OSCARコンパイラを用いた医用画像3Dノイズリダクションの自動マルチグレイン並列処理

柴崎 大侑[1,a)]　桑島 昂平[1]　奥村 万里子[1]　見神 広紀[1]　木村 啓二[1]　門下 康平[2]　中野 恵一[2]
笠原 博徳[1,b)]

概要：医用画像 3 次元ノイズリダクション（3DNR）は，動画像データ内の連続するフレームを時間軸方向に分析し，動画像データに含まれるノイズ成分を検出・除去するデジタルノイズ除去手法の一種である．従来ではリアルタイム制約を守るために，3DNR 処理はハードウェアで実装されることが多かったが，近年では生産性，柔軟性の観点からソフトウェア実装が注目されてきており，ソフトウェア実装の最大の課題である実行時間の改善が求められている．また，現在のマルチコアプロセッサの普及に伴い，プログラム並列化による高速処理が有効になっている．プログラムのループ並列化は最も一般的な並列化手法として知られているが，3DNR プログラムはメニーコア実行に十分なループ並列性を持たないため，ループ並列化のみの適用では十分な性能向上は得られない．以上の背景から，本稿では 3DNR プログラムから更なる並列性を利用するための，色処理別ループ分割を用いた階層的並列化手法を提案する．3DNR プログラムに提案手法を適用することで，プログラム全域の並列性を階層的に利用したマルチグレイン並列処理を実現できる．提案手法を適用し，POWER7 ベースの 128 コア SMP サーバ Hitachi SR16000 上で性能評価を行った結果，3DNR プログラムの根幹処理である動き補償 及び コントラストマップ取得において，128 コア使用時に 99.86 倍の性能向上，ベクトル平滑化 及び 巡回 NR において，128 コア使用時に 79.64 倍の性能向上を得ることが出来た．

1. はじめに

一般的にカメラから入力された画像信号において一定のノイズは不可避であり [1]，ノイズを低減し画像への悪影響を抑えることが画像処理において非常に重要な課題である [2]．特に医用画像では，医師が診断を下す上で可能な限り正確な情報を示す必要があるため，高精度なノイズ除去が望まれる．

動画像に対するノイズ除去手法として一般的に普及している 3 次元ノイズリダクション（3DNR）は，動画像データ内の連続するフレームを時間軸方向に分析し，動画像データに含まれるノイズ成分を検出・除去するデジタルノイズ除去手法の一種である．この 3DNR 処理は，リアルタイム制約を守るためにハードウェアで実装されるのが一般的である [3]．一方，ハードウェアでの実装は，頻繁な仕様変更や機能の追加に迅速に対応できないことが問題となっており，その低い柔軟性が問題となっている．また，組み込み系への搭載を想定した場合，専用ハードウェアの高価な開発コストが製品開発時のハードルになっており，より柔軟で安価なソフトウェア実装の発展が望まれている．ソフトウェアでの実装は仕様変更等に柔軟であるといった利点があり，開発コストを抑え，生産性を向上させたい場合には非常に効果的である．しかし，従来のソフトウェアは単一プロセッサ上での逐次処理を想定して実装されることが多く，マルチコアプロセッサ上での並列処理を実現できていないために, 実行時間が非常に大きいという問題があった．

以上の背景から，本稿では OSCAR 自動並列化コンパイラを用いた 3DNR プログラムの並列化を行うとともに，マルチコアプロセッサ上で性能を最大限に引き出す並列化手法の検討を行った．従来のプログラム並列化では，ループ内部の並列性を利用するループ並列化が一般的であり，様々な並列化技術，リストラクチャリング手法が研究されてきた [4]，[5]，[6]．しかし，ループに十分な並列性が存在しない場合，プロセッサ数を増加させても十分な性能向上

[1] 早稲田大学
Waseda University.
[2] オリンパス株式会社
Olympus Corporation.
[a)] tomo@kasahara.cs.waseda.ac.jp
[b)] kasahara@kasahara.cs.waseda.ac.jp

は得られない．特に 60 コアあるいは 100 コア以上を搭載するようなメニーコアでの並列処理の場合，3DNR プログラムには全てのコアを有効利用するのに十分なループ並列性が存在しておらず，効率的な並列処理を実現できていなかった．そういった背景を踏まえ，本稿では ループとループ間の 2 種類の階層並列性を抽出する階層的並列化手法を 3DNR プログラムに適用した．

ソフトウェアの並列化手法には，様々な手法が提案されており，並列化対象のプログラム構造やアルゴリズムごとに最適な手法が存在する．本稿で扱う 3DNR プログラムは，デジタルカメラ等で使用されるイメージセンサで標準とされているベイヤ配列方式 [7] を採用している．ベイヤ配列方式で表現された画像は 1 つの画素が R, G, B のいずれかの色の情報のみを扱っており，画像処理を行う際においても色別に処理を行う場合が多い．その特徴を考慮し，本稿では 色処理別分割を用いた階層的並列化手法を提案する．本手法ではベイヤ配列に対して処理を行うループを，R 画素の処理ループ，G 画素の処理ループ，B 画素の処理ループの 3 種類のループに分解し，色別ループ間にタスク並列性を生成する．ここで抽出されたタスク並列性を，従来使用されていたループ並列性と階層的に組み合わせることで，プログラム全域の並列性を効率的に利用したマルチグレイン並列処理を実現できる．

一般的に手動でのソフトウェア並列化は非常に時間がかかる作業であり，その作業を製品開発に取り入れるのは開発コストの面で好ましくない．そこで，本提案手法では OSCAR コンパイラを用いて，マルチグレイン並列処理を可能な限り素早く適用することで，3DNR プログラムの性能向上と開発期間の短縮を目指す．

以下, 2 章で 3DNR プログラムの概要について，3 章で本稿で提案する 3DNR プログラムの並列化手法について, 4 章で性能評価結果について述べ，最後に 5 章で本研究のまとめを述べる．

2. 3DNR プログラム

ここでは，動画像用ノイズ除去手法の一種である 3 次元ノイズリダクション（3DNR）を，ソフトウェアとして実装した 3DNR プログラムについて述べる．

2.1 3DNR プログラムの概要

本稿で扱った 3DNR プログラムは，動画像データ内のフレームを想定した複数の RAW 画像を入力ファイルとして扱い，それらの RAW 画像を対象にノイズ除去処理を行うことで各 RAW 画像に含まれるノイズ成分を軽減するプログラムである．ノイズの低減手法は，基盤となる技術に基づいて，空間的な方法と時間的な方法の大きく 2 種類に分類される．2 次元ノイズリダクション (2DNR) 等に代表される空間的ノイズ除去手法では，動画の個々のフレーム内を分析し，信号を空間方向に平滑化することでノイズ成分を除去する．一方，3DNR のような時間的ノイズ除去手法では，動画内の連続するフレームを分析対象とし， 複数のフレームで同じ位置にある画素の値を比較し，信号を時間軸方向に平滑化する方式を採用している [8]．本プログラムのノイズ除去処理の際には，RAW 画像 1 枚毎に一連の関数群が実行され，実行毎に現在のフレームとその前後に連続するフレームを時間軸方向に分析することで，各画像に含まれるノイズ成分を検出・除去する．

本プログラムは，図 1 に示すような 8 つの過程から構成される．プログラム内では，8 つの過程はそれぞれ関数として記述され，各関数内ではフレームの画素に対してループ文で処理を行う．プログラム内の多くのループは 2 重にネストされたループで構成され，ループの回転数が画像サイズに依存している．特に 3DNR プログラムの根幹となるのは，動き補償，動きベクトル平滑化，巡回 NR の 3 処理であり，動き補償処理でフレーム間の各画素における動きベクトルの算出を行った後に，動きベクトル平滑化処理で特異なデータやノイズを修正，最後に巡回型 NR 処理でノイズ除去のフィルタ処理を行う．これら 3 つの処理を含む，各関数ごとの実行時間プロファイル結果を図 2 に示す．

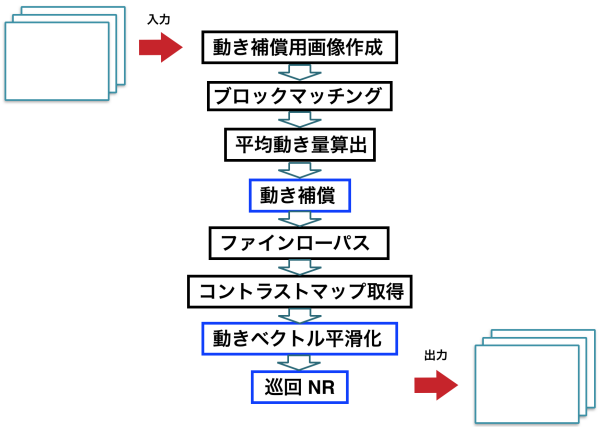

図 1 3DNR プログラム 実行フロー

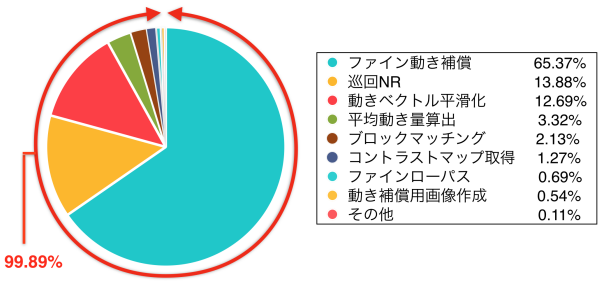

図 2 3DNR プログラム プロファイル結果

図 2 を見ると，実行時間のほとんどを, 処理の根幹である動き補償（65.37% ），巡回 NR（13.88% ），動きベクトル平滑化（12.69% ）の 3 関数が占めていることが分かる．

よって，3DNR プログラムの高速処理を実現するためには，これらの関数の実行時間を短縮することが必須であるといえる．本稿においては，これら 3 つの関数に提案手法を適用し，3DNR プログラムの性能向上を図った．

3. 3DNR プログラムの並列化

本節では，3.1 で 3DNR プログラムの並列化に用いた OSCAR コンパイラについて，3.2 で従来のループ並列化で生じた問題点について，3.3 及び 3.4 では 3.2 で述べた問題を解決する階層的並列化手法について述べる．

3.1 OSCAR 自動並列化コンパイラ

ここでは，OSCAR コンパイラの概要について述べる．OSCAR コンパイラは，従来のループ並列化に加え，ループ間，関数間の並列性を利用する粗粒度並列化，ステートメント間の並列性を利用する近細粒度並列化を効果的に組み合わせたマルチグレイン並列処理を実現しており，複数の並列性を階層的に利用する自動並列化コンパイラである [9]，[10]．

ループ並列化は最も一般的な並列化手法であり，ループ内の並列性を利用する並列化である．それに対し，OSCAR コンパイラにおける粗粒度並列化ではループ間，関数間の並列性を，近細粒度並列化では代入文等のステートメント間の並列性を利用する．

OSCAR コンパイラの粗粒度並列化では，対象のソースプログラムを基本ブロックやループ，サブルーチン呼び出しの 3 種類のタスクに分解し，タスク間の制御フロー，データ依存を解析してマクロフローグラフを作成後，最早実行可能条件解析の適用により，タスク間の粗粒度並列性を表現するマクロタスクグラフを生成する．また，粗粒度タスクがループ，サブルーチン呼び出しである場合には，その内部を粗粒度タスクに分割して，プログラムの全域にわたって階層的な並列性を抽出する [11]．

OSCAR コンパイラは，C 言語や Fortran 言語で記述された逐次ソースコードから並列性を抽出し，並列化を施したソースコードを出力する．並列化版ソースコードは，逐次ソースコードが C 言語の場合には，同一の C 言語で記述され，OpenMP と互換性を持つ OSCAR_API 指示文が挿入される．並列化版ソースコードのコンパイルの際には，逐次コンパイラのコンパイルオプションに OpenMP 用のオプションを使用することで，各ハードウェア用の並列化された実行バイナリを得ることができる．

3.2 ベイヤ配列方式プログラムのループ並列化

ベイヤ配列とは，図 3 のような周期性を持つ画像の配列パターンのことであり，1 つの画素で R,G,B のいずれかの色の情報のみを数値化する [12]．

このベイヤ配列は，3DNR プログラムの全域で使用される配列パターンであり，特に動き補償，ベクトル平滑化，巡回 NR の 3 関数はフレームの全画素に対して，R,G,B の色ごとに処理を切り替えるループで構成されている．このループは入力画像の縦の画素数×横の画素数の回転数を持つ 2 重のネストループ構造になっており，ループ並列化が有効である．しかし，ループ並列化のみで高速処理を図る場合，各ループの十分な並列性が必要となる．

YX	0	1	2	3	4	5
0	B	G	B	G	B	G
1	G	R	G	R	G	R
2	B	G	B	G	B	G
3	G	R	G	R	G	R
4	B	G	B	G	B	G
5	G	R	G	R	G	R

図 3 ベイヤ配列

ここで，ループ並列化における並列性能を確認するために，OSCAR コンパイラを用いてループ並列化を適用した時点での使用コア数毎の並列性能を図 4 に示す．図 4 では縦軸に速度向上率，横軸に使用コア数を設定し，動き補償，ベクトル平滑化，巡回 NR における使用コア数毎の速度向上率を示している．ここでの速度向上率は逐次実行時間を 1.00 としたときの並列処理時の速度向上倍率である．

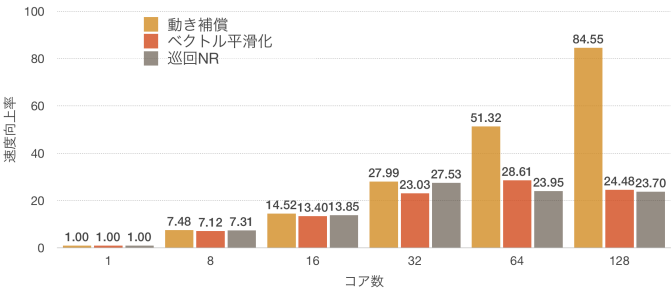

図 4 Hitachi SR16000 上での OSCAR コンパイラによるループ並列化適用時の 3DNR プログラムの速度向上率

図 4 を見ると，動き補償関数ではループ並列化が効果的に働き，128 コアで 84.55 倍とスケーラブルな性能向上を得られている．また，ベクトル平滑化関数と巡回 NR 関数に関しては，32 コア使用時にはベクトル平滑化関数が 23.03 倍，巡回 NR 関数が 27.53 倍とスケーラブルに性能が向上し，64 コア使用時にはベクトル平滑化関数が 28.61 倍，巡回 NR 関数が 23.95 倍，128 コア使用時にはベクトル平滑化関数が 24.48 倍，巡回 NR 関数が 23.70 倍と性能が悪化していることが確認できる．この性能悪化は，最外側ループが 1000 回程度のため 128 コア実行に適した並列性を持っていないことや，ループ内の処理コストが小さいた

めに並列実行時の同期やリモートメモリアクセス等のオーバーヘッドが相対的に大きくなってしまったことなどが原因として考えられる．

また，ループ並列化の別の事例として，XL C コンパイラによる自動並列化を試行した．XL C コンパイラでは各関数内の主要ループを並列化可能と判定できず，以下に示す図 5 のように，使用コア数に依らず 1.00 から 1.20 倍程度の速度向上率に留まり，コア数の増加に伴った性能向上を確認できなかった．GCC コンパイラの逐次実行時間を 1.00 としたときの，XL C コンパイラによる速度向上率を図 5 に示す．

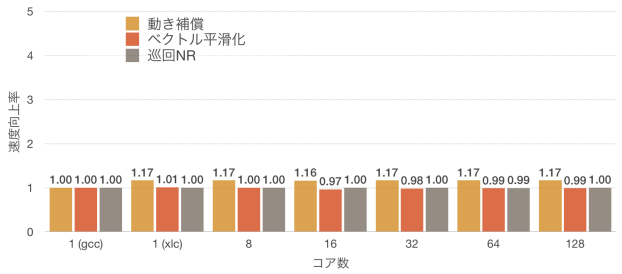

図 5 Hitachi SR16000 上での XL C コンパイラによるループ並列化適用時の 3DNR プログラムの速度向上率

以上の結果から，3DNR プログラムの更なる性能向上のためには，ループ以外の並列性を利用する必要があると考えられる．

3.3 色処理別ループ分割による階層的並列化

ここでは, 3DNR プログラムから潜在的な粗粒度並列性を抽出する手法について述べる．

図 4 の評価時には動き補償，ベクトル平滑化，巡回 NR の 3 関数はそれぞれ 1 つのループのみで構成されていたため，ループ並列性以外の並列性を利用できていなかった．そこでここでは，ベイヤ配列の全画素に対する処理を行うループを，R 画素の処理ループ，G 画素の処理ループ，B 画素の処理ループの 3 種類の色処理別ループに分解し，これらの関数内に粗粒度並列性を抽出した．

以下，具体的な抽出手法を説明する．図 3 の外側の Y ループに着目すると，この 2 重ループは Y の値が偶数である場合には BG 処理, 奇数である場合には GR 処理を行う方式になっていることが確認できる．このベイヤ配列における処理パターンの周期性に着目し，図 6 のように色処理別ループ分割を行う．この分割された 2 つのループは互いに依存を持たず，並列実行できる．すなわち，ベイヤ配列を色別ループに分割することで，関数内に粗粒度並列性を生成できた．また，本手法は画素の色を判断する際の条件文の分岐オーバーヘッドを軽減する効果も持ち，粗粒度並列性の抽出とともに逐次プログラムの性能向上を図ることができる．

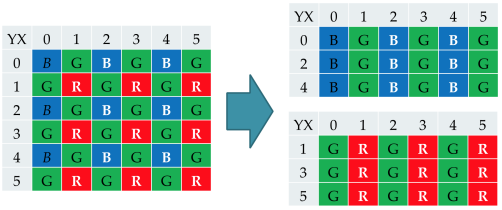

図 6 ベイヤ配列方式ループの色処理別ループ分割

さらに，図 6 のように，BG 間や GR 間に依存が存在しなければ，X ループに関しても色別分割を適用することができる．図 7 のように Y，X ループで 2 段階に分割すると，B 処理ループ，G 処理ループ，G 処理ループ，R 処理ループの 4 つのループ間での粗粒度並列性を生成することができるため，更に高い並列性を利用できる．

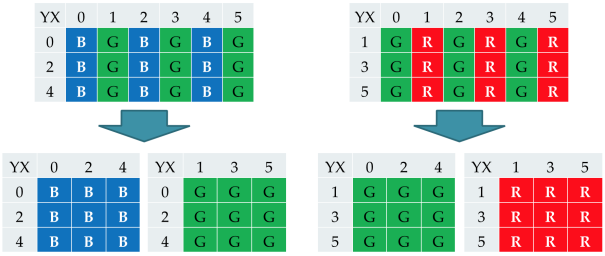

図 7 2 段階 色処理別ループ分割

本稿で提案する階層的並列化は，上述のように生成された粗粒度並列性を，従来持っていたループ並列性と階層的に組み合わせることで，プログラム全域の並列性を効率的に利用することを目的としている．例えば，図 7 に階層的並列化を適用する場合，B, G, G, R の色別ループをそれぞれ 32 分割し，それを 4 色分並べて粗粒度並列処理を行うことで，128 コアを利用するのに十分な並列性を確保できる．このように階層的に 2 種類の並列性を利用することで，ループ内のループ並列性が低い場合でも，関数全体としての並列性を十分に確保することができる．上述した色処理別ループ分割は，ベイヤ配列特有のループ構造を持つ動き補償，ベクトル平滑化，巡回 NR に適用可能であり，一般的なループ並列化手法に比べて，より効率的な並列実行を実現できる．

3.4 ループフュージョンによる並列実行効率の向上

3.3 の色別分割は，粗粒度並列性を得られるという利点があるが，ループ内の処理量が少なくなってしまうために，ループ並列化の効率が低下してしまうという特性を持つ．今回その対応として，関数間のループフュージョンを 4 つの関数に実施した．ループフュージョンとは，複数のループ内にある実行文を 1 つのループに融合する最適化のことをいう．

一般的に，プログラム実行時に高い並列性能を得るため

には，並列化対象タスクの並列性を上げることが有力である．特に本プログラムの場合，ループ並列化が有効な並列化手法であるため，並列実行効率を向上させるためにはループ内の処理量とメモリアクセス，同期等のオーバーヘッドとのバランスを改善することが有効となる．

今回は，動き補償 及び コントラストマップ取得の 2 つのループ，ベクトル平滑化 及び 巡回 NR の 2 つのループを対象にループフュージョンを適用した．

4. 性能評価

本節では，3DNR プログラムの性能評価の際に用いた評価環境及び，提案手法を適用したときの評価結果について述べる．また，ループ並列化の適用時と提案手法の適用時の並列性能を比較し，提案手法の有効性について検討する．

4.1 評価環境

本項では，3DNR プログラムの性能評価を行う際に用いた評価環境について述べる．

本研究で評価に用いた，Hitachi SR16000(モデル VM1) の各種性能を表 1 に示す [13]．SR16000 は 8 コア集積 POWER7 プロセッサを 16 個搭載した 128 コア cc-NUMA サーバである．各コアは 4.0GHz で動作し，256KB の L2 キャッシュを持ち, 8 コアで 32MB のオンチップ L3 キャッシュ を共有している．SR16000 で使用した gcc コンパイラのバージョンは 4.4.7 であり, コンパイルオプションは O3, fopenmp, mtune=power7, mcpu=power7 を使用した．

表 1 SR16000 性能

CPU	IBM POWER7
CPU core	128 cores (8cores / chip)
CPU Frequency	4.0GHz
L1-I cache (1core)	32KB
L1-D cache (1core)	32KB
L2 cache (1core)	256KB
L3 cache (1chip)	32MB
Native Compiler	gcc 4.4.7
Compiler Option	-O3 -fopenmp -mtune=power7 -mcpu=power7

次に 3DNR プログラムに用いた入力データの詳細を表 2 に示す．本評価では入力ファイルとして，1920x1080 のフル HD サイズの RAW 画像を 3 枚利用している．現在フレームと前後に連続したフレームの 3 枚を入力することで，動画を入力に用いた場合の実行環境を想定している．

表 2 3DNR プログラムの評価に用いた入力ファイル

ファイル形式	RAW イメージ
フレーム数 (入力画像数)	3
解像度	1920x1080

4.2 性能評価結果

本項では 3.3，3.4 で述べた並列化手法を適用したときの性能評価について述べる．

本評価では，動き補償 及び コントラストマップ取得のループフュージョン後の関数，ベクトル平滑化 及び 巡回 NR のループフュージョン後の関数の計 2 関数の各々の実行時間を計測し，逐次実行時の実行時間と並列実行時の実行時間を比較することで，並列実行時の各関数毎の速度向上率を算出した．

3.3 の階層的並列化，3.4 のループフュージョンを適用したときの，逐次実行と 128 コア使用時の並列性能について，計測した評価結果をループ並列化の結果とともに図 8, 図 9 に示し，ループ並列化と階層的並列化の性能比較を行う．

まず，図 8 より，動き補償 及び コントラストマップ取得のループフュージョン後の関数について，評価結果を比較する．図 8 では，縦軸に速度向上率，横軸に使用コア数を設定し，動き補償 及び コントラストマップ取得のループフュージョン後の関数について，ループ並列化を適用した時点での速度向上率及び，階層的並列化を適用した時点での速度向上率を示している．ここでの速度向上率は逐次実行時間を 1.00 としたときの並列処理時の速度向上倍率である．

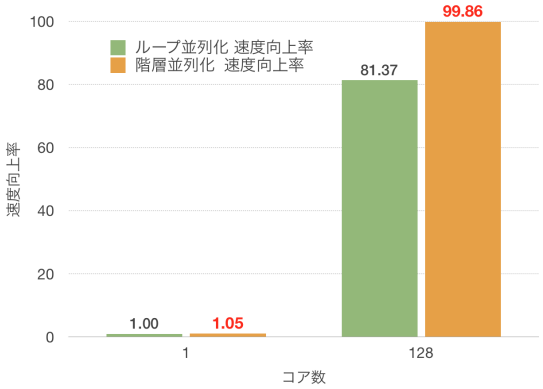

図 8 Hitachi SR16000 上での動き補償 及び コントラストマップ取得におけるループ並列化と階層的並列化の性能比較

図 8 の速度向上率に着目すると，ループ並列化のみの適用時には 128 コア使用時に 81.37 倍の速度向上が限界だったものが，提案手法の適用後には 128 コア使用時に 99.86 倍の速度向上を実現できている．元々高いループ並列性を持っていた動き補償処理であるが，粗粒度並列性とループ並列性を合わせて使用したこと，ループフュージョンにより色別ループ内部のループ並列性を高めたことでより高い並列化効率とオーバーヘッドの削減を実現できた．また，図 8 の 1 コア使用時の速度向上率に着目すると，ループフュージョンの適用によるループ特有のオーバーヘッドの削減等が効果的に働き，1 コア使用時の性能が適用前の 1.05 倍に改善されていることが分かる．

続いて，図 9 より，ベクトル平滑化 及び 巡回 NR のループフュージョン後の関数について，評価結果を比較する．図 9 では，ベクトル平滑化 及び 巡回 NR のループフュージョン後の関数について，ループ並列化を適用した時点での速度向上倍率及び，階層的並列化を適用した時点での速度向上倍率を示している．

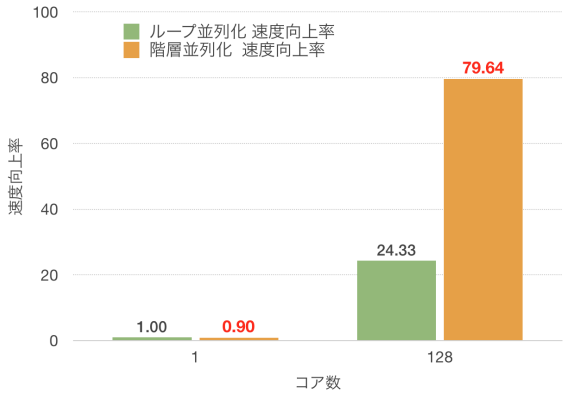

図 9 Hitachi SR16000 上でのベクトル平滑化 及び 巡回 NR におけるループ並列化と階層的並列化の性能比較

図 9 の速度向上率に着目すると，ループ並列化のみの適用時には 128 コア使用時に 24.33 倍の速度向上が限界だったものが，提案手法の適用後には 128 コア使用時に 79.64 倍の速度向上を実現できている．元々はループ並列性を十分に持っていない ベクトル平滑化 及び 巡回 NR に関しても，提案手法の適用により高い並列性の抽出を実現し，結果として並列性能を大幅に改善できた．一方，図 9 のループフュージョン適用後の 1 コア使用時の速度向上率においては，ループフュージョン適用後の性能が適用前に比べて 0.90 倍に悪化していることが確認できる．これはループフュージョンに伴い，ループ内で使用するデータ量が増加し，ワーキングセットが大きくなったことで，キャッシュミス数が増加したことが原因である．ただし，並列実行時には各コアが必要とするデータ量が少なくなるため，この問題は並列性能に影響しない．また，元々はループ並列性を十分に持っていなかった ベクトル平滑化 及び 巡回 NR には，ループフュージョンによるループ並列性抽出が並列性能に大きく影響するため，この逐次実行時間の増加を十分に隠すことができる．

5. まとめ

本稿では，3DNR プログラムの並列化手法として，ベイヤ配列方式の色処理別分割，ループフュージョンを用いた階層的並列化手法を提案した．また，提案手法を 3DNR プログラムに適用した場合の性能評価を POWER7 ベースの 128 コア cc-NUMA サーバ Hitachi SR16000 上で行い，ループ並列化と提案手法の性能比較を行うことで提案手法の有効性を検証した．

本稿では，動き補償関数，コントラストマップ取得関数，ベクトル平滑化関数，巡回 NR 関数に提案手法を適用した．これらは 3DNR プログラムの実行時間の大部分を占め，3DNR 処理の根幹部分といえる．提案手法を上記の関数に適用した結果，動き補償 及び コントラストマップ取得のループフュージョン後の関数では，128 コア使用時に 1 コア使用時に対して 99.86 倍の速度向上を達成でき，実行時間を大幅に短縮できた．ループ並列化適用時には，128 コア使用時に 81.37 倍の速度向上に留まっていたが，提案手法の適用で 99.86 倍にまで性能を改善でき，提案手法の有効性が確認できた．同様に，ベクトル平滑化 及び 巡回 NR のループフュージョン後の関数では，128 コア使用時に 79.64 倍の速度向上を達成できた．この関数は，ループ並列化適用時には 128 コア使用時に 24.33 倍の速度向上に留まっていたため，本手法の適用により大幅な性能向上が得られることが確かめられた．よって，元々の並列性が低い場合にも，提案手法が有効であることが確認できた．

並列性の抽出は依存性の有無の確認，抽出によって引き起こされるオーバーヘッドの考慮など，抽出時の条件が厳しく，十分な検討が必要な作業である．しかし，ベイヤパターンを持つ配列は，多くの場合で色処理別に配列を分割可能であり，色数に相当した粗粒度並列性を比較的容易に抽出できる．そのため，本稿で実施した色処理別ループ分割による階層的並列化はベイヤ配列に対応したプログラムの多くに利用でき，汎用性が非常に高い手法だといえる．

本稿では，手動で提案手法を適用したプログラムに対し OSCAR コンパイラの自動並列化を適用したが，今後は完全自動並列化に向けて コンパイラに提案手法の実装を行う予定である．

参考文献

[1] 尾崎弘，谷口慶治：画像処理: その基礎から応用まで，共立出版 (1988).
[2] 市川忠男：シャノン・ノイマン・ディジタル世界，森北出版 (2005).
[3] 田村秀行：コンピュータ画像処理:応用実践編: 第 1 巻，総研出版 (1990).
[4] Wolfe, M. J.: High Performance Compilers for Parallel Computing, *Addison-Wesley Longman Publishing Co.* (1995).
[5] Banerjee, U. K.: Loop Parallelization, *Kluwer Academic Publishers Norwell* (1994).
[6] APC: http://www.apc.waseda.ac.jp/.
[7] 木内雄二：イメージセンサの基礎と応用，日刊工業新聞社 (1991).
[8] 前田友英，芹沢正之：動き適応型ノイズ除去フィルタによる高画質化，パナソニック技報 Vol.56 No.4 (2011).
[9] 小幡元樹， 白子準，神長浩気，石坂一久，笠原博徳：マルチグレイン並列処理のための階層的並列処理制御手法，情報処理学会論文誌 (2003).
[10] 本多弘樹，岩田雅彦，笠原博徳：Fortran プログラム粗粒度タスク間の並列性検出手法，電子情報通信学会論文誌

(1990).
[11] 笠原博徳：並列処理技術，コロナ社 (1991).
[12] 黒田隆男：イメージセンサの本質と基礎, コロナ社 (2012).
[13] 日立製作所：SR16000：技術計算向けサーバ，http://www.hitachi.co.jp/Prod/comp/hpc/.