Innovative Multigrid methods II

藤井昭宏

2020 年 12 月 25 日

工学院大学

目次

Innovative Multigrid methods II

Innovative Multigrid methods II

Co-PI:

K. Nakajima[2,4], M. Bolten[8]

Members:

M.Kawai[4], A.Ida[2],
G.Wellein[9], C.Alappat[9],
M.Schreiber[11], T.Hoshino[2],
S.Ohshima[6], T.Hanawa[2],
O.Marques[10], K.Ono[5],
M.Miyashiro[13], M.Iizuka[5],
T.Iwashita[1], Y.Takahashi[7],
R.Speck[12], R.Yoda[2], Y.Chen[2]

GMG and AMG

- Smoothers with SELL-C-sigma
- Lower precision utilization
- Evaluation with large sized problems

PinST

- New approaches
- Convergence analysis of MGRIT preconditioning

1: Hokkaido U., 2: U. Tokyo, 3: Kogakuin U., 4: RIKEN R-CCS,
5: Kyushu U., 6: Nagoya U., 7: Doshisha U., 8: U. Wuppertal*, 9: FAU*,
10: LBNL, USA, 11: Technical U., of Munich*,
12: Juelich Supercomputing Centre*, 13: Taisei Co. *:Germany

Innovative Multigrid methods II

Co-PI:

K. Nakajima[2,4], M. Bolten[8]

Members:

M.Kawai[4], A.Ida[2],
G.Wellein[9], C.Alappat[9],
M.Schreiber[11], T.Hoshino[2],
S.Ohshima[6], T.Hanawa[2],
O.Marques[10], K.Ono[5],
M.Miyashiro[13], M.Iizuka[5],
T.Iwashita[1], Y.Takahashi[7],
R.Speck[12], R.Yoda[2], Y.Chen[2]

GMG and AMG

- Smoothers with SELL-C-sigma
- Lower precision utilization
- Evaluation with large sized problems

PinST

- New approaches
- Convergence analysis of MGRIT preconditioning

1: Hokkaido U., 2: U. Tokyo, 3: Kogakuin U., 4: RIKEN R-CCS,
5: Kyushu U., 6: Nagoya U., 7: Doshisha U., 8: U. Wuppertal*, 9: FAU*,
10: LBNL, USA, 11: Technical U., of Munich*,
12: Juelich Supercomputing Centre*, 13: Taisei Co. *:Germany

背景と目的

- 代数的多重格子法は問題行列から小さいサイズの問題行列（粗いレベル）を生成し，緩和法を交互に適用する反復解法
  - 計算量 $O(n)$：未知数の個数を n
  - 大規模な問題ほど優位．
- 粗いレベルでも各プロセスの未知数は一定の個数の未知数を保持できるようにプロセス数を調整する処理がよく利用される．
  - その粗格子集約手法を提案し，主にストロングスケーリングの設定で有効性を確認 [iWAPT2018]
  - 現在の実装でどこまで大規模な問題が解けるか，実装のスケーラビリティや問題点も含めて確認したい

目的

非均一な拡散係数を持つ大規模ポアソン問題を対象に粗格子集約手法 (CGA) の有効性を調べる．未知数の個数は 10^{10} 程度までの問題とする．

代数的多重格子法

代数的多重格子法 (2 レベルの場合)

マルチレベル生成部のあとで，反復解法部が続く

反復解法部：V-cycle による前処理付クリロフ部分空間法

マルチレベル生成部,
In: A, **Out:** A_c, P

1: $\tilde{P} \leftarrow aggregate(A)$;
2: $P \leftarrow smooth(\tilde{P})$;
3: $A_c \leftarrow P^T AP$;

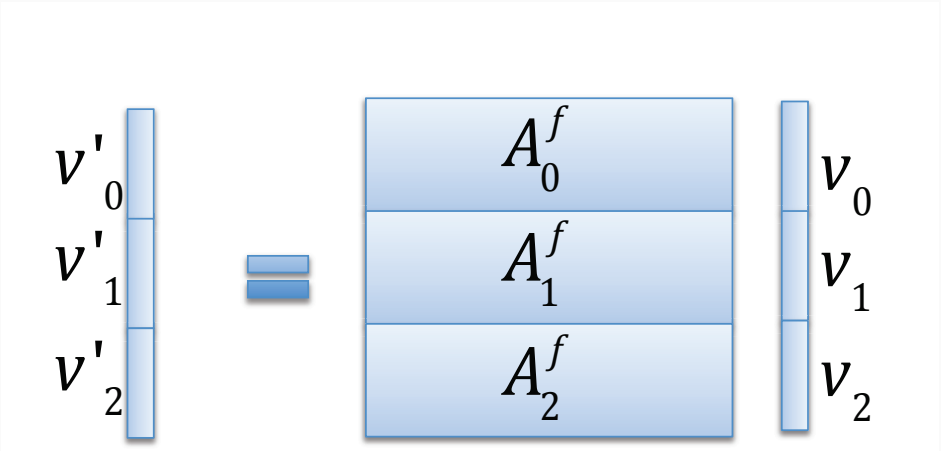
Line.3：A_c は 3 つの疎行列の積

V-cycle,
In: A, x, b, A_c, P, **Out:** x

1: $x \leftarrow smoother(A, x, b)$;
2: $b_c \leftarrow P^T(b - Ax)$;
3: $x_c \leftarrow 0$;
4: $x_c \leftarrow solve(A_c, x_c, b_c)$
5: $x \leftarrow x + Px_c$;
6: $x \leftarrow smoother(A, x, b)$;

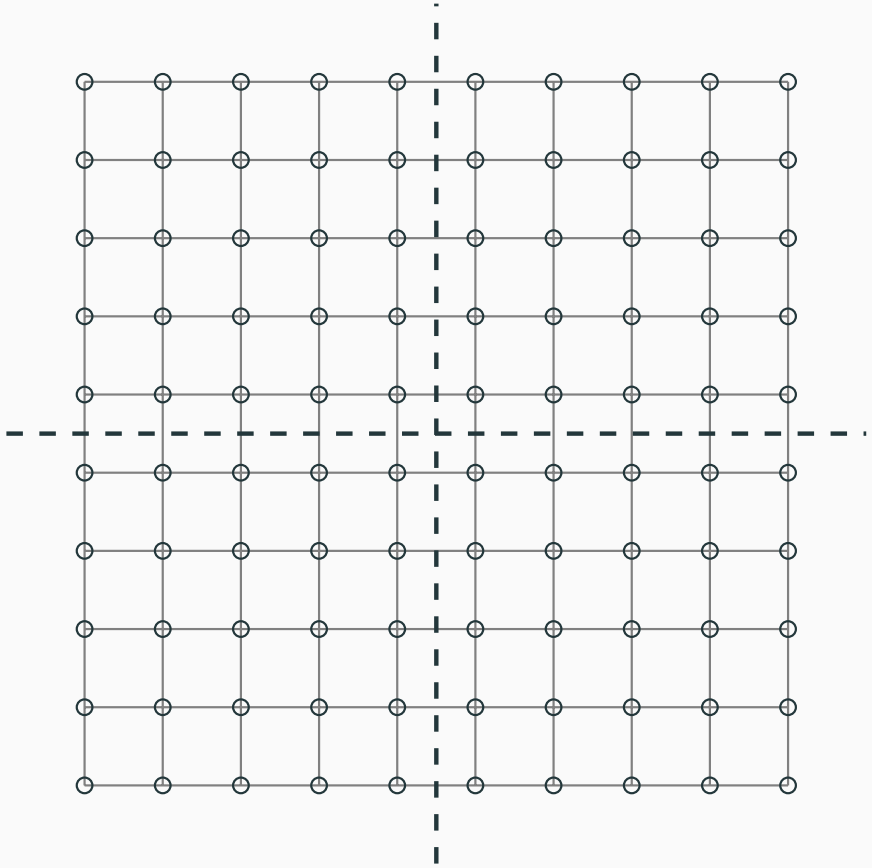
各プロセスが保持する分散疎行列データ構造

- ブロック行分散でプロセスランク番号順にブロック行を割当
  - ブロック行: CSR 形式
  - 疎行列ベクトル積: 自分の保持していない未知数の値を取得してから並列に疎行列ベクトル積
- 行番号、列番号ともに 32bit 整数型
  - 行番号, 列番号はプロセスごとに 1 から数える.
  - 行列全体をまたがる番号は保持されない
  - 1 プロセスの保持する行列の非ゼロ要素数が 32 ビットを超えなければ，全体サイズは並列度数分だけ 32 ビットより大きく表現できる.



粗格子集約手法と粗格子補正戦略

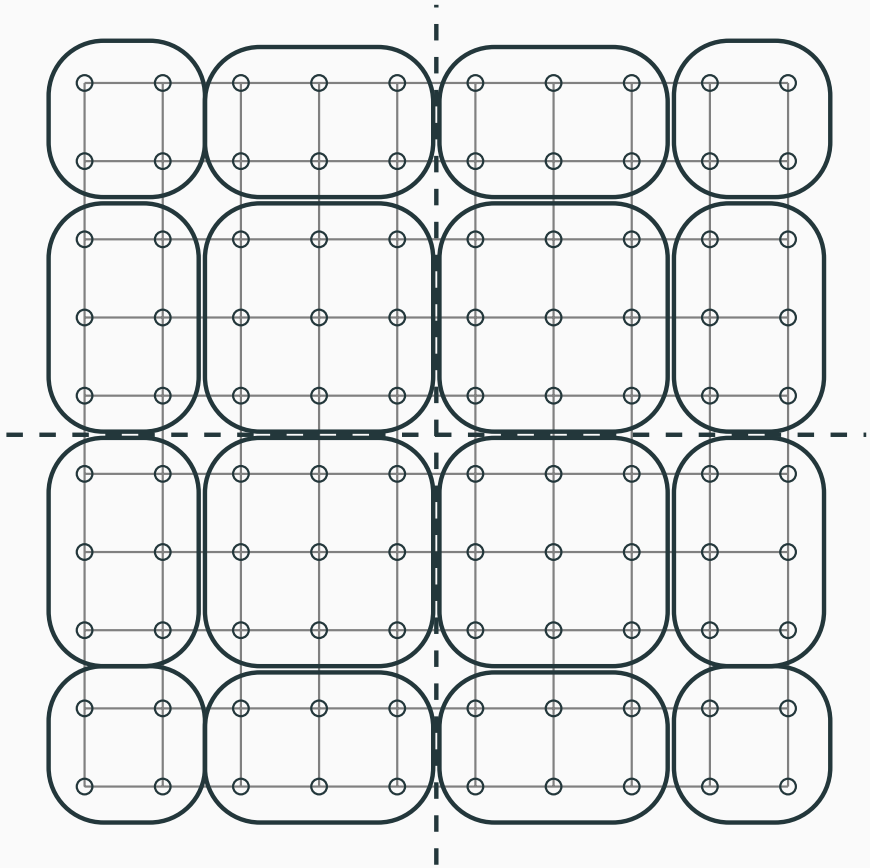
アグリゲート生成手法



2 次元 5 点差分の未知数間の関係を表すグラフ．
10×10 の 100 個の未知数を 4 プロセスに分散

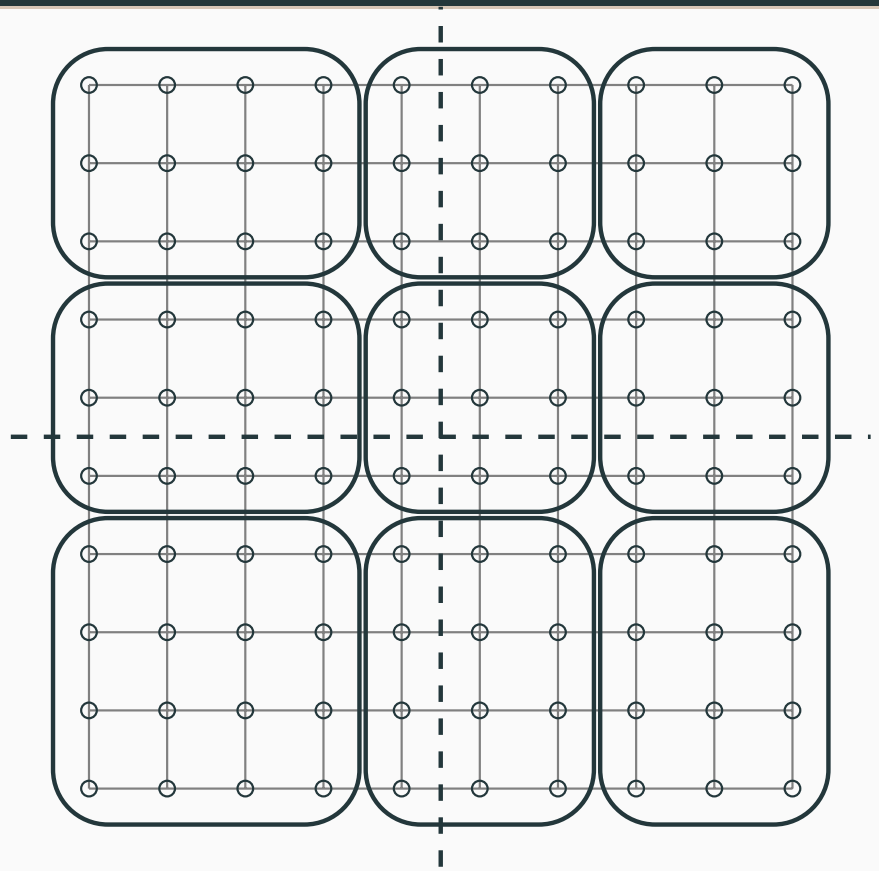
- 格子点:　未知数
  格子点間の枝:　非ゼロ要素
- 破線: プロセス担当領域境界

アグリゲート生成手法



独立アグリゲート生成

プロセス境界を越える未知数集合は生成しない．1 プロセス領域に少なくとも 1 個はアグリゲートが作られる．



共有アグリゲート生成

プロセス境界の上から未知数集合を作成し，その後、プロセス領域内部のアグリゲート生成．

粗格子集約手法

粗格子を再分散する通常の実装

1: アグリゲート生成
2: 補間行列 P 生成
3: $A_c = P^T AP$
4: 疎行列 A_c の再分散

本研究の手法

1: 独立アグリゲート生成
2: プロセス間隣接グラフの生成
3: アグリゲートの個数から粗いレベルのプロセス数を決め，ParMETIS によりプロセスのグループ分け
4: 結合するプロセス間でアグリゲート番号が連続になるようにアグリゲート番号の付け替え
5: 補間行列 P を生成
6: $A_c = P^T AP$

本研究の手法は，粗いレベルの生成と粗いレベルのプロセス数の調整を同時に行う手法．

最粗格子レベルの設定と解法の適用

- 最粗レベルとするかどうか
  - 最大レベル数に到達
  - そのレベルの未知数の個数が閾値以下となる
- 最粗レベルの解法
  - LU 分解
  - 反復解法
- よく利用される設定
  - 最粗レベルの問題サイズを十分小さくして 1 プロセスで LU 分解
  - 最粗レベルでも未知数をある程度残し，反復解法の適用

粗格子戦略一覧

	アグリゲート生成	最粗レベル未知数閾値	最粗レベル解法	領域集約
LU	独立	500 個	LU	なし
CGA_LU	独立	500 個	LU	あり
w/o_LU	独立	40000 個	SGS 法 20 回	なし
coupled_LU	共有	500 個	LU	なし

デフォルトは独立アグリゲート
LU とあれば、サイズを小さくして 1 プロセスで LU 分解
w/o_LU の場合、最粗レベルでも並列の反復解法

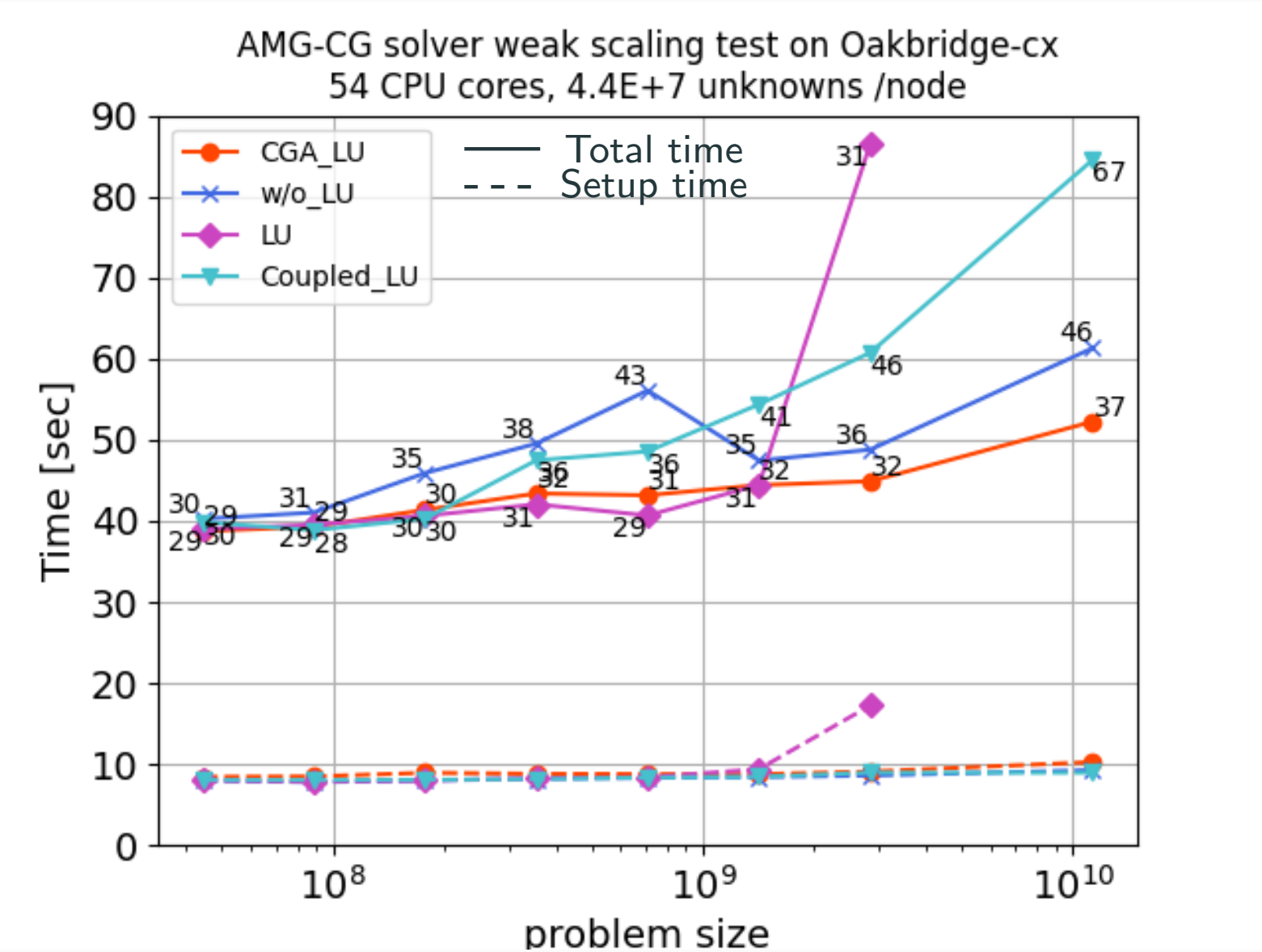
各レベルのスムーザはプロセス領域境界の依存関係は無視した (マルチカラー)SSOR を 1 回適用.

数値実験

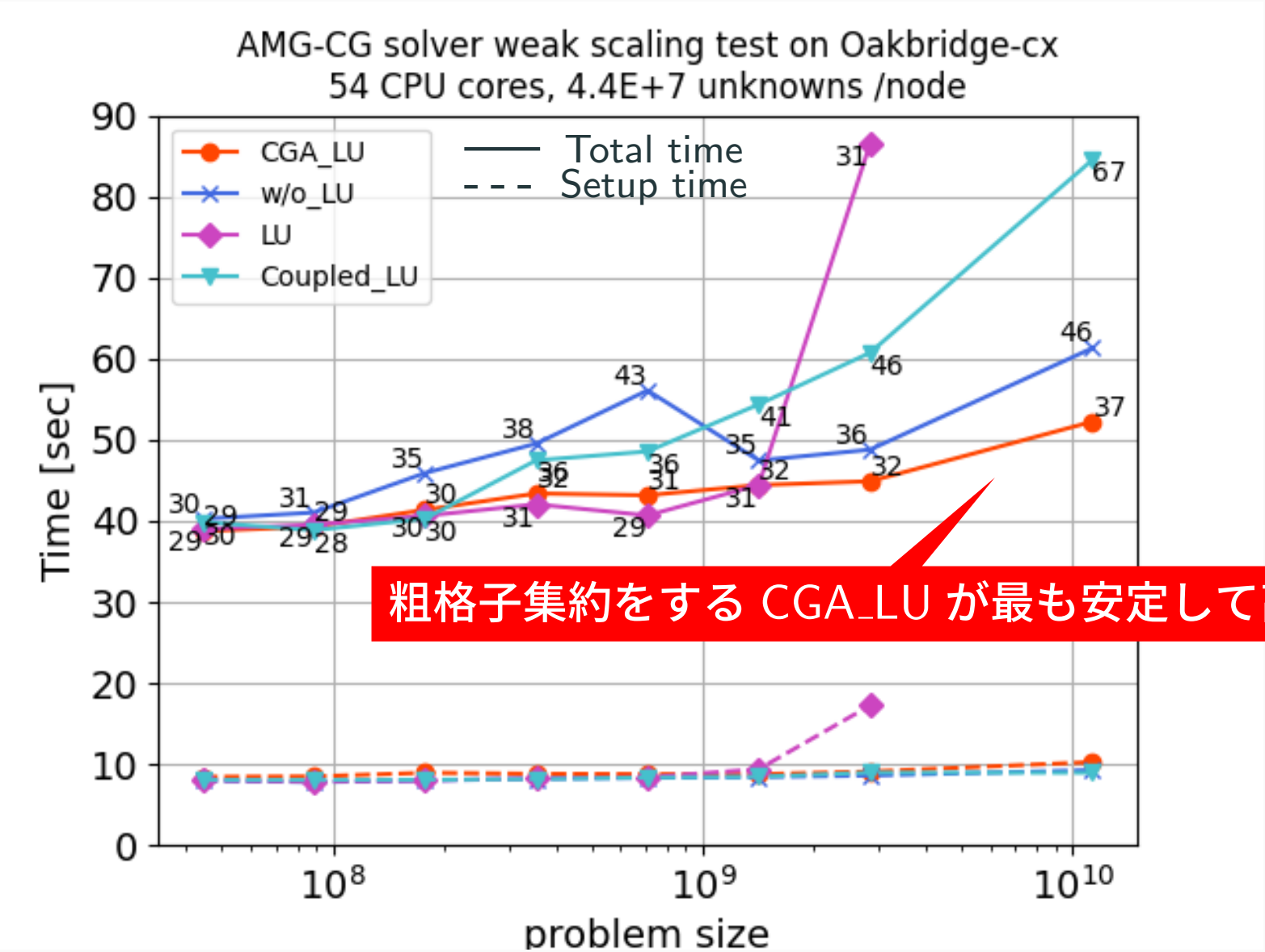
問題設定と環境

- Darcy Flow に基づくポアソン方程式の問題
- 3 次元立方体領域で 10^{-5} から 10^{5} の範囲で不連続に拡散係数が変化する
  - 実験 1：様々な粗格子補正戦略の比較
  - 実験 2：PETSc-gamg との比較
  - 実験 3：GPU solver の性能評価
- 計算環境：Oakbridge-CX（東京大学）と Flow-CX(名古屋大学)
  - Oakbridge-CX：28 core CPU×2
  - Flow-CX: Tesla V100×4 と 20 core CPU×2

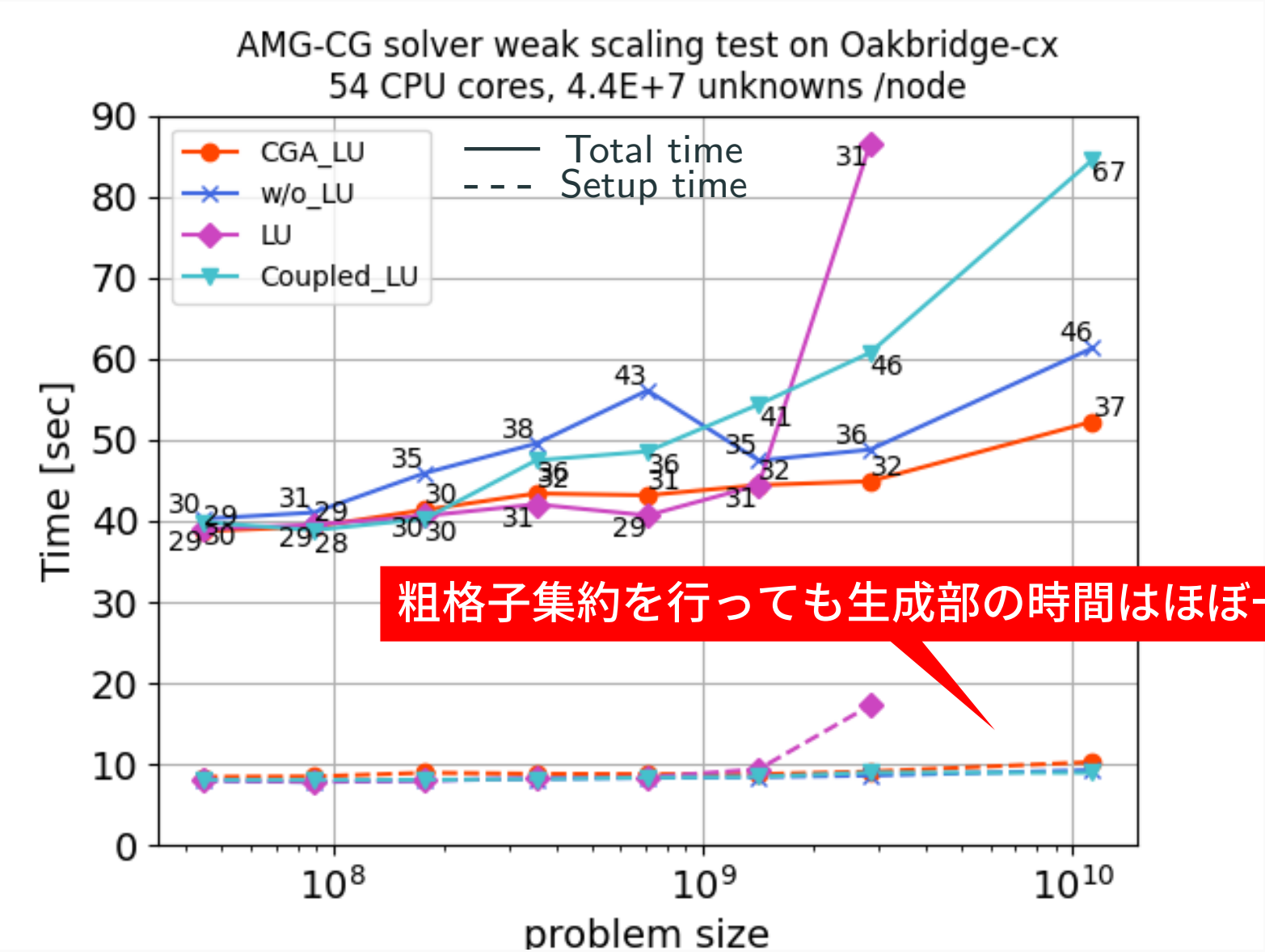
粗格子生成戦略の評価 [最大 256 ノード（18 プロセス 3 スレッド/node）]



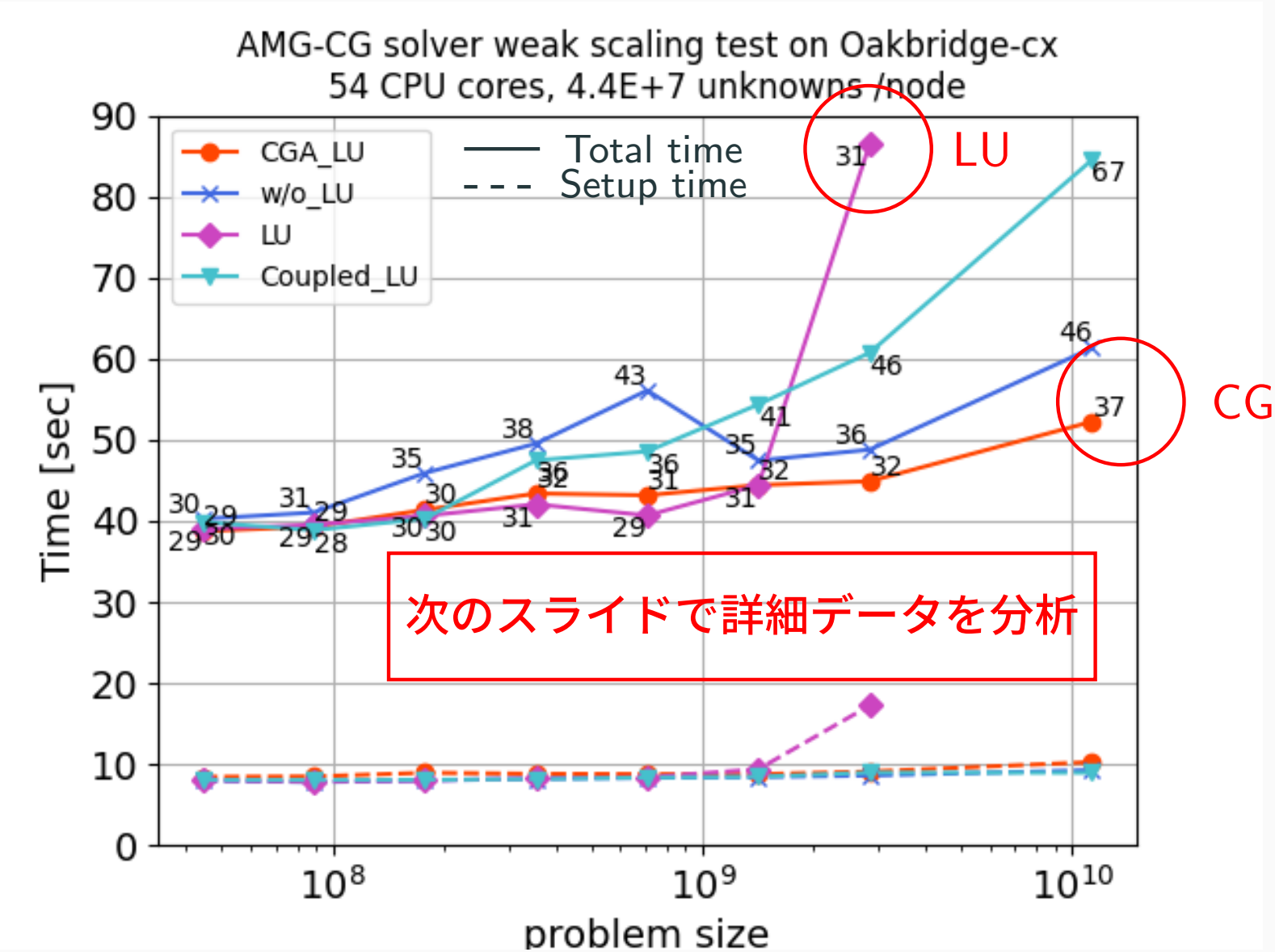
粗格子生成戦略の評価 [最大 256 ノード（18 プロセス 3 スレッド/node）]



粗格子生成戦略の評価 [最大 256 ノード（18 プロセス 3 スレッド/node）]



粗格子生成戦略の評価 [最大 256 ノード（18 プロセス 3 スレッド/node）]



粗格子生成戦略とウィークスケーリング性能

LU　[独立アグリゲート生成で最下層で LU]

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	2834352000	1152	26.90	26		26.500	
2	104157013	1152	69.40	26	26	3.020	2.1
3	2978157	1152	92.03	26	26	0.254	0.251
4	129966	1152	113.38	26	26	0.107	0.083
5	8453	1152	143.57	80	26	0.060	0.032
6	1317	1152	254.63	435	26	1.800	0.019
7	1251	1152	941.20	1151	79	10.500	2.277
8	1244	1152	1244.00	1151	404	12.700	6.6
9	1240	1152	1240.00	1151	785	5.700	6
10	1229	1	1229.00	0	1151	0.030	7.32

CGA_LU　[独立アグリゲート生成に CGA を適用]

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	11337408000	4608	26.90	26		31.70000	
2	416679696	4608	69.70	26	26	5.14000	3.09
3	11919993	4608	92.89	26	26	2.05000	1.29
4	522267	1045	116.73	33	65	1.51000	2.84
5	25947	52	137.30	26	124	0.08900	0.318
6	1062	3	121.00	2	40	0.05300	0.073
7	37	1	35.40	0	2	0.00163	0.0037

粗格子生成戦略とウィークスケーリング性能

LU　[独立アグリゲート生成で最下層で LU]

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	2834352000	1152	26.90	26		26.500	
2	104157013	1152	69.40	26	26	3.020	2.1
3	2978157	1152	92.03	26	26	0.254	0.251
4	129966	1152	113.38	26	26	0.107	0.083
5	8453	1152	14				32
6	1317	1152	25				9
7	1251	1152	941.20	1151	79	10.500	2.277
8	1244	1152	1244.00	1151	404	12.700	6.6
9	1240	1152	1240.00	1151	785	5.700	6
10	1229	1	1229.00	0	1151	0.030	7.32

途中からサイズが小さくならない

CGA_LU　[独立アグリゲート生成に CGA を適用]

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	11337408000	4608	26.90	26		31.70000	
2	416679696	4608	69.70	26	26	5.14000	3.09
3	11919993	4608	92.89	26	26	2.05000	1.29
4	522267	1045	11				2.84
5	25947	52	137				318
6	1062	3	121.00	2	40	0.05300	0.073
7	37	1	35.40	0	2	0.00163	0.0037

プロセス数が徐々に少なくなる

粗格子生成戦略とウィークスケーリング性能

LU [独立アグリゲー

平均非ゼロ要素数/行の増大が止まらない

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	2834352000	1152	26.90	26		26.500	
2	104157013	1152	69.40	26	26	3.020	2.1
3	2978157	1152	92.03	26	26	0.254	0.251
4	129966	1152	113.38	26	26	0.107	0.083
5	8453	1152	143.57	80	26	0.060	0.032
6	1317	1152	254.63	435	26	1.800	0.019
7	1251	1152	941.20	1151	79	10.500	2.277
8	1244	1152	1244.00	1151	404	12.700	6.6
9	1240	1152	1240.00	1151	785	5.700	6
10	1229	1	1229.00	0	1151	0.030	7.32

CGA_LU [独立アグ

平均非ゼロ要素数/行は 120 程度で抑えられる

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	11337408000	4608	26.90	26		31.70000	
2	416679696	4608	69.70	26	26	5.14000	3.09
3	11919993	4608	92.89	26	26	2.05000	1.29
4	522267	1045	116.73	33	65	1.51000	2.84
5	25947	52	137.30	26	124	0.08900	0.318
6	1062	3	121.00	2	40	0.05300	0.073
7	37	1	35.40	0	2	0.00163	0.0037

粗格子生成戦略とウィークスケーリング性能

LU　[独立アグリゲート生成で最下層で LU]

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	2834352000	1152	26.90	26		26.500	
2	104157013	1152	69.40	26	26	3.020	2.1
3	2978157	1152	92.03	26	26	0.254	0.251
4	129966	1152	113.38	26	26	0.107	0.083
5	8453	1152	143.57	80	26	0.060	0.032
6	1317	1152	254.63	435	26	1.800	0.019
7	1244	1152	941.20	1151	79	10.500	2.277
8	1244	1152	1244.00	1151	404	12.700	6.6
9	1240	1152	1240.00	1151	785	5.700	6
10	1229	1	1229.00	0	1151	0.030	7.32

最大 1000 プロセス以上

CGA_LU　[独立アグリゲート生成に CGA を適用]

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	11337408000	4608	26.90	26		31.70000	
2	416679696	4608	69.70	26	26	5.14000	3.09
3	11919993	4608	92.89	26	26	2.05000	1.29
4	1062	4608	116.73	33	65	1.51000	2.84
5	1062	4608	137.30	26	124	0.08900	0.318
6	1062	3	121.00	2	40	0.05300	0.073
7	37	1	35.40	0	2	0.00163	0.0037

最大 100 プロセス程度

粗格子生成戦略とウィークスケーリング性能

LU　[独立アグリゲート生成で最下層で LU]

Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	2834352000	1152	26.90	26		26.500	
2	104157013	1152	69.40	26	26	3.020	2.1
3	2978157	1152	92.03	26	26	0.254	0.251
4	129966	1152	113.38	26	26	0.107	0.083
5	8453	1152	143.57	80	26	0.060	0.032
6	1317	1152	254.63	435	26	1.800	0.019
7	1244	1152	1244.00	1151	79	10.500	2.277
8	1244	1152	1244.00	1151	404	12.700	6.6
9	1240	1152	1240.00	1151	785	5.700	6
10	1229	1	1229.00	0	1151	0.030	7.32

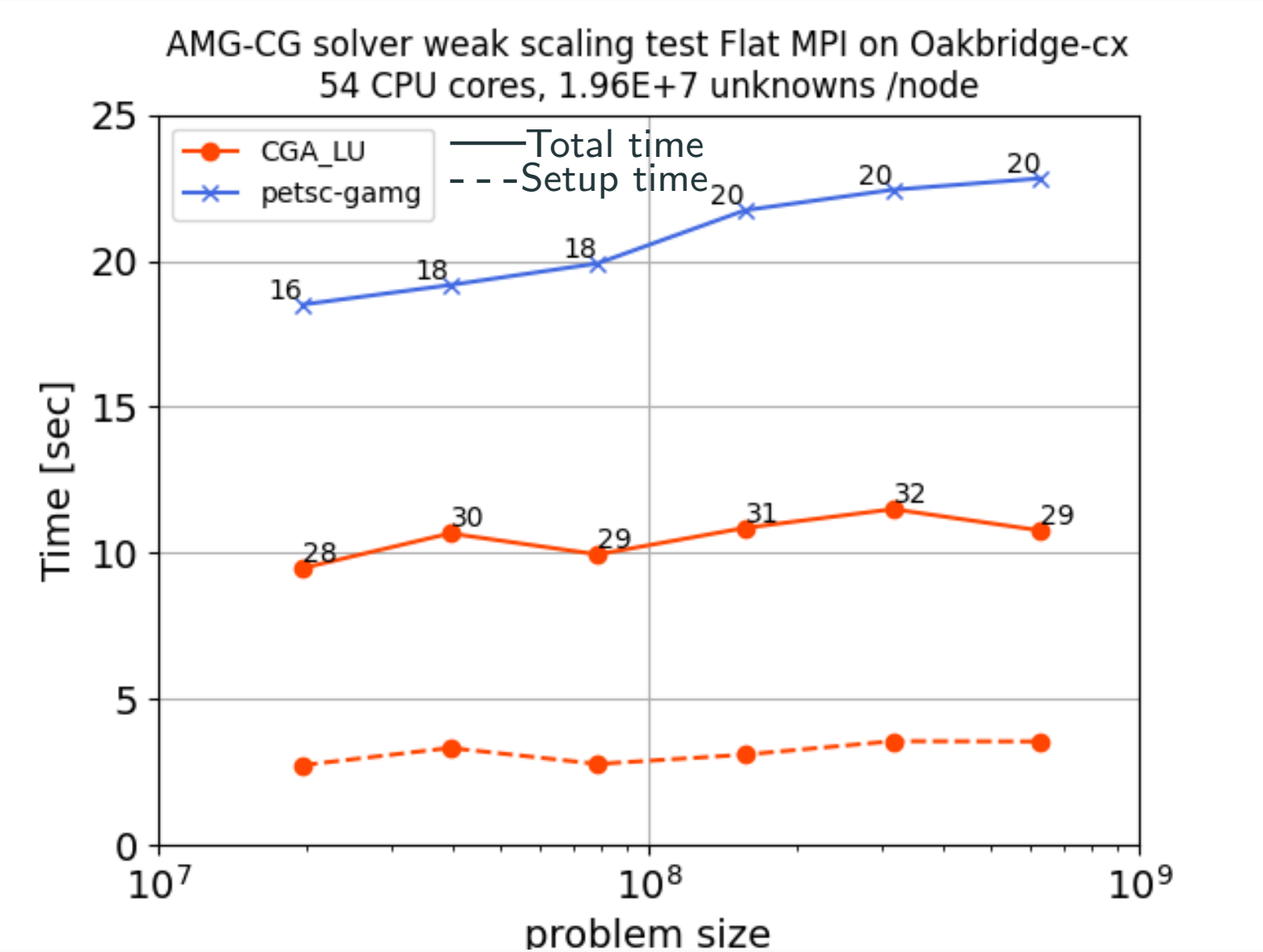
粗いレベルの所要時間が増えている．

CGA_LU　[独立アグリゲート生成に CGA を適用]

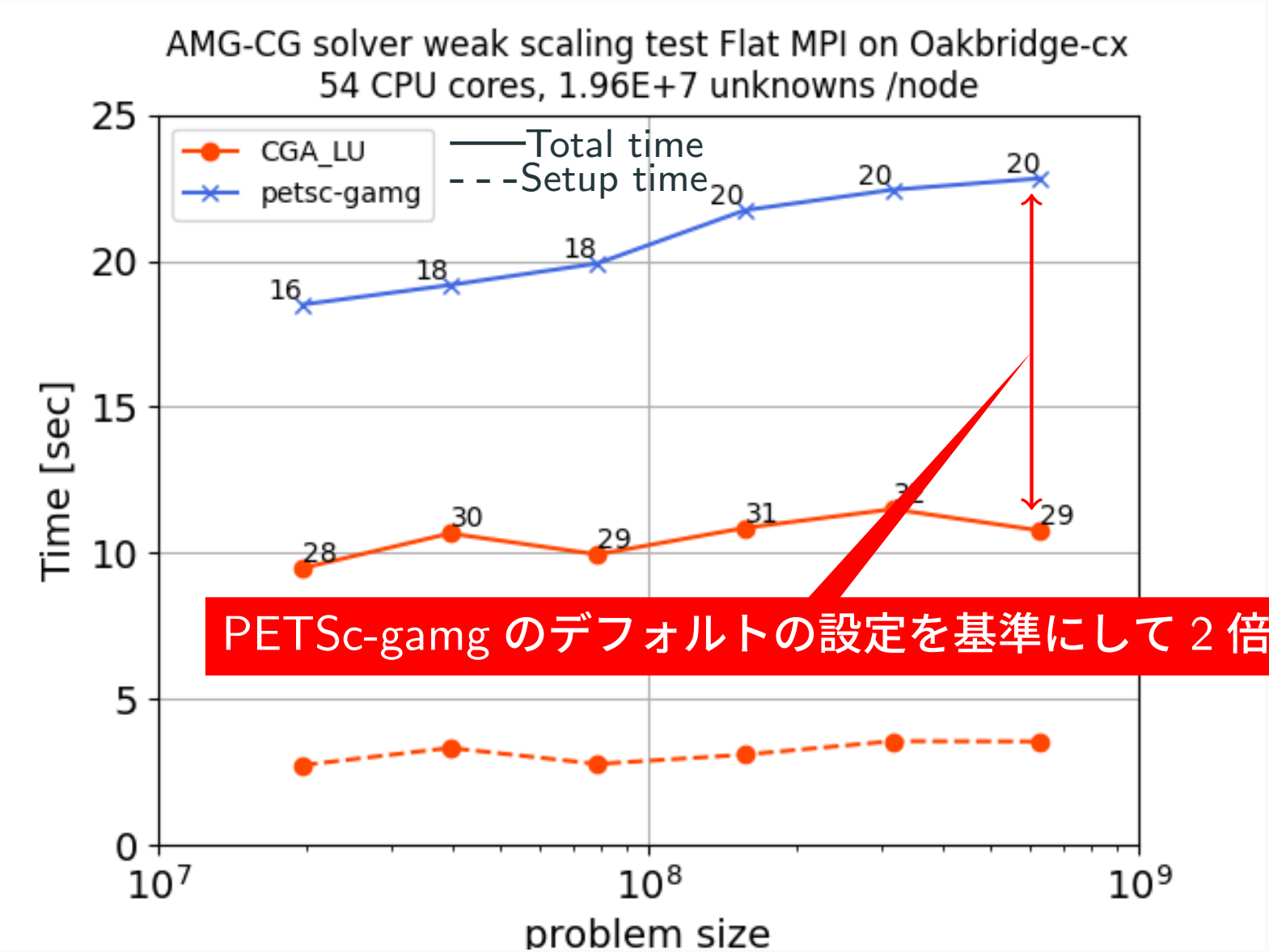
Lev.	DOF	# of P	ave. nonz	max neib	bet levs	max time[s]	bet lev[s]
1	11337408000	4608	26.90	26		31.70000	
2	416679696	4608	69.70	26	26	5.14000	3.09
3	11919993	4608	92.89	26	26	2.05000	1.29
4	1062	3	121.00	2	65	1.51000	2.84
5	1062	3	121.00	2	24	0.08900	0.318
6	1062	3	121.00	2	40	0.05300	0.073
7	37	1	35.40	0	2	0.00163	0.0037

レベルが上がると実行時間も小さくなっている．

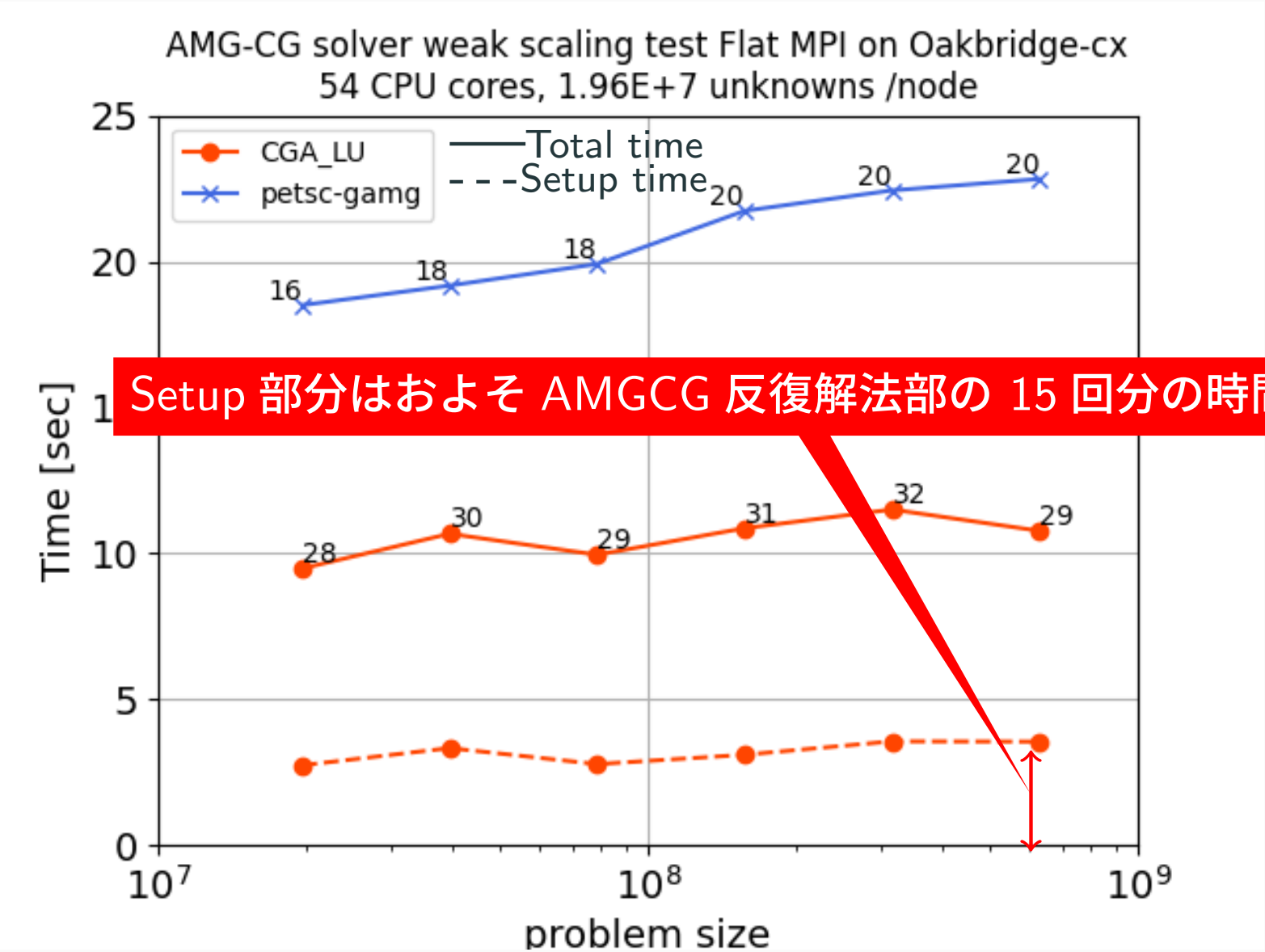
PETSc-gamg との性能比較 [最大 32 ノード (54 プロセス 1 スレッド /node)] -同じデータ分散、プロセス、スレッド数での解法部時間での比較-



PETSc-gamg との性能比較 [最大 32 ノード (54 プロセス 1 スレッド /node)] -同じデータ分散、プロセス、スレッド数での解法部時間での比較-



PETSc-gamg との性能比較 [最大 32 ノード (54 プロセス 1 スレッド /node)] -同じデータ分散、プロセス、スレッド数での解法部時間での比較-



GPU ソルバ評価実験

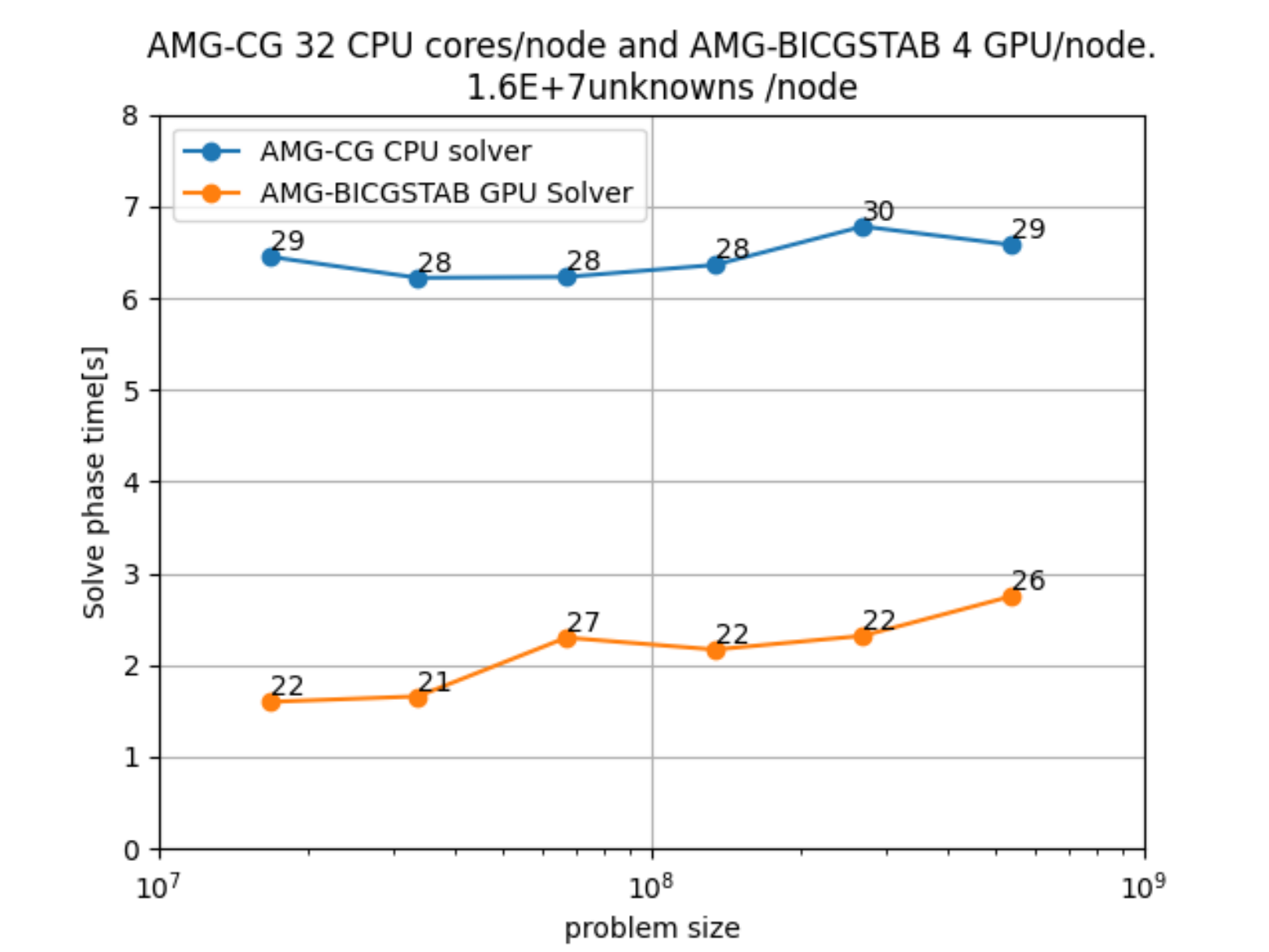
ソルバ設定

- マルチレベル生成部
  - CPU で生成。CGA-LU
- 反復解法部
  - AMG-BiCGSTAB 法
    レベル 1：マルチカラーガウスザイデル
    レベル 2 ー：ヤコビ法
    最下層：直接解法

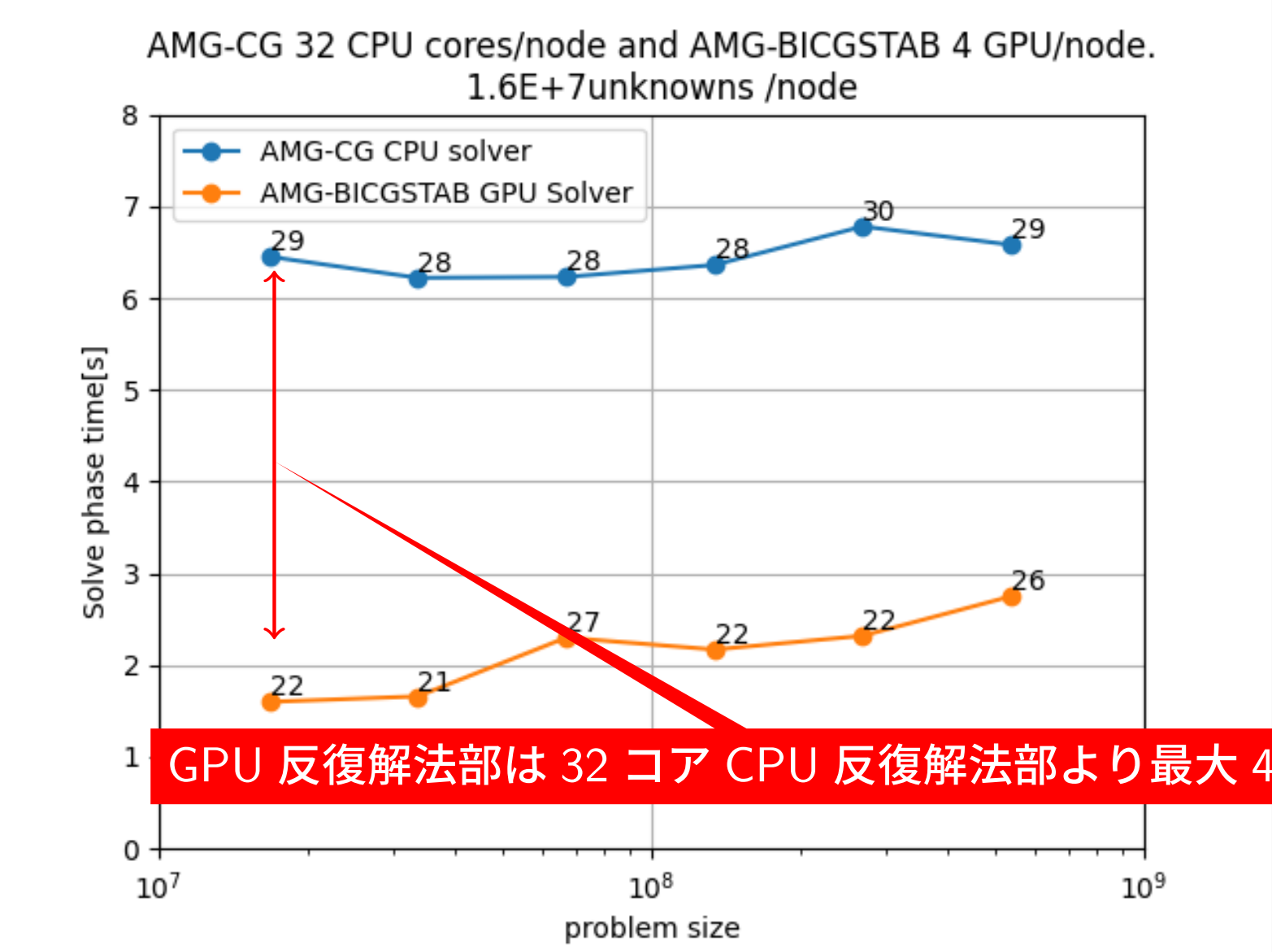
実験設定 **1node** あたりの情報

- GPU ソルバ (Flow-CX)
  4GPU, 4 process, 5 thread
- CPU ソルバ (Oakbridge-CX)
  32 processes, 1thread

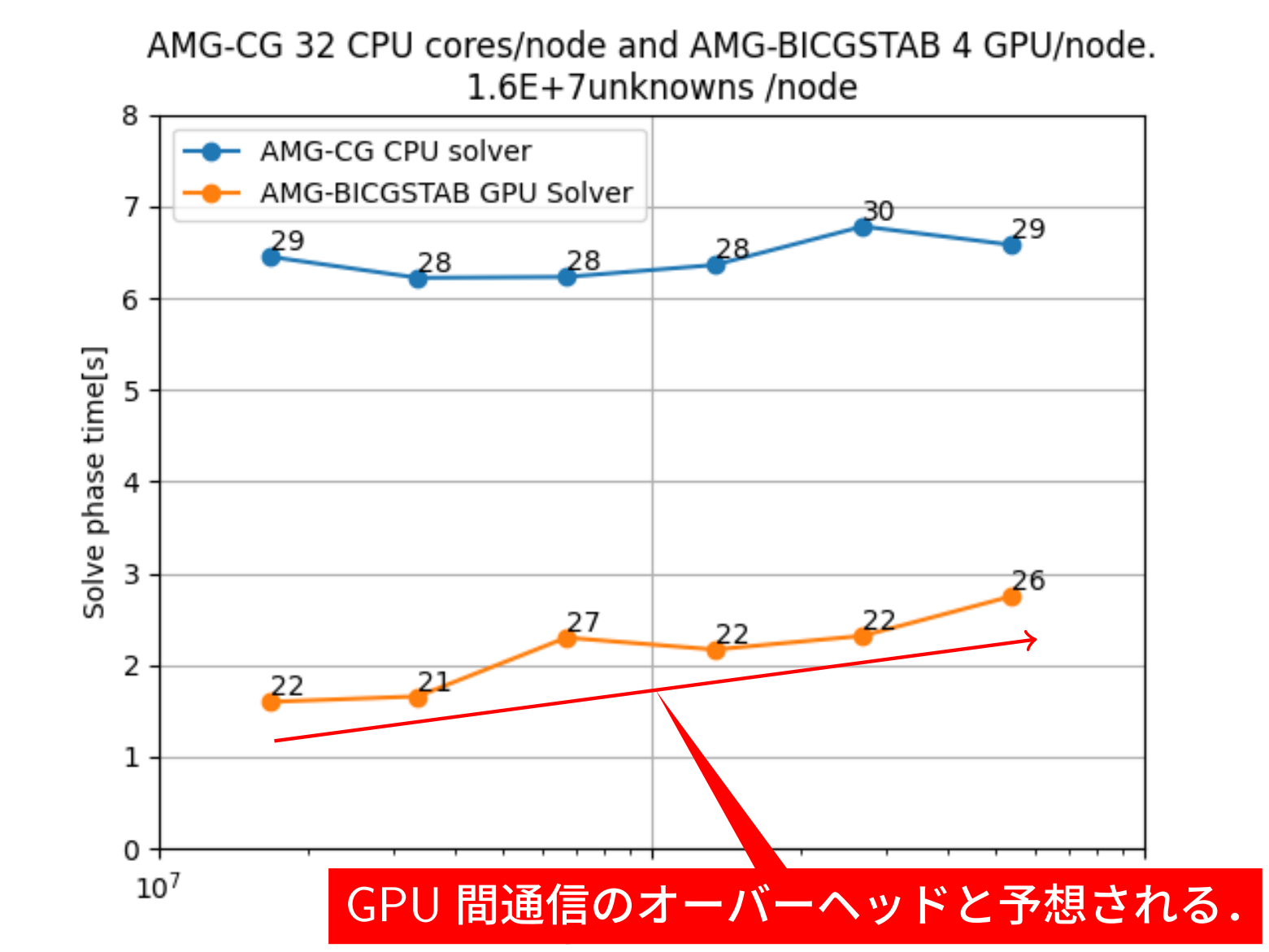
GPU ソルバと CPU ソルバの反復解法部性能比較



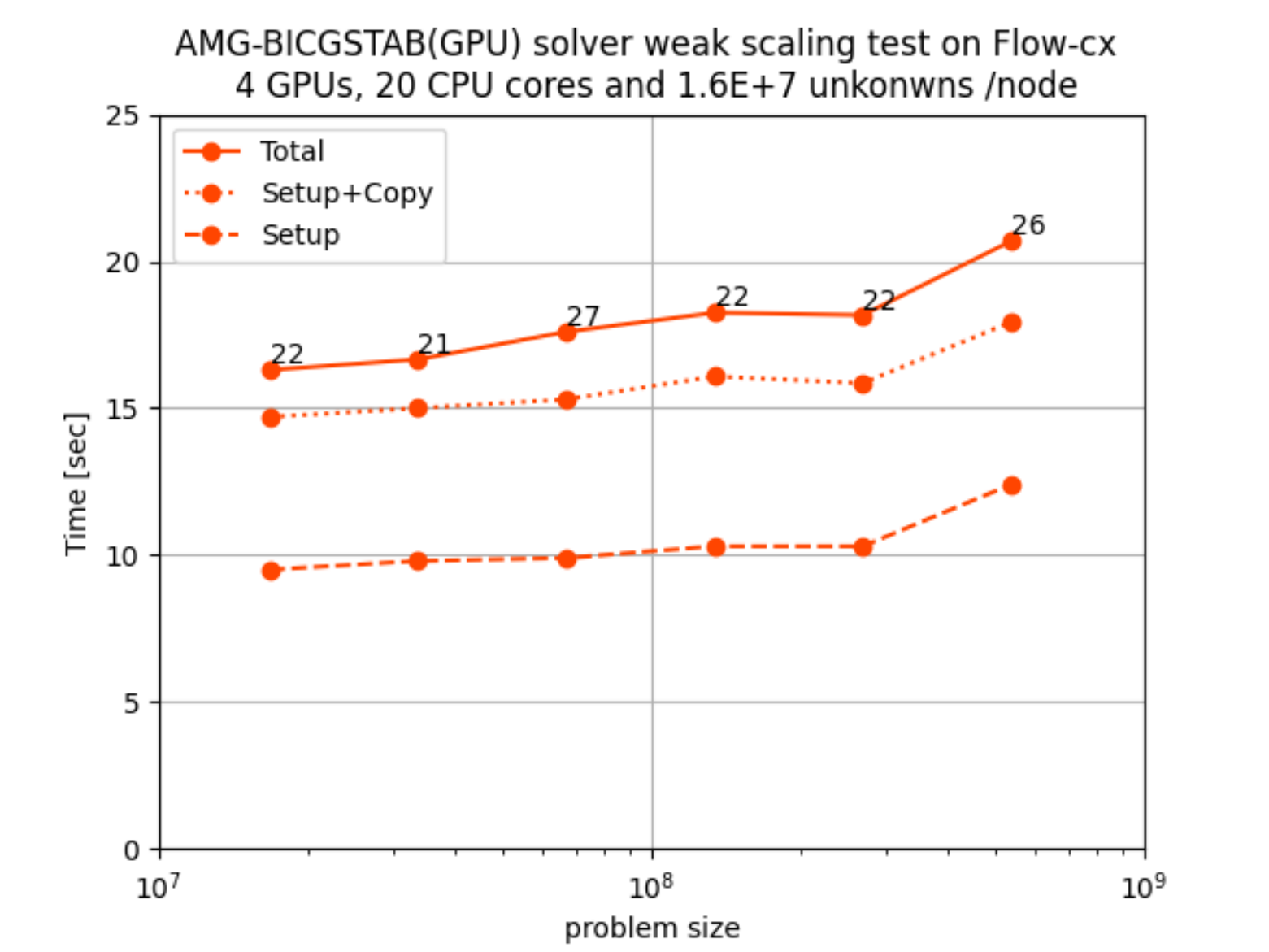
GPU ソルバと CPU ソルバの反復解法部性能比較



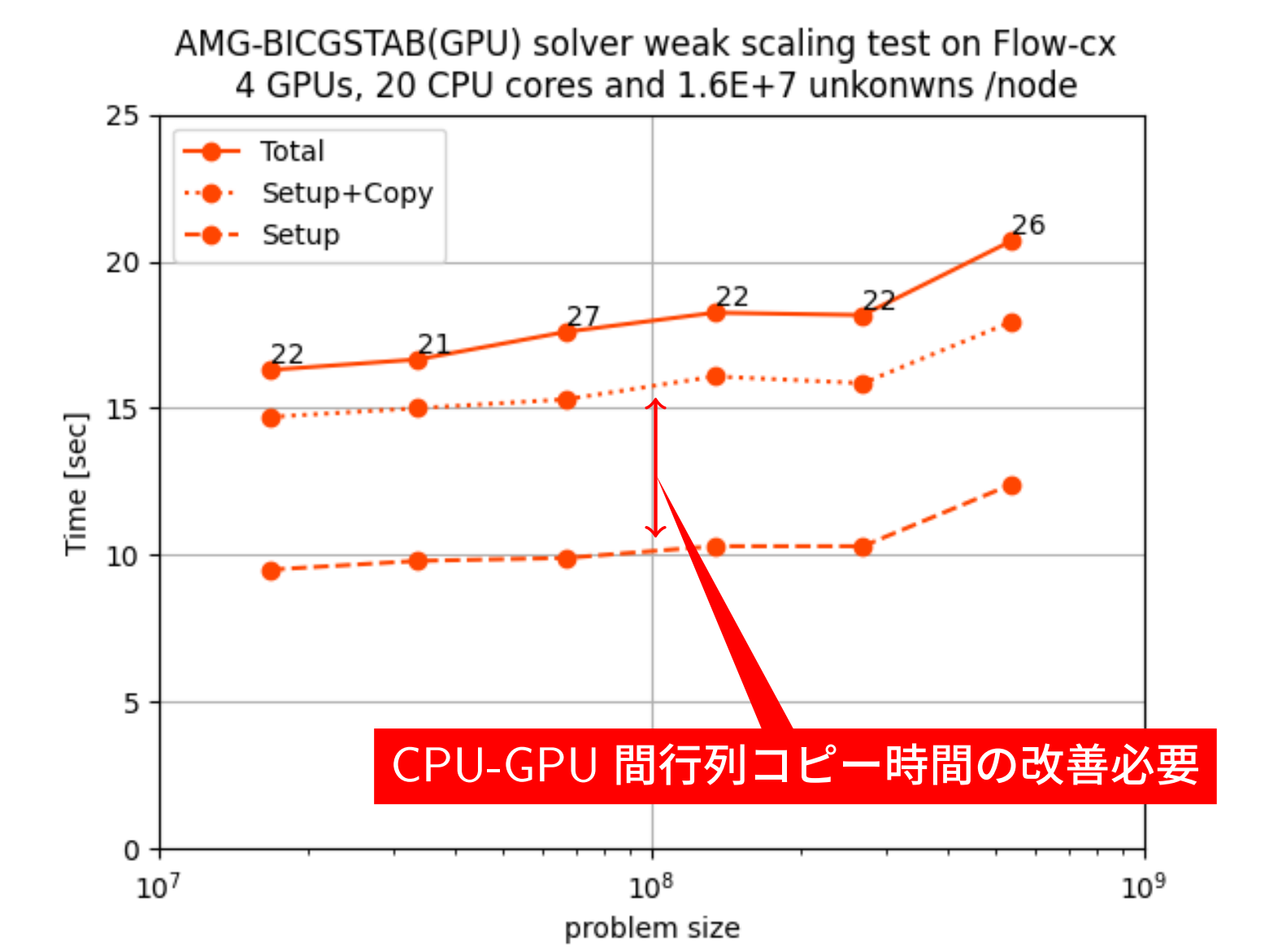
GPU ソルバと CPU ソルバの反復解法部性能比較



GPU ソルバの全体時間構成



GPU ソルバの全体時間構成



まとめと今後の課題

まとめ

- Darcy Flow により現れる拡散係数が不連続に変化するポアソン方程式を題材に 10^{10} 程度の問題に対し，Weak scaling test を行った．
- 提案手法の粗格子集約のオーバーヘッドはほとんどなく、最も高速な設定の一つとなっていた．
  - マルチレベル生成部は問題サイズが大きくなっても実行時間は上昇することはなかった．
  - 収束に要する反復回数が問題サイズが最大の場合，37 回と増大した．スムーザをより強力なものを選択すれば反復回数は抑えられより高いスケーラビリティが実現できると考えられる．
- PETSc のデフォルトのパラメタ設定と行列の分散状態、プロセス数、スレッド数を合わせて実行しても高速に収束していた．
- GPU 反復解法部にも適用でき，CPU 版よりも高速に収束していた．

今後の課題

- アグリゲート番号の管理に全プロセスで共通するグローバル番号を使っており、2 レベル目の未知数の個数が 32 ビットの範囲内に収まるなら、現状のソルバで対応可能

- 10^{11} 以上の DOF の問題の場合，このアグリゲート番号の管理を 64 ビット整数型で行うなど、より大規模な問題を解くためには対応が必要になる．