重粒子線がん治療用線量計算エンジンの自動並列化

林　明宏[†1]　松本卓司[†1]　見神広紀[†1]
木村啓二[†1]　山本啓二[†2]　崎　浩典[†2]
高谷保行[†3]　笠原博徳[†1]

粒子線によるがん治療は臨床レベルで実用化されており，外科的侵襲を伴わず患者への負担が少なく，また X 線放射線治療の様に皮膚からがん患部までの正常細胞に損傷を負わせることもなくその高い治癒率から注目を集めている．治療にあたっては医師が事前に計算機を使用してがん細胞にのみ照射するための照射計画のシミューレーションを行うが，標的に必要な投与線量が集中するように各種機器の設定条件を調整するには，シミュレーションを繰り返して評価することが必要であり時間が非常にかかるなどの問題点があった．本論文では，この治療計画を高速に行う重粒子線治療用線量計算エンジンの並列化手法を提案する．具体的には逐次プログラムをコンパイラで並列化しやすい Parallelizable C によって記述された計算エンジン本体を開発することにより OSCAR 自動並列化コンパイラにより自動並列化を行う．これにより一度だけの書き換えで任意の SMP サーバーで任意プロセッサ数に対応できるようにした．その結果 IBM Power 7 プロセッサを搭載した日立 SR16000 SMP サーバー上において．64CPU 使用時に約 50 倍，そして Intel Xeon X5670 プロセッサを搭載した日立 HA8000/RS220 SMP サーバー上において，12CPU 使用時に約 9 倍の性能向上を実現し，提案手法が高いスケーラビリティを実現可能であることを確認した．

Automatic Parallelization of Dose Calculation Engine for A Particle Therapy

Akihiro Hayashi,[†1] Takuji Matsumoto,[†1] Hiroki Mikami,[†1]
Keiji Kimura,[†1] Keiji Yamamoto,[†2] Hironori Saki,[†2]
Yasuyuki Takatani[†3] and Hironori Kasahara [†1]

Particle radiotherapy has been attracted much attention since it is very effective to treat cancer. However, it takes a long time to simulate the dose calculation before the treatment. It is essential to gain the performance of a treatment simulation by using multicore processors. In this paper, we realize an automatic parallelization of the dose calculation engine for particle radiotherapy. This dose calculation engine is based on the clinically used program developed by National Institute of Radiological Sciences (NIRS) and Mitsubishi Electronics. This paper proposes a processing scheme for the application and enables OSCAR compiler to exploit the parallelism of the dose calculation engine. As a result, the proposed method attains good speedups of 9.0 times with 12 CPUs on Hitachi HA8000/RS220 system based on the Intel Xeon Processor and 50.0 times with the 64 CPUs on Hitachi SR16000 system based on the IBM Power 7 processor.

1. はじめに

がんは日本人の死因のうち最も多く，厚生労働省の調査によると 3 人に 1 人はがんにより命を落としている[1)]．近年，がんの放射線治療法の 1 つとして，重粒子線を用いた治療が臨床レベルで高い治療効果を実現しており広く注目を集めている．それは，重粒子線が従来の X 線による放射線治療に比べがん細胞への影響力が高く，かつ正常細胞への影響が少ないためである．重粒子線において国内では，独立行政法人放射線医学総合研究所を中心として実際に治療が行われている．

重粒子線治療を実施する際には，事前に計算機を用いた線量分布の予想が必要であり，患部に対して重粒子線をどのような条件で照射すると，がん細胞に対し

†1 早稲田大学
WASEDA UNIVERSITY
†2 三菱スペース・ソフトウエア
MITSUBISHI SPACE SOFTWARE CO., LTD.
†3 三菱電機
MITSUBISHI ELECTRIC CORPORATION

てどれくらいの線量を与えることになるのかをソフトウェアを用いて検証している[2]．このようなソフトウェアの中核部分は線量計算エンジンと呼ばれており，線量値の計算に 3 次元の物理シミュレーションを実行している．粒子線治療では，治療前に治療計画を策定する中で，線量計算を行い，得られた線量分布を確認しながら，シミュレーションを繰り返すことが必要である．しかしながらこのような計算には非常に時間がかかり，きめ細かい治療実現の為に計算精度を上げると，患者一人あたりのシミュレーションに膨大な時間がかかってしまうという欠点がある．

一方，近年メニーコア及びマルチコアは爆発的に普及しており，エンドユーザ向けの携帯電話，スマートフォンといった組込み機器用途[3][4][5][6] から科学技術計算に使用するスーパーコンピュータ用途[7][8] まで広く利用されている．このような背景から，マルチコア上で線量計算エンジンを並列処理により高速実行することで治療計画を立てるオペレータの負荷の減少が見込まれる．

しかしながら，広く知られているようにマルチコア上で高いスケーラビリティを実現することは大変な苦労が伴う．並列化する際にはプログラム中で並列実行可能なタスクの抽出や，そのスケジューリング，及びバリア同期等のコア間通信コードの挿入に関してプログラマが熟考する必要があるためである．本計算エンジンに関していえば先行研究[2] は OpenMP を用いた手動並列化を試みている．しかしながら，critical による lock を行なっているため，2CPU，4CPU を用いた場合でそれぞれ 1.5 倍，2.3 倍と使用する CPU 数が増大にするにつれ性能が鈍化し，8CPU を用いた場合，逐次実行時に比べ 2.8 倍の速度向上にとどまっている．このような問題に対処し，CPU 数に依存せずに高いスケーラビリティを実現するため，我々は従来より OSCAR 並列化コンパイラを開発している[9]．

OSCAR コンパイラは C 言語のポインタ利用に制限を加えた Parallelizable C や Fortran 77 言語で記述された逐次ソースプログラムを入力とし，並列プログラムを自動生成する自動並列化コンパイラである．並列化プログラムの生成は source-to-source で行うため，OSCAR コンパイラの生成したコードを各プロセッサのネイティブコンパイラでコンパイルすることにより，様々なプラットフォームで自動並列化が実現可能である．また，OSCAR コンパイラではループイタレーションレベルの並列処理を行うのみでなく，ループ・手続き間の粗粒度タスク並列処理[10]，ステートメント間の近細粒度並列処理を組み合わせたマルチグレイン並列処理[11]，メモリウォール問題に対処するための複数ループにわたるキャッシュあるいはローカルメモリの最適利用[12] が実現されている．さらに，プログラム中の各並列処理部に対する適切なリソース割り当てや，各リソースの周波数・電圧・電源制御による消費電力の自動削減[13] が実現されている．そして，OSCAR コンパイラには種々のポインタ解析ルーチンが実装されており，C 言語のポインタを利用したプログラムの並列化も可能である．しかし，C 言語のあらゆる仕様を網羅するポインタ解析の実装は事実上不可能であり，自動並列化コンパイラを現実的に利用していく上では，解析可能な C プログラムの記述を明確化することが重要である．そのため，ポインタ解析器を実装したコンパイラによる自動並列化が可能な基準として，Parallelizable C (level2) を C 言語のサブセットとして定義している[14]

本論文では臨床に使用されている重粒子線治療用線量計算エンジンという実プログラムに対して，計算精度の向上及び実行時間の短縮を目指し，OSCAR コンパイラによって並列性を抽出しやすい逐次 C プログラムを開発し，任意の SMP サーバーにおいて任意のプロセッサ台数で自動並列化できる方式の確立を目指す．

以下 2 章で重粒子線がん治療線量計算エンジンの概要，3 章で線量計算エンジンプログラムのチューニングについて 4 章で SMP サーバーにおける評価結果について述べる．

2. 重粒子線がん治療線量計算エンジン概要

本章では線量計算エンジンの概要及び，実行プロファイル結果について述べる．本線量計算エンジンは放射線医学総合研究所・三菱電機によって開発されたものである[15],[16]．本アルゴリズムはペンシルビーム法という線量計算アルゴリズムをベースに散乱現象など実際に起こりうる物理現象を模擬する様々な計算モデルを追加することで，高精度な線量計算を実現している．

本エンジンは，3 次元空間上の各点をボクセルで表現し，線量を計算する．入力は 3 次元 CT 画像や病巣の輪郭情報，出力は各ボクセルでの線量である．計算精度はボクセルの一辺のサイズに依存しており，そのサイズを小さくすることで，より精度の高いシミュレーションが可能となる．元のプログラムは C++言語で記述されていたが，前述の Parallelizable C (level2) に書き換えられている．計算の流れを以下に示す．

step 1: データ構造の初期化

ボクセルの配列を確保し，全てのボクセルの線量

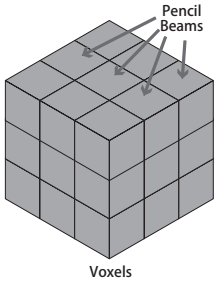

図 1 ペンシルビーム法を用いた線量計算イメージ
Fig. 1 Dose Calculation using pencil beam algorithm

値を 0 で初期化する．

step 2: ペンシルビーム法を用いた線量計算

図 1 に線量計算のイメージを示す．線量計算においては，照射する粒子ビーム全体をペンシルビームと呼ばれる局所的なビームへの集合体として表現している．図 1 においては，Voxels が 3 次元のボクセル配列を表し，Pencil Beams はペンシルビームを表しており，各ペンシルビームが通過するボクセルに与える線量を物理計算している事を示している．以下が線量計算部分のコードイメージであり，各ペンシルビームが通過するボクセルに対して dcalc 関数で計算した線量値を配列 Voxel1 に足しこんでいる様子を示している．

```
for (ペンシルビーム数) {
    for (ペンシルビーム通過ボクセル) {
      // 線量値の計算
      線量 = dcalc();
      Voxel1[xyz] += 線量
    }
}
```

step 3: 散乱計算

図 2 に散乱計算のイメージを示す．散乱計算においては，step 2 で計算した各ボクセル線量値に対して散乱現象を考慮して，その影響分を周囲のボクセルに足しこむ．散乱現象による影響は x 軸，y 軸，z 軸の各方向におよぼされ，各方向それぞれいくつのボクセルに影響を与えるかは当該ボクセルの線量値によって異なる．図 2 においては，ペンシルビームがとあるボクセルに到達した際の散乱部分が他のボクセルに影響を与えていることを示している．以下が散乱計算部分のコードイメージであり，各ボクセルに対して scalc 関数で計算した散乱値を散乱値を格納する配列 Voxel2 に足しこんでいる様子を示している．

```
for (z 軸) {
  for (y 軸) {
    for (x 軸) {
      散乱値 = scalc(Voxel1);
      for (周辺ボクセル) {
        // 散乱値の足しこみ
        Voxel2[xyz] += 散乱値;
      }
    }
  }
}
```

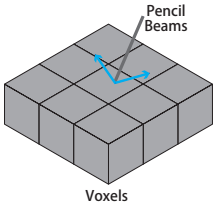

図 2 散乱計算イメージ
Fig. 2 Scatter Calculation

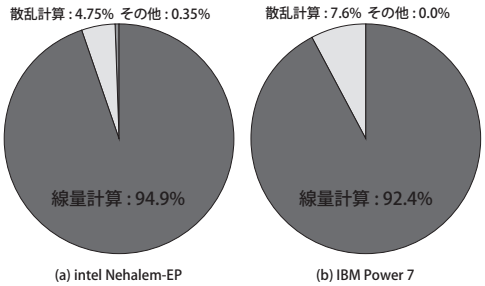

図 3 Intel/IBM プロセッサ上での性能プロファイル結果
Fig. 3 Profile results on Intel/IBM processor

step 4: 補正計算

線量分布の補正計算を行う．

並列化に先立って，実行プロファイルにより，上記の各ステップの処理時間を計測した．ボクセルの一辺の長さ (以下ボクセルサイズとする) は Intel プロセッサは 1.5mm，Power 7 プロセッサは 0.5mm の場合でそれぞれ gcc でコンパイルを行い実行プロファイルを行った．プロファイル環境の詳細なマシンパラメータに関しては 4 章を参照されたい．図 3 に Intel Xeon プロセッサ及び Power 7 プロセッサにおけるプロファイル結果を示す． 図 3 は本線量計算エンジン全体の実行時間の内，前述の各ステップがどのくらいの割合を占めているかを示している．図 3 よりどちらのプロセッサにおいても，step 2 の線量計算部分が全体の 90%以上を占めており，次に処理の割合が大きいのは

step 3 の散乱計算部分であることもわかる．図中「その他」の部分は初期化部分と補正計算部分の割合を示している．

3. 逐次線量計算エンジンプログラムに対するチューニング

本章では 2 章で述べた線量計算エンジンに対する自動並列化コンパイラによる高いスケーラビリティを得ることができる逐次チューニングについて述べる．既に述べた様に，本線量計算エンジンは，初期化，線量計算，散乱計算，補正計算の各ステップからなる．元のプログラムは Parallelizable C (level 2) で記述されているため，並列化を行いやすいコードになっているが，このままで十分な並列性が無いため，並列性を抽出しやすい逐次プログラムの書き換えを検討する．以下の各節で各ステップの並列化について述べる．

3.1 線量計算の逐次チューニング

線量計算部分は各ペンシルビームが与える線量をボクセル配列に足しこんでいる．ペンシルビームの本数は数万から数十万のオーダーのため，ペンシルビームレベルで並列化を行うことが性能向上の鍵である．しかしながら，各ペンシルビームが通過するボクセルは重複し，かつそのアクセスは間接参照を用いているため，データの競合が発生し，並列化コンパイラは各ボクセルの計算を並列実行不能であると判定する．そこで，ボクセル配列に対して配列の次元を 1 つ増やした記法をとり，プロセッサ台数分ボクセル配列を共有メモリに確保することで並列性を増加させる．これにより，各プロセッサはそれぞれが担当するペンシルビームの線量計算を他のプロセッサと独立して実行することが可能である．これにより各プロセッサが計算した値をマージするコードが余分に必要である．これに関しては後述する．下記に線量計算のコードイメージを示す．

```
/* 並列化可能ループ */
for (p = 0; p < プロセッサ数; p++) {
  for (ペンシルビーム数/プロセッサ数) {
    for (ペンシルビーム通過ボクセル) {
      // 線量値の計算
      線量 = dcalc();
      Voxel1[xyz][p] += 線量
    }
  }
}
```

上記のコードイメージにおいて Voxel1 は線量値を格納する配列，xyz はペンシルビームが通過するボクセルの座標を示し，p はプロセッサ番号である．このような記述により，最外側ループを並列実行可能であると自動並列化コンパイラは判定可能である．また，本処理は dcalc 関数によって行われる線量計算が線量計算全体の 96%と大部分を占め，Voxel1 配列へのアクセス頻度は非常に少ないため，メモリアクセスが比較的少なく，スケーラビリティの向上が期待される．

3.2 散乱計算の逐次チューニング

散乱計算は各ボクセルの線量値に応じて，周囲のボクセルに与える散乱の影響を計算している．本計算は全てのボクセルを走査する処理となっているため，各ボクセルを走査する際にまず線量計算で各プロセッサが計算した線量値を総計することでできるだけ余計な計算コストがかからない様にしている．そして，その値に応じて散乱値の影響を計算するがその際に 3 次元のボクセル配列の z 軸方向で各プロセッサに処理を分担する．しかし，散乱の影響を受けるボクセル数は実行時まで不明のため，データの競合が起きる可能性があり，コンパイラは並列化不能と判定する．これに関しても次元拡張で対応する．以下に散乱計算のコードイメージを示す．

```
/* 並列化可能ループ */
for (p = 0; p < プロセッサ数; p++) {
  for (z 軸/プロセッサ数) {
    for (y 軸) {
      for (x 軸) {
        for (q = 0; q < プロセッサ数; q++) {
          // 線量計算部で各プロセッサが計算した結果のマージ
          Voxel1_1[xyz] += Volel1[xyz][q];
        }
        散乱値 = scalc(Voxel1_1);
        for (周辺ボクセル) {
          // 散乱値の足しこみ
          Voxel2[xyz] += 散乱値;
        }
      }
    }
  }
}
```

上記のコードイメージにおいて Voxel1_1 は線量計算において各プロセッサ個別に計算された配列 Volxel1 をマージした結果を格納する配列，Voxel2 は Voxel1_1 の値を元に，散乱計算値を周囲に足しこんだ値を格納する配列である．

このような記法をとることにより，最外側ループを

並列実行可能であると自動並列化コンパイラは判定可能である．ここでも，線量計算の場合と同様に各プロセッサが計算した散乱計算による線量値 (上記 Voxel2) をマージする必要がある．また，本処理は配列への要素アクセスが大部分を占めるため，SMP 環境においては，スケーラビリティが阻害される可能性がある．

このマージ部分に関しても以下のようなコードを書くことで並列化が可能である．

```
/*並列化可能ループ*/
for (p = 0; p < プロセッサ数; p++) {
  for (全てのボクセル/プロセッサ数) {
   Voxel3[xyz] += Voxel2[p][xyz];
  }
}
```

3.3 初期化部分の逐次チューニング

この部分のオリジナルコードは実行時に指定されるボクセルサイズに応じて線量値を格納する 3 次元データを malloc で動的に確保し，0 で初期化している．本プログラムは動的メモリ確保の必要性が少ないプログラムであるので 3.5 節で後述するように，並列化することによって生じる malloc のオーバーヘッドを低減する為にスタティックにメモリを確保する．

3.4 補正計算の逐次チューニング

本部分は散乱計算の計算結果 (Voxel3) を元に補正計算を行い，最終的な計算結果を求めているが，図 3 に示されているように，処理の割合が Intel プロセッサで 0.15%，IBM プロセッサで限りなく 0%に近いため今回は並列化の対象としていない．これらの処理の並列化が必要になるのは 512CPU を使用して補正計算以外の部分が理想的に 512 倍に加速された際に，加速後のコストと補正計算のコストがほぼ一致するため，本論文の並列化対象には含めていない．

3.5 スケーラビリティ向上の為のプログラム記述方法

SMP サーバにおいて CPU 数の増加に応じた高いスケーラビリティを実現するためには，プログラムの全体にわたり並列実行可能部分を増やす必要がある．本論文でこれから評価する様な 64CPU までを対象とした場合，微小な逐次部分が性能向上を阻害する可能性があるが，必ずしも全ての部分を並列化することが性能向上に寄与するわけではない．今回の知見としては特に malloc による動的確保の並列化はできる限り避ける事が重要であることがわかった．malloc 自体はスレッドセーフな関数であるが，ヒープ領域の確保時にヒープのロックを行う為に複数スレッドから呼ばれる場合に速度が低下する要因となってしまうため，静的配列を使う様に逐次プログラムを修正している．

表 1 評価環境及びパラメータ
Table 1 Evaluation environment

	SR16000	HA8000/RS220
CPU	IBM Power 7 ((4.00GHz × 8) × 4) × 4	Intel Xeon X5670 (2.93GHz × 6) × 2
L1 D-Cache	32KB / 1 CPU	32KB / 1 CPU
L1 I-Cache	32KB / 1 CPU	32KB / 1 CPU
L2 cache	256KB / 1 CPU	256KB / 1 CPU
L3 cache	32MB / 8 CPU	12MB / 6 CPU
Operating System	Red Hat Enterprise Linux	Ubuntu Linux
Native Compiler1	GNU C Compiler version 4.4.5	GNU C Compiler version 4.4.3
Compile Option 1	-O3 -m32 -fopenmp	-O3 -m32 -fopenmp
Native Compiler2	IBM XLC Compiler 11.0	Intel Parallel Studio 12.0
Compile Option 2-1	-q64 -qsmp=omp -O4 -qarch=pwr7 -qmaxmem=-1	-m32 -fast -openmp
Compile Option 2-2	-q64 -qsmp=auto -O4 -qarch=pwr7 -qmaxmem=-1	-m32 -fast -parallel
Voxel Size	0.5mm	1.5mm
# of Beam	165018	18369

4. SMP サーバー上での性能評価

本章では 3 章で述べた手法によりチューニングされたアプリケーションを自動並列化コンパイラで並列化し，その性能を SMP サーバーにおいて評価し，その結果について分析を行う．

4.1 評 価 環 境

各環境における評価パラメータを表 1 に示す．本評価においては，IBM Power 7 を搭載した日立製作所製のスーパーテクニカルサーバ SR16000 及び，Intel Xeon を搭載した日立製作所製の SMP サーバ HA8000/RS220 で評価を行う．すでに述べたように，OSCAR コンパイラは source-to-source コンパイラとして動作するため，各サーバーのネイティブコンパイラを使用することで並列性能の評価が可能である．表 1 の Native Compiler の欄にあるように，gcc 及び各プロセッサベンダーが提供している商用コンパイラを使用した場合でそれぞれスケーラビリティの評価を行う．また，商用コンパイラは自動並列化機能も持っているため，その機能を利用して自動並列化を行った場合の OSCAR コンパイラとの性能比較も行う．なお，Intel 環境においては，線量計算エンジンに使用するライブラリが 32bit 版のみしか存在しないため，32bit 用バイナリを生成するオプションを付与している．

4.2 HA8000/RS220(Intel Xeon) 上での性能評価

図 4 に HA8000/RS220 上での性能評価結果を示す．

図 4 において横軸はプロセッサ構成で，original はオリジナルコードを intel コンパイラ (以下 icc)，及び GNU Compiler Collection(以下 gcc) でコンパイルした場合の逐次性能を示している (OSCAR コンパイラは使用していない)．nCPU は 3 章で述べた並列度増加のためのチューニングを施されたコード (チューニング版) で n 基の CPU を計算資源として OSCAR による並列化を行い，ネイティブコンパイラに icc および gcc を用いた場合の性能をそれぞれ示している．また，icc による自動並列化はオリジナルコードとチューニング版を両方自動並列化した結果，性能がよかったオリジナルコードの性能を記載している．縦軸は OSCAR と gcc によるチューニング版の 1CPU 実行時間を 1 とした場合の速度向上率である．

図 4 より，icc による自動並列化では台数効果は無いこと，また，OSCAR コンパイラで並列化を行うと，バックエンドのマシンコード生成に icc と gcc のいずれを使用してもプロセッサ数の増加に応じて性能が向上していることがわかる．icc 使用時に 12CPU で icc の 1CPU 実行時に比べて 6.89 倍 (1CPU で gcc 使用持に対して 13.39 倍)，gcc 使用時に 12CPU で gcc の 1CPU 実行時に比べて 8.96 倍の性能向上をそれぞれ得ている．

まず，この OSCAR コンパイラによる性能向上に関して分析を行う．図 5 は線量計算エンジン内の各処理の処理時間の内訳を示している．横軸はプロセッサ構成であり，オリジナルコードを gcc 及び icc でコンパイルした場合の時間，各 CPU 構成においてそれぞれ OSCAR コンパイラのネイティブコンパイラとして用いた場合の時間を示している．なお，縦軸はオリジナルコードを gcc コンパイルして作成した実行バイナリの実行時間を 100 として正規化している．既に 2 章で述べた様に，本計算エンジンは線量計算及び散乱計算が計算時間の殆どをしめており，値としては CPU 数が 1CPU，2CPU，6CPU，12CPU と変化すると，gcc の場合，線量計算部分は 93.7，47.95，16.55，9.05，散乱計算部分は 5.25，3.9，2.85，1.9，icc の場合，線量計算部分は 47.60, 24.00,8.45,5.45, 散乱計算部分は 3.35,2.50,1.95,1.45 とスケーラブルに実行時間が減少しているため，図 5 の様なスケーラブル結果が得られたことが裏付けられる．また，6CPU から 12CPU の速度向上率が鈍化している理由としては，本アーキテクチャは 1 チップ当たり 6CPU 搭載しており，QPI で接続されているため，チップ間の通信にコストがかかるからである．

次に icc による自動並列化の性能について述べる．

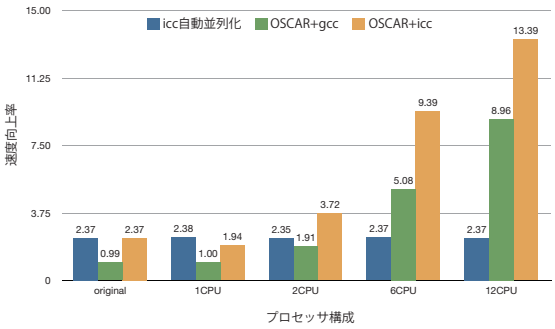

図 4 Intel Xeon プロセッサ上での性能評価結果
Fig. 4 Evaluation Result on Intel Xeon Processor

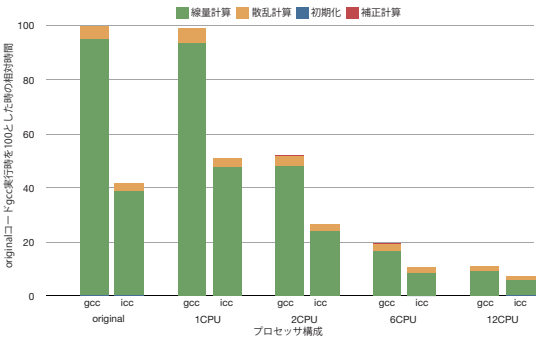

図 5 Intel Xeon プロセッサ上での性能分析結果
Fig. 5 Performance Analysis on Intel Xeon Processor

既に図 4 に示したように，icc による並列化効果は出ていない．icc の並列化レポートによると，線量計算部や散乱計算部などの主要なループは依存解析の結果，イタレーション間依存があるため並列化不能であるとしている．icc が並列化可能であると判定したのは，散乱計算部および補正計算の内の一部のループであるが，回転数が少ないと icc が判断し実際には並列処理を行なっていない．結果的に線量計算と散乱計算の一部のベクトル化を行い，オリジナルコードを gcc で実行した場合と比較して線量計算を 2.45 倍，散乱計算を 1.6 倍高速化している．2 章のプロファイル結果とも併せて icc によるオリジナルコードに対する速度向上率は妥当であるとわかる．これに対し，OSCAR コンパイラはポインタ解析等の解析系の充実により，主要なループを並列化可能としている．

4.3 Hitachi SR16000 上での性能評価

図 6 に SR16000 上での性能評価結果を示す．図 6 において横軸はプロセッサ構成で，original はオリジナルコードを IBM XLC コンパイラ (以下 xlc)，及び gcc でコンパイルした場合の逐次性能を示している

(OSCAR コンパイラは使用していない)．nCPU は 3 章で述べたチューニング済みコードで n 基の CPU を計算資源として OSCAR による並列化を行い，ネイティブコンパイラに xlc および gcc を用いた場合の性能を示している．また，xlc による自動並列化はオリジナルコードと並列度を増加させた記法を取ったコードを両方自動並列化した結果，結果のよかったオリジナルコードの性能を記載している．縦軸は OSCAR と gcc による 1CPU 実行時間を 1 とした場合の速度向上率である．

図 6 より，xlc による自動並列化では台数効果は無いこと，また，OSCAR コンパイラで並列化を行うと，Intel プロセッサの場合と同様，どちらのネイティブコンパイラを使用してもプロセッサ数の増加に応じて性能が向上していることがわかる．xlc 使用時に 64CPU で xlc の 1CPU 実行時に比べて 48.06 倍 (1CPU で gcc を利用した時に対して 67.58 倍)，gcc 使用時に 64CPU で gcc の 1CPU 実行時に比べて 49.93 倍の性能向上を得ている．

まず，この OSCAR コンパイラによる性能向上に関して分析を行う．図 7 は線量計算エンジン内の各処理の処理時間の内訳を示している．横軸はプロセッサ構成であり，オリジナルコードを gcc 及び xlc でコンパイルした場合の時間，各 CPU 構成においてそれぞれ OSCAR コンパイラのネイティブコンパイラとして用いた場合の時間を示している．なお，縦軸はオリジナルコードを gcc コンパイルして作成した実行バイナリの実行時間を 100 として正規化している．既に 2 章で述べた様に，本計算エンジンは線量計算及び散乱計算が計算時間の殆どをしめており，値としては CPU 数が 1CPU，32CPU，64CPU と変化すると，gcc の場合，線量計算部分は 95.9，3.21，1.69，散乱計算部分は 7.60，0.45，0.35，xlc の場合，線量計算部分は 62.6，2.1，1.2，散乱計算部分は 11.0，0.27, 0.29 と xlc で 64CPU 使用時の散乱計算部分を除いてスケーラブルに実行時間が減少しているため，図 5 の様なスケーラブル結果が得られたことが裏付けられる．また，32CPU から 64CPU の速度向上率が鈍化している理由としては，本アーキテクチャは 1 プレーン当たり 32CPU 搭載しており，それらがプロセッサ内バスに対して相対的に低速なネットワークで結合されているためである．特に散乱計算部は単位時間当たりのメモリアクセスが多いため，性能が低下している．

次に xlc による自動並列化の性能について述べる．既に図 6 に示したように，xlc による並列化効果は出ていない．xlc の並列化レポートによると，線量計算

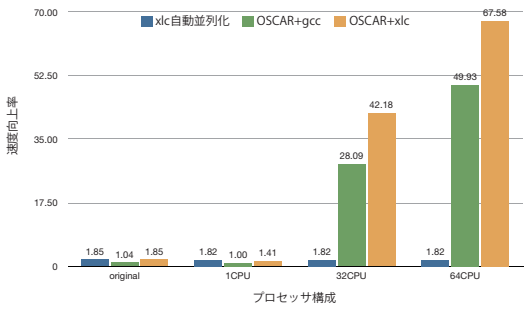

図 6 IBM Power7 プロセッサ上での性能評価結果
Fig. 6 Evaluation Result on IBM Power7 Processor

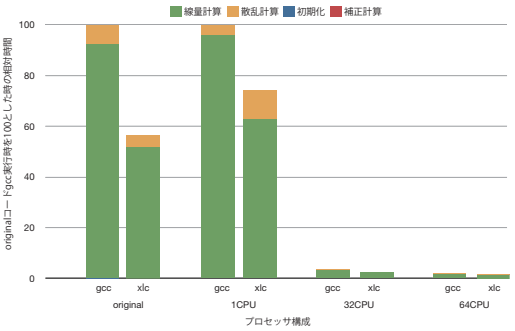

図 7 IBM Power7 プロセッサ上での性能分析結果
Fig. 7 Performance Analysis on IBM Power7 Processor

部や散乱計算部などの主要なループは依存解析の結果，イタレーション間依存があるため並列化不能であるとしている．xlc が並列化可能であると判定したのは，初期化，線量計算部，散乱計算部内の一部のループである．オリジナルコードを gcc で実行した場合と比較して初期化部分を 1.3 倍，線量計算を 1.77 倍，散乱計算を 1.57 倍高速化している．2 章のプロファイル結果とも併せて xlc によるオリジナルコードに対する速度向上率は妥当であるとわかる．これに対し，Intel コンパイラの場合と同様に，OSCAR コンパイラはポインタ解析等の解析系の充実により，主要なループを並列化可能としている．

5. ま と め

本論文では臨床で使用されている重粒子線治療用線量エンジンという実プログラムを OSCAR 自動並列化コンパイラを用いて異なる SMP 上で任意台数で並列チューニングの負荷なく自動で高速化できる手法を提案した．その並列化性能を Intel Xeon プロセッサを搭載した SMP サーバー及び IBM Power 7 プロセッサを搭載した SMP サーバー上で評価を行った．その結

果，先行研究では Intel Xeon プロセッサ 8CPU を指定して 2.8 倍までしか速度向上出来なかった対象プログラムを Intel プロセッサ上で 12CPU を使用した場合に約 9.0 倍，IBM Power 7 プロセッサ上で 64CPU を使用した場合に約 50.0 倍の性能向上が得られ，提案手法が高いスケーラビリティを実現可能であることを確認した．また，同時に Parallelizable C によってアプリケーションを記述することにより，OSCAR コンパイラによって実アプリケーションにおいても，並列性の抽出が可能であることも確認した．この研究により線量計算をより高精度，より高速に実施できるようになれば，治療計画を作成するのにかかる時間を短縮できる共に，いずれは 1 日当たりの治療人数を増加させたりするなど，治療サービスの向上に繋がっていく可能性を開くことができた．また提案手法による性能向上は，より高精度な計算手法であるモンテカルロ法を用いた線量計算の実用化や患者状態に合わせた Adaptive Therapy Planning にも道を開く可能性を示唆している．

今後の課題としては本手法の臨床現場での適用，評価が挙げられる．

謝辞 本研究の一部は早稲田大学グローバルＣＯＥプログラム「アンビエントＳｏＣ教育研究の国際拠点」（文部科学省研究拠点形成費補助金）の支援により行われた．

参 考 文 献

1) 厚生労働省：人口動態調査，http://www.mhlw.go.jp/toukei/list/81-1.html.
2) 山本， 足立：粒子線用線量計算エンジンの開発，MSS 技報， Vol.21, pp.36–42 (2010).
3) Yoshida, Y., Kamei, T., Hayase, K., Shibahara, S., Nishii, O., Hattori, T., Hasegawa, A., Takada, M., Irie, N., Uchiyama, K., Odaka, T., Takada, K., Kimura, K. and Kasahara, H.: A 4320MIPS Four-Processor Core SMP/AMP with Individually Managed Clock Frequency for Low Power Consumption, *IEEE International Solid-State Circuits Conference, ISSCC*, pp.100–590 (2007).
4) Yuyama, Y., Ito, M., Kiyoshige, Y., Nitta, Y., Matsui, S., Nishii, O., Hasegawa, A., Ishikawa, M., Yamada, T., Miyakoshi, J., Terada, K., Nojiri, T., Satoh, M., Mizuno, H., Uchiyama, K., Wada, Y., Kimura, K., Kasahara, H. and Maejima, H.: A 45nm 37.3GOPS/W heterogeneous multi-core SoC, *IEEE International Solid-State Circuits Conference, ISSCC*, pp. 100–101 (2010).
5) Masayasu, Y., Takeshi, S., Toshiaki, T., Yasuhiko, K. and Toshinori, I.: NaviEngine 1, System LSI for SMP-Based Car Navigation Systems, *NEC TECHNICAL JOURNAL*, Vol. 2, No.4 (2007).
6) Nakajima, M., Yamamoto, T., Yamasaki, M., Hosoki, T. and Sumita, M.: Low Power Techniques for Mobile Application SoCs Based on Integrated Platform "UniPhier", *ASP-DAC '07: Proceedings of the 2007 Asia and South Pacific Design Automation Conference* (2007).
7) Maruyama, T., Yoshida, T., Kan, R., Yamazaki, I., Yamamura, S., Takahashi, N., Hondou, M. and Okano, H.: Sparc64 VIIIfx: A New-Generation Octocore Processor for Petascale Computing, *Micro, IEEE*, Vol. 30, No. 2, pp.30 –40 (2010).
8) Kalla, R., Sinharoy, B., Starke, W. and Floyd, M.: Power7: IBM's Next-Generation Server Processor, *Micro, IEEE*, Vol.30, No.2, pp.7 –15 (online), DOI:10.1109/MM.2010.38 (2010).
9) Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc of The 13th International Workship on Languages and Compilers for Parallel Computing(LCPC2000)* (2000).
10) 本多， 岩田， 笠原：Fortran プログラム粗粒度タスク間の並列性検出法，信学論 (D-I), Vol.J73-D-I, No.12, pp.951–960 (1990).
11) Kimura, K., Wada, Y., Nakano, H., Kodaka, T., Shirako, J., Ishizaka, K. and Kasahara, H.: Multigrain Parallel Processing on Compiler Cooperative Chip Multiprocessor, *Proc. of 9th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9)* (2005).
12) 中野， 桃園， 間瀬， 木村， 笠原：マルチコアプロセッサ上での粗粒度並列処理のためのローカルメモリ管理手法，情報処理学会論文誌コンピューティングシステム， Vol.2, No.2, pp. 63–74 (2009).
13) 白子， 吉田， 押山， 和田， 中野， 鹿野， 木村， 笠原：マルチコアプロセッサにおけるコンパイラ制御低消費電力化手法 (2006).
14) 間瀬， 木村， 笠原：マルチコアにおける Parallelizable C プログラムの自動並列化，情報処理学会研究報告,Vol.2009-ARC-184, No.15,pp.1-10 (2009).
15) 山本， 足立， 高谷， 阿部， 坂本， 兼松：重粒子線用線量計算エンジンの開発，第 99 回 日本医学物理学会学術大会報文集 (2010).
16) 高谷， 阿部， 坂本， 近藤， 冨士， 山路， 濱田， 高橋， 山本， 足立， 兼松：

普及型重粒子線用治療計画装置の開発，第 99 回
日本医学物理学会学術大会報文集 (2010).