格子 *QCD* における混合精度ソルバー

金森 逸作 (理研 R-CCS)

2023 年 12 月 8 日 (online)
第 20 回 HPC-Phys 勉強会

Outline

- 混合精度ソルバーのアルゴリズム
- ドメインウォール型格子フェルミオンでの比較

  benchmark には Bridge++ に手を加えたものを利用 

  開発リソース：「富岳」ra000001, Wisteria Odyssey (筑波大学際共同利用)
- (近似式の検証：精度が足りずに（？）最近ハマってしまったこと)

格子 QCD での線形ソルバー

格子 QCD （格子上の場の理論）

- 時空を 4 次元の格子に切って、場の量子論記述
  無限自由度の系の正則化とか、対称性とか。単に計算機に載せるための近似ではない
- 計算機上で、経路積分を実行（モンテカルロ積分）
  $\int dU d\psi \exp(-S_{\text{gauge}}[U] - \bar{\psi} D[U] \psi) = \int dU \det D[U] \exp(-S_{\text{gauge}}[U]$
  - U: ゲージ場 (グルーオン)
  - ψ: フェルミオン（クォーク）。フェルミオン 2 つ (u,d quark) の寄与なら $\det D^\dagger D$
  - D: Dirac 演算子。離散化の仕方はいろいろある。　Wilson, Domainwall, staggered etc.
    格子上では巨大な疎行列 (rank: $O(10^8)$–$O(10^{10})$)

ソルバー？ $Ax = b \Rightarrow x = A^{-1}b$

- 分布の作成（配位生成）：$\det D^\dagger D = \int d\bar{\phi} d\phi \exp(-\bar{\phi}(D^\dagger D)^{-1}\phi)$
- フェルミオンの相関関数：$\langle \psi \bar{\psi} \rangle = D^{-1}$
  $D(x,y)\eta(y) = \delta(x, x_0)$ を解くと $\eta(y) = D^{-1}(y, x_0) = \langle \psi(y)\bar{\psi}(x_0) \rangle$ が求まる
- 計算時間の 8 割程度はソルバー　パラメータによる

なぜ混合精度か

- 単精度演算のほうが倍精度演算より速い: SIMD, GPU
- AI の影響で、単精度や半精度の演算性能が大きく向上
- メモリアクセスでも単精度は倍精度の半分　　QCD の計算はメモリバンド幅・通信律速
  「京」は倍精度演算しかなかったが、データを単精度で保持することで B/F 比を稼いだ (LDDHMC)
- 隣接通信の通信量も、単精度は倍精度の半分
- ソルバーの反復数（行列・ベクトル積の回数）は増える

残差反復 (preconditioned Richardson iterations)

欲しいもの: x s.t. $Dx = b$
近似解は手元にある: x_n (approx. x)
$r = b - Dx_n$ として、$x_n + D^{-1}r$ は解になっている:

$$D(x_n + D^{-1}r) = Dx_n + r = b$$

- $D^{-1}r$ は無理でも、近似的な $D^{-1}_{\text{app.}}r$ なら計算できるものとする
- input: x_0

---

1 $r_0 := b - Dx_0$
2 $i = 0$
3 **while** $|r_i|$ *is not small enough* **do**
4 $\quad x_{i+1} := x_i + D^{-1}_{\text{app}} r_i$
5 $\quad r_{i+1} := b - Dx_{i+1}$
6 $\quad i := i + 1$

この D^{-1}_{app} に単精度（低精度）のソルバーを用いる

混合精度ソルバー

黒：倍精度　　赤： 単精度

1 $r_0 := b - Dx_0$
2 $i = 0$
3 **while** $|r_i|$ *is not small enough* **do**
4 　$r_i^{(\text{single})} :=$ to_single$(r_i/|r_i|)$
5 　$\delta x^{(\text{single})} := D_{\text{app}}^{-1} r_i^{(\text{single})}$　　Krylov 部分空間を使った反復法で解く（単精度ソルバー）
6 　$x_{i+1} := x_i +$ to_double$(\delta x^{(\text{single})}) \times |r_i|$
7 　$r_{i+1} := b - Dx_{i+1}$
8 　$i := i + 1$

- 演算のほとんどが、単精度ソルバー
- 残差を倍精度で計算することが重要

単精度（低精度）ソルバーの問題点

反復法: （例）CG 法で $Ax = b$ を解く

1 $r_0 = b - Ax_0$
2 $p_0 = r_0$
3 $i = 0$
4 **while** $|r_i|$ *is not small enough* **do**
5 $\quad \alpha_i = \frac{(r_i, p_i)}{(p_i, Ap_i)}$ 内積 (a, b) の計算
6 $\quad x_{i+1} = x_i + \alpha_i p_i$
7 $\quad r_{i+1} = r_i - \alpha_i A p_i$
8 $\quad \beta_i = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)}$ ノルムの計算
9 $\quad p_{i+1} = r_i + \beta_i p_i$
10 $\quad i := i + 1$

- 内積、ノルムの計算 : 要素数は $O(10^8)$–$O(10^{10})$
- 単精度の有効桁 : 7 桁程度 （半精度 : 3 桁程度）
- 単精度だと、$10^8 + \underbrace{1}_{\text{小さ過ぎる}} = 10^8$

  ⇒内積やノルムが正しく求まらない
- 部分和を足し上げるなどの工夫が必要
- | 内積 | ≪ | ノルム | のときは桁落ちの恐れ

  とくに BiCGStab 法
- 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:

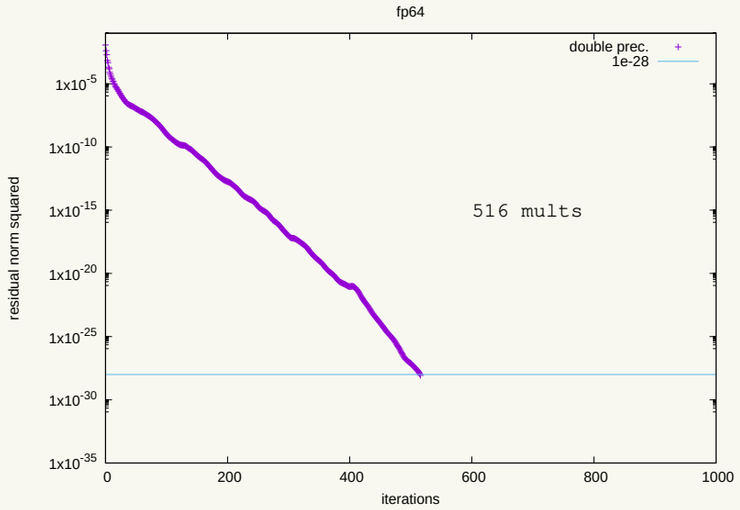
  Lattice 2023, Kate Clark

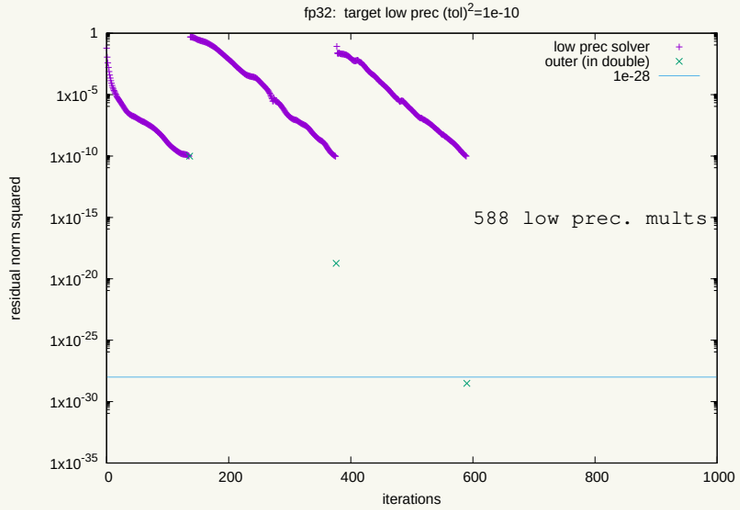
  https://indico.fnal.gov/event/57249/contributions/270632/

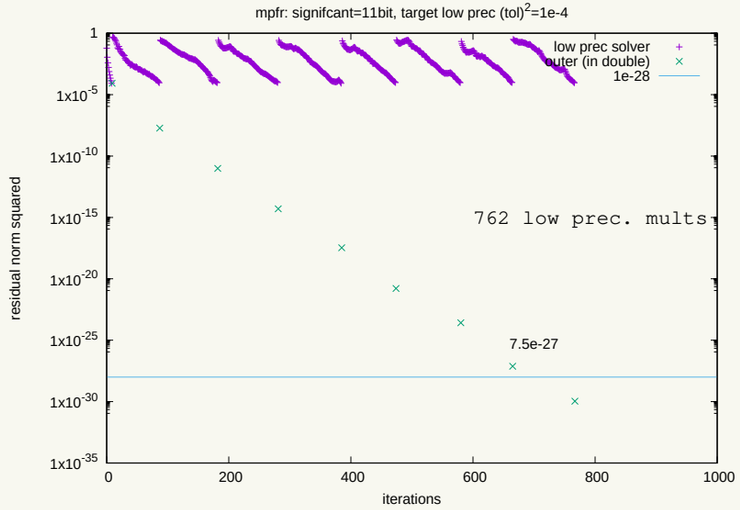
倍精度、単精度 + 倍精度、 半精度 + 倍精度

フェルミオンは Möbius Domainwall fermion: $32 \times 8 \times 8 \times 12$ (行列のランクは 3.5×10^6)
半精度は任意精度ライブラリ mpfr で代用 (*) (mpreal `https://github.com/advanpix/mpreal` 経由)
幸谷さんの「多倍長精度数値計算」（森北出版）が参考になりました

(*) 仮数部分は 11bit, 指数部分は半精度より大きい



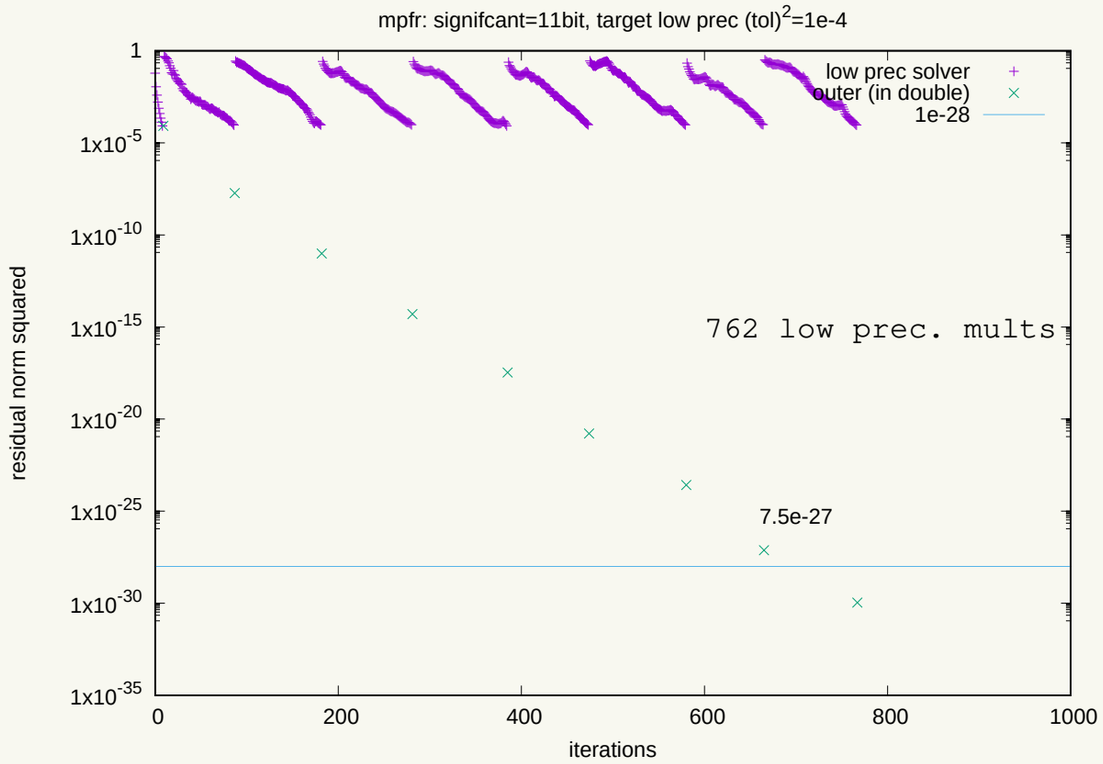


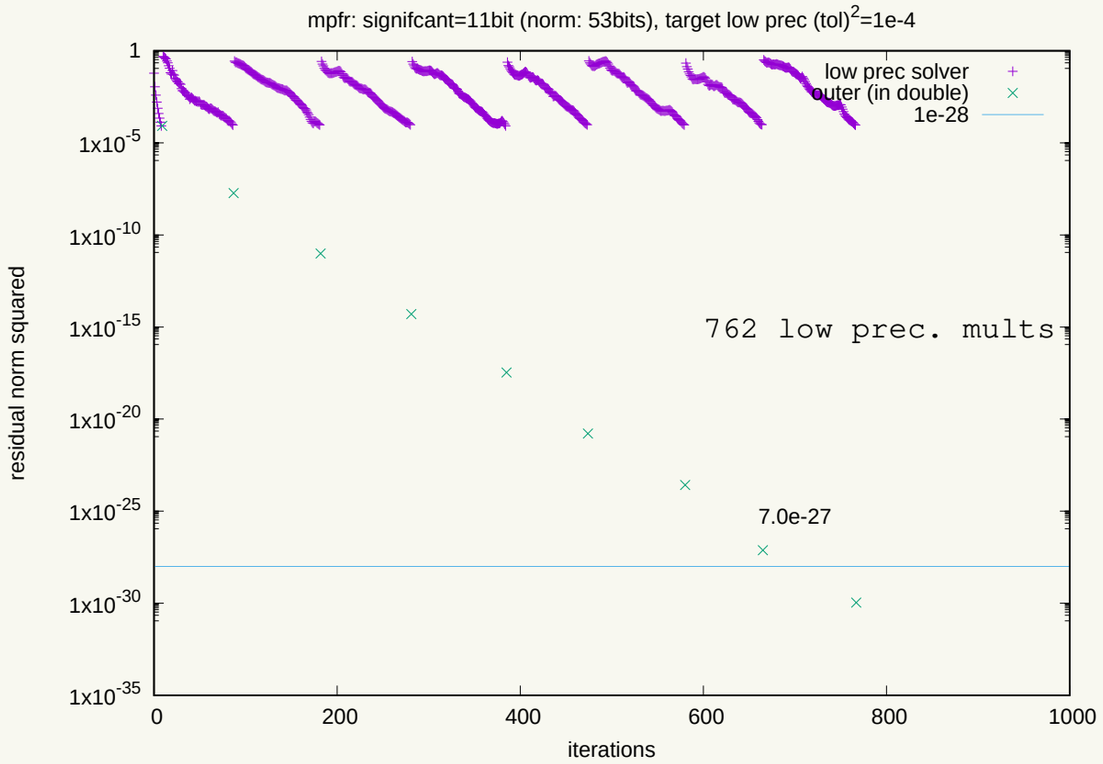


- 収束条件：$|r_i|^2 < 10^{-28}$
- 単精度：$|r_i|^2 < 10^{-10}$
- 半精度：$|r_i|^2 < 10^{-4}$

内積とノルムの精度を上げる

- （左）low prec: 半精度相当 (11 bits) ＋内積とノルムは倍精度相当 (53 bits)
- （左）low prec: すべて半精度相当 (11 bits)





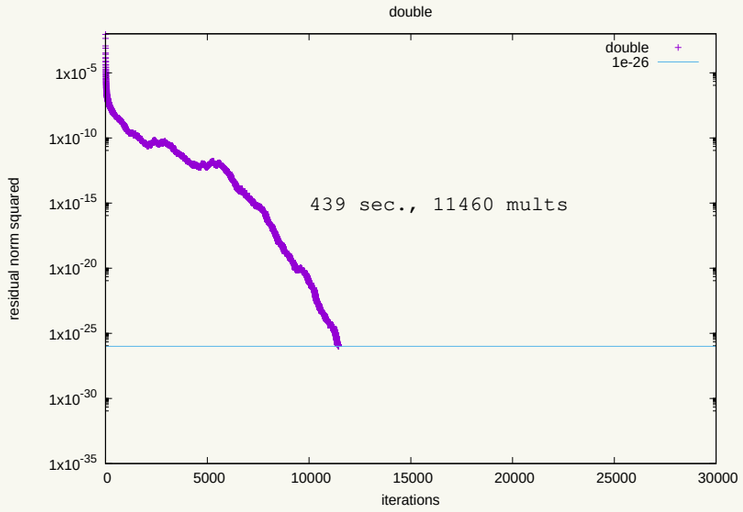
差はわずか：収束直前で、7% (7.5×10^{-27} vs. 7.0×10^{-27})
系が大きくなると差も広がると予想

より現実的な場合

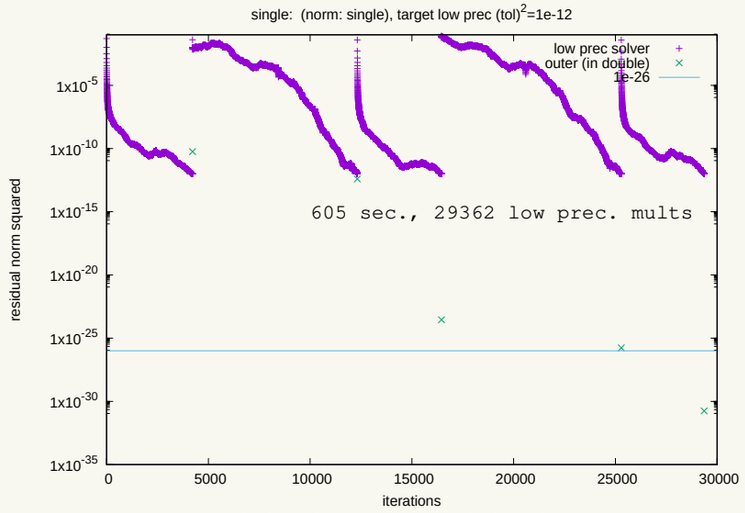
Wisteria Odssey 12 nodes で実行 (筑波大学際共同利用)

格子サイズ：$48^3 \times 12$ （行列ランク 1.9×10^8）

- 収束条件：$|r_i|^2 < 10^{-26}$
- 単精度：$|r_i|^2 < 10^{-12}$
- ノルム・内積が倍精度の方が多少速い
- 単精度はもっと早く打ち切った方がよさそう

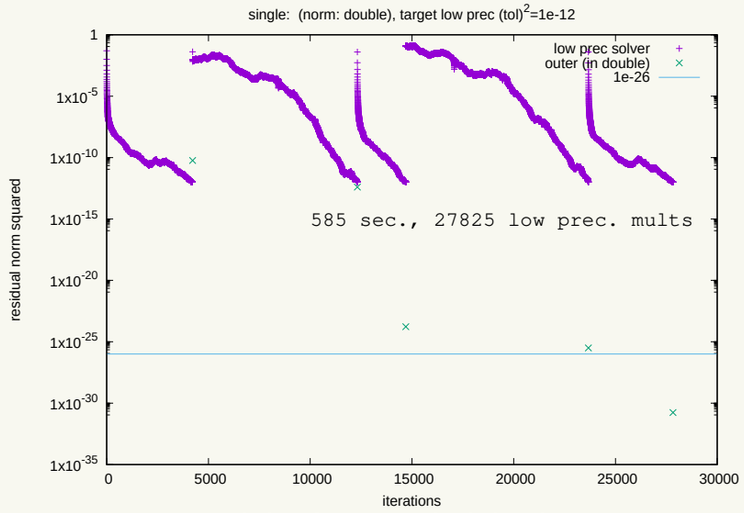


倍精度



単精度＋倍精度

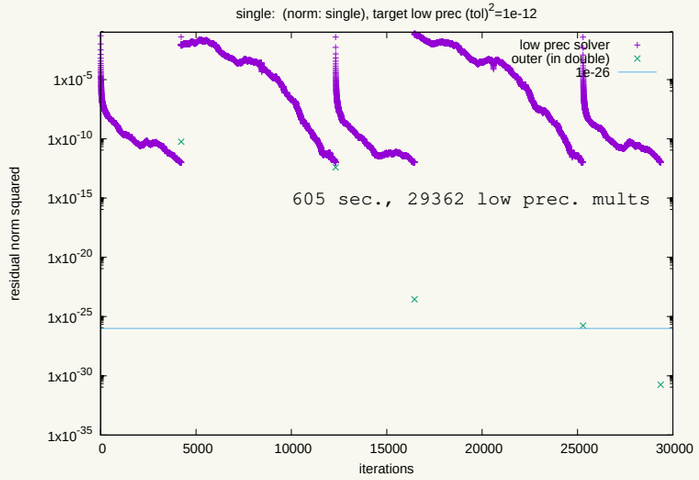
ノルム・内積は単精度

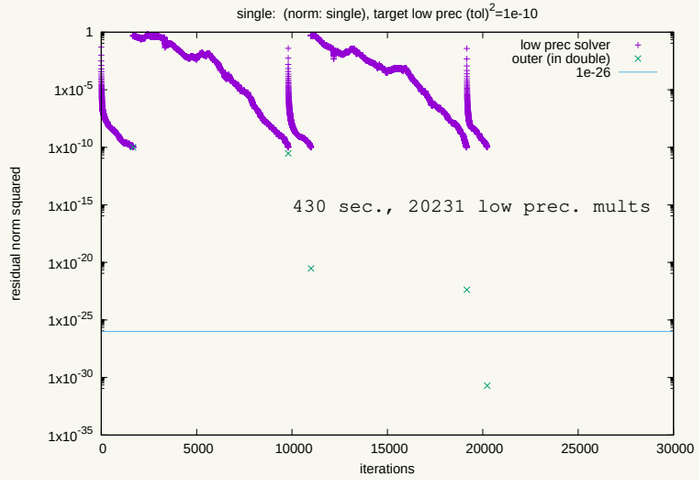


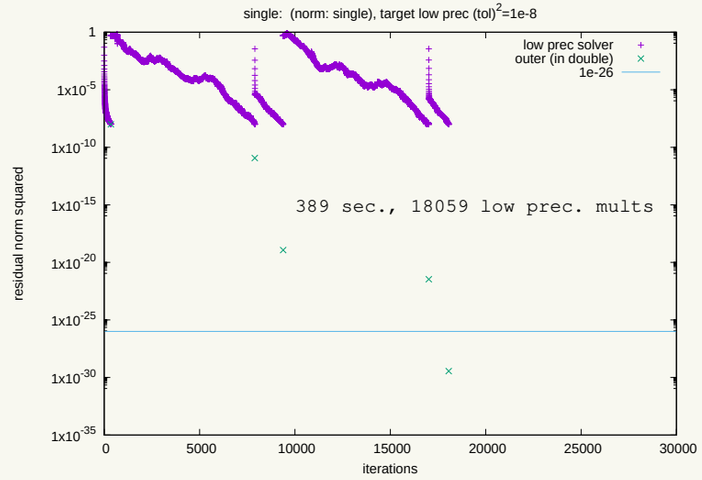
単精度＋倍精度

ノルム・内積は倍精度

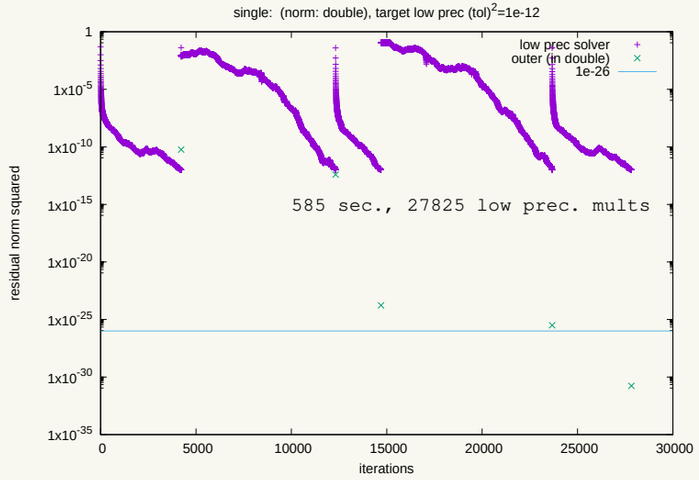
単精度のソルバーの収束精度依存性

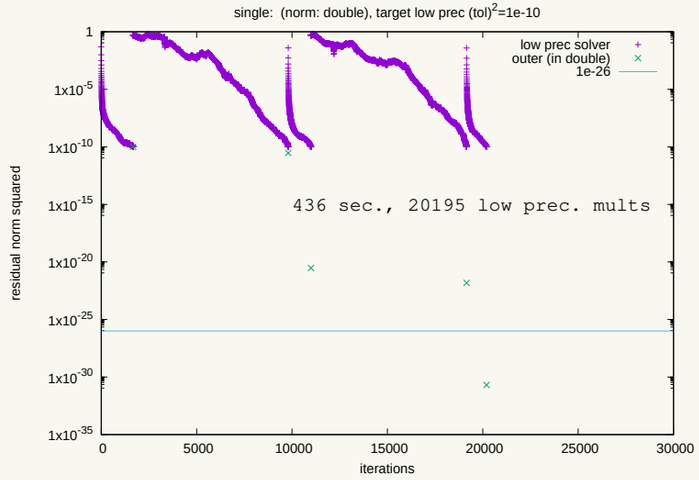


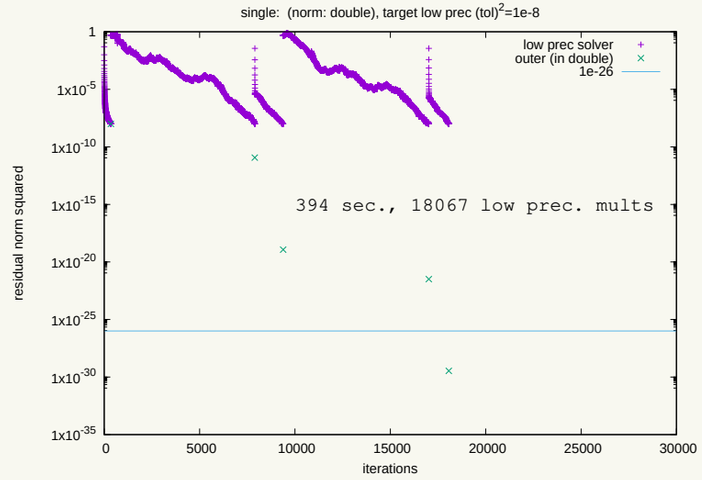




ノルム・内積は単精度







ノルム・内積は倍精度

単精度：$|r_i|^2 < 10^{-12}$

単精度：$|r_i|^2 < 10^{-10}$

単精度：$|r_i|^2 < 10^{-8}$

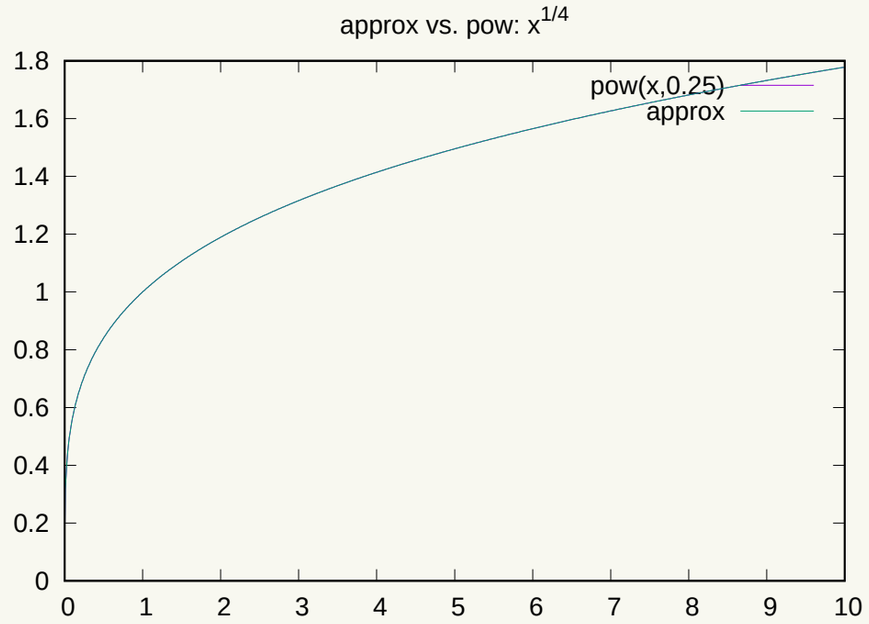
※ 上記より更に精度を落としたら実行時間が増加した

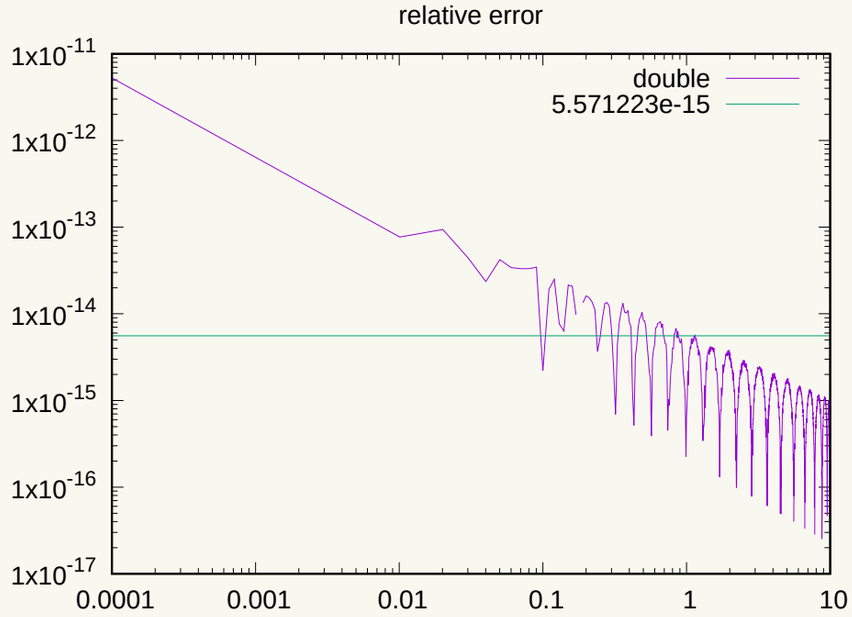
精度に関わる小ネタ

分数べき乗の近似

分数べき乗の近似：$x^{1/4} = a_0 + \sum_{i=1}^{n} \frac{a_i}{x+b_i}$
(QCD の Rational HMC で 行列 $D^\dagger D$ の分数べき乗が必要になる)

- 係数を求めるプログラム AlgRemez `https://github.com/maddyscientist/AlgRemez`
- 近似式の精度を確認したい （正しい係数を使っていることを確認したい）



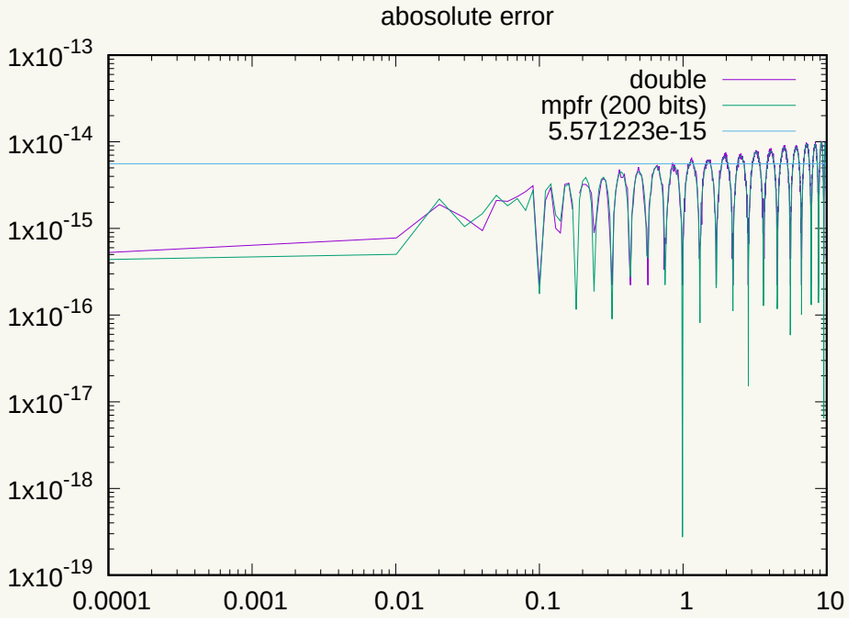


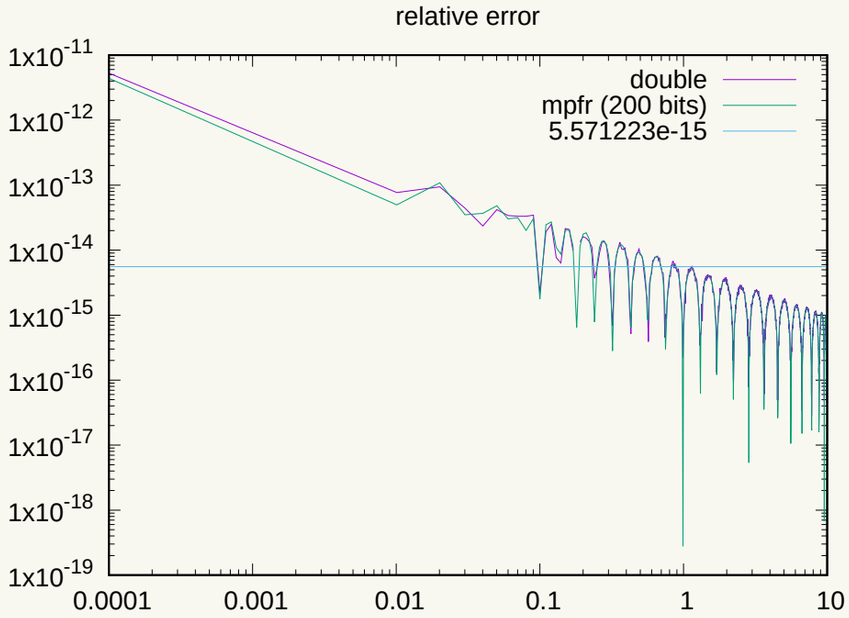
相対誤差が AlgRemez が出した 5.571223e-15 より大きい⇒桁落ち？

$$a_0 = 5.5803685240057019$$

おそらく、、、

mpfr を使った検証





おそらく $x < 1$ は相対誤差（左）、$1 < x$ は絶対誤差（右）になっている。ドキュメントで仕様を確認する必要あり。

まとめと議論

- 格子 QCD で用いている混合精度ソルバー紹介
- 単精度＋倍精度：単精度ソルバーの収束条件に注意
- 内積、ノルムの精度も大事：精度が落ちる前に打ち切るのが良さそう

  今回は調べていないが、多倍長も有効かも
- 半精度もたぶん行ける？ To do: A64FX の半精度演算
- 体積が大きくなるともっと大変になる

  cf. メトロポリステストで $\Delta H = H - H_0 = O(10^{10}) - O(10^{10}) \sim 0.1$　　$H \propto$ volume

Backup Slides

Bridge++ での例: ノルムの計算 (「富岳」向けのコードから)

```
1   set_threadtask(ith, nth, is, ns, Nouter);          // range for each thread
2   svreal_t yt;   set_vec(pg, yt, 0.0);               // yt=0 (SIMD variable)
3   for (int ex = 0; ex < Nex; ++ex) {
4     svreal_t tmp_yt;
5     set_vec(pg, tmp_yt, 0.0);                        // tmp_yt=0
6     for (int i = is; i < ns; ++i) {
7       real_t   *v = &vp[VLEN * Ninner * (i + Nouter * ex)];   // real_t: float or double
8       svreal_t xt, vt;
9       set_vec(pg, xt, 0.0);                          // xt=0
10      for (int in = 0; in < Ninner; ++in) {
11        load_vec(pg, vt, &v[VLEN * in]);
12        add_norm2_vec(pg, xt, vt);                   //  xt += vt*vt   (1)
13      }
14      add_vec(pg, tmp_yt, xt);                       //  tmp_yt += xt  (2)
15    }
16    add_vec(pg, yt, tmp_yt);                         //  yt += tmp_yt  (3)
17  }
18  real_t a;
19  reduce_vec(pg, a, yt);                             // summation in SIMD  (4)
20
21  #pragma omp barrier
22   // summation over threads then over MPI processes          (5) and (6)
23    ThreadManager::reduce_sum_global(a, ith, nth);
24    return a;
```

場は 3 次元配列のような構造を持つ
（1 次元配列 + index で疑似 3 次元配列）
そのため自然に 3 重ループになる