# 富岳上での FDPS の性能デバッグ

牧野淳一郎
神戸大学

**2021/10/26** 富岳計算宇宙惑星内部ワークショップ

# 話の構成

1. 問題はなにか:**FDPS** を使ったプログラムの富岳上での性能と安定性
2. 測定結果: 何が遅いか?
3. 問題と対応
4. **GPLUM** における問題と対応
5. まとめ

# 問題はなにか:FDPS を使ったプログラムの富岳上での性能と安定性

**FDPS** を使って富岳で動いている/動いて欲しいプログラムは沢山ある。

**ASURA/FDPS**、**GPLUM**、**FDPS-SPH**、**PETAR**、あと他にも。あと神戸大学/理研の大石先生のとこで防災シミュレーションで **SPH**。多分 **DEM** とかも。

どうも色々問題があってあんまり使えてない。

- 遅い。
- しばらく回すと落ちる。

というわけで、「ある程度直した」という話を今日は。

# 測定

- 測定に使ったコード**: nbody-with-center (https://github.com/jmakino/nbody-with-center)** 牧野が書いた、惑星リング用のコード。書いたというか **N** 体サンプルプログラムをちょっといじって中心天体重力と粒子同士の非弾性衝突をいれただけ。
- **FDPS** の領域分割とツリーは円筒座標で作るモードで動作。
- とりあえず簡単なコードで様子を見るため。
- ノード数**: 1k-4k**
- 粒子数**: 2-200M**
- 見込み角 $\theta = 0.5$、リング半径 **1**、幅**=0.1**、**4** ステップに **1** 回領域分割 **(**いらないけど時間測定のため**)**

# 最初の測定結果

**1** ステップの計算時間

| 項目 | 時間 (ミリ秒) |
|---|---|
| 全体 | **105** |
| 領域分割 | **1.5** |
| 粒子交換 | **10.5** |
| 相互作用計算 | **54** |
| 「その他」 | **38** |

- とにかくものすごく遅い。ハードウェアの性能がちゃんとでたら **1** ミリ秒のオーダー、まあ通信が××だとしても　**10** ミリ秒くらいになってて欲しい。
- この測定はアプリケーションプログラム側で、あんまりバリアとかちゃんといれてないので例えば「その他」はなにかとか怪しい。

# とりあえずいれた修正

- **LET(**相互作用計算に必要な他のノードの情報。**Local Essential Tree)** の交換に **MPI_ALLTOALLV** を使う標準モードの代わりに **P2P_EXACT** に切り替え。これは、領域のトップレベルの情報は **ALLGATHER** で共有して、それより細かい情報が必要なところだけ **P2P** で交換する
- 相互作用計算を **SIMD** 化。あんまり気合いはいってなくて最内側ループが連続アクセスでベクトル化されるように成分毎の配列にコピーしてから計算するくらい。

# 書換え後の性能

**External times**

| 項目 | 時間 (ミリ秒) |
|---|---|
| **Total** | **59** |
| 領域分割 | **1.7** |
| 粒子交換 | **8.4** |
| 相互作用計算 | **44** |
| その他 | **4.6** |

**Internal times**

| 項目 | 時間 (ミリ秒) |
|---|---|
| 領域分割 | **3.6** |
| 粒子交換 | **22** |
| プロセス内/全体ツリー構成 | **9.4/9.5** |
| 相互作用計算 | **3.2** |
| **LET** 構築/通信 | **6.5/15** |

**SIMD** 化で相互作用計算カーネル「だけは」すごく速くなった。

# 問題と対応

- 粒子交換
- **LET** 構築/通信
- ツリー構築
- 領域分割

# 粒子交換

これは、粒子が動いた/領域分割が変わったことで、属するべきところに送るところ。

- リング用円筒座標コードで、粒子はそんなに遠くまでいかないはず。
- 実際に通信している相手をみたら何故かほぼ全部になっていた。角度方向で両隣以外は、実際にはそこまで動く粒子がないがチェックしていた。
- これは **FDPS** の性能バグで、**(**この円筒座標用の**)** 周期境界での粒子交換アルゴリズムが間違っていた。修正した。

修正後の性能**:**

$$\mathbf{22ms \rightarrow 2.2ms}$$

# LET 構築/通信

- P2P_EXACT はリング向けではあんまり、、、
- P2P_FAST に変更。

**LET** 構築時間: $\mathbf{6.5ms \rightarrow 3.5ms}$
**LET** 通信時間: $\mathbf{15ms \rightarrow 0.3ms}$

# ツリー構築

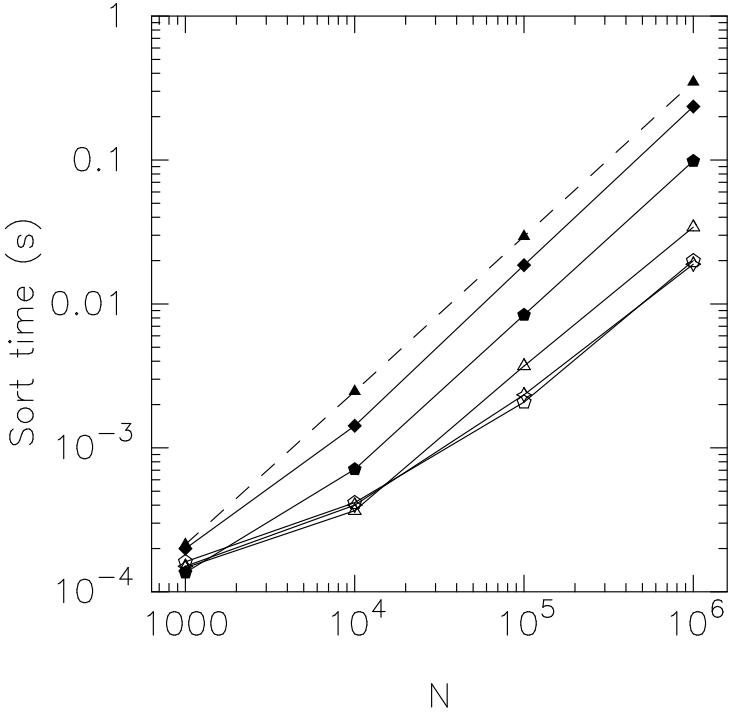
- **FDPS** では、ツリーはモートンキー作ってからソートする。**OpenMP** で性能だすために並列マージソートを使っていたが、これが富岳では非常に遅くなっていた。
- とりあえず **1** コアで **std::sort** にしてみた

  プロセス内ツリー構築時間: $9.4\mathrm{ms} \rightarrow 0.56\mathrm{ms}$
- あと、並列サンプルソートも実装した。

  プロセス内ツリー構築時間: $0.56\mathrm{ms} \rightarrow 0.44\mathrm{ms}$

  **(1** ノードあたりの粒子数が多いとちゃんと **20** 倍くらい速くなる**)**

# 並列サンプルソートの性能




**160** バイトのデータを **64** ビットキーでソートする時間 **(**経過時間**)**

破線**+**黒三角は **1** コア **std::sort**。黒四角、五角、中抜き三角、四角、五角は **2**、 **4**、**12**、 **24**、**48** スレッドでの並列サンプルソート。

**24** スレッドまではいいが **48** スレッドは、、、どこが遅いかとかもうちょっと調べて性能あげられるかもしれない。

サンプルソートライブラリとして **https://github.com/jmakino/sortlib** で公開

# 領域分割

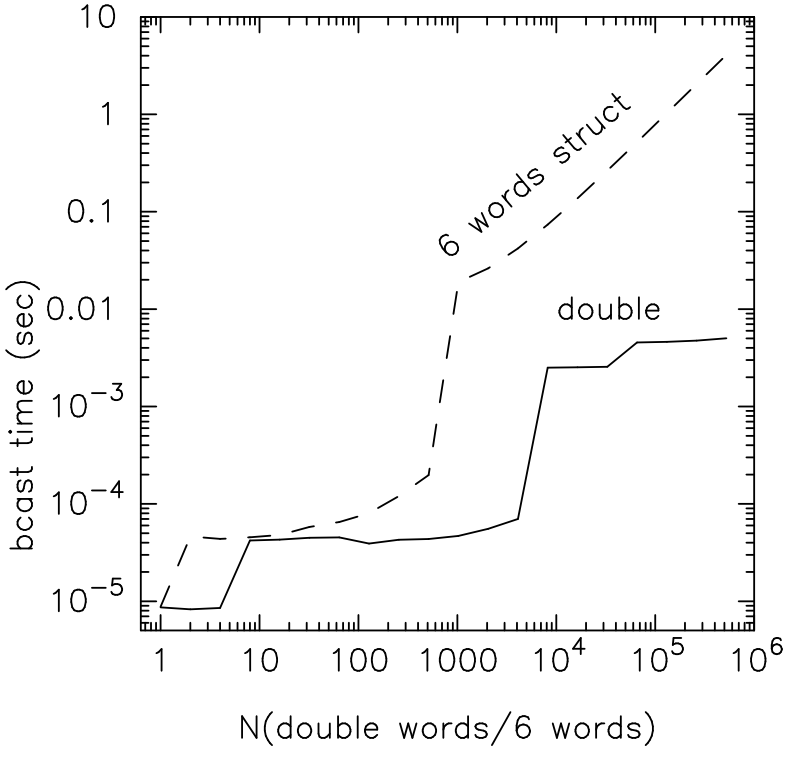
領域分割の操作毎にバリアいれて時間測った結果わかったこと:

- サンプル粒子あつめるところ (何故か MPI_AllgatherV) は十分速い。
- ランク 0 での計算はそこそこ。
- 作った領域分割の放送が滅茶苦茶に遅い。単に MPI_Bcast 1 度呼ぶだけでデータ量も上のサンプル粒子もらうところより桁で少ないが時間は桁で遅い。MPI のバグとしか思えない (詳しくは細野君の発表で)

とりあえずの対応: MPI_Bcast やめて全プロセスで同じ計算するようにした。あと、計算ではソートは並列サンプルソートに変えて、他にも OpenMP 化さぼってたところをちゃんといれた。

領域分割時間: $8\mathrm{ms} \rightarrow 0.4\mathrm{ms}$

# 富岳 MPI_Bcast の性能(詳しくは細野君発表で)




**1024** ノード、 **48** スレッド/プロセス

**6 words=** 倍精度 **6** 語の構造体を送る

まともな **MPI** 実装なら、構造体 $N$ 個送るのは $6N$ 語送るのと同じ時間ですむはず。あと、必要時間がジャンプしないように適切なアルゴリズムが選ばれるはず。

はずなんだけどねえ、、、

# 現状

| 項目 | 時間 (ミリ秒) |
|---|---|
| 全体 | **7.2** |
| 領域分割 | **0.41** |
| 粒子交換 | **0.78** |
| 相互作用計算 | **5.8** |
| ツリー構築 | **1** |
| **LET** 構築 | **1.6** |
| **LET** 交換 | **0.4** |
| ツリー辿る | **0.44** |
| 相互作用カーネル | **1.57** |
| その他 | **0.1** |

大分速くなった。
**200** 万粒子に **1024** ノードは実際上は無理で、**72** ノード **9.4ms**、**144** ノード **8.35ms** あたりがまあ使ってもいいかなくらい。
**144** ノード以下だと **1** ノード **4** プロセスのほうが速い。
**2G** 粒子 **1152** ノードで **0.85** 秒。この辺で相互作用カーネルの効率が低いのがみえてくる。

# FDPSとしての改善の余地

- 領域分割には $O(p^{1/3})$ ($p$:プロセス数) のアルゴリズムがあるのでそっちを発掘して切り替える。
- 粒子交換: 粒子を見つけるところがなんか遅いのでもうちょっとなんとかする。**slow**
- **LET** と 相互作用計算のツリー辿るところが遅いのをなんとかする。これはだいぶ書き直して、**x86** では **1.5** 倍くらい速くなったが富岳ではまだ。

# GPLUM

- **FDPS** 速くなったのでまともに速く動くようになったかというとそうでもなく、、、
- 富岳では少なくとも **12** スレッド/プロセス、大ノード数で性能と安定性を維持するには **48** スレッド/プロセスが望ましいが、その辺想定してなかった結果色々なことが。

# GPLUM—具体的な問題

- 近接粒子リストを **std::vector** を使って作っている。
- さらに、 **std::map** を使って粒子 **ID** からランクやインデックスをひけるようにしている

これらがスレッド数多いとボトルネックになる。あと細かいことが色々。

- **std::vector** は全然別の **vector** をアクセスしてても、サイズが変わると共通のヒープからメモリとってくるので、 **push_back** とかで要素追加する操作はスケールしない
- **std::map** はそもそも要素追加が並列にできない

現行の **GPLUM** では、これらを「なるべく」使わないように修正。 **vector** は残ってるかも。**100** 万粒子で **1** ステップ **10ms** くらい？

# 色々わかったこと

- 非常に注意深く使えば富岳の通信はかなり速い。
- 注意深く=測定して、理屈に合わない性能のとこはドキュメント読んで理屈にあう性能がでるものに変える。
- 特に **ALLTOALL(V)** は避けるべき。
- あと、**MPI derived data type** は富岳では使ってはいけない。
- 「これ使うと遅いんですが」「では、使わないようにして下さい」的
- 何故かこの作業のあと落ちないで何ステップでも問題なく走るようになった。**ALLTOALL(V)** が悪い気がする。

# 色々わかったこと2

- **STL** は便利だがメモリ管理周りは容易に **OpenMP** 環境で性能ボトルネックになるので、そうならないようにする必要あり。使わなければ問題は起きない。
- **OpenMP parallel for** のスレッド起動オーバーヘッドは富岳でもあんまり小さくない。サンプルソートの実行時間に影響があったので、**10** マイクロ秒くらいある？**(**要調査**)** 性能だすためにはなるべく **parallel section** の単位を大きくするべき。

# まとめ

- 富岳での **FDPS** の性能の問題点を **1024** ノード **200** 万粒子で詳細測定し、比較的容易に直せるところは直した。
- **SIMD** 化はしたが最適化はできてない相互作用カーネルの実行時間が見えるくらいにやっとなった。まだ実行効率としては低い。
- 惑星リングなら **(**まだ試してないが**)** 相互作用リスト再利用モードにすれば相互作用カーネルがメインにできるはず。
- 再利用モード使えない計算だとツリー辿るところをもうちょっとなんとか、、、
- 大雑把にいって **100** ノード超えると **1** ノード **1** プロセスのほうが通信性能的に安全。

- この辺の改良はいった **FDPS 7.1** 公開されました。