BAMR 法を用いた並列 LES コードの性能評価

Performance Evaluation of a Parallel LES Code with Blocked Adaptive Mesh Refinement

○ 松尾裕一，宇宙航空研究開発機構，東京都調布市深大寺東町 7-44-1, matsuo.yuichi@jaxa.jp
桑原匠史，アドバンスソフト株式会社，東京都港区赤坂 1-9-20, kuwabara@advancesoft.jp
池知直子，アドバンスソフト株式会社，東京都港区赤坂 1-9-20, ikezi@advancesoft.jp
中森一郎，アドバンスソフト株式会社，東京都港区赤坂 1-9-20, nakamori@advancesoft.jp
Yuichi MATSUO, Japan Aerospace Exploration Agency, 7-44-1, Jindaiji-higashi, Chofu, Tokyo, JAPAN
Takuhito KUWABARA, Advancesoft Corporation, Akasaka 1-9-20 Minato-ku, Tokyo, JAPAN
Naoko IKEZI, Advancesoft Corporation, Akasaka 1-9-20 Minato-ku, Tokyo, JAPAN
Ichiro NAKAMORI, Advancesoft Corporation, Akasaka 1-9-20 Minato-ku, Tokyo, JAPAN

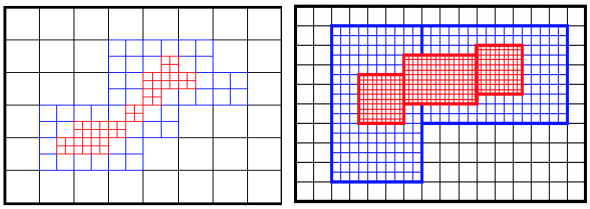
In order to accurately solve the unsteady fluid motion by large eddy simulation, we have to use a grid-resolution as uniform as possible to the whole flow field. It is often the case that the grid-resolution becomes insufficient off the wall when the structured grid system such as an O-type is employed. There have been many researches on the adaptive mesh refinement (AMR) since 1980s. Even in the AMR code, however, we should take advantage of the knowledge acquired in the structured grid approach. Then, we employ the blocked adaptive mesh refinement (BAMR) to remain the grid-resolution to be preferably unchanged in the computational domain. In this study, we made the performance evaluation tests to the parallel BAMR LES developed by the authors' group, and found that some improvements related to data copy and communication should be made to get the better performance up to higher parallel numbers.

１．はじめに

LESにおいては，格子サイズ自体がSubgrid Scale（SGS）渦粘性モデルのパラメータとなるため，捉えるべき現象に対し空間格子解像度をできるだけ一定に保つ必要がある[1]．しかし，物体に対して単一のO型格子を使うような場合には，遠方で解析の解像度が劣化し，特に，ウェークなどを精度良く捉えることが困難となる．このような場合に，マルチブロック格子法や非構造格子法とともに，適合格子（Adaptive Mesh Refinement; AMR）法が有効である．AMRについては，1980年代から現在に至るまで様々な手法が提案されているが，これらは主に，①直交デカルト格子系を基盤として局所的に細分化する方法と，②構造格子上にブロック領域を定義し，その領域内を細分化する方法(Blocked Adaptive Mesh Refinemen; BAMR)の2種類に大別される．

前者のAMRは，BergerとOligerの研究[2]にまで遡ることができ，諸量の空間変化や時間変化に応じてセルベース(Fig. 1(a))で，随時，分割（refinement）と結合（coarsening）を繰り返すものである．主に火炎や噴流，自由界面の解析等に有効な手法であり，Aftosmis[3]やWang[4]により航空宇宙分野にも応用されている．しかし，物体壁面を含む解析には，カットセルや境界層専用格子を使うなどの工夫が必要となり，データ構造や前後処理についても独自の開発が必要となる．

一方，後者のBAMRは，AMRを点単位ではなくブロック単位(Fig. 1(b))で行うものであり，ブロックの中では既存の構造格子ソルバーが使えるため比較的簡単にAMRのメリットを享受できる．翼周りの遷音速非粘性流れに適用した構造格子の例としてDudekら[5]の計算やNS方程式を支配方程式として翼まわりと鈍頭物体まわりの圧縮性流れを計算したSteinthorssonらの例[6]がある．



(a) Cell-based AMR grid　　(b) Blocked AMR grid

Fig. 1 Two types of AMR grids.

本研究は，LES解析における格子解像度の影響を改善することを視野に，BAMR法を用いて自動的に格子解像度を計算領域内で一定に保つような手法を構築するとともに，それを実装した並列化LESコードを開発し，燃焼等の問題に適用することを目指している．筆者らは，前報告までに，8分木のブロック化アルゴリズムに基づくBAMR法を開発・検証するとともに，MPI並列化及び初期マルチブロックへの適用やメモリの削減などの実応用に向けた改善に取り組んできた．本報告では，並列時の計算時間，処理の内訳，ロードバラン等の調査により，開発したLESコードの性能評価を行い，課題と解決法について考察した結果を示す．

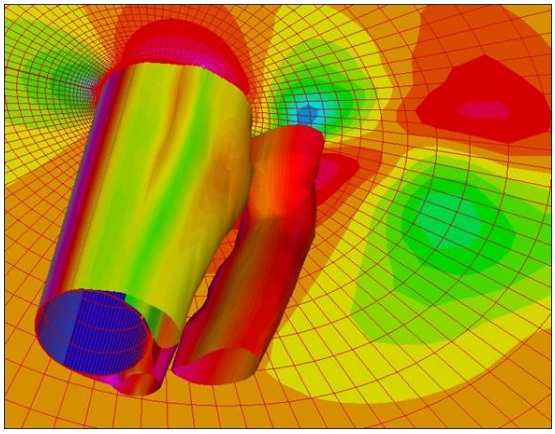
２．数値解法の要点と MPI 並列化

本研究で用いたBAMR法によるベースラインAMR解法においては，各ブロック内での数値解法は，既存の構造格子用の解法及びコードを用いている．構造格子ソルバーであれば他のものに交換することも比較的容易と思われ，そのあたりが本手法のメリットでもある．Fig. 2 は，ベースライン解法により円柱まわりの流れを解いたものである．ベースライン解法の概要やその他の解析事例については，文献 7 を参照されたい．

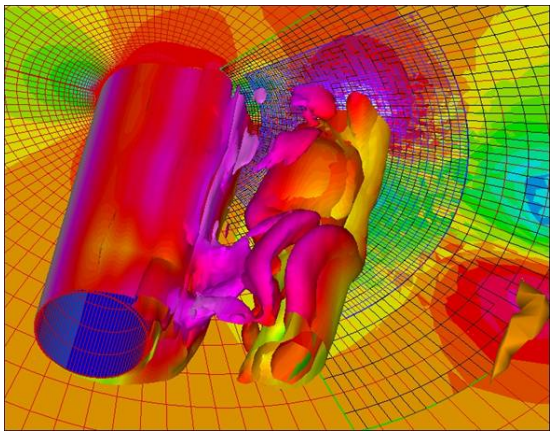
ベースライン解法における格子の AMR 再分割（細分化）方法では，最初の分割がその後の分割数に最後まで影響してしまうだけでなく，領域毎の負荷バランスが悪くなる可能性があり，並列計算に向いていないなど，実用的観点からは課題が多い．そこで，並列化に合わせて負荷分散に都合よい均等なブロック再分割を見据えた AMR のデータ構造やアルゴリズムを採用した．BAMR を並列計算に対応させるには，ブロックの再分割を単に各座標軸方向に二分し，木構造（三次元では 8 分木 Octree，二次元では 4 分木 Quadtree）を構築する処理を繰り返すことによって，BAMR におけるデータ構造とアルゴリズムを比較的単純に構築することができる．（Figs 3-4，文献 10 等を参照．）

木構造の下では，各ブロックの格子点数は同一数になるので，「領域分割」の考え方で並列化を行うことができる．その際，一番簡単なのは，1 ブロック→1CPU という割付の方法であるが，ブロック数は計算途中で再分割によって変わってしまうのと，再分割でブロック数は急激に増加するので，この割付は現実的でない．

Copyright © 2009 by JSFM



(a) Single grid



(b) Bamr grid

Fig. 2: Magnitude of vorticity colored by static pressure, and Mach number contours in the xy-plane.

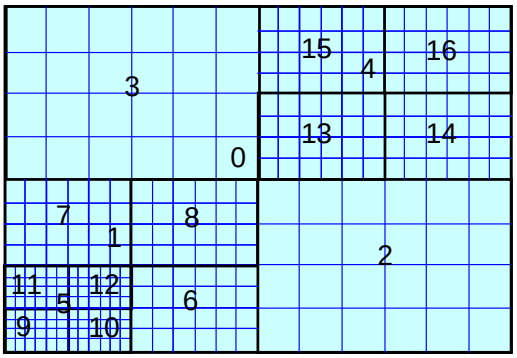


Fig. 3 An example of a blocked AMR grid in 2D.

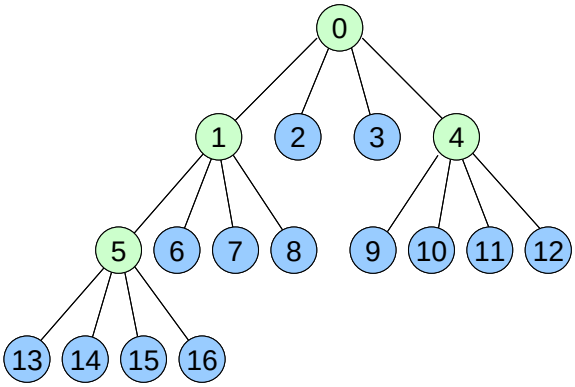


Fig. 4 Data structure by a tree structure.

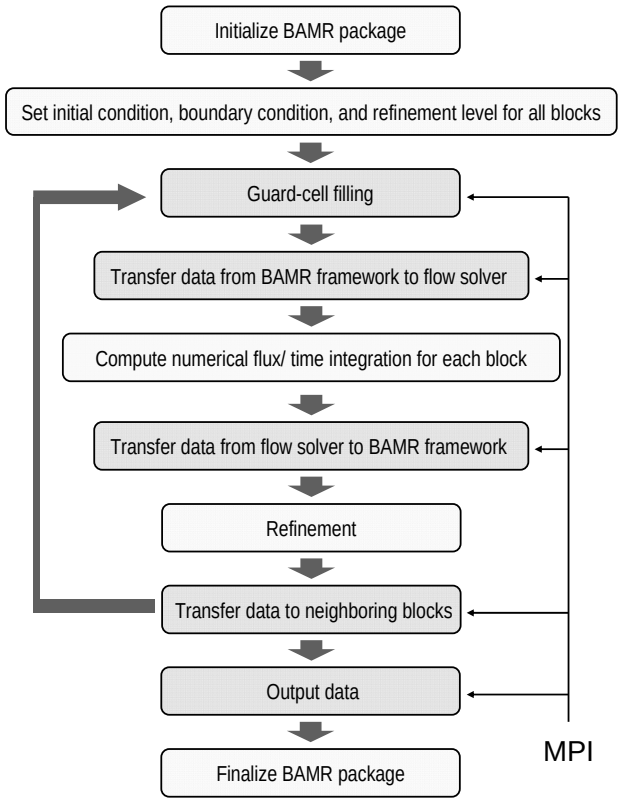


Fig. 5 Workflow of the currently developed parallel BAMR code.

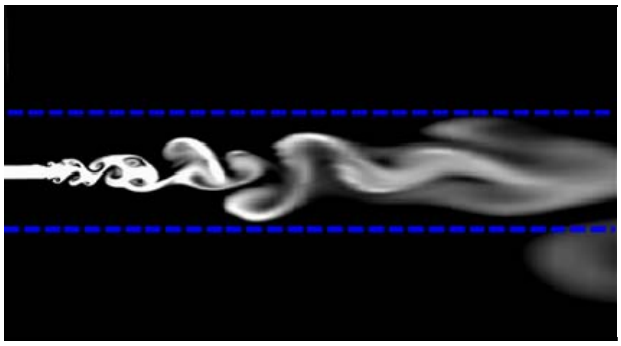


Fig. 6 Use of the multi-block for a free jet flow.

本研究では，複数ブロック→1CPU という割付を可能とするとともに，再分割によってブロックが新たに生成される際，各 CPU の負荷バランスを一定に保つように，各 CPU にできるだけ均等にブロックを配分するアルゴリズムを組み込んでいる．また，ガードセル充填において，各面の転送データ量は同じでないので，面 ID が 1 と 2，3 と 4，5 と 6 の 3 グループに分けて通信を行うようにしている．並列化コードは，MPI と Fortran90 で作成し，流体ソルバーの部分を交換可能なように，ソルバー部と AMR 部は分離したプログラム構造としている．Fig. 5 に，並列化コードの処理の流れを示す．詳細は文献 11 を参照されたい．

３． 実用適用に向けた課題と改良

3.1 マルチブロックへの対応

8 分木のアルゴリズムを用いた BAMR 法では，格子解像度が必要な個所を含むブロックの細分化によって，あまり細分化の必要のない部分にも細分化ブロックが生成されてしまうことがある．これを避けるために，初期ブロックをマルチブロックに対応させた．すなわち，元（親）格子を単一ではなく，複数ブロック（マ

Copyright © 2009 by JSFM

ルチブロック）から出発できるようにし，細分化する必要のあるブロックだけに AMR をかけられるようにした．これにより，あらかじめブロックの細分化の必要な部分と必要でない部分に領域を分けることが可能となり，過剰なブロックの細分化を抑止することができる．例えば，Fig. 6 のようなジェット噴流の場合を考えると，計算領域を 8 分木の方法によって細分化して行くと，1 回目の格子の細分化で，格子の細分化の必要のない噴流の外側領域まで格子の解像度が倍になってしまう．しかし，初期格子をマルチブロック化し，Fig. 6 の青色ラインの領域の中と外に分けておけば，ジェットの外側の領域の不必要な細分化を防ぐことができる．この考え方を実現するために，従来の BAMR 法のデータ構造をそのまま利用でき，かつ大胆なコード改変とならないように，マルチブロック化された初期ブロックは，すべて同じセル数を持つ場合に適用範囲を絞り，この制約の下でマルチブロック化において， 隣ブロック neigh に隣接するブロックの番号が代入されるようにコードの改良を実施した．

3.2 使用メモリ量の軽減

以前の BAMR 法コードでは，全プロセッサにおいて生成されたブロックの数だけ物理量の変数を確保する方式を取っていたため，メモリをかなり余計に使用していた．そのために，ブロック数の増大に伴いメモリ使用量が急速に増加し，計算実施の障壁となる場合があった．これを解消するために，ブロック数が増大する際にネックとなる物理量の変数と格子の変数のメモリ使用量を，各プロセッサで必要最小限に抑える工夫を施した．すなわち，ブロックには，各プロセッサに割り当てられたブロック数分だけの連続的な番号を持つローカルなブロック番号を与え，それと BAMR 法の木構造におけるブロック番号 LB とを対応付ける 1 次元リスト lb_local(LB)を導入した．このリスト配列によって，物理量の変数 u と格子の変数 grid の配列は次のように変更される．

改良前	→	改良後
u(i, j, k, LB)	→	u(i, j, k, lb_local)
grid(i, j, k, LB)	→	grid(i, j, k, lb_local)

LB　全体ブロック番号．各プロセッサ内でとびとびの値を持つ．(0～lbmax：全体ブロックの総数)

lb_local　ローカルブロック番号．各プロセッサ内で, 0 から始まる連続的な値を持つ．(0～localIDmax：各プロセッサに配分されたブロックの総数)

これらの物理量の変数 u や格子の変数 grid を参照する際の，ブロック番号についてのループは， 1 次元リスト lb_local(LB)を通して， ローカルなブロック番号に変換される．Fig. 11 に各プロセッサへのメモリ割り当てのイメージを示した．

以上の改良を行った結果，次に示すようにメモリ使用量を軽量化することができた．検証計算において，16 x 16 x 2 のセルを含む格子を用いて 4CPU での計算を行った場合のメモリ使用量の総和を Table. 1 に示す．細分化したブロック数が少ない場合は，メモリ使用量に大きな差は見られないが， 細分化ブロック数が 400 に近くなってくると, 改良前のコードはおよそ 3GB のメモリを使用しているのに対し，改良後のコードでは， 1 GB 程度に収まっている．並列計算において，流体計算の変数や定数および BAMR 法のデータ構造すべてを必要な分だけ n 個のプロセッサに配分するようにすれば，メモリの使用量を 1/n にすることができるが，実際にはすべてのメモリをプロセッサに分散することはできないため，メモリの使用量は厳密に 1/n にすることは難しい．ここで

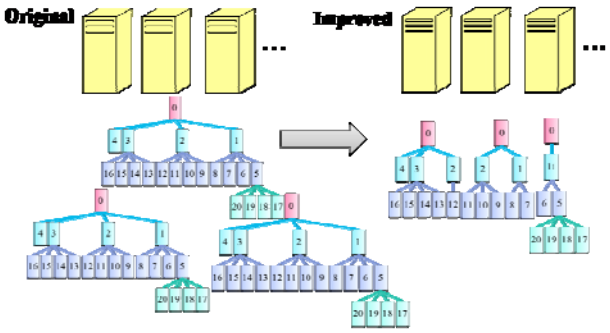


Fig. 7　Image of the memory usage in the original BAMR code and the improved one.

は，流体計算の変数についてはメモリを分散させたが，BAMR 法の木構造を保持するデータ構造のメモリは全プロセッサで同様に持っているため，ブロック数が小さいときには， 木構造の部分のメモリ使用量と流体計算の変数のメモリ使用量が同等となり，流体計算の変数のメモリの分散の効果が見えにくい．しかしながら，ブロック数が増加すると流体計算の変数のメモリ消費が支配的になるために，改良前と改良後のコードではメモリ使用量の違いが顕著に表れる．実際に，検証計算の例では，細分化ブロック数が 5383 の場合は 改良前のコードでは約 32.8 GB，改良後のコードでは 9.7GB となっており，70.5%のメモリを削減することができた．また，改良後のコードでは 8CPU で計算した場合のメモリ使用量が 10.1 GB であった．この場合，改良前のコードのメモリ使用量はプロセッサ数に比例するため 65.5 GB であり，改良後のコードは 84.6 % のメモリを削減できていることになる．このメモリの軽量化によって，さらに多数のブロック数を使用した計算も可能であることが分かった．

上記と同じ検証計算において，1 セル 1 ステップあたりの処理時間について調べた結果を Table 2 に示す 表から分かるように，処理時間については ほとんど変化がみられなかった．2 次キャッシュミス率については，改良前， 改良後いずれのコードも 0. 1%程度，TLB ミス率についても 0 に近い値であるため，メモリの軽量化によって処理時間に差がでなかったと考えられる．

Table 1　Total memory usage of the BAMR code.

Number of leaf blocks	Original	Improved
79	768 MB	704 MB
463	3072 MB	1024 MB
1487	9408 MB	3136 MB
5383	32768 MB	9664 MB

Table 2　Effective CPU Time / cell / step.

Number of leaf blocks	Original	Improved
79	0. 108 msec	0. 107 msec
463	0. 109 msec	0. 107 msec
1487	0. 113 msec	0. 108 msec

Copyright © 2009 by JSFM

4． 並列 BAMR コードの性能評価

前報[12]では，並列化 BAMR 法コードによる混合層[13]の計算を対象に検証計算を行った．そこでは，単一格子から計算を出発させ，10000 ステップ毎にブロック再分割判断を行いブロック数が増加していく場合の性能調査を実施し，単一格子の場合と BAMR の場合における各々の格子数削減のメリットや使用メモリ量削減についての優位性を示す結果を得た．

本報告では，900 タイムステップの間で強制的にブロック再分割を行った場合の性能調査を行った．計算開始時の単一格子数は (nx, ny, nz)=(72, 54, 2)を用い，ジェットの主流方向を x，高さ方向を y，奥行き方向を z とした．また，分割ブロック数については，前回の検証例題においては，最終的なブロック分割数は 40 程度であったが，今回は更に一桁多い 440 ブロックまでの性能評価を行った．計算は，終了時のブロック分割数が 8 ブロック，56 ブロック，440 ブロックになる 3 つのケースについて，2CPU，4CPU，8CPU，16CPU，32CPU を用いた並列計算を行った．(ただし，8 ブロックのケースについては，16CPU，32CPU の場合，CPU が無駄になるため計算は行わない．)

Fig.8 は，計算終了時のブロック分割数に対する計算時間を示す．青線が 2CPU，ピンク線が 4CPU，黄色線が 8CPU，水色線が 16CPU，茶色線が 32CPU の場合の結果である．この図より，16CPU まではブロック分割数によらず概ね良好な並列化効率が得られていることが分かる．32CPU については，初期に 8 ブロックしかないため，56 ブロックに再分割されるまでは CPU 数に無駄が生じるため並列化効率が落ちている可能性がある．

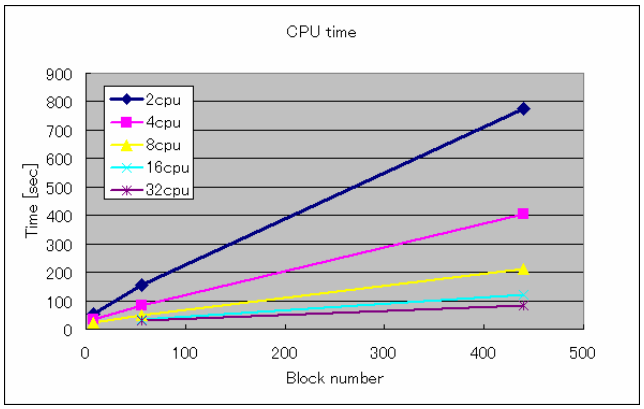
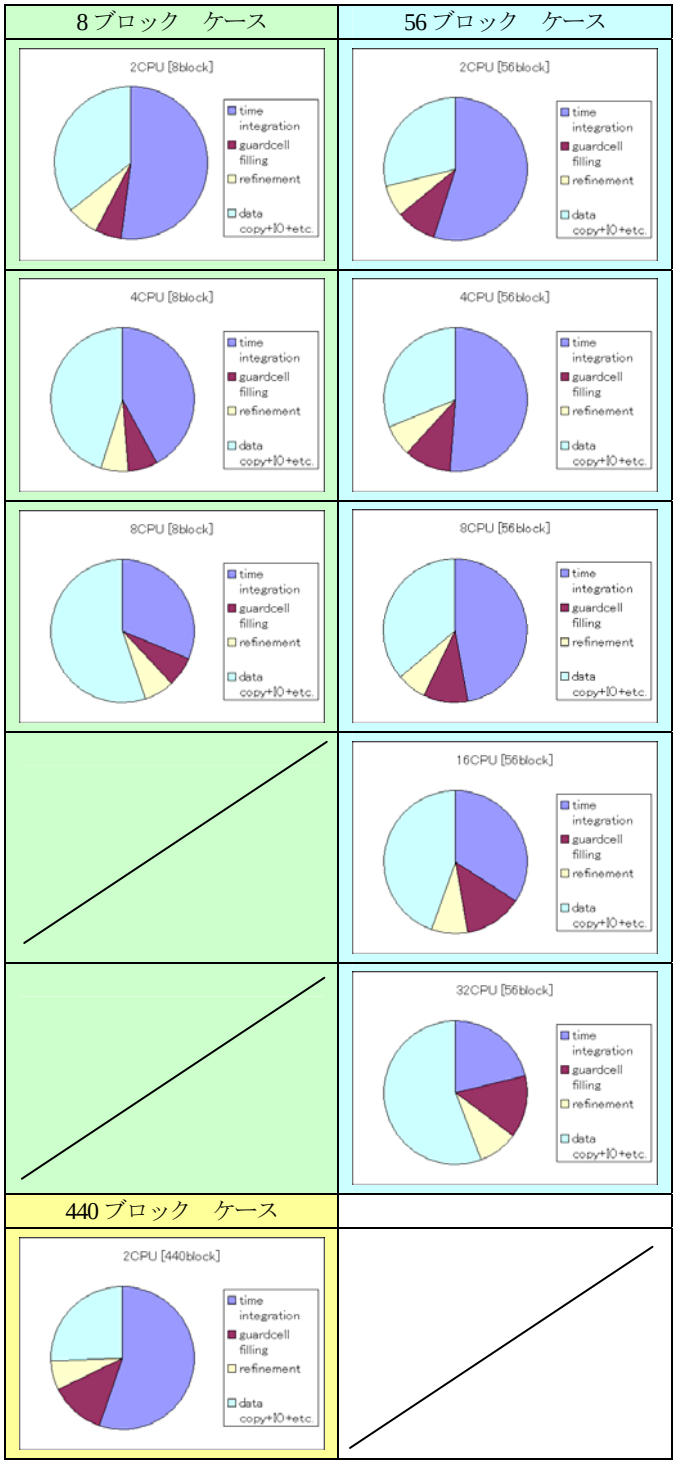


Fig. 8　Comparison of elapse time for parallel computations (2CPU, 4CPU, 8CPU, 16CPU, and 32CPU).

次に，３種のケースにおける計算コストの内訳を，計算に用いた CPU 数毎に Fig. 9 に示す．ブロック数が 8 ブロックのケースに注目すると，全体の計算時間におけるデータ補間等にかかる割合（水色部分）が，CPU 数の増加に伴い大きくなることが分かる．これは，ブロック数が少ないために，CPU 数が増えるにつれて流体計算部分（紫色部分）にかかる時間に対して細分化レベルの異なるブロック間のデータ補間や通信にかかる時間が相対的に増加しているためと思われる．他のケースについても同様の傾向が見られるので，CPU 数を多くした場合は問題が顕在化する可能性がある．一方，新たに分割された領域の袖の部分を境界上面として設定する guardcell filling（茶色部分）と，分割された領域内を細分化する refinement（黄色部分）にかかる時間の割合については，同じケース内では，CPU 数が変わってもそれ程大きな違いがないことが分かる．

32CPU 使用時の各 CPU の経過時間を Fig. 10 に示す．ブロックの増加とともに，各 CPU にできるだけ均等にブロック数を割り当てるようにしているため，ロードバランスは比較的良好である．一方，Fig.11 には MPI によるデータ待ち時間の割合を示した．これを見ると，通信時間に多少のばらつきがある．より効率を向上させるためには，データ通信や IO 部分の割合を減少させる必要があることがわかる．全体の処理時間に占める流体計算以外の部分（特にブロック再分配に伴う，担当 CPU へのブロックデータの MPI 通信）の割合が高いことがはっきりしたので，今後重点的に改善して行く予定である．



Copyright © 2009 by JSFM

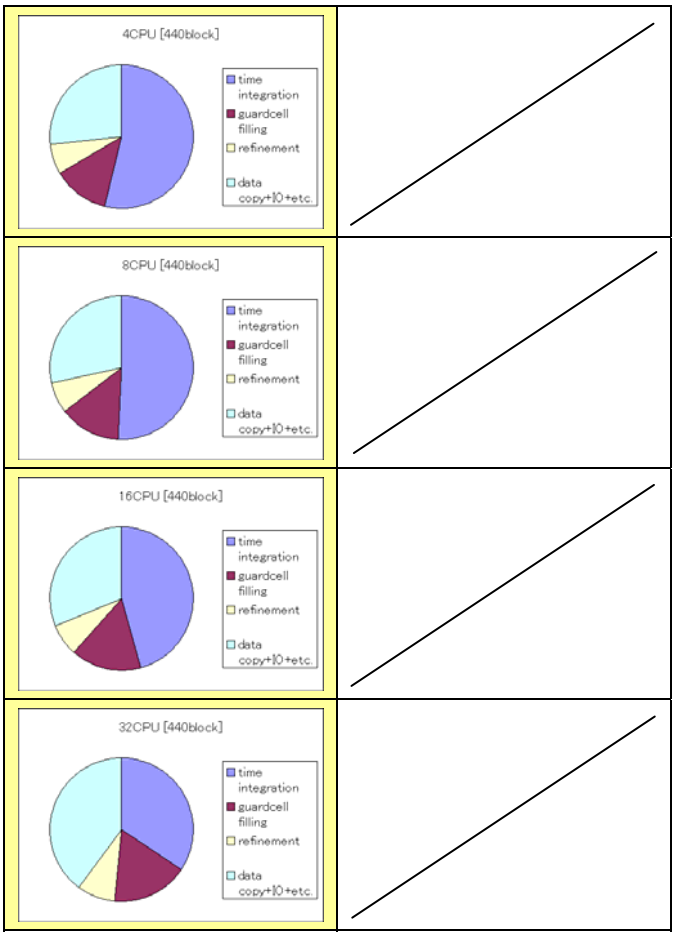


Fig. 9　Cost comparison for the case of different block number and used CPU-numbers.

Elapsed(s)	User(s)	System(s)		
92.8380	2767.1899	58.4000	Application	
92.8380	86.3400	2.9900	Process	4
92.6092	85.7800	2.5800	Process	0
92.5257	85.8900	2.7500	Process	1
92.4699	86.3300	2.3800	Process	2
92.2714	86.4300	2.3600	Process	3
92.1651	86.2400	2.1000	Process	5
92.0584	86.4900	2.0900	Process	6
91.9173	86.3100	2.1800	Process	7
91.8732	86.2400	2.1000	Process	8
91.7750	86.6000	1.9900	Process	9
91.7631	86.2900	1.9500	Process	10
91.6933	86.6500	1.9500	Process	12
91.6896	86.2000	1.9500	Process	11
91.5674	86.6500	1.8300	Process	13
91.4550	86.4200	2.0000	Process	14
91.3888	86.5200	1.7500	Process	15
91.3710	86.4400	1.6400	Process	16
91.2143	86.2500	1.8300	Process	17
91.1658	86.7200	1.7100	Process	18
91.1279	86.6800	1.6400	Process	19
91.0846	86.6500	1.5600	Process	20
91.0294	86.7700	1.4600	Process	21
90.9461	86.5400	1.4700	Process	22
90.9176	86.6500	1.4000	Process	23
90.8057	86.7600	1.4900	Process	24
90.7037	86.5400	1.5500	Process	25
90.6375	86.5200	1.4500	Process	26
90.5749	86.6300	1.2500	Process	27
90.5229	86.5300	1.3100	Process	28
90.4125	86.6600	1.3100	Process	29
90.3292	86.7100	1.2100	Process	30
90.2972	86.7600	1.1700	Process	31

Fig. 10　Time information (Elapsed time, User time, and System time) for each CPU

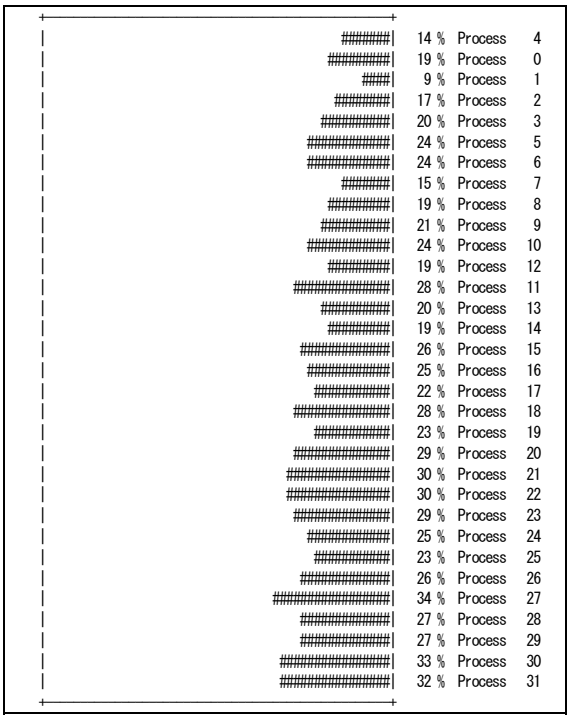


Fig. 11　Percentage of time waiting for a received and get

最後に，コードの特性と AMR の効果を調べるために，1 次元衝撃波管問題へ本 BAMR 法を適用した際の結果を Fig. 13 に，BAMR 法を適用しない場合の結果を Fig. 14 に示す．Sod の標準問題を扱っており，圧力比が 10，密度比が 8 とした．Fig. 13 における厳密解は赤い実線であり，初期の格子数は 128 分割とし，これを初期（レベル 0）として 2 段階（レベル 2）までの AMR 格子を自動的に生成している．ここで，格子の細分化レベルの様子は，Fig. 13 の下図に示している．衝撃波が領域の右側へ進行するにつれてその波面を含むブロックにおいて細分化がなされ，膨張扇についても同様の操作がなされていることが確認できる．なお，接触不連続面については，時間を追って格子解像度を向上させても急峻な勾配には回復しない．これについては，計算開始後の早いタイミングにおいて細分化の候補となるブロックを特定し，格子の細分化が達成されるように本 BAMR コードの user-adjustable parameter を調節することで対処した．

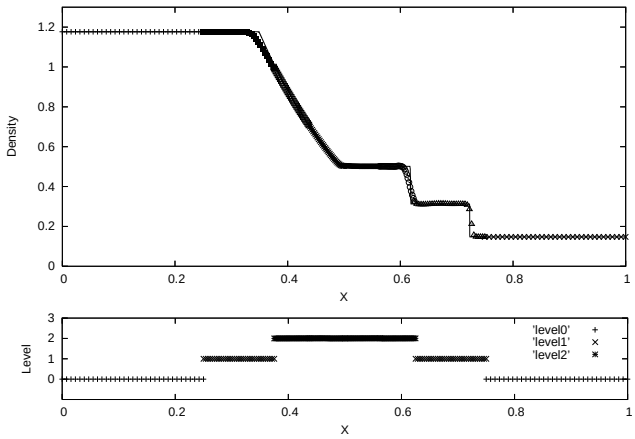


Fig.12　Top: Comparison of numerical results for Sod's shock-tube problem : a solid line(red-line) shows analytic solution; markers show the BAMR solution.　Bottom; Grid spacing for each level created by BAMR splitting.

Copyright © 2009 by JSFM

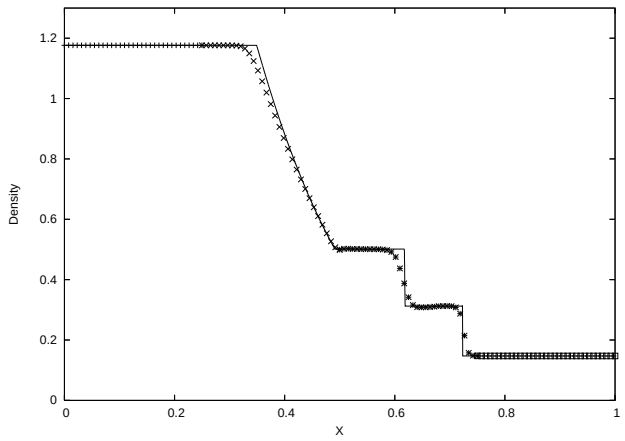


Fig. 13 Comparison of the numerical result of no-BAMR calculation for Sod's shock-tube problem; a solid line (red-line) shows the analytic solution, and markers show the result by no-BAMR calculation.

５．まとめ

LES解析における，解像度確保のための格子数増大による計算負荷を改善するために並列化BAMR-LES解析コードを開発し，その有効性の調査を行ってきた．本BAMR-LESコードは，自動的に格子解像度を計算領域内で一定に保ち，並列計算に適した８分木のデータ構造を持つ手法を実装しており，メモリ使用量についても削減を行い，多数のブロックを用いた計算において効果的に抑えられるように改良を行ってきた．今回は，短い計算ステップの間にブロック分割が行われるケースについての性能調査を行った．今回行った計算で，より多くのCPUを用いて効率よく並列計算を行う場合には，初期ブロックの数と分割を行うタイミングに注意が必要ということと，効率よく計算できるCPU数と格子数の関係が見えてきたため，今回の結果は，より多くの複数CPUで計算を行う場合の格子数を決めていく指標になる．

本報告までの開発と性能評価を通じて，BAMR法の特性や効果を明らかにすることができた．一方で，コードの問題やブロックの細分化と担当ブロックのCPUへの再分配に伴うデータ補間やデータ通信に時間を要するなどのボトルネックも表面化した．今後は，ボトルネックを解消する改善を継続に実施するとともに，機能を精査し，アルゴリズムの見直しも含めたコードの再構築を行なっていく予定である．

参考文献

1) P. R. Spalart : International Journal of Heat and Fluid Flow, Vol. 21, pp. 252-263, 2000.
2) M. J. Berger, et al. : J. Comput. Phys. Vol. 53, pp. 484-512, 1984.
3) M. J. Aftosmis, et al. : AIAA Paper 2000-0808, 2000.
4) Z. J. Wang, et al. : AIAA Paper 2000-0395, 2000.
5) S. Dudek, et al. : AIAA Paper 98-0543, 1998.
6) E. Steinthorsson, et al. : NASA TM 106704, 1994.
7) R. Löhner: John Wiley & Sons Ltd. , West Sussex, 2001.
8) R. Löhner: "An Adaptive Finite Element Scheme for Transient Problems in CFD, " Comp. Meth. Appl. Mech. Eng. Vol. 61, pp. 323-338, 1987.
9) 松尾裕一，中森一郎：ブロック AMR 法による LES 解析の局所的高解像度化の試み，第 20 回数値流体力学シンポジウム，講演番号 E1-2, 2006.
10) P. MacNeice et al. : "PARAMESH : A parallel adaptive mesh refinement community toolkit. ", Computer Physics Communications, Vol. 126, pp. 330-354, 2000.
11) 松尾裕一，中森一郎，池知直子 : LES 解析のための並列化 BAMR コードの開発，第 21 回数値流体力学シンポジウム，講演番号 E1-6, 2007.
12) 松尾裕一，中森一郎，池知直子：BAMR 法による圧縮性自由噴流の LES 解析，第 22 回数値流体力学シンポジウム，講演番号 J9-6, 2008.
13) 梶昭次郎 : 超音速ジェットスクリーチについて，ながれ 20, pp. 204-212, 2001.

Copyright © 2009 by JSFM