３　計算プログラムの最適化

３．１　最適化に関わる条件

「地球シミュレータ」では利用ノード数制限緩和申請に当たり、下記の条件が課されている。

・ベクトル化率：95%以上

・並列化効率：50%以上

ここで、ベクトル化率とは、プログラムの実行において全てスカラー命令で実行した場合の総計算時間に対するベクトル命令で実行可能な時間の割合のことである。並列化効率とは並列加速率（プロセッサ１台に対する実行性能）を使用しているプロセッサ台数で割った値である。

アムダールの法則より、

$$T_{VECTOR} = T_{SCALOR}\left[(1-\beta_r)+\beta_r / N_V\right] \qquad (217)$$

T_{VECTOR}：ベクトルプロセッサの総計算時間

T_{SCALAR}：スカラプロセッサの総計算時間

β_r：ベクトル化率(0≦ β_r ≦1)

N_V：ベクトルプロセッサによる高速化係数

同様に

$$T_{PARALLEL} = T_{SINGLE}\left[(1-\alpha_r)+\alpha_r / N_P\right] \qquad (218)$$

$T_{PARALLEL}$：複数のプロセッサによる総計算時間

T_{SINGLE}：シングルプロセッサによる総計算時間

α_r：並列化率(0≦ α_r ≦1)

N_P：プロセッサ台数

並列化効率は次式で表される。

$$E_r = \frac{T_{SINGLE}}{N_P T_{PARALLEL}} \qquad (219)$$

E_r：並列化効率(0≦ E_r ≦1)

３．２　必要になる計算機資源の推定

３．２．１　実行環境

計算プログラムの最適化に当たり、「地球シミュレータ」上における計算プログラムの性能を確認する必要がある。最終的にターゲットとする領域は東京23区(約30km四方)であるが、最適化の方針を策定するため1km四方のデータを活用し、16CPUと64CPUで計算して実行時の性能を評価した。測定した環境を以下に記す。

実行マシン：「地球シミュレータ」

計算領域：1km(x)×1km(y)

格子点間隔：dx=dy=5[m]

計算格子数：200(x)×200(y)×100(z)

=4,000,000 点

測定ケース：2ノード16CPU(ケース1)利用と8ノード64CPU(ケース2)利用の2ケース

３．２．２　ベクトル性能

ケース１、２の実施結果からベクトル処理に関する情報を抜き出したものが、表９である。ベクトル長はできるだけ256に近い方が望ましい。CPU台数が増えることで領域を細分割したため、ベクトル長が短くなり、ベクトル演算率低下の一因となっている。一般に、並列台数を増やすと逐次処理部分のスカラ部分の割合が増え、ベクトル演算率は低下する。この低下を最小限に抑えるために、逐次処理部分の高速化およびループの一重化、ループ長を長く取れるような分割方法の検討が重要になる。

表９　ベクトル性能

ケース	ケース1	ケース2
CPU台数	16	64
経過時間(sec)	202.978	81.445
ベクトル演算率(%)	98.49	97.07
平均ベクトル長	153.186	105.902
GFLOPS（全体）	28.038	70.847
ピーク比(%)	21.90	13.80

３．２．３　並列性能

並列処理に関する情報を抜き出したものが、表１０である。式(218)より、

$$\begin{aligned} 202.978 &= T_{SINGLE}\left[(1-\alpha_r)+\alpha_r/16\right] \\ 81.445 &= T_{SINGLE}\left[(1-\alpha_r)+\alpha_r/64\right] \end{aligned} \qquad (220)$$

になり、α_r=0.9845、T_{SINGLE}=2633.638 が得ら

れる。また、並列化効率を見ると、64 台で 50%まで下がっていることがわかる。現状では、1km 四方の問題サイズに対して「地球シミュレータ」の最適化の条件を満たすのは 64CPU（=8 ノード）による実行が限界である。

３．２．４　30km 四方

16CPU と 64CPU のメモリサイズ(16CPU:12GB、64CPU：38GB）から、30km 四方のシミュレーションを行うときのメモリサイズの試算を行った。並列プログラムは、分割されるデータと分割されないデータに分けられる。並列プログラムが消費する全体のメモリサイズを考えると、分割されないデータは、MPI プロセス数（＝CPU 数、以下プロセス数と略記）に比例して大きくなり、分割されるデータは袖領域を考えなければ、プロセス数で割られてプロセス数倍されるので、プログラム全体では一定となる。それぞれ、比例部分（β_F）と固定部分（B_C）と呼ぶことにして、消費されるメモリサイズの概算を行った。

$$\begin{aligned}\beta_F + 16B_C &= 12\\ \beta_F + 64B_C &= 38\end{aligned} \qquad (221)$$

より、β_F=3.33 (GB)，B_C= 0.54 (GB)が得られる。

解析領域が 1km 四方の S 倍になる場合、計算格子数に対応するメモリ量 m は以下のように表される。

$$m = \beta_F S + B_C N_P \qquad (222)$$

したがって 1 プロセス数当たりのメモリ量 m_r は以下のようになる。

$$m_r = \frac{m}{N_P} = \frac{\beta_F S}{N_P} + B_C \qquad (223)$$

「地球シミュレータ」の 1 プロセス当たりのメモリ量は 2GB であるので、

$$\frac{\beta_F S}{N_P} + B_C \le 2 \qquad (224)$$

次に 30km 四方の場合を考える。比例部分に分割領域依存性はないとして、単純に固定部分のみを S=900(=30*30)倍する。表１１はいくつかのノード数において 30km 四方の計算格子数で必要になるメモリ量を見積もったものである。「地球シミュレータ」の場合、1 ノード当たり、16GB 以下でなくてはいけない。すなわち、少なくとも 257 ノード以上が必要であることがわかる。

計算時間は概ね次のように見積もることができる。現在の計算プログラムを用いた場合、5km

表１０　並列性能

ケース	架空の測定ケース	ケース1	ケース2
CPU 台数	1	16	64
経過時間(sec)	2633.638（推定）	202.978	81.445
加速率	1	12.97	32.34
理想的な加速率	1	16	64
並列化効率	100.00%	81.10%	50.50%

表１１　メモリ使用量（30km 四方を想定）

node	128	256	384	450
CPU(=N_P)	1024	2048	3072	3600
メモリ量(GB)	3,555	4,109	4,664	4,950
単位メモリ量(GB/node)	27.8	16.1	12.1	11.0

四方の数値解析ではノード数８ノード、経過時間 9.360 ノード時間の計算機資源を要する。

30km 四方の数値解析を実施するにあたって経過時間を 36(=30/5*30/5)倍、更に流入風が通り抜ける回数が計算領域の相似比に比例することから 6(=30/5)倍することにより、9.360*36*6=2021.76 ノード時間の計算機資源を要する。これを先ほど推定したノード数 257 で割ると計算時間は 7.867 時間になる。

並列化効率 50%を超えるためには、257 ノードを使用する場合、1CPU 当たり 99.95%の並列化率が必要であることがわかる。これは、10,000 秒のプログラムを並列化する場合、シリアル実行が許されるのがわずか 4.9 秒だけと言うことである。

３．３　計算プログラムの診断

３．３．１　検討の視点

東京 23 区（30km 四方を想定）のシミュレーションで、実行時の性能を推定するために、以下の視点で対象問題を検討する。

A)30km 四方に領域サイズ拡大による実行性能への影響

B) ソルバとして ICCG(+BiCGSTAB)法が採用されているが、この部分を MultiGrid 法に置き換えた場合の実行性能への影響

３．３．２　CPU 台数の違いによる分析

図１２は、加速率の順でソートしたサブルーチン毎の 16CPU と 64CPU での最大経過時間を比較したものである。加速率は、(16CPU での最大経過時間）／（64CPU での最大経過時間）で求めた。x 軸のルーチン名の先頭に付加した番号は、Rank0 での経過時間順にソートした場合の順位である。理想の加速率は 4 倍であるため、加速率が 4 倍を越えるものはカットした。経過時間の大きなものは以下の通りである（全てのサブルーチンの定義等の記載は省略する)。

A) 01_k3d_iccg(ICCG 法), 03_k3d_bcgstb,04_k3d_imdqdt(BiCGSTAB 法)

B) 06_k3d_module_rsdprf.setrsdprf(residual の set)

C) 56_k3d_out_restart(I/O 処理)

D) 05_k3d_dsr_dc2(19_k3d_bc_symprv,29_k3d_bc_symprs,62_k3d_bc_symprc からの呼び出し)

E) 02_k3d_m_warsm(MPI_AllReduce の呼び出し)

F) 08_k3d_dsr_dc1(12_k3d_bcgstb_pre,03_k3d_bcgstb,01_k3d_iccg からの呼び

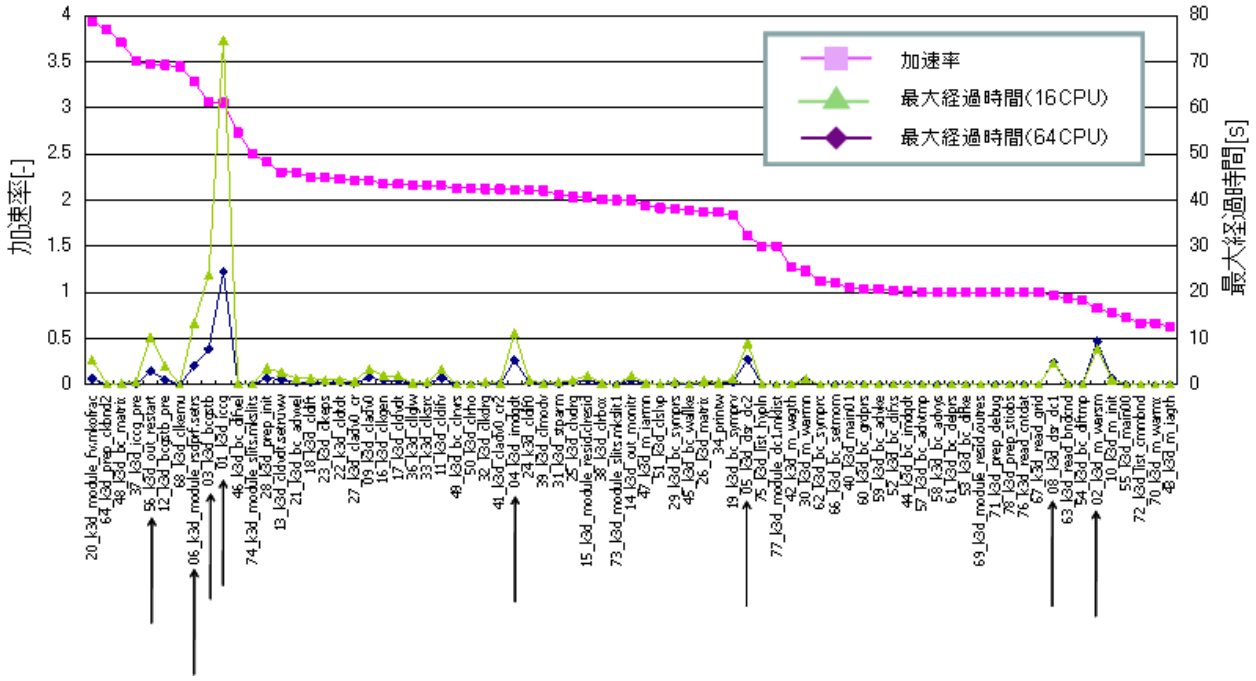

図１２　加速率と最大経過時間
（最大経過時間が大きなものを矢印で表示）

出し）

図１３は、加速率の順でソートしたサブルーチン毎の 16CPU と 64CPU での平均ベクトル長を比較したものである。CPU の増加に伴い、平均ベクトル長が短くなる傾向が見られる。平均ベクトル長は最内側ループの長さに依存しているため、1CPU が担当する領域を分割すると一般に短くなる。さきほど挙げた 01_k3d_iccg、03_k3d_bcgstb、12_k3d_bcgstb_pre、06_k3d_module_rsdprf.setrsdprf、04_k3d_imdqdt および 05_k3d_dsr_dc2 がこのケースに相当する。これらの平均ベクトル長の低下は、1km 四方の領域を 16CPU と 64CPU で分割したためと考えられる。

コストの高い 01_k3d_iccg や 03_k3d_bcgstb、12_k3d_bcgstb_pre 等のサブルーチンは、担当する領域の面積(mx×my)に比例するループを持つ。例えば、

【16CPU】

50*50=2,500［1CPU 当りの計算担当領域］

2,500/256=9.77［ベクトルループ実行回数］

2,500/10=250［平均ベクトル長］

【64CPU】

25*25=625［1CPU 当りの計算担当領域］

625/256=2.44［ベクトルループ実行回数］

625/3=208［平均ベクトル長］

となり、ほぼ実測と等しい値が得られる。領域を分割して処理量は削減されたが、平均ベクトル長が短くなることで加速率の低下を招くことがわかる。

平均ベクトル長が全く変化しない事例として、例えば 1CPU 150 メッシュのケースを計算して見ると、

150*150=22,500［1CPU 当りの計算担当領域］

22,500/256=87.89［ベクトルループ実行回数］

22,500/88=255.68［平均ベクトル長］

その 4 倍の 1CPU 300 メッシュの領域を計算した場合には、

300*300=90,000［1CPU 当りの計算担当領域］

90,000/256=351.56［ベクトルループ実行回数］

90,000/352=255.68［平均ベクトル長］

となる。MultiGrid 法においても同様の効果を

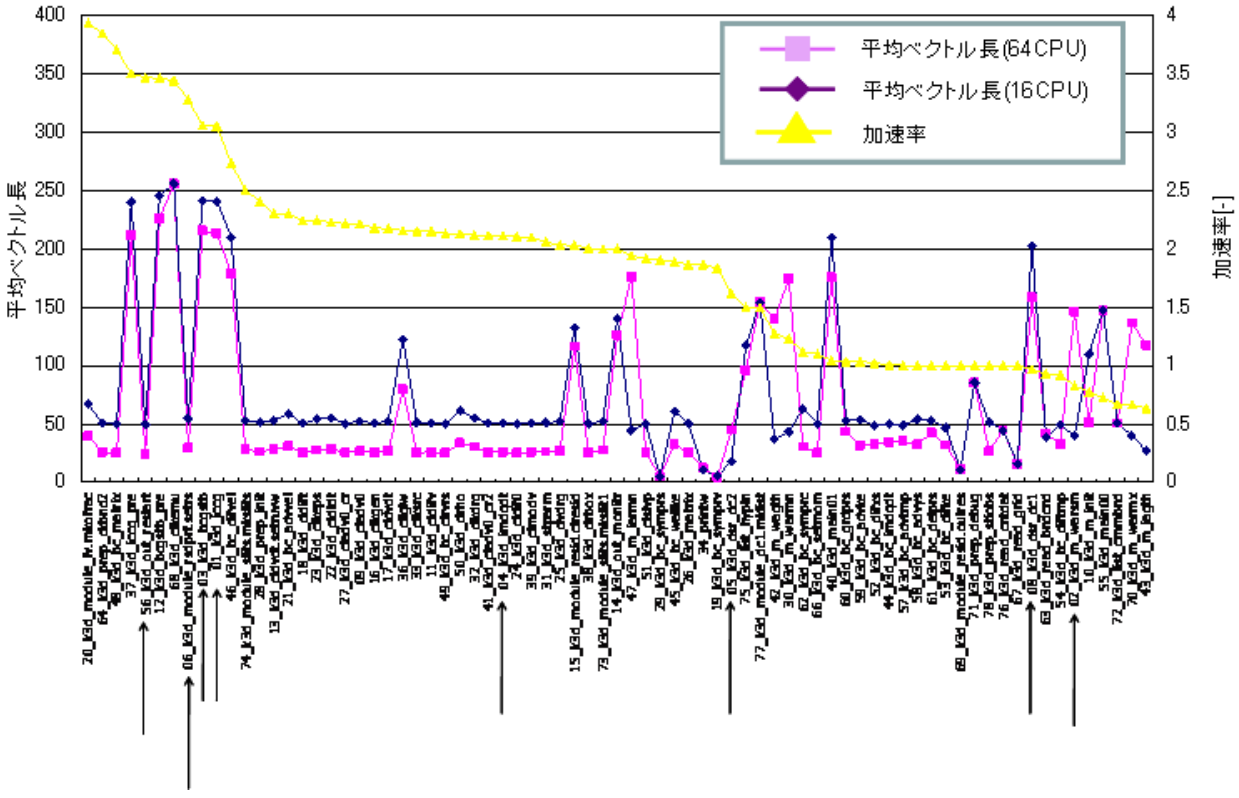

図１３　加速率と平均ベクトル長
（最大経過時間が大きなものを矢印で表示）

期待できると考えられる。

平均ベクトル長が半減しているルーチン群については以下のように考える。分割数を 16 から 64 に上げることにより、50 のループ長が 25 に減少している。しかし、30km 四方を 1CPU 当たり 150 メッシュで行う場合、平均ベクトル長は 150 と推定される。4 倍の領域(300 メッシュ)を 1CPU で担当する場合は、300/2=150 より平均ベクトル長は変化しない。したがって、今回のような加速率の低下は起こらないと予想される。

３．３．３　分割台数毎の分析

ばらつきが大きいルーチンは並列化効率を下げてしまうので原因の究明を行う必要がある。経過時間にばらつきが大きいルーチンの洗い出しを行った。図１４は、16CPU 実行による経過時間の最大値から最小値を引いた差の順に主なルーチンをソートしたものである。差が

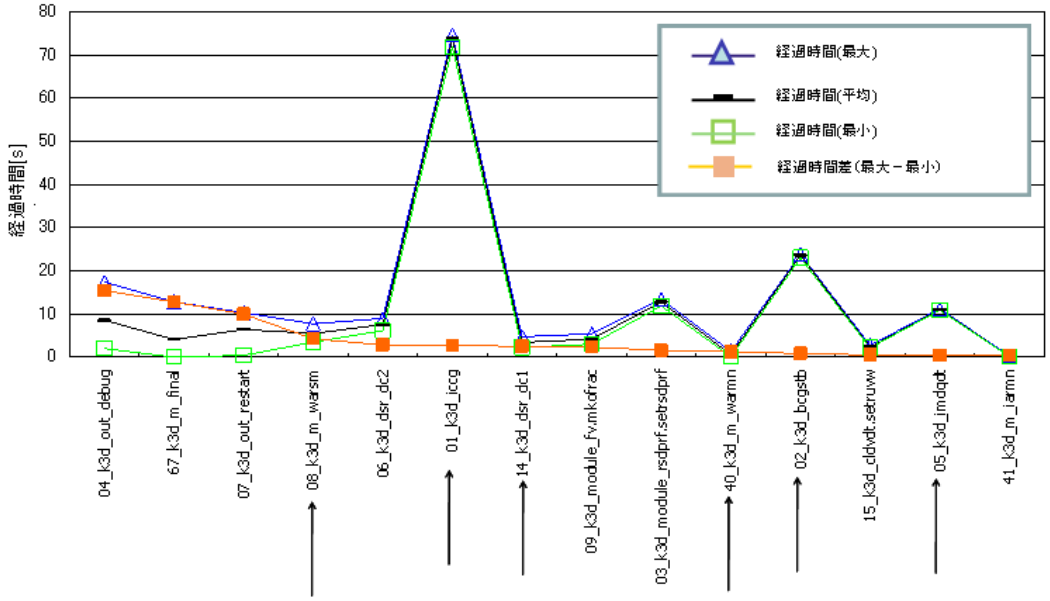

図１４　ロードインバランス(16CPU)
(ICCG(+BiCGSTAB)関連を矢印で表示)

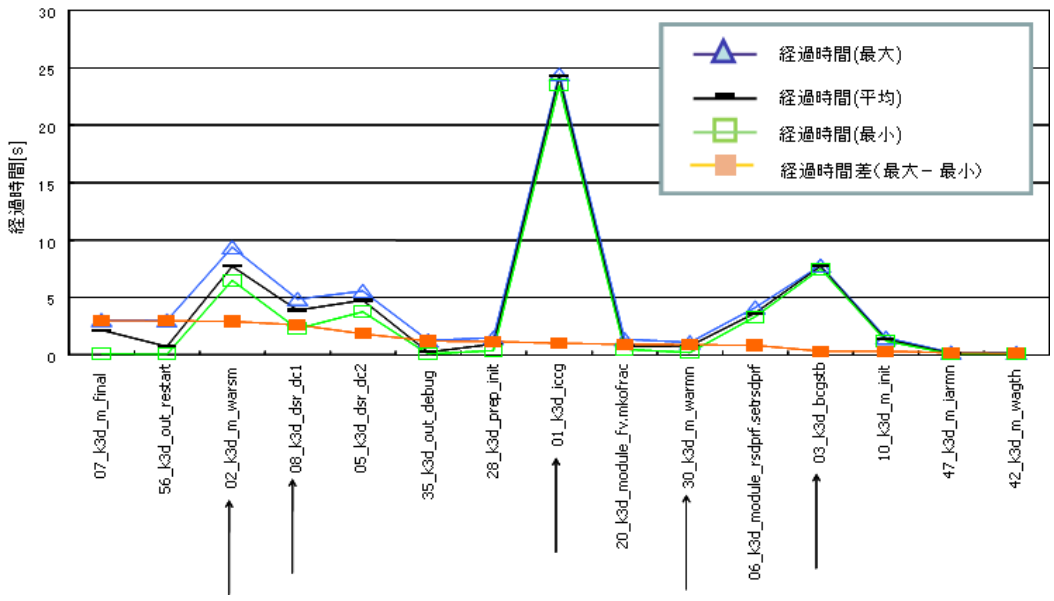

図１５　ロードインバランス(64CPU)
(ICCG(+BiCGSTAB)関連を矢印で表示)

大きい最初の3つのサブルーチンは、I/O処理(04_k3d_out_debut,07_k3d_out_restart)とその待ち合わせを含む終了処理(67_k3d_m_final)であるが、性能測定時にはI/Oを止めて評価すれば問題にはならない。矢印で示すサブルーチン群(08_k3d_m_warsm、01_k3d_iccg、14_k3d_dsr_dc1、40_k3d_m_warmn、02_k3d_bcgstb、05_k3d_imdqdt)は、ICCG(+BiCGSATB)法関連のサブルーチンであり、これらは、MultiGrid法を選択した場合、ほぼ置き換わるものの、汎用的なMPI_AllReduce系ルーチンの08_k3d_m_warsm、40_k3d_m_warmnと1対1通信isend,irecvを行っている14_k3d_dsr_dc1は残る。09_k3d_module_fv.mkofracは初期処理であるため、今は気にする必要がない。06_k3d_dsr_dc2、41_k3d_m_iarmnが境界処理から呼ばれる通信処理ルーチンである。境界処理(bcルーチン)にインバランスが見られる。同様に64CPUのロードインバランスを図１５に示す。

３．４　最適化の作業内容

３．４．１　ベクトル化

プログラムのベクトル化を主にICCG法を用いた行列ソルバーに対して実施する。ベクトル化要領を以下に示す。

（１）不完全Cholesky分解行列の反転処理

一般にLU分解行列の反転処理はハイパープレーン法によりベクトル化されるが、構造格子の場合でも間接アドレス参照になる上、ループ長を十分に取れないためベクトル演算の効率はあまり高くなく、最適化前のプログラムではこの前処理にあたる反転処理を削除していた。しかし、大規模な連立1次方程式を解く上では前処理を削除したCG法の反復計算回数が大幅に増大し、ベクトル演算による高速化ではカバーし切れないほどに計算効率が悪くなったため、前処理の導入を実施した。

（２）間接アドレスの廃止

領域分割による並列化を行う上で良いロードバランスを保つために、間接アドレス処理を用いて各プロセスの演算処理が同じによるようにしていた。ベクトル化は行われるものの、間接アドレスの参照による実効性能の低下を招いていたため、間接アドレスを廃止した。

３．４．２　並列化

プログラムの並列化を領域分割法により行った。x、y、z軸方向に任意の分割数で領域分割することができるようになっている。N_x、N_y、N_z（N_x、N_y、N_zは整数)をそれぞれx、y、z軸方向の分割数とすると、全体の領域数は$N_p = N_x \times N_y \times N_z$で、$N_p$は同時にプロセス数となる(図１６)。

数値解法として差分法を使用していることにより、分割領域の境界付近では隣接領域のデータが必要となる。その部分は双方のプロセス領域でデータを重複して保持することになる。その境界付近のデータを保持する部分を「仮想領域」と呼ぶ(図１７)。この仮想領域に隣接領域の数値を通信により保持し、差分計算を行うようにすることが並列化の前提となる。

その他、次のような並列化作業を実施する。

- 1領域1プロセッサで並列化する。プログラムは、SPMD(Single Program Multi Data)であり、使用される全てのプロセッサで、同じ実行体がロードされ、取り扱う値はプロセス毎に異なる。
- MPIコマンドで指定する実行プロセス数は

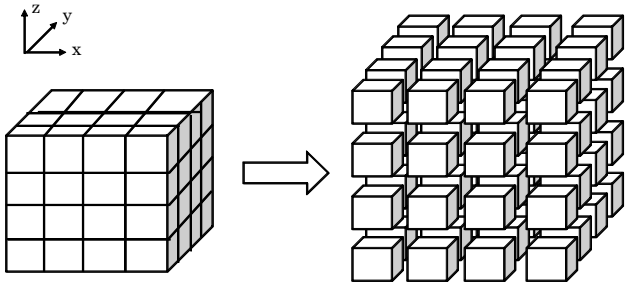

図１６　領域分割法

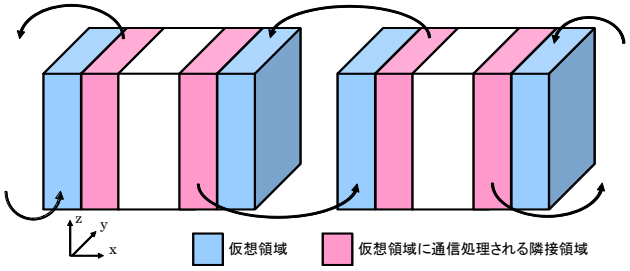

図１７　領域分割法における通信処理

領域分割の総数 N_p と同数でなければならない。一致していない場合はエラーとして実行を停止する。

・ 入力および出力については、分散するものとしないものが存在する。基本的に有次元で配列を持つデータの入出力は分散し、物理定数や計算制御データなど次元のない値は非分散としている。

３．４．３　メモリの削減

所要メモリを削減することはノード数の低減につながり、必要とされる並列化率が緩和されることになる。使用メモリ量を以下のように約 3 割削減した。

従来版：2,306MB（最大）、180GB（10 ノードのトータル）

削減版：1,656MB（最大）、129GB（10 ノードのトータル）

プログラムのメモリを削減するために次の作業を実施した。

(a)計算途中で一時的に確保される一時配列の縮小、削減

(b)恒久配列のうち、再計算することで一時配列に変更する。

```
call sub1(work01)
→sub01でwork01に風速uの2乗を定義する。
call sub2(work01)
→sub02でwork01を風速uの2乗として利用する。
call sub3(work01)
→sub03でwork01に風速vの2乗を定義する。
call sub4(work01)
→sub04でwork01を風速vの2乗として利用する。
```

図１８　配列の「使い廻し」の例

```
Allocate(uu(mx,my,mz))
call sub1(uu)
→sub01でuuに風速uの２乗を定義する。
call sub2(uu)
→sub02でuuを風速uの２乗として利用する。
deallocate(uu)
Allocate(vv(mx,my,mz))
call sub3(vv)
→sub03でvvに風速vの２乗を定義する。
call sub4(vv)
→sub04でvvを風速vの２乗として利用する。
deallocate(vv)
```

図１９　動的割付機能を利用する例

運動方程式の計算ルーチン k3d_cldvdt 以下では多くの一時配列を利用しており、このルーチンを中心に一時配列を削減した。一時配列を削減する場合、図１８に示すような同じ配列を異なる用途で使用する、いわゆる「使い廻し」が行われることがある。ただし、このような処理の場合、一時配列の利用範囲が不明瞭になり、第3者がプログラムを読みにくくなるばかりでなく、再度プログラムを変更する場合にバグを作り易くなってしまう。

Fortran90 の配列機能を利用することで「配列の使い廻し」は回避することができる。割付配列の機能を利用すれば図１８の例は図１９のようにコーディングすることが可能である。

この方法はプログラムの変更も少なく、したがって元々のプログラムの「筋」を壊さないため安全、確実な方法と言える。ただし、動的割付配列の場合、メモリ管理のための余計なオーバーヘッドがかかるため、演算量に比べて allocate､deallocate の頻度が高くなってくると計算効率に影響が出る場合もあると考えられる。

３．４．４　ICCG(+BiCGSTAB)法の MultiGrid 法への置き換え

係数行列から代数的に、より粗い計算格子における方程式を導出できる代数的多重格子法（AMG 法）は、計算格子のトポロジーに依存しないため、汎用的な実装が可能な計算手法である。本報告で使用した AMG 法の特徴は以下の通りである。

(a)CG 法の前処理に AMG 法を適用する

(b)多重格子法：V サイクル

(c)緩和法：不完全 Cholesky 分解

(d)並列化：局所前処理行列

(e)ベクトル化:Hyper plane 法

(f)各格子レベルでの反復法:最密=1、中間=5、最粗:10

実際の計算プログラムに AMG 法を組み込み、CFD 解析を実施することにより、導入効果を確認した。計算条件を以下に示す。

・計算領域：5km(x)×5km(y)×500m(z)

・領域分割：5×16=80

・格子点間隔：dx=dy=5[m]

・計算格子数：1000(x)×1000(y)×100(z)
=100,000,000 点

・測定ケース：AMGCG 法利用と AMG 法利用の 2 ケース

AMGCG 法利用と AMG 法利用による計算結果を比較したのが表１２である。AMGCG 法の適用により反復回数は 74 分の 1 程度に縮減される。AMGCG 法の 1 反復当たりの演算量は ICCG 法に比べて約 3 倍であるため、実効の加速率は約 20 倍程度である。

なお、ICCG(+BiCGSTAB)法のルーチンはコストの 3 割弱を占めるが、MultiGrid 法に置き換わった場合にコスト分布が大きく変わる。これにより境界処理や集団通信系のチューニングが必要になる。双方共に取りうる手段があるので、できるだけ本番データに近いデータによる性能測定を行うことで対策をより具体化する必要がある。

表１２　AMGCG 法と ICCG 法の比較

解法	反復回数	計算時間（s)
AMGCG 法	44(　1)	5.5(　1)
ICCG 法	3,240(73.6)	110.3(20.1)

CG 法の収束判定は $\|r\|_\infty < 10^{-12}\|b\|_\infty$ とした。

AMGCG 法は 10 タイムステップの平均値、AMG 法は 3 タイムステップの平均値である。

３．４．５　インバランスの是正

各サブルーチンのトレース情報を取得し、バランスが悪化しているサブルーチンに対して、最適化処理を実施した。表１３にインバランスの是正事例を示す。この作業に伴い、並列化率は表１４のように改善している（5km 四方、40PE+80PE、400STEP)。

３．５　最適化の行程

３．５．１　最適化の行程

計算プログラムに温位対応等のモデル拡張、AMG ソルバーの組み込み、メモリ削減等を実施する。この計算プログラムに対してノード数の拡張を可能とするため、インバランスの是正等

表１３　インバランス状況のサブルーチン比較

No.	subroutine name	tune	処理前				処理後			
			40PE T1(秒)	80PE T2(秒)	ratio T1/T2	imbalance T2-T1/2(秒)	40PE T1(秒)	80PE T2(秒)	ratio T1/T2	imbalance T2-T1/2(秒)
1	k3d_bcgstb		2935	1544	1.9	76.5	2937	1545	1.9	76.5
2	amg_forwd		553	273	2.03	-3.5	554	273	2.03	-4
3	amg_bakwd		464	228	2.04	-4	465	228	2.04	-4.5
4	amg_smooth		314	150	2.09	-7	314	150	2.09	-7
5	k3d_bc_imdqdt	*1	282	189	1.49	48	77	51	1.51	12.5
6	k3d_imdqdt_makcoef		261	133	1.96	2.5	261	133	1.96	2.5
7	amg_rest		223	104	2.14	-7.5	223	104	2.14	-7.5
8	k3d_bcgstb_pre		209	107	1.95	2.5	209	107	1.95	2.5
9	amg_cg		180	99	1.82	9	178	100	1.78	11
10	amg_ax		166	79	2.1	-4	142	79	1.8	8
11	k3d_bc_advtmp	*1	151	101	1.5	25.5	54	35	1.54	8
12	amg_prol		142	66	2.15	-5	142	66	2.15	-5
13	k3d_cladv0		122	62	1.97	1	122	62	1.97	1
14	k3d_bc_difxs	*1	120	82	1.46	22	46	31	1.48	8
15	k3d_bc_advvel	*1	105	64	1.64	11.5	89	53	1.68	8.5
16	k3d_bc_difke	*1	96	65	1.48	17	36	25	1.44	7
17	k3d_cldifv		98	50	1.96	1	98	50	1.96	1
18	k3d_bc_diftmp	*1	89	60	1.48	15.5	32	21	1.52	5
19	k3d_dsr_dc2		86	52	1.65	9	94	53	1.77	6
20	k3d_bc_grdprs	*1	80	53	1.51	13	22	15	1.47	4
21	k3d_bc_advke	*1	79	53	1.49	13.5	29	19	1.53	4.5
22	k3d_bc_delprs	*1	74	50	1.48	13	19	13	1.46	3.5
23	amg_cedger		73	65	1.12	28.5	71	68	1.04	32.5
24	k3d_clkgen		65	33	1.97	0.5	65	33	1.97	0.5
25	k3d_cldvdt_setruvw		63	32	1.97	0.5	63	32	1.97	0.5
26	k3d_m_warsm		62	80	0.78	49	59	81	0.73	51.5
27	k3d_bc_difvel	*1	56	36	1.56	8	54	33	1.64	6
28	k3d_bc_advys	*1	55	36	1.53	8.5	31	20	1.55	4.5
29	k3d_out_debug	*2	43	23	1.87	1.5	0	0	-	0
30	k3d_cldift		41	21	1.95	0.5	41	21	1.95	0.5
31	k3d_module_exner.calexnr		39	22	1.77	2.5	39	22	1.77	2.5
32	k3d_cldtdt		36	19	1.89	1	36	19	1.89	1
33	k3d_bc_freeke	*1	36	23	1.57	5	22	14	1.57	3
34	k3d_module_resid.clresid		35	18	1.94	0.5	35	18	1.94	0.5
35	k3d_bc_wallke		31	21	1.48	5.5	32	21	1.52	5
36	k3d_clkeps		31	16	1.94	0.5	31	16	1.94	0.5
37	k3d_cllglw		31	14	2.21	-1.5	31	14	2.21	-1.5
38	k3d_bc_rstexnr	*1	29	20	1.45	5.5	15	10	1.5	2.5
小計						365.5				251

*1：指示行挿入、*2:ソース修正

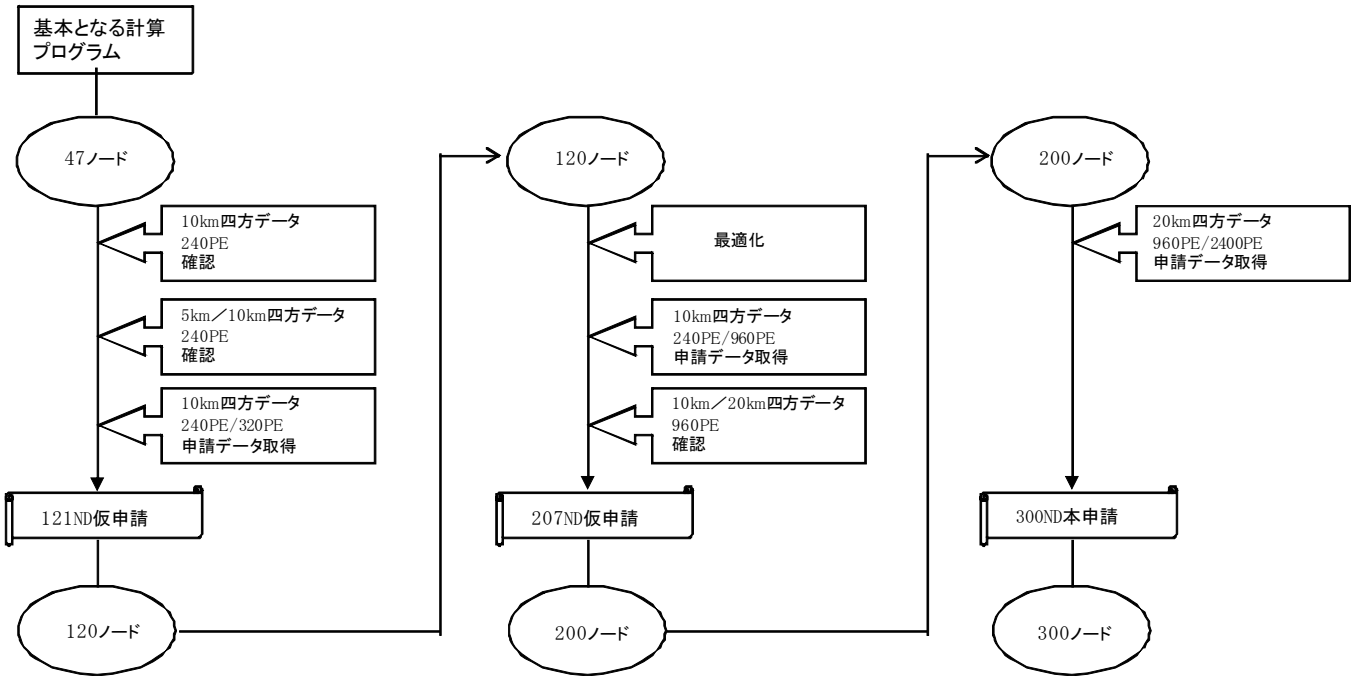

図２０　最適化の行程

の最適化処理を行いながら図２０に示す行程で作業を実施した。

３．５．２　最終的な最適化状況

200 ノードクラスの最適化を施した後、20km 四方データを利用して並列性能を調査した。その結果、表１４に示す通り並列化効率が 300 ノードを利用しても 50%を超えていることから、300 ノードクラスの最適化が達成された。

３．５．３　高分割時の主要サブルーチンの状況

120ノード利用時と300ノード利用時における主要サブルーチンの状況(Rank0)を表１５に示す。高分割時でも平均ベクトル長の劣化が比較的小さいことと、通信ルーチンが含まれないために960PE／2400PE=2.388倍と効率的に実行されている。なお、並列化効率100%では960PE／2400PE=2.5倍、並列化効率50%では1.75倍である。

その他の高コストを占めるサブルーチンについて表１６に一覧する。これらのサブルーチン群には実行効率の悪いものも含まれるが、最大コストルーチンの全体に占める割合が大きいため、他ルーチンの性能をカバーしているものと考えられる。

３．５．４　２３区データの実行可能性

表１４　並列性能

20km四方・800STEP	960 PE（120ND）	2400 PE（300ND）
経過時間	13039.7 秒	6297.1 秒
ベクトル化率	99.13%	98.97%
平均ベクトル長	241.2	235.3
GFLOPS(ピーク比)	1965.4(25.5%)	4209.3(21.9%)
並列化率(%)	99.983	
並列化効率(%)	71.394	

表１５　最大コスト(約 40%)を占める k3d_bcgstb の状況　(Rank0)

項目	960 PE（120ND）	2400 PE（300ND）	RATIO
経過時間	5218.7 秒	2185.0 秒	2.388
ベクトル化率	99.79%	99.78%	−
平均ベクトル長	252.5	247.0	0.978
MFLOPS	2692.5	2620.4	0.973

表１６　高コストを占めるサブルーチン（k3d_bcgstb 以外）

サブルーチン	960 PE（120ND）	2400 PE（300ND）	RATIO
amg_forwd	771.5 秒	352.2 秒	2.191
k3d_m_warsm	680.2 秒	504.2 秒	1.349
amg_bakwd	645.7 秒	293.5 秒	2.200
amg_smooth	447.3 秒	204.4 秒	2.189
amg_cg	429.6 秒	330.9 秒	1.298

20km四方解析の実行性能情報から120ND実行時の1プロセスのメモリ所要量は、1,744MBであった。よって、20km四方データの全体のメモリ所要量は、1,674.24GB（＝1.744(GB)×120(ND)×８(PE/ND)）である。

ここで、所要メモリ量がデータの面積に比例すると考えた場合、30km四方データのメモリ所要量は、袖部分を1割含むとき、3,767GB（＝1,674.24(GB)×（33×33／22／22））となる。また、23区全域（33km四方）データのメモリ所要量は、袖部分を1割含むとき、4,483GB（＝1,674.24(GB)×（36×36／22／22））と見積もられる。

300ノードを利用した場合の利用可能総メモリ量は、4,800 GB（＝300(ND)×16(GB/ND)）であるから、最適化の条件として300ノードを達成しておけば、23区全域データを実行可能になると考えられる。

３．５．５　300ノード申請時の情報

300ノード申請時の情報を図２１、図２２に示す。

```
MPI Program Information:
========================
Note: It is measured from MPI_Init till MPI_Finalize.
      [U,R] specifies the Universe and the Process Rank in the Universe.

Global Data of 960 processes:          Min [U,R]             Max [U,R]           Average
============================

Real   Time (sec)           :      13039.042 [0,279]      13039.690 [0,912]      13039.505
User   Time (sec)           :      12832.937 [0,667]      12971.455 [0,296]      12937.662
System Time (sec)           :          3.043 [0,328]          8.666 [0,0]            4.919
Vector Time (sec)           :      11083.217 [0,380]      12052.044 [0,888]      11587.315
Instruction Count           :  1095334992582 [0,928]  1321892000237 [0,940]  1139780911644
Vector Instruction Count    :   339783061785 [0,382]   384859803728 [0,896]   367759549610
Vector Element Count        : 81699498270562 [0,382] 93025183622168 [0,896] 88721847894711
FLOP Count                  : 24995212643261 [0,382] 27295302077202 [0,438] 26487528790375
MOPS                        :       6380.423 [0,382]       7244.010 [0,96]        6917.281
MFLOPS                      :       1932.227 [0,360]       2113.605 [0,579]       2047.314
Average Vector Length       :        239.767 [0,465]        243.045 [0,739]        241.234
Vector Operation Ratio (%)  :         98.868 [0,940]         99.238 [0,136]         99.135
Memory size used (MB)       :       2238.126 [0,0]         2239.141 [0,34]        2238.900
MIPS                        :         84.660 [0,928]        102.059 [0,940]         88.098
Instruction Cache miss (sec):         27.463 [0,19]          43.417 [0,144]         37.440
Operand     Cache miss (sec):         62.750 [0,698]        108.765 [0,959]         71.804
Bank     Conflict Time (sec):         78.771 [0,620]        117.148 [0,192]        100.014


Overall Data:
=============

Real   Time (sec)           :    13039.690
User   Time (sec)           : 12420155.665
System Time (sec)           :     4722.610
Vector Time (sec)           : 11123822.321
GOPS   (rel. to User Time)  :     6640.621
GFLOPS (rel. to User Time)  :     1965.427
Memory size used (GB)       :     2098.968
```

図２１　300ノード利用申請（960プロセス）

```
MPI Program Information:
========================
Note: It is measured from MPI_Init till MPI_Finalize.
      [U,R] specifies the Universe and the Process Rank in the Universe.

Global Data of 2400 processes:          Min [U,R]             Max [U,R]         Average
==============================

Real   Time (sec)           :       6296.231 [0,31]        6297.121 [0,2200]        6296.804
User   Time (sec)           :       6228.745 [0,1822]      6278.447 [0,87]          6257.484
System Time (sec)           :          1.603 [0,842]          5.941 [0,0]              2.405
Vector Time (sec)           :       5181.171 [0,2380]      5626.437 [0,576]         5475.353
Instruction Count           :   529491568085 [0,2328]  712760511538 [0,19]      554210437031
Vector Instruction Count    :   150191332005 [0,900]   167388737985 [0,176]     160991911922
Vector Element Count        : 35110426322510 [0,982]  39571801122450 [0,176]  37892030039814
FLOP Count                  : 10356147043641 [0,982]  11306991699979 [0,19]   10974941300868
MOPS                        :       5676.561 [0,982]       6382.276 [0,136]         6118.299
MFLOPS                      :       1653.842 [0,920]       1812.617 [0,1817]        1753.888
Average Vector Length       :        232.591 [0,985]        237.701 [0,1000]         235.346
Vector Operation Ratio (%)  :         98.486 [0,3]           99.085 [0,2256]          98.971
Memory size used (MB)       :       1259.720 [0,0]         1260.641 [0,1448]        1260.454
MIPS                        :         84.569 [0,2328]       113.768 [0,19]           88.568
Instruction Cache miss (sec):         30.199 [0,19]          47.216 [0,2192]          40.272
Operand     Cache miss (sec):         49.266 [0,447]        118.903 [0,2396]          61.513
Bank     Conflict Time (sec):         42.190 [0,1540]       118.182 [0,2034]          65.747


Overall Data:
=============

Real   Time (sec)           :     6297.121
User   Time (sec)           : 15017961.453
System Time (sec)           :     5771.110
Vector Time (sec)           : 13140847.184
GOPS   (rel. to User Time)  :    14683.952
GFLOPS (rel. to User Time)  :     4209.337
Memory size used (GB)       :     2954.189
```

図２２　300ノード利用申請（2,400プロセス）