格子 QCD
富岳からポスト富岳、CPU から GPU へ向けて

理化学研究所 計算科学研究センター
金森 逸作

2024 年度 第 2 回計算科学フォーラム
2025 年 3 月 31 日
オンライン

もくじ

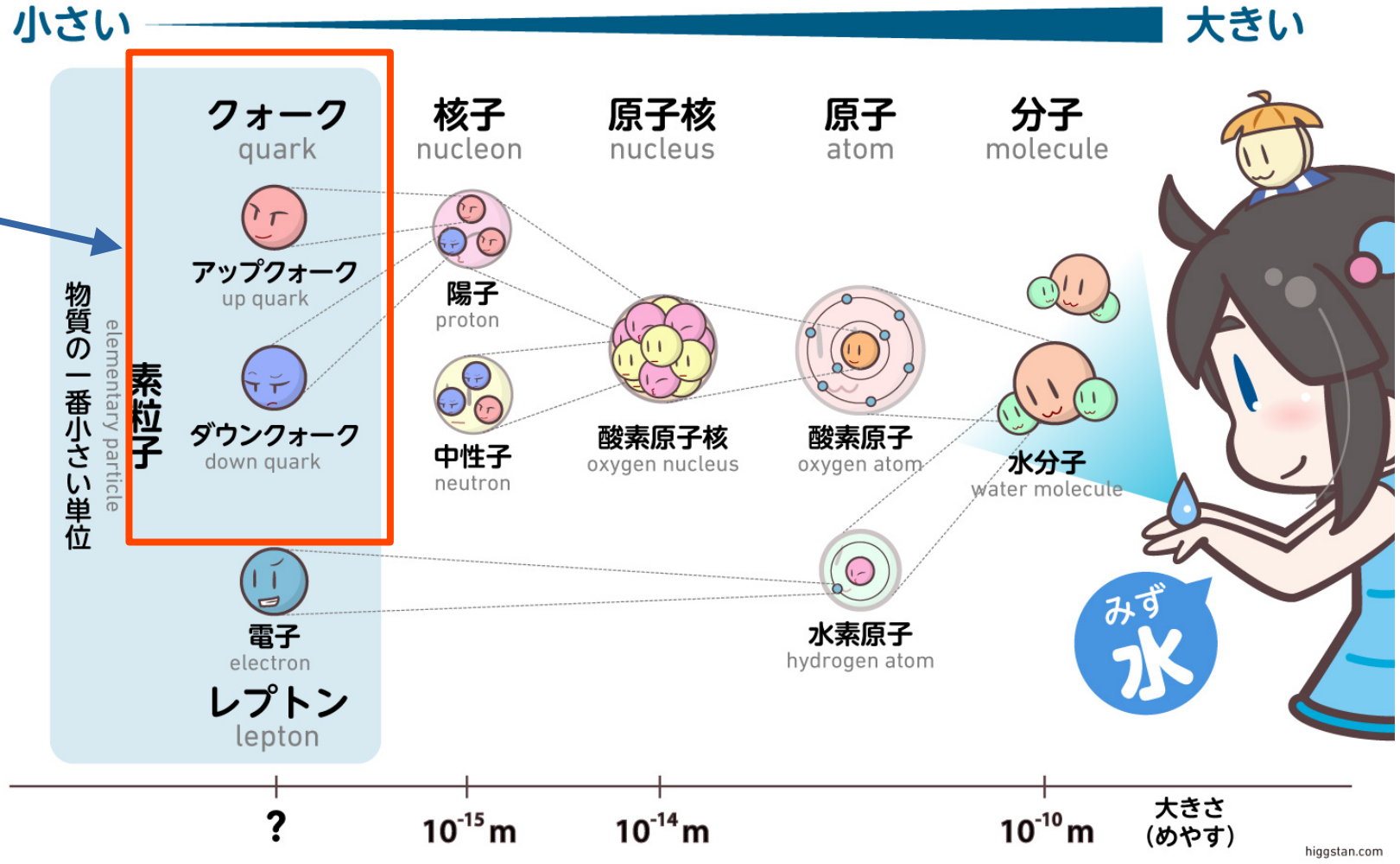
お断り：ポスト富岳の調査研究（理研）に参加しましたが、本講演は個人的な見解に基づくものであり、調査研究を代表するものではありません

対象：微小世界の物理学

クォークと
グルーオン

量子色力学
Quantum
ChromoDyanimcis
(QCD)

素粒子標準模型
の精密検証
（KEK加速器実験）
核力の導出



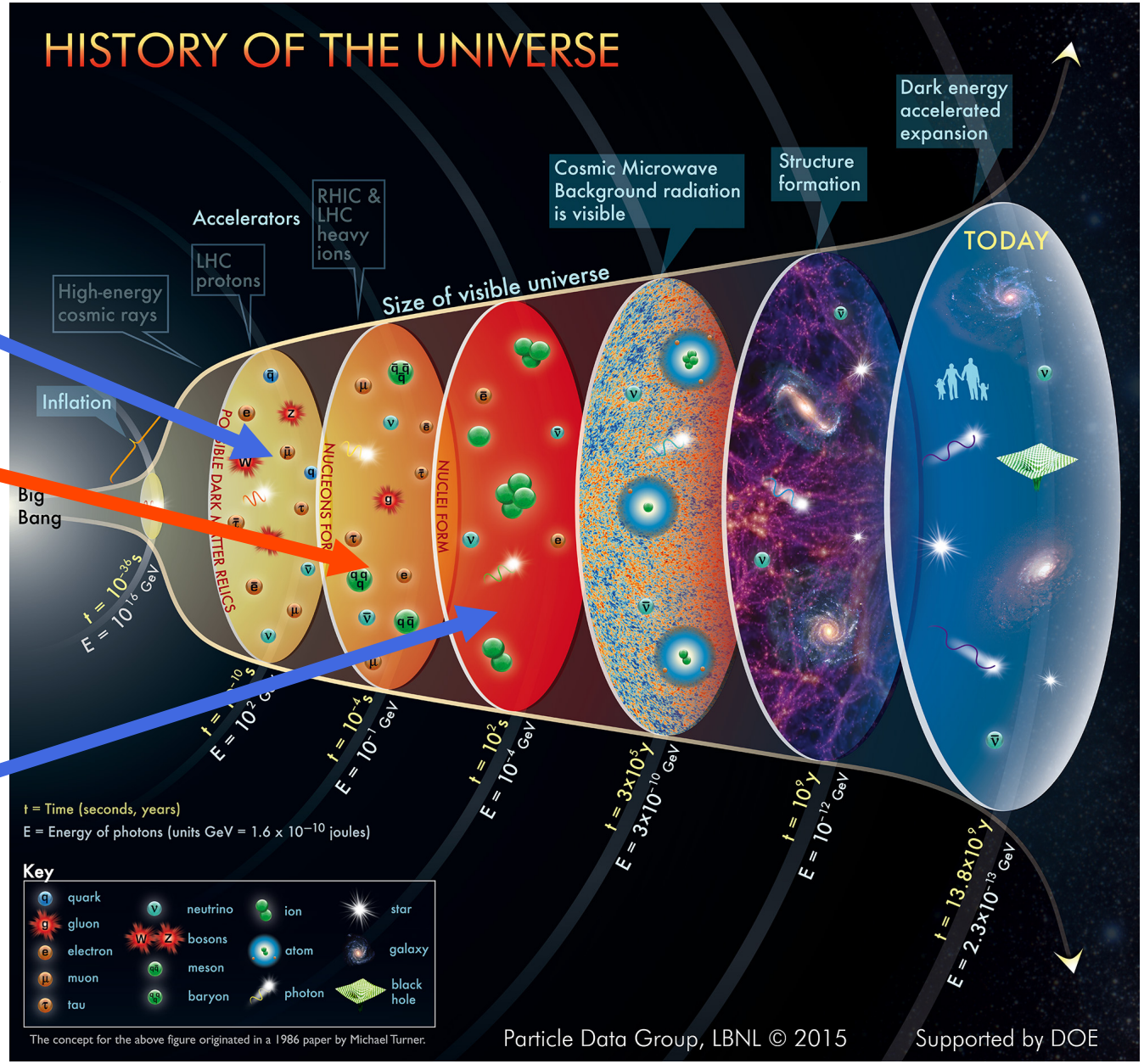
初期宇宙

10^{-10}秒後：1000兆度
クォークたちがバラバラの世界

10^{-4}秒後：1.75兆度
(～150 MeV)
陽子や中性子ができる

クォーク・グルーオンのプラズマ相からハドロン相へ
cf. 水蒸気から水へ

100秒後：10億度
原子核ができる



格子QCD

- 4次元格子上でQCDを記述
- 量子論：経路積分をモンテカルロ法で評価（シミュレーション）

$$\int \mathcal{D}U \det D[U] \mathcal{O}[U] \exp(-S[U])/Z = \frac{1}{N}\sum_{i=1}^{N} \mathcal{O}[U_i] + O\left(\frac{1}{\sqrt{N}}\right)$$

誤差

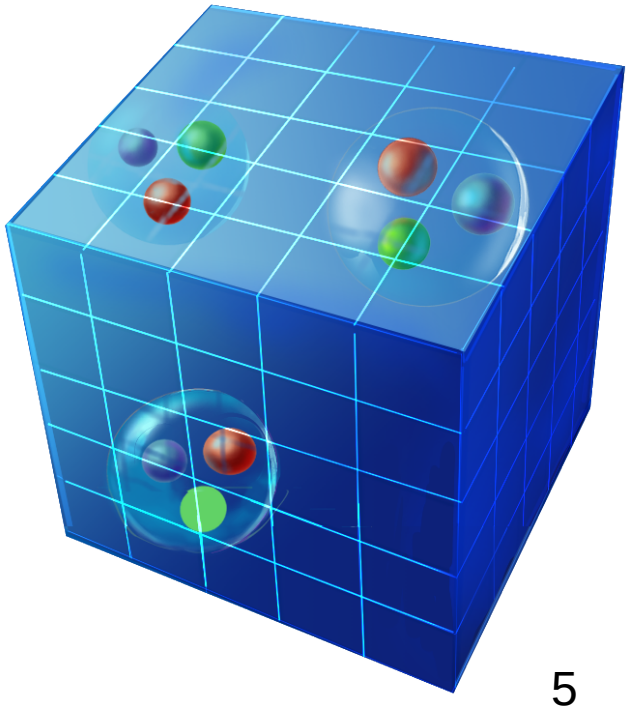
真空分極の効果

律速：線形ソルバー、反復法

測りたいもの

配位

- **配位** U_i の質、**測りたいもの** $\mathcal{O}$ の複雑さ、**統計** N
- パラメータ
  - 格子間隔
  - 格子サイズ
  - クォークの種類、質量
  - クォーク離散化の方法（対称性）



計算のボトルネック

- Dirac 方程式 $Dx=b$ を解くソルバー
- Dirac演算子D の作用（疎行列ベクトル積、ステンシル計算）

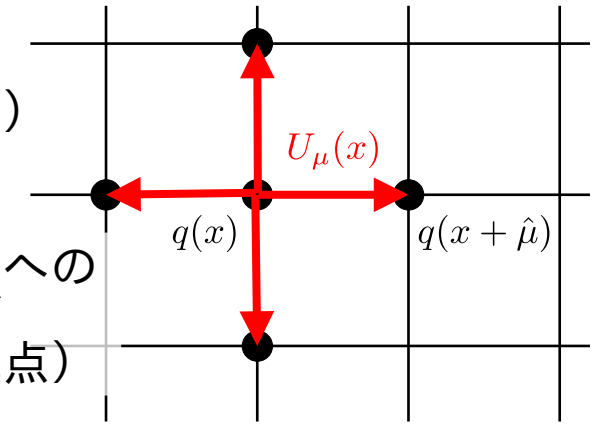
$$\det D^\dagger D = \int d\phi^\dagger d\phi \exp[-\frac{1}{2}\phi^\dagger (D^\dagger D)^{-1}\phi]$$

$$\text{cf.} \quad \int \mathcal{D}U \det D[U]\mathcal{O}[U]\exp(-S[U])/Z = \frac{1}{N}\sum_{i=1}^{N}\mathcal{O}[U_i] + O\left(\frac{1}{\sqrt{N}}\right)$$

$U_i \to U_{i+1}$ へ進むのに、数百〜数万回解く（自己相関距離にも依存）
多くの場合、$\mathcal{O}[U_i]$ の計算にも必要：U_i あたり、１０〜数千回

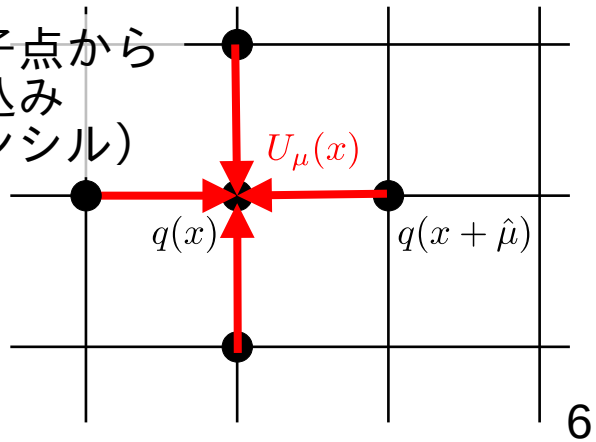
- 反復法で解く: CG法、BiCGStab 法など
- 連続極限に近づくにつれて、ill-conditioned になる: 効率のよい前処理が重要

格子点には複素12成分
（カラー x スピノール）
リンク上に複素 3x3 行列（カラー）



隣の格子点への
ホッピング
（物理屋視点）

隣の格子点から
の足す込み
（ステンシル）



「京」から「富岳」へ

CPU の比較

	京	富岳	比
	SPARC 64 VIIIfx	A64FX	
Core の数	8	48 (+2 or 4)	6
SIMD幅	128 bits	512 bits	4
レジスタ数	256	32	1/8
精度	fp64	fp64/fp32/fp16	
メモリバンド幅	64G B/s	1024 GB/s	16
メモリ容量	16 GB	32 GB	2
ピーク性能	128 GFlops	3072 GFlops (2.0GHz, fp64)	24
B/F 比	0.5	0.33	0.67
ノード間通信	20 GB/s	40 GB/s	2

富岳の特徴
- 単精度(fp32)のサポート
- SIMD幅の拡大
- レジスタ数の減少、長い命令レイテンシ

個人的な感想
- 京：レジスタ変数を多数使って高速化、複素数向けのSIMD命令は潤沢
- 富岳：SIMD化必須、メモリバンド幅を使い切るのは難しい

富岳向けのコデザインでやったこと（格子QCD）

cf. 富岳コデザインレポート(2022)

- Tofu barrier を使ったMPIのreduction 命令の高速化
  - ３要素までの global reduction が高速化（「京」では１要素）
  - （少数要素の）MPI global reduction を使うすべてのアプリに恩恵
- QCD Wide SIMD ライブラリ（QWS）の開発 https://github.com/RIKEN-LQCD/qws
  - Wilson 型格子フェルミオン向けの混合精度ソルバー
  - SIMDと uTofu（通信の低レベルインターフェース）を利用
  - 供用開始時に、QWSを利用できるアプリはなかった
    QWS の中身を独自に再実装した LDDHMC が用意された 石川健一さん
    BQCD 向けにインターフェースが用意された 中村宜文さん

QWS：性能は良いが、そのままでは使えない・使いにくい

QWS のその後

- 開発時のノウハウを用いた rankmap tool の開発 金森
  https://github.com/RIKEN-LQCD/rankmap4d
- Bridge++（国内で開発されている汎用格子QCDコードセット）に組み込まれた
  

  2023年3月
- 富岳向け Bridge++ の複素数のデータ構造は、QWSと同じ rrrriiii
- Intel AVX-512 版（富岳NEXTのFS中に、インテルの人が移植）

QWSの開発には関わったが、「QWS」という問題設定には関わらなかった立場からの反省点

- ノウハウは生きたかもしれないが、直接的な利用が限定的
- アプリの継続性の問題
  - 使っているコードを富岳向けに改造 LDDHMC
  - 既存のコードから呼び出し Bridge++, BQCD　　特にここが弱かった
- 国内の大規模シミュレーション：Wilson と Domainwall フェルミオンのうち、Wilson のみ
  Domainwall: 主に、ドイツ レーゲンスブルク大のQPACE4 （FX700）向けに最適化されたものを利用（Grid）

「富岳NEXT」へ

富岳NEXTの目標性能

	富岳	富岳NEXT CPU	富岳NEXT 加速部	比（加速部/富岳）
ノード数	158976	3400+	3400+	
FP64	488 PFlops	48+ PFlops	3.0+ EFlops	6.2+
FP16/BFP16 行列演算	1.95 EFlops	1.5+ EFlops	150+ EFlops	77.7+
FP8行列演算	N/A	3.0+ EFlops	300+ EFlops	
メモリサイズ	4.85 PiB (32 GiB / node)	10+ PiB (~3 TiB/node)	10+ PiB (~3 TiB/node)	4.1 +
メモリバンド幅	163 PB/s	7+ PB/s	800+ PB/s	4.9+
B/F 比	0.33	0.15	0.27	0.8

富岳NEXTの数字はhttps://www.r-ccs.riken.jp/media/20250204/より

- 倍精度演算の性能は6倍程度
- 性能を出すには、低精度演算の利用が必須
- B/F比（倍精度）はあまり変わらない
- ネットワークは、加速部の同士の直接接続あり

富岳NEXTの特徴

- 演算加速器（GPU）
  何で書くかはベンダーに依存？ CUDA, OpenACC, SYCL, kokkos, …
- 低精度演算は、倍精度演算より格段に速い
- 行列演算（NVIDIA だったら、テンソルコア）だともっと速い
- メモリバンド幅は5倍程度
  QCDはメモリバンド幅律速なので、何もしなければ5倍にしかならない
- 通信の高速化（バンド幅・遅延）もあまり多くは期待できない

混合精度ソルバー　富岳でも使われている

- 低精度のほうが速い、メモリアクセスも少ない　メモリバンド幅律速
- 隣接通信の通信量も少なくなる
- 基本は残差反復
  - $Ax=b$ を解きたい、x の近似解x_n は手元にある
  - $r=b-Ax_n$ として、$x_n+A^{-1}r$ は解になっている:　$A(x_n+A^{-1}r)=Ax_n+r=b$
  - $A^{-1}r$ は無理でも、近似的な $A_{\mathrm{app}}^{-1}r$ なら計算できるとする

$r_0 := b - Ax_0$
$i = 0$
while $|r_i|$ *is not small enough* **do**
　$x_{i+1} := x_i + A_{\mathrm{app}}^{-1} r_i$
　$r_{i+1} := b - Ax_{i+1}$
　$i := i + 1$

$A_{\mathrm{app}}^{-1} r_i$ に低精度のソルバーを用いる

混合精度ソルバー

1 $r_0 := b - Dx_0$ 黒：倍精度　赤：単精度
2 $i = 0$
3 **while** $|r_i|$ *is not small enough* **do**
4 $\quad r_i^{(\mathrm{single})} := \mathrm{to\_single}(r_i/|r_i|)$
5 $\quad \delta x^{(\mathrm{single})} := D_{\mathrm{app}}^{-1} r_i^{(\mathrm{single})}$ low prec. solver
6 $\quad x_{i+1} := x_i + \mathrm{to\_double}(\delta x^{(\mathrm{single})}) \times |r_i|$
7 $\quad r_{i+1} := b - Dx_{i+1}$
8 $\quad i := i + 1$

- 演算のほとんどが、低精度ソルバー
- 残差を倍精度で計算することが重要

低精度（単精度）ソルバーの問題点

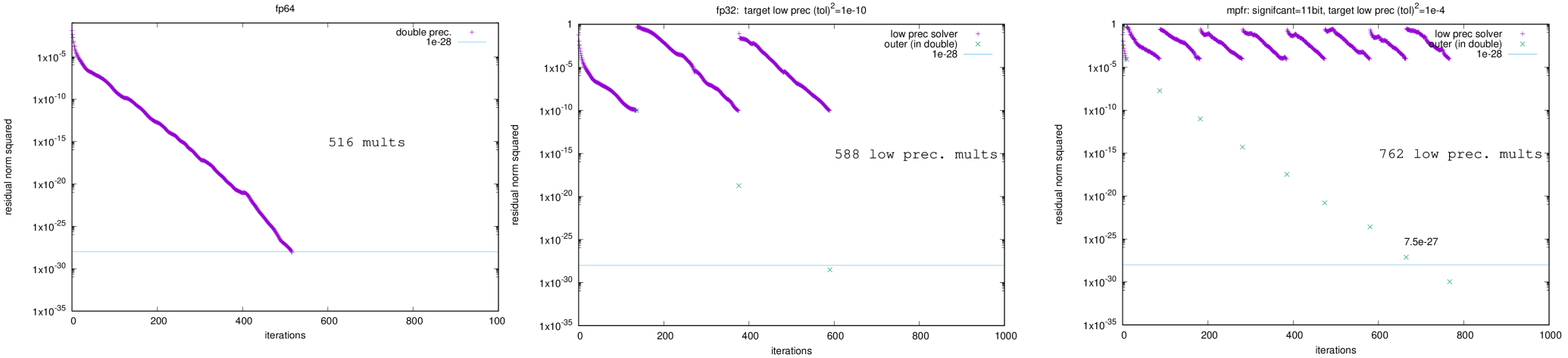
$r_0 = b - Ax_0$
$p_0 = r_0$
$i = 0$
while $|r_i|$ *is not small enough* **do**
  $\alpha_i = \frac{(r_i, p_i)}{(p_i, Ap_i)}$ inner prod (a, b)
  $x_{i+1} = x_i + \alpha_i p_i$
  $r_{i+1} = r_i - \alpha_i A p_i$
  $\beta_i = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)}$ norm
  $p_{i+1} = r_i + \beta_i p_i$
  $i := i + 1$

Conjugate Gradient (CG) solver

- 内積・ノルムの計算：要素数は 10^8-10^9
- 単精度の有効桁：7桁程度（半精度は3桁程度）
- 単精度だと、10^8 + 1 = 10^8
  小さすぎる
  →内積やノルムが正しく求まらない
- 部分和の足し上げなどの工夫が必要
- | 内積 | << | ノルム | のときは桁落ちの恐れ
  とくに BiCGStab
- 内積やノルムの計算だけ倍精度（あるいは多倍長）にするのも選択肢
- 並列計算では、結合則の破れ
  $(x+y)+z \neq x+(y+z)$ もある
  reproducible summation (Ahrens+ 2020) の利用:
  Lattice 2023, Kate Clark
  https://indico.fnal.gov/event/57249/contributions/270632/
- Bridge++ では、部分和 ＆ global reduction は倍精度

倍精度、単精度＋倍精度、半精度＋倍精度

- Moebuis Domainwall フェルミオン 32x8x8x12 (行列ランク 3.5×10^6)
- 半精度は任意精度ライブラリ mpfr を利用: 指数部は半精度より大きい mpreal (https://github.com/advanpix/mpreal 経由)



収束条件：$|r_i|^2<10^{-28}$
- 単精度：$|r_i|^2<10^{-10}$
- 半精度：$|r_i|^2<10^{-4}$

半精度を利用すると反復数が50%増加
ダイナミカルレンジは、本物の半精度より大きい

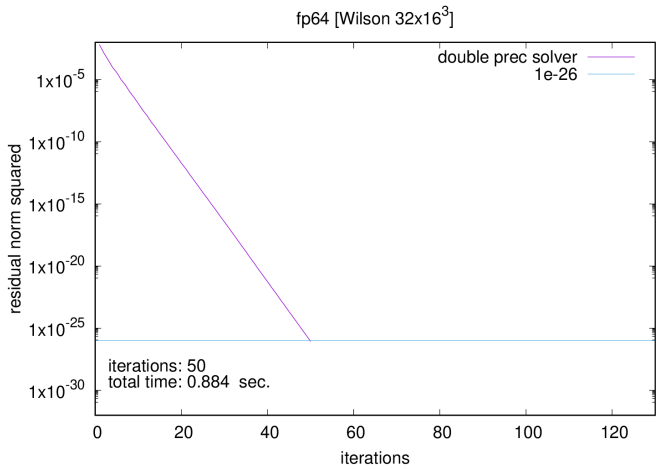
$$\begin{pmatrix} D_{ee} & D_{eo} \\ D_{oe} & D_{oo} \end{pmatrix} \begin{pmatrix} x_e \\ x_o \end{pmatrix} = \begin{pmatrix} b_e \\ b_o \end{pmatrix}$$

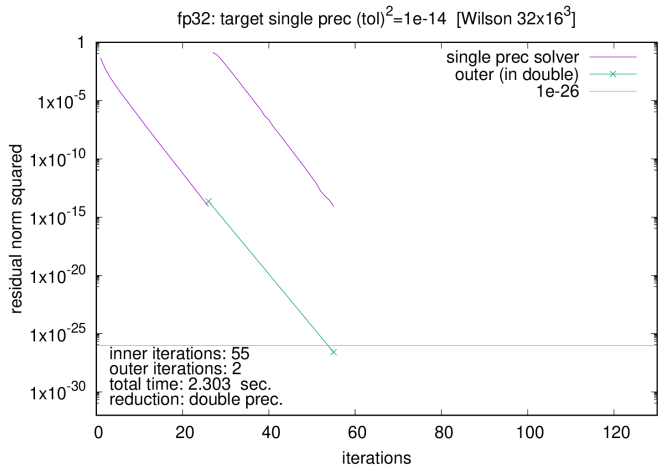
$$(1 - D_{ee}^{-1} D_{eo} D_{oo}^{-1} D_{oe}) x_e = \tilde{b}_e$$

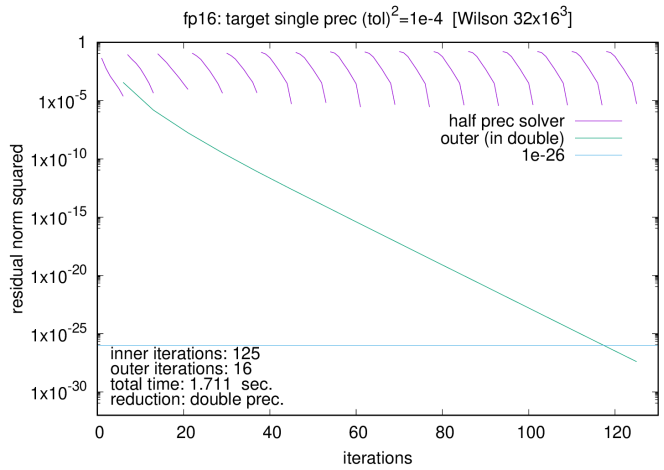
をCGNRで解いている

半精度ソルバー（富岳）

- 本当の半精度+シンプル＆解きやすい設定　Wilson fermion, 32x32x32x16







富岳 １ノード、４MPI processes
半精度を利用すると、反復数は2.5倍
倍精度・半精度間の変換（tuningしてない）のためか、実行時間は2倍
　行列ベクトル積の性能：176 GFlops (double), 382 GFlops (single), 935 GFlops (half)

半精度ソルバー（富岳）

- より複雑な行列（Domainwall fermion）&現実的な設定

  収束しなかった

- 実装は単純な行列ベクトル積にはなっていない（次のスライド）

$$\mathrm{D}_{DW} = \begin{pmatrix} D_+^{(1)} & D_-^{(1)}P_- & 0 & \cdots & 0 & -mD_-^{(1)}P_+ \\ D_-^{(2)}P_+ & D_+^{(2)} & D_-^{(2)}P_- & & & 0 \\ 0 & D_-^{(3)}P_+ & D_+^{(3)} & D_-^{(3)}P_- & & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & & & D_-^{(L_s-1)}P_+ & D_+^{(L_s-1)} & D_-^{(L_s-1)}P_- \\ -mD_-^{(L_s)}P_- & 0 & \cdots & 0 & D_-^{(L_s)}P_+ & D_+^{(L_s)} \end{pmatrix}$$

$$D_W(x,y;M) = (4+M)\delta_{x,y} - \frac{1}{2}\sum_{\mu=1}^{4}\left\{ (1-\gamma_\mu)U_\mu(x)\delta_{x+\hat{\mu},y} + (1+\gamma_\mu)U_\mu^\dagger(x-\hat{\mu})\delta_{x-\hat{\mu},y}\right\}$$

$P_\pm = \dfrac{1\pm\gamma_5}{2}$　　スピノール（4成分）を自由度半分に射影

Domainwall 型の詳細

- D(格子点, 5th, spinor, color) : 添字は4つ
- 構造　$D = M + HG$

隣接格子点を結ぶ（5th は対角的）

格子点とcolorは対角的

途中で多くの演算：精度が落ちる危険性
途中の演算マージするべき
$(1 \pm \gamma_5)$ の射影を、上半分・下半分に

準備中

解の 5th coordinate 依存性が指数関数的
（軽減するような前処理があれば嬉しい）

- 実装：M, H, G それぞれをかけている
  その中で、たとえば H では
  - $(1+\gamma_\mu)$で4成分スピノールを２成分スピノールに射影
    $s_1 = \psi_1 + \psi_3, s_2 = \psi_2 + \psi_4$ など。方向によって具体形は異なる
  - ２成分スピノールにcolorの3x3行列 U_μ をかける
  - ４成分スピノールに足し込む
- Even-odd 前処理：$D_{ee}^{-1} = M^{-1} = U^{-1} L^{-1}$　解くのは $(1 - D_{ee}^{-1} D_{eo} D_{oo}^{-1} D_{oe}) x_e = \tilde{b}_e$

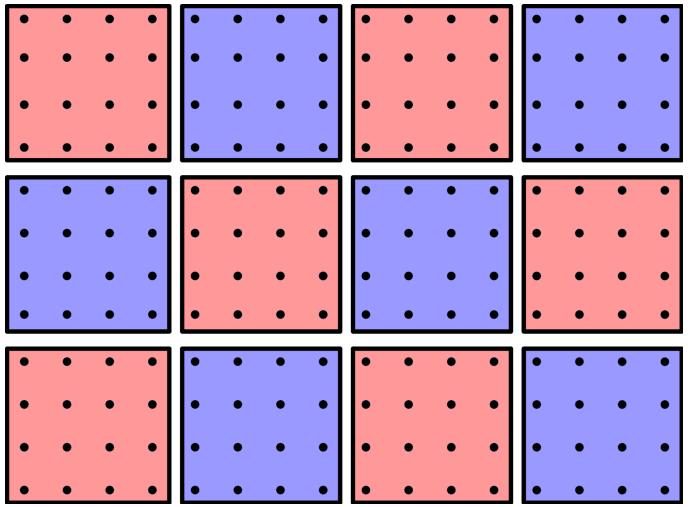
行列エンジン: 利用できる場所あるか？

- 同一の U_μ が複数の2成分スピノールにかかる: (3x3 行列) x (3x2 行列)
- （Domainwall 型限定 ）: (3x3 行列) x (3x(2xLs) 行列) Ls: 5th extent
- D_{ee}^{-1} についても使えるかもしれない
- 懸念事項: メモリバンド幅律速なので、演算を加速しても効果は限定的
  期待:キャッシュの利用効率があがることで速くなるかも

通信を避ける工夫の例: SAP

- 演算に比べて、通信の高速化は期待薄
- 通信頻度を減らすアルゴリズムが重要
- 既に使われている: QWS など

1 $x := x_0, r := b - Dx$
2 **while** $|r|$ *is not small enough* **do**
3 $\quad x := x + D^{-1}_{(\mathrm{EVEN})} r$ /* solve in even domains */
4 $\quad r := b - Dx$
5 $\quad x := x + D^{-1}_{(\mathrm{ODD})} r$ /* solve in odd domains */
6 $\quad r := b - Dx$

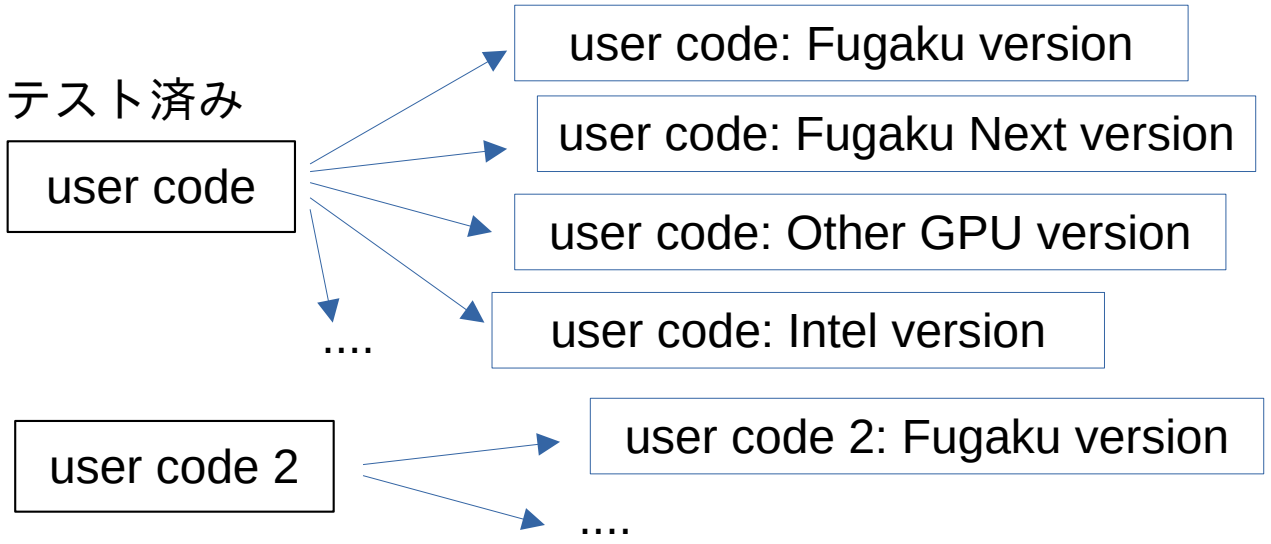


Schwartz Alternating Procedure

- EVEN 領域と ODD 領域を交互にとく　　EVEN-ODD 間のリンクは忘れて解く
- 各領域の solver は適当な反復法
- 短距離モードはすぐに解に近づく
- 長距離モードを解くのは苦手

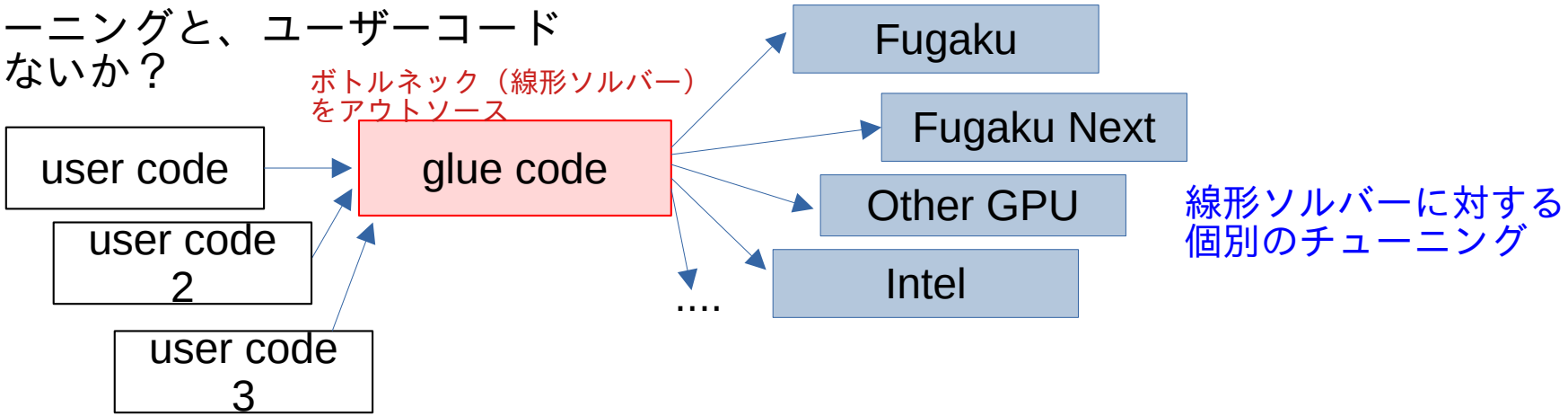
コデザインとユーザーコードの親和性: Glue code?

富岳での反省を踏まえて、コードの可搬性を保ちながら富岳NEXTにどうつなげるか



ターゲットごと書き換え
＆ テストが必要

個別のチューニングと、ユーザーコード
を切り離せないか？



線形ソルバーに対する
個別のチューニング

GPUコードの現状： Bridge++

Bridge++ の GPU 版（OpenACC ）: 近日公開予定

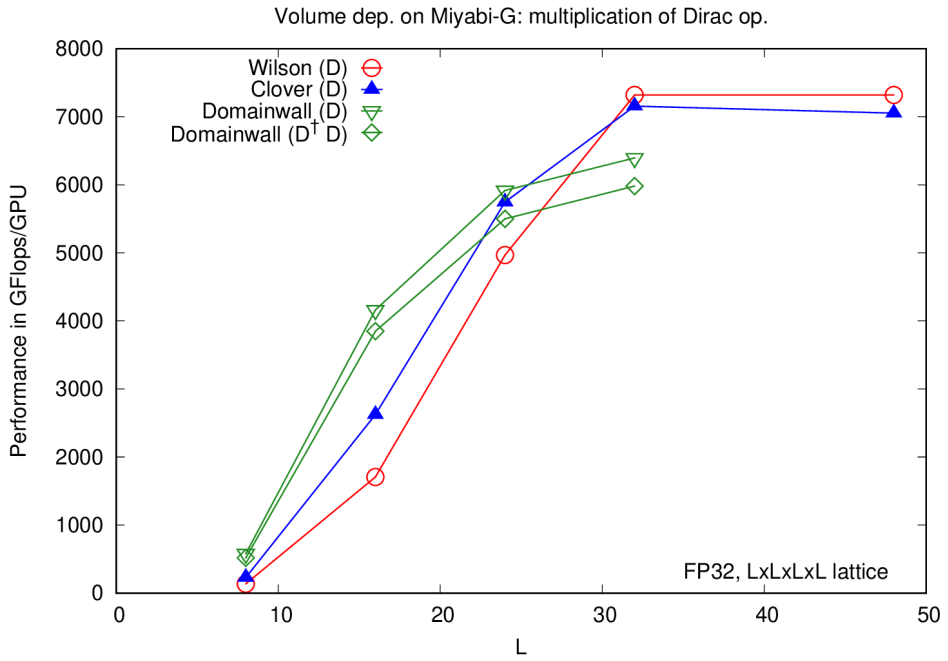
- 主なチューニング
  - データの並び替え: coalesced access
    v[site][in] → vv[s1][in][s0], site = s0 + 32*s1
  - マルチスレッド
    - thread0: MPI通信担当
    - thread1: GPU カーネル呼び出し担当
  - 小さなカーネルの融合（domanwall）: $D_{ee}^{-1} = U^{-1} L^{-1}$
    L^{-1}, U^{-1} の呼び出しから、統合した D_{ee}^{-1} の呼び出しへ
- CUDA版も開発中 W-L.Chen, I.K., H.Matsufuru HPC-Asia2025

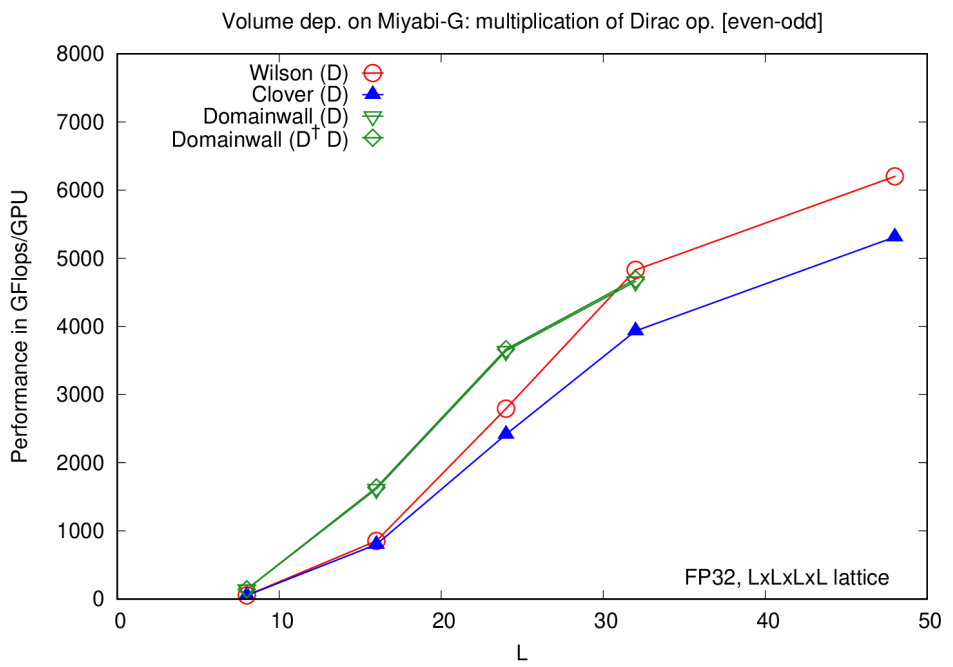
パフォーマンス: 行列ベクトル積、1GPU (Miyabi-G)

筑波大学際共同利用

- NVIDIA GH200（CPU: Grace+ GPU: Hopper）
  加速部の理論ピーク:37/64TFlops (倍精度/単精度)、メモリバンド幅: 4TB/s
- ルーフライン性能（理論値、GFlops/GPU, 単精度）
  3570 (Wilson), 4350 (Clover), 5190 (Domainwall eo)

Better

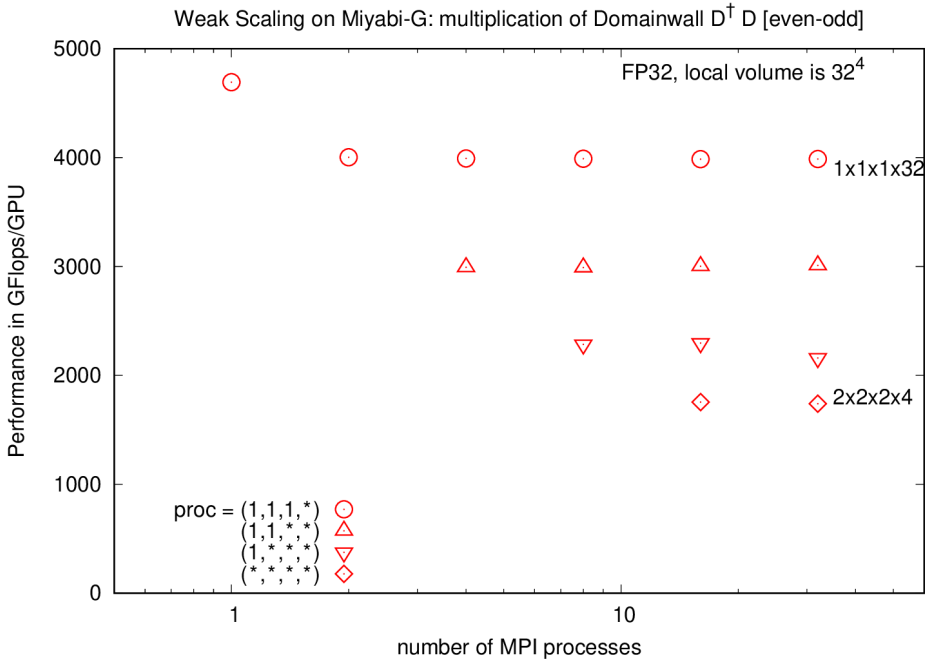




十分に体積が大きければ、ルーフライン性能（メモリバンド幅由来の性能見積もり）を超えるケースもある

パフォーマンス: 行列ベクトル積、multi-GPU (Miyabi-G)

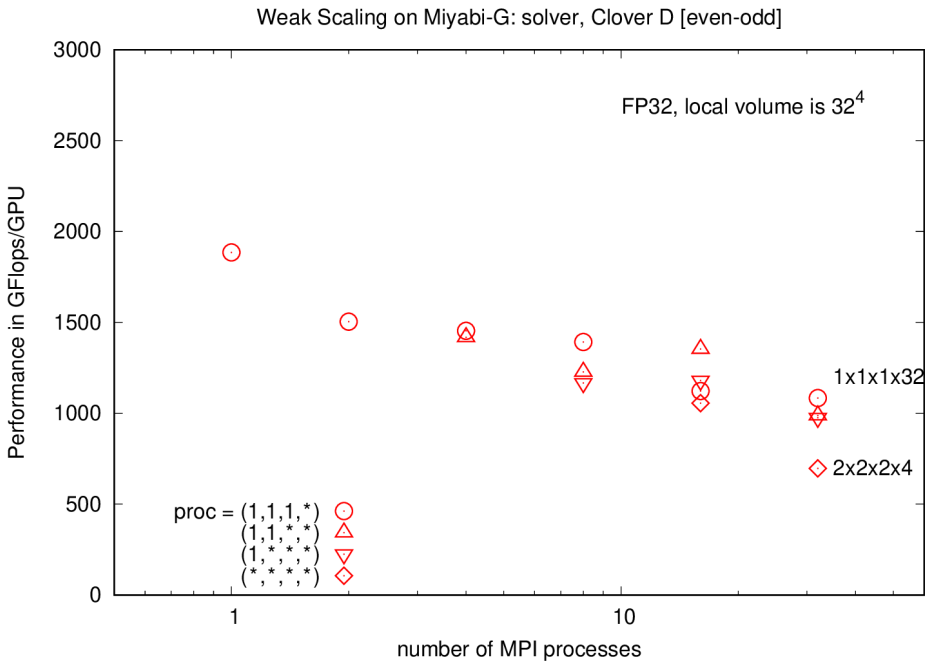
Clover (Improved Wilson)



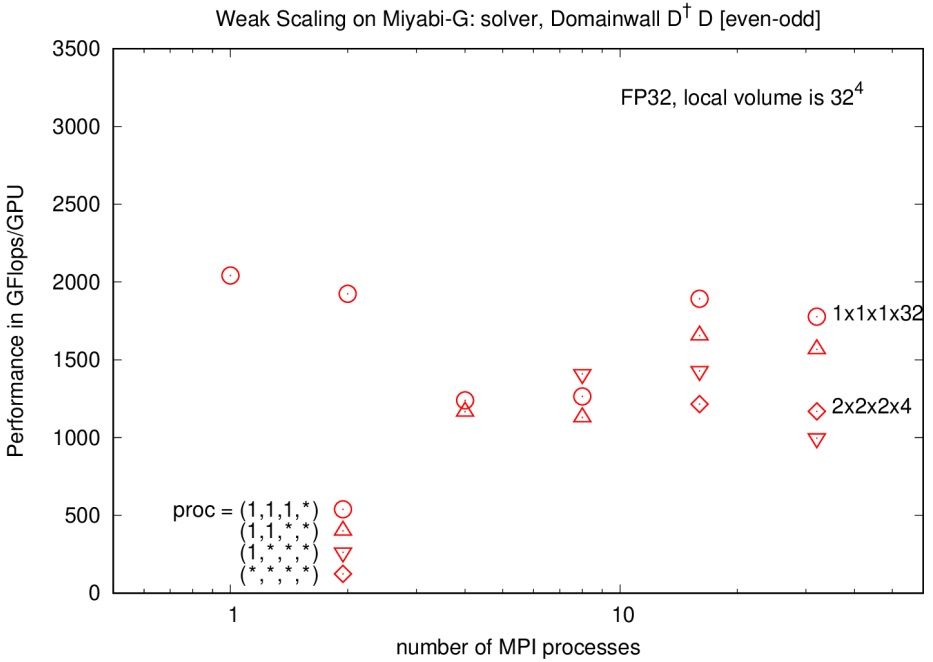
Domainwall

- 通信のオーバーヘッドは大きい

パフォーマンス: ソルバー、multi-GPU (Miyabi-G)



Clover (Improved Wilson)



Domainwall

性能ゆらぎ？
要調査

- 最大で「富岳」5-6ノード相当
  clover eo: 220GFlops/node, Domainwall eo: 360 GFlops/node

まとめ

まとめ

- 演算加速器（GPU）向けの格子QCDコード: 開発は進んでいます
- 低精度演算を使いこなす必要あり
  単精度はOK、半精度や行列エンジンはどこまで使えるか？
- アルゴリズムの工夫
  - 通信を避けるもの： SAP
  - （マルチグリッド法： domainwall 向けは発展途上）
- ユーザーコードとの連携は重要（富岳のコデザインの反省）
- 富岳よりもルーフライン性能は出しやすそう
- 触れなかったこと: AIを用いた加速の提案もあるが、大規模なシミュレーションで使えるようになるかは不明