdx().x - 1) * blockDim().x + threadIdx().x
26      stride = blockDim().x * gridDim().x
27      for t1 in index : stride : length(y)
28          @inbounds begin
29              ρ = 1.0 / axialdata[t1, k1]
30              y[t1] = sqrt(1.0 - ρ ^ 2) * x[u1listdata[t1, k1]] - ρ * x[t1]
31          end
32      end
33      nothing
34  end
```

τ 行列をかける関数

発展：CUDA-aware MPIについて

今回は必要ないが、CUDAとMPIを組み合わせる必要があることもある

- CUDA-aware MPIとはCPUを経由せずにGPU間でMPI通信する機能。使うのは簡単で、Juliaの側から特にすることはない。
- CUDA対応Open MPIをインストールするのが少々面倒（詳細略）。
- 使い方は簡単で、MPI.jlの関数（一対一通信・集団通信）にCuArrayを投げれば良い。CPUとGPUで共通のコードが使える。
- 注意点としては、CUDA-aware MPIの関数を呼ぶ直前にCUDA.synchronize()を呼ぶ必要がある。←忘れがちなので注意！

実際の計算

- 18,432 flat MPI並列（N=24）on 物性研スパコンohtaka
  - ハイブリッド並列をした方がいいんじゃないかと思われるかもしれないが、今回あまり効果的ではなかった。
- GPUを使った並列（N≦20）on 研究室の計算機（NVIDIA A100 x 2）
  - NVIDIA A100には40GB/GPUメモリが載っているので、かなり大きなサイズまでcTPQ計算ができる。今回はCUDA-aware MPIは使わず、二つのGPUで独立した計算を回している。

まとめ

MAGMA.jl	公開済み
SUNWignerSymbols.jl	公開済み
SUNED.jl（仮）	**公開予定**

- JuliaはスパコンでもGPUでも使うのはとても簡単！
- 今まで専用のコードを書いて動かしていたCUDAがJITコンパイルでスクリプト言語のように動かせるのは感動もの。未経験者でもすぐに使えます。
- MPI.jlを使ってコードを書くのは、MPIの経験さえあれば難しくないです。MPI未経験者にはMPIそのものに慣れるのに時間がかかるかと思いますが、それさえ理解すれば実装は単純明快です。
- 最終的に最速を目指すならば当然カリカリにチューニングはしなければいけないです。逆に言うと最速を目指さなければとても読みやすいコードのまま十分な速度を出せます。Monte Carlo load balancerのような簡単な工夫でもかなりの効果があります。