ルーフラインモデルに基づくベクトルプロセッサ向けプログラム最適化戦略

佐藤義永[†] 永岡龍一[†] 撫佐昭裕[††]
江川隆輔[†] 滝沢寛之[†]
岡部公起[†] 小林広明[†]

ベクトルプロセッサにおけるピーク演算性能に対するメモリバンド幅 (Bytes/Flop, 以下，B/F) は年々減少している．このため近年のベクトルプロセッサは，低下する B/F を補うためにキャッシュメモリを搭載している．本研究の目的は，キャッシュメモリを有するベクトルプロセッサにおいて高い実行効率を実現するプログラム最適化手法を確立することである．複数のプログラム最適化手法を適用する場合，各々の最適化パラメータにおいてトレードオフが存在する．さらに，これらの最適化を併用する場合には互いの最適化パラメータが影響しあうため，体系的に最良のトレードオフを探索するプログラム最適化戦略が求められる．

本論文では，キャッシュを有するベクトルプロセッサの性能を引き出すためのプログラム最適化戦略を提案する．最適化戦略では，最適化の対象となるプログラムのボトルネックをルーフラインモデルにより解析し，ボトルネックを改善する最適化手法を対象プログラムに施す．また，最適化手法として本論文では，ループ変換によるプログラム最適化であるループアンローリングとキャッシュブロッキングに着目する．さらに適用する最適化パラメータは，グリーディサーチアルゴリズムによる探索で決定する．そして，複数のアプリケーションを用いて実効性能と消費エネルギを評価し，本提案手法の優位性を示す．評価結果より，提案手法を用いることで実効性能が改善でき，さらに消費エネルギを大幅に削減できることが明らかになった．

A Performance Tuning Strategy Based on the Roofline Model for Vector Processors

YOSHIEI SATO,[†] RYUICHI NAGAOKA,[†] AKIHIRO MUSA,[††]
RYUSUKE EGAWA,[†] HIROYUKI TAKIZAWA,[†] KOKI OKABE[†]
and HIROAKI KOBAYASHI [†]

Over the last decade, the ratio of memory bandwidth to computational performance (Bytes/Flop, B/F) of vector processors has decreased. To cover the insufficient B/F, modern vector processors are equipped with an on-chip vector cache. The purpose of this work is to establish a performance tuning strategy to exploit the potential of modern vector processors. When several tuning techniques are applied to an application, there is an explicit trade-off between individual tuning techniques. Therefore, a tuning strategy which finds a good trade-off between individual tuning techniques is required.

In this paper, a tuning strategy based on the roofline model for modern vector processors is proposed. We focus on two important loop transformations. One is loop unrolling and the other is cache blocking. To decide which of loop unrolling and cache blocking is performed first, the roofline model is employed to analyze the performance bottleneck of a target application. Then, the optimization effective to remove the bottleneck is applied to the application preferentially. To determine the number of loop unrolls and the cache blocking size, we employ the greedy search algorithm. The superiority of the strategy is evaluated with several applications. The evaluation results show that the strategy can improve the performance and also drastically reduce the energy consumption.

1. はじめに

先端科学や工学分野において不可欠な科学技術シミュレーションの発展には，ベクトル型スーパーコンピュータが大きく貢献している[1]．大規模科学計算においてベクトル型スーパーコンピュータが有している優位性には，ベクトルプロセッサが持つ高い実効メモリバンド幅が大きく寄与している．しかし近年，CPU の演算性能は半導体加工技術の微細化に伴い向上している一方で，CPU に実装可能な入出力ピン数の物理的限界からメモリバンド幅のさらなる向上が困難になってきている．このため，メモリバンド幅と演算性能の比である B/F (Bytes/Flop) は減少傾向にある．結果として，演算性能に見合ったデータ供給がなされず，理論最大演算性能に対する実効性能の比である**実行効率**が大きく低下する[2]．

低下する B/F の影響を軽減するために、ベクトルプロセッサ用のキャッシュメモリが提案されている[3]．プロセッサ内部にキャッシュメモリを設けることで，キャッシュメモリとベクトルレジスタ間のメモリバンド幅を，メインメモリのメモリバンド幅よりも高く保つことが可能となる．現在，ベクトルプロセッサ向けのキャッシュメモリは Cray X1 や NEC SX-9 に搭載されている．これらのスーパーコンピュータの性能評価に関する研究はすでに多数報告されており，その高い演算性能が明らかとなっているが，ベクトルプロセッサに新たに導入されたキャッシュメモリ向けのプログ

† 東北大学
TOHOKU University
†† 日本電気株式会社
NEC Corporation

ラム最適化手法は未だ確立されていない[4)5)]．大規模科学技術計算ではキャッシュメモリ容量を超える大量のデータを扱うことから，限られたキャッシュメモリ容量を効率よく利用するプログラム最適化手法が高い実効性能を達成するための重要な鍵となる．

従来のベクトルプロセッサ向けの最適化では，ベクトル長を長くして最内ループのベクトル演算量を増加させてきたが，キャッシュメモリを用いる場合にはベクトル長を長くすることで逆に性能が低下する恐れがある．これは，一度の命令で扱うベクトル演算量を増加することにより，メモリアクセスレイテンシを隠蔽できる一方で，参照の局所性の減少により，キャッシュヒット率が低下するためである．また，キャッシュブロッキングはキャッシュヒット率を向上させるプログラム最適化手法であるが，ブロッキングによりベクトル長が短くなるため，性能が低下する恐れがある．したがって，ベクトル長とキャッシュヒット率の最良のトレードオフを体系的に探索するプログラム最適化手法が求められる．

本研究の目的は，キャッシュメモリを有するベクトルプロセッサのためのプログラム最適化戦略を構築することである．本論文ではルーフラインモデルに基づいたベクトルプロセッサの性能モデルを用いて，性能のボトルネック解析を行う．ボトルネック解析に基づき，ループ変換による最適化を組み合わせるプログラム最適化戦略を提案する．最後に，実アプリケーションを用いて実効性能と消費エネルギの両面から提案手法の有効性を評価する．

本論文では，2 章において関連研究について述べる．その後，3 章においてベクトルプロセッサのためのルーフラインを用いた性能モデル，および 4 章において性能解析に基づくプログラム最適化戦略について述べる．そして，5 章において実アプリケーションによる性能評価を示す．最後に，6 章で本論文のまとめを述べる．

2. 関 連 研 究

性能向上を実現するプログラム最適化手法として，ループアンローリングやキャッシュブロッキング等のループ変換手法が広く利用されている．しかし，これらの最適化手法はハードウェアの特徴と密接に関連しており，最も高い性能を得る最適化の組合せを導くためには多大な労力を要する．

そこで，Wolf らはキャッシュミス数やレジスタ数，ループのオーバーヘッドに着目して，実行時間が最も短縮される最適化の組合せの見積りを行っている[6)]．この見積りによって得られた最適化の組合せを用いて，SPEC と RICEPS のカーネルプログラムを用いた評価を行った結果，コンパイラによる自動最適化よりも高い実効性能が得られることを示している．Gannon らは各配列に対して再利用されるデータサイズを見積り，キャッシュヒット率の予測に基づくプログラム最適化戦略を提案した[7)]．配列ごとのキャッシュヒット率を予測することで，データの局所性を変化させるループ交換やキャッシュブロッキングなどのプログラム最適化を活用できる．また，McKinley らはループ内の時間的・空間的局所性を導くコストモデルを構築し，これを基にデータの局所性が向上するようプログラム最適化を施す手法を提案した[8)]．

しかし，これらの手法ではスカラプロセッサを想定しており，ベクトルプロセッサ向けの最適化の指針とは大きく異なっている．例えばベクトルプロセッサでは，データレベル並列性を有する演算に対してベクトル命令を適用するベクトル化を行うことにより，多数の演算器を並列に稼働させて演算の高速化を図っている．したがってベクトルプロセッサでは，ベクトル化の対象となる最内ループのベクトル長を維持するために，最外ループに対してアンローリングを適用している．さらに，スカラプロセッサでは扱えるレジスタ数が少ないことを想定してアンローリング段数の推定を行っているが，ベクトルプロセッサは大容量のベクトルレジスタを備えているため，アンローリング段数はベクトルレジスタサイズの制約を受けにくい．一方で，アンローリング段数の増加に伴いベクトルレジスタ間のデータ転送が増加して性能が低下する．これらの影響は，プログラムに依存するため，事前に把握することは困難である．そのため，NEC SX-9 向け Fortran コンパイラ[9)] による自動最適化では，アンローリングの自動化は可能であるが，適用するアンローリング段数はプログラマが明示的に指定しなければならない．

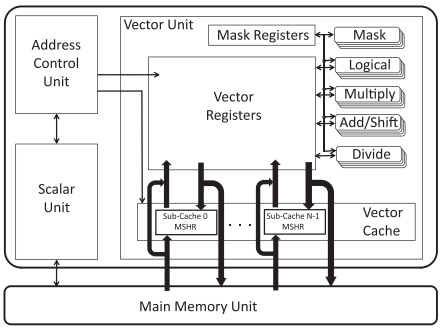

図 1 ベクトルキャッシュを有するベクトルプロセッサ

また，スカラプロセッサと異なりベクトルプロセッサではベクトル命令のオーバーラップによりメモリレイテンシを隠ぺい可能であるため，レイテンシに対するキャッシュの効果は低い．さらに，ベクトルプロセッサはスカラプロセッサと比べて高いメモリバンド幅を有しているため，キャッシュヒット率が低くても極端な性能低下は起こりにくい．一方で，キャッシュブロッキングの適用に伴うループ変換によりベクトル長が低下し，実効性能が減少する恐れがある．[10)]．したがって本研究では，キャッシュを有するベクトルプロセッサの特徴を踏まえ，効果的なプログラム最適化の組合せ手法について検討する．

3. ベクトルプロセッサのための性能モデル

3.1 ベクトルキャッシュを有するベクトルプロセッサ

初めに，本研究で想定するプロセッサモデルについて述べる．ここで，本研究で用いるオンチップメモリを**ベクトルキャッシュ**と呼称する[11)]．ベクトルキャッシュを備えるベクトルプロセッサの構成を図 1 に示す．ベクトルキャッシュは，各メモリポートに対応したサブキャッシュとして分割されている．ロードするデータをベクトルキャッシュに保持して，再度データを利用する際に，ベクトルキャッシュからベクトルレジスタに短いメモリアクセスレイテンシと高いメモリバンド幅でデータを供給する．またベクトルキャッシュには，限られたキャッシュ容量を効率的に利用するために，時間的局所性の高いデータのみをキャッシュに格納してそれ以外のデータはキャッシュをバイパスする機構を有している．さらに，先行メモリアクセスと後続のメモリアクセス間で同一メモリアドレスへのアクセスを防ぐ Miss Status Handling Register (MSHR)[12)] を備えており，高いキャッシュヒット率と，限られたオフチップメモリバンド幅をオンチップキャッシュの高

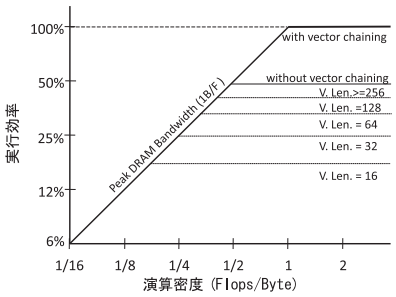

図 2 ベクトルプロセッサのためのルーフラインモデル

いバンド幅で補うことにより，高い実効メモリバンド幅を実現している．

3.2 ベクトルプロセッサのためのルーフラインモデル

科学技術アプリケーションでは大量のメモリアクセスが行われることから，実効性能はメモリバンド幅に強く依存している．メモリバンド幅を考慮したプロセッサの性能モデルとして，ルーフラインモデルが Williams らにより提案されている[13]．このモデルでは，演算性能とメモリバンド幅のどちらか一方がアプリケーションの実効性能を律速すると仮定する．アプリケーションに含まれる演算量とメインメモリから転送されるデータ量の比を**演算密度** *(Flops/Byte)* として定義した場合，演算密度が低いアプリケーションではメモリバンド幅が実効性能を律速し，演算密度が高いアプリケーションでは演算性能が実効性能を律速する．そこで，演算密度から得られる理論実効性能をルーフラインとして表現し，プロセッサの特徴によって律速される性能を上限 (シーリング) としてルーフラインモデル中で表現する．

図 2 に本論文で対象とするベクトルプロセッサのルーフラインモデルを示す．本研究で想定するベクトルプロセッサは，図 1 に示すように加算器と乗算器を独立に備えている．加算命令と乗算命令が連続する場合は，加算器の出力を直接乗算器の入力として利用するベクトルチェイニング動作を行い，加算命令と乗算命令の並列処理を行う．そこで，ルーフラインモデルでは，加算・乗算の両パイプラインが並列に稼働する際の実効性能をピーク性能とする．逆に，カーネルプログラム中に含まれる加算命令と乗算命令の割合が異なる場合では，チェイニング動作ができず一方の演算パイプラインのみの利用となる．したがって，チェイニング動作が不可能な場合の実行効率はピーク性能の 50%まで低下する．さらに，演算性能とメモリバンド幅の比を 1B/F と想定すると，アプリケーション実行時の演算密度が 1 以上あればデータ転送に比べ演算量が多くなるが，1 を下回る場合ではデータ転送量が演算量を上回りメモリバンド幅がボトルネックとなる．そのため，演算密度の減少に応じて，メモリバンド幅が制約となり実行効率が低下する．

またベクトルプロセッサの性能は，ベクトル演算命令で並列に処理を行うデータ量を表すベクトル長の影響を受ける．ベクトル長の影響について，Hockney らがベクトル長と性能の関係を定式化している[14]．ベクトル命令の実行に要する時間は，ベクトル命令のフェッチとデコードに要する時間，ベクトル演算パイプラインの立ち上がり時間，各ベクトル要素を演算する時間の 3 種類に分けられる．ベクトル長が短い場合では，ベクトル要素を演算する時間に比べ，フェッチやデコードに要する時間とベクトル演算パイプラインの立ち上がり時間が相対的に大きくなるため性能が低下する．以上の要素から Hockney の公式をまとめると，実効性能 r は式 (1) で与えられる．なお，ベクトル命令のフェッチとデコードに要する時間を s，演算パイプラインの立ち上がり時間を l，ベクトル長を n，そしてピーク演算性能を r_∞ として示す．

$$r = \frac{r_\infty}{1 + \frac{s+l-1}{n}} \quad (1)$$

これにより，ベクトル長が小さくなるにつれ，実行効率も大幅に低下することがわかる．以上の制約条件から，図 2 に示すルーフラインモデルを構築することができる．次章において，構築したルーフラインモデルを用いてアプリケーションのボトルネックを解析し，プログラム最適化に応用する．

4. ボトルネック解析に基づくプログラム最適化戦略

本研究の目的は，キャッシュを有するベクトルプロセッサの性能を引き出すために，複数のループ変換によるプログラム最適化手法を効果的に組み合わせる最適化戦略を構築することである．そこで本章では，初めに前章で述べたルーフラインモデルを用いてボトルネック解析を行う．その後，ボトルネックを解消するようにプログラム最適化を適用する．プログラム最適化として，ループ変換によるプログラム最適化であるループアンローリングとキャッシュブロッキングに着目し，これら最適化手法の特徴と性能との関係について述べる．最後に，ボトルネック解析に基づいて二種類のプログラム最適化手法を適用するプログラム最適化戦略について述べる．

4.1 ルーフラインモデルを用いたボトルネック解析

ルーフラインモデルとアプリケーション実行時の実行効率，及び演算密度から性能のボトルネックを類推することが可能である．図 3 に実行効率がメモリバンド幅により律速される場合を示す．このようなメモリのボトルネックを解消するためには，キャッシュヒット率を高めて演算密度を向上するプログラム最適化手法が有効であると考えられる．

一方，実行効率がメモリバンド幅に律速されていないにもかかわらず，十分な性能が得られない場合の例を図 4 に示す．演算密度が高くとも，ベクトル長が短かったり，並列演算パイプラインを十分に利用できない場合は実行効率が低くなる．そこで，ベクトル長や最内ループの演算命令数を増やす等により，性能のボトルネックを解消する必要がある．

4.2 プログラム最適化手法

本研究では，演算のボトルネックを解消するプログラム最適化手法としてループアンローリングに着目し，メモリバンド幅のボトルネックを解消するプログラム最適化手法としてキャッシュブロッキングに着目する．これらループ変換によるプログラム最適化は，性能に対し一長一短の性質を有している．そのため，各最適化を組み合わせて最大の性能を得るには，効果的な組合せで最適化を施す必要がある．例えば，ループアンローリングを適用した後にキャッシュブロッキングを適用する場合，ベクトル長とキャッシュヒット率の最適なトレードオフを実現できない恐れがある．これは，キャッシュヒット率を考慮せずにループアンローリングを施すこととなり，その後キャッシュブロッキングを適用しても十分なキャッシュヒット率を得られないためである．したがって，本研究では最適化を適用する順序を考慮した最適化の組み合わせを検討する．そこで本節では，ループアンローリングとキャッシュブ

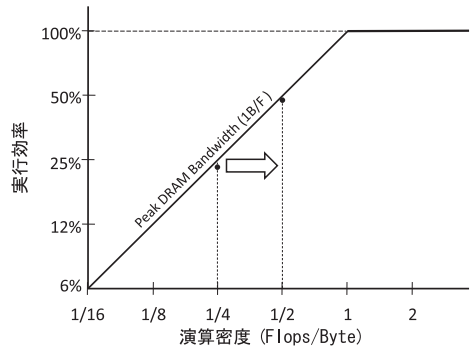

図 3 メモリバンド幅ネックの場合のルーフラインモデル

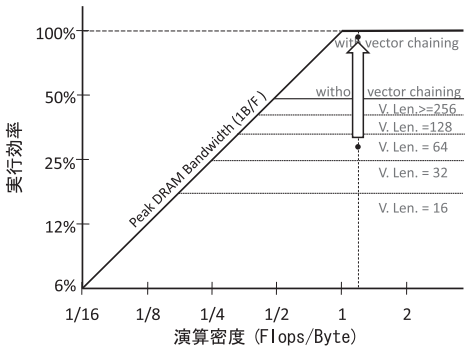

図 4 演算ネックの場合のルーフラインモデル

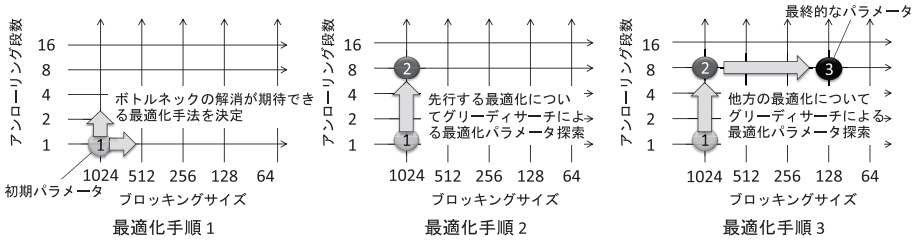

図 5 最適化戦略の手順

ロッキングの性質について述べる.

4.2.1 ループアンローリング

ループアンローリングは，ループを展開することで複数の繰り返し演算を一回の繰り返しで処理する手法である．ループアンローリングにより，ループの繰り返しに要する分岐命令が削減される. さらに，複数の繰り返しに含まれるメモリアクセスのうち共通するメモリアドレスが，一回のアクセスで済むためメモリアクセス数を削減できる．したがって，最内ループにおける演算数が増加し，演算パイプラインを効率的に利用することが可能となり実効性能が向上する[15].

一方で，複数の繰り返しにおいて共通するメモリアクセスがなく，ループアンローリングによってメモリアクセス数が削減できないアプリケーションでは，ループの展開により内側ループに含まれるメモリアクセス数が増加する．その結果，外側ループにおける時間的局所性が減少し，キャッシュヒット率が低下する. したがって，ループアンローリングは，最内ループにおける演算量向上とキャッシュヒット率減少のトレードオフを有するプログラム最適化手法である.

4.2.2 キャッシュブロッキング

キャッシュブロッキングは，キャッシュサイズに合わせてループを分割することで，参照の局所性を高める最適化手法である．ループを分割することで，再利用性のあるデータをより多くキャッシュに格納することができ，キャッシュヒット率の向上が期待できる.

一方，ブロッキングを行うためのループ分割によりベクトル長が短くなり，一度のベクトル命令で処理するデータ数が減少する．また，ループ分割によりループ分岐回数が増加するため，ループのオーバーヘッドも増加する．このため，ブロッキングするループ長に応じてキャッシュヒット率が向上する一方，ベクトル長の短縮やループのオーバーヘッドによる性能低下というトレードオフの関係が存在する.

4.3 プログラム最適化戦略

前節で述べた通り，ループアンローリングとキャッシュブロッキングの最適化パラメータにはトレードオフの関係がある. したがって，二種類の最適化を併用する場合には，各最適化間において最適化パラメータが影響を及ぼしあう. そこで，本最適化戦略ではループアンローリングとキャッシュブロッキングの適用順序をルーフラインモデルにより検討する. 例えば，ルーフラインモデルによる解析によりメモリバンド幅がボトルネックである場合には，キャッシュブロッキングを先に適用し，その後ループアンローリングを適用する. 性能のボトルネックを解消するプログラム最適化の効果を最大限に引き出し，その後もう一方のプログラム最適化を適用することで二種類の最適化における最良のトレードオフを導く.

次に，最適化パラメータの決定方法について述べる. アンローリング段数やブロッキングサイズを変化させた際の実効性能の傾向は，最大値を頂点として単調増加から単調減少となることが報告されている[16] そこで本手法では，グリーディサーチアルゴリズム[17] を用いて最適化パラメータの探索を行う. このアルゴリズムは，最適化前のパラメータを初期条件として徐々に最適化パラメータを変えていき，実効性能が最も高くなる最適化パラメータを探索する. これにより，各最適化における性能の傾向から，最適化パラメータの全探索を行う必要がなく最適解を得ることができる[18]. しかし二種類の最適化を組み合わせる場合，独立に探索して得られた最適解は，他方の最適化の影響で最適解でなくなる恐れがある. したがって，先行する最適化を適用した後に，後続の最適化における最適化パラメータを探索する必要がある.

図 5 に最適化戦略の手順を示す. 初めにルーフラインモデルを用いたボトルネック解析により適用するプログラム最適化の順序を決定する. そして先に適用するプログラム最適化のパラメータをグリーディサーチ

表 1 評価パラメータ

Base System Architecture	NEC SX-8
Main Memory	DDR-SDRAM
Vector Cache	SRAM
Vector Cache Size	1MB
Number of Sub-caches	32
Associativity	2-Way
Cache Policy	LRU，Write-through
Cache Bank Cycle	5% of memory cycle
Cache Latency	15% of memory latency
Line Size	8B
Total MSHR Entries (Per Sub-caches)	8192(256)
Memory-Cache bandwidth per flop/s	1B/F
Cache-Register bandwidth per flop/s	4B/F

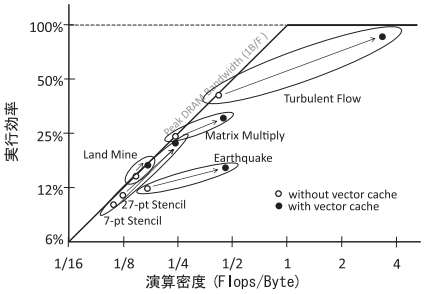

図 6 各アプリケーションにおけるルーフラインモデル

により探索して適用する. その後，他方のプログラム最適化について再度グリーディサーチによるパラメータ探索を行い最適化を適用する. 結果として，本最適化戦略により二種類の最適化パラメータの効果的な組み合わせが実現できる.

5. 性能評価

5.1 評価環境

本評価では，新たに開発したソフトウェア制御の可能なオンチップキャッシュを導入した SX ベクトルプロセッサのタイミングシミュレータを用いて評価を行う. 本評価で用いるタイミングシミュレータは，実行される各命令の実行タイミングや，実行時のサイクル数，メモリアクセスサイクル数等を計測することで，全実行時間やキャッシュヒット率を算出するシミュレータである．さらに，SX タイミングシミュレータに与えるパラメータを変更することで，ベクトルキャッシュの有無やプロセッサの構成，メモリバンド幅の変更も可能である．本評価で用いるシミュレーションパラメータを表 1 に示す.

5.2 評価アプリケーション

評価には，行列積 (Matrix Multiply) と 7 点および 27 点のステンシル計算 (7-pt Stencil, 27-pt Stencil) を行うカーネルに加え，東北大学サイバーサイエンスセンターのスーパーコンピュータで利用されている実アプリケーションを用いる. 以下に実アプリケーションの詳細について述べる.

Earthquake Earthquake[19] は地震滑りの伝搬速度における観測を検証するための，3 次元プレート沈み込みモデルの数値解析シミュレーションプログラムである．この数値シミュレーションでは大規模な地震が発生する単独のアスペリティを想定したシミュレーションが可能であり，余効すべりの到着時間を再現することが可能である. アスペリティ領域の外側で摩擦パラメータが均一であると仮定し，岩石実験結果から得られた速度・状態依存摩擦構成則を適用している. この常微分方程式を解くために，本アプリケーションでは適応刻み幅制御を伴ったルンゲクッタ法を用いている.

Land Mine Land Mine[20] は地雷検知を行う地中レーダの性能評価を行うプログラムである．本アプリケーションは，地雷検知用アレイアンテナと不均質地下媒質のモデル化を行い，3 次元 FDTD(Finite-Difference Time-Domain) の解法を用いている. 実験室を用いた実験では不可能な土中水分含有率の精密制御をシミュレーション空間で行うことにより，土中水分含有率が計測に及ぼす影響を定量的に評価することが可能である.

Turbulent Flow Turbulent Flow[21] はチャネル乱流における流体線の解析を行う直接数値計算のプログラムである．支配方程式は，連続の式，およびナビエストークス方程式であり，計算アルゴリズムには Fractional step 法を用いている．空間離散化には有限差分法を用い，差分精度としては流れ方向とスパン方向に 4 次精度中心差分法を，壁垂直方向には 2 次精度の不等間隔格子を用いている. 時間進行は，壁垂直方向の粘性項に対しては 2 次精度クランク・ニコルソン法を，その他の項は 2 次精度アダムス・バシュフォース法を用いている.

5.3 各アプリケーションにおけるボトルネック解析

本節では，各アプリケーションにおける性能のボトルネック解析を行う．図 6 に評価から得られたルーフラインモデルを示す. 図中の白丸はキャッシュを用いない場合，黒丸はキャッシュを用いる場合を示している. キャッシュを用いない場合では，全てのアプリケーションにおいて演算密度が 1 を下回っており演算量に比べてデータ転送量が多くなっている. そのため，Matrix Multiply，各 Stencil，Land Mine，および Turbulent Flow のアプリケーションはメモリバンド幅に律速されていることがわかる. 逆に Earthquake ではメモリバンド幅の上限からは離れており，演算がボトルネックとなってデータ転送性能を十分に活かせていないことがわかる.

次にキャッシュメモリを用いる場合では，全てのアプリケーションで性能向上しているが，各アプリケーション毎に得られる性能向上の割合が異なることが見て取れる. まず各 Stencil と Turbulent Flow では約 2 倍の性能向上が得られている. これは，両アプリケーションとも時間的局所性が大きく，高いキャッシュヒット率を得ることにより，実効メモリバンド幅が増加したためである．しかし，各 Stencil は未だにメモリバンド幅に律速されている．また Land Mine では，時間的局所性が小さくキャッシュを活かすことができず性能向上はわずかである. したがって，各 Stencil と Land Mine はプログラム最適化による時間的局所性の性能改善が求められる.

Matrix Multiply ではキャッシュを用いることで実行効率は向上するが，演算密度の向上と比べて実行効率の向上は少ない. これは，キャッシュの利用によりメモリバンド幅のボトルネックが解消する一方で，最内ループの演算量が少なく分岐命令処理の遅延が生じているためと考えられる. また，Earthquake では元々メモリネックではないためキャッシュによって得られる効果は小さい. したがって，これらのアプリケーションはループ内部の演算を増やし，演算パイプラインを効率的に稼働させる必要がある.

以上のルーフラインによる解析結果で得られたアプリケーションの特徴を基にして最適化を施す. 次節で各アプリケーション毎の最適化の評価について述べる.

5.4 最適化戦略の評価

図 7 に各アプリケーションへ最適化戦略を適用した際の評価結果 (proposed strategy) を示す. 縦軸はキャッシュを用いない場合との性能向上率を示している. な

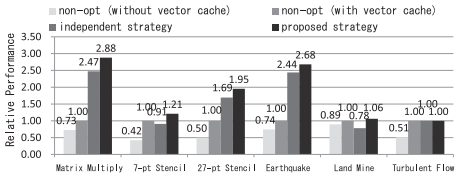

図 7 最適化戦略による性能向上

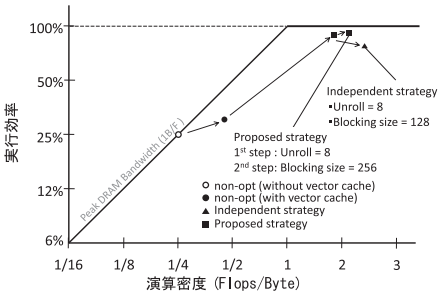

図 8 最適化戦略適用時のルーフラインモデル (Matrix Multiply)

お本評価では，提案する最適化戦略の有効性を示すために，各プログラム最適化において独立にパラメータ探索を行ってプログラム最適化を適用する戦略 (independent strategy) も評価し，提案手法と比較する．

評価結果から，全アプリケーションに渡り提案戦略において最も高い性能を得られることがわかる．次節では各アプリケーションに対する最適化戦略の影響をボトルネックで分類して考察する．

5.4.1 演算がボトルネックとなるアプリケーションに対する最適化戦略の効果

演算がボトルネックとなるアプリケーションである Matrix Multiply と Earthquake に提案戦略を適用すると，図 7 で示すように大幅な性能向上が得られる．最適化戦略の効果を詳細化するために，最適化戦略によって適用されるプログラム最適化の効果をルーフラインモデルで解析する．

図 8 に Matrix Multiply に最適化戦略を適用した際のルーフラインモデルを示す．演算がボトルネックであるため，初めにループアンローリングを適用することになり，実行効率は 8 段のループアンローリングにより 33.8%から 97.2%まで向上する．ループアンローリングが大きな効果を持つ理由は，ループの分岐命令削減に加え，ベクトルロード命令が削減されるためである．本アプリケーションでは，外側ループ間でデータを共有し，キャッシュメモリからデータ転送できる配列が存在する．この配列はアンローリングを適用することにより最内ループで共有され，レジスタ間でアクセス可能となる．その結果，ロード命令が削減されカーネル内の演算命令の割合が増加し，演算パイプラインを効率的に稼働させることができる．

ループアンローリングを適用した後，さらにキャッシュブロッキングを適用することで，残りのロード命令においてキャッシュが利用されて 97.9%まで性能が向上する．その一方で，独立に適用する場合では，キャッシュブロッキングにより性能が低下している．

アンローリングとブロッキングにおける性能のトレードオフを解析するために，図 9 にアンローリングとブロッキングの性能の関係を示す．図より，アンローリングを行わない場合はブロッキングサイズが 128 の場合が最適であるが，アンローリング段数を増やすに

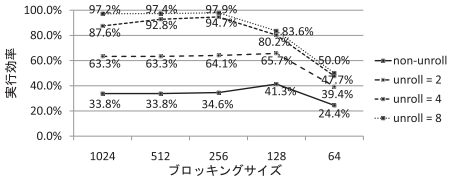

図 9 アンローリング段数とブロッキングサイズの関係 (Matix Multiply)

したがって，128 のブロッキングサイズは最適でなくなる．これは，アンローリング段数が増加するにしたがってベクトルロード命令が減少し，演算命令の割合が増加することが影響していると考えられる．ロード命令の減少によりキャッシュメモリの効果が小さくなる一方で，演算命令の割合が増加し演算パイプラインの立ち上がり時間が影響してくる．そしてブロッキングサイズを短縮する，すなわちベクトル長を短縮することにより演算パイプラインの立ち上がり時間が増加し，性能の低下を招く．このように，ループアンローリングによってプログラムの特徴が変化する可能性があるため，キャッシュブロッキングを適用する場合にはアンローリング適用後に再度性能評価を行い，最適解を導く必要がある．

5.4.2 メモリバンド幅がボトルネックとなるアプリケーションに対する最適化戦略の効果

メモリバンド幅がボトルネックとなる各 Stencil と Land Mine に提案戦略を適用する場合，図 7 で示すように性能の向上率は大きく異なる．

Stencil では，7-pt Stencil に比べて 27-pt Stencil の方が性能向上の割合が大きい．提案戦略にしたがって適用された最適化を比較すると，どちらもキャッシュブロッキングによる最適化のみを適用している．しかし，7-pt に比べて 27-pt の方がより差分で参照する配列数が多く，7-pt ではブロッキングサイズを 128 にすることで実行効率が 23.4%から 28.6%へと向上するのに対して，27-pt では同様のブロッキングサイズで実行効率が 23.4%から 45.7%まで向上する．一方，独立に最適化を適用する場合，7-pt ではループアンローリングを適用することでキャッシュヒット率が低下し，その結果，実効性能が低下する．また，27-pt ではアンローリングによって性能が向上するが，ブロッキングを併用するとベクトル長の影響により性能が低下する．

しかし，7-pt と異なり 27-pt では，ブロッキングのみを施す場合よりもアンローリングのみを施す場合の方が高い性能が得られる場合がある．これは，ステンシルの形の違いによって，アンローリングによる影響が異なるためである．アンローリングがメモリアクセスパターンへ与える影響を考えると，アンローリングによって隣接する格子を同時に計算するため，共通するメモリアドレスが 1 回のアクセスで済み，メモリアクセスの削減が可能である．n 段のアンローリングにより削減可能な要素数は，7-pt Stencil では $2(n-1)$ 要素であるのに対して，27-pt Stencil では $18(n-1)$ 要素である．このように，大きなステンシルの方がアンローリングに伴うメモリアクセス削減量が多く，アンローリングの利点を享受しやすい．また，アンローリングがキャッシュヒット率へ与える影響を考えると，アンローリングを行わない場合では各 Stencil 共に連続する 3 平面を保持できれば容量性ミスを回避できるが，アンローリングを行う場合では各 Stencil 共に隣接する格子の平面も保持する必要があるため，データ量が増加し，容量性ミスによりキャッシュヒット率が低下

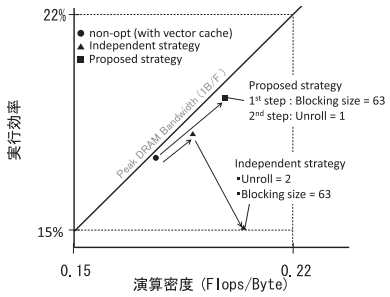

図 10　最適化戦略適用時のルーフラインモデル (Land Mine)

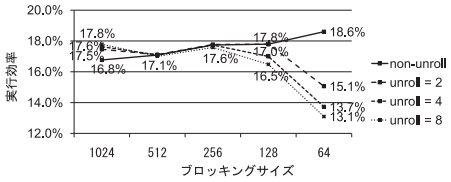

図 11　アンローリング段数とブロッキングサイズの関係 (Land Mine)

する．このことから，ステンシルの構造によって，アンローリングの有効性が異なるため，構造に合わせた最適化パラメータの設定が必要となる．

次に，Land Mine について最適化戦略を適用した際のルーフラインモデルを図 10 に示す. 提案戦略を用いる場合では，まずボトルネックであるメモリバンド幅を改善するためにキャッシュブロッキングが適用され，その結果，実行効率は 16.8%から 18.6%まで向上する. 一方で，独立に適用する場合では，各最適化で最適解である 2 段のアンローリングと 63 のブロッキングサイズを適用している. しかし，アンローリングによるキャッシュヒット率の低下に加え，短ベクトルによる立ち上がり時間の増加を招き，お互いに悪影響を及ぼしあい，結果として大幅な性能低下を招く. このように，メモリバンド幅がボトルネックのアプリケーションではアンローリングが悪影響を及ぼす恐れがあるため，キャッシュブロッキング適用後にループアンローリングの最適なパラメータを探索する必要がある.

図 11 に Land Mine におけるアンローリング段数とブロッキングサイズの関係を示す．図より，アンローリングを初めに適用する場合では，アンローリングにより性能は向上するが，次にブロッキングを適用したとしても性能向上は得られないため，探索によって得られる最適化パラメータは，8 段のアンローリングのみの局所解に陥り，最良のパラメータを得ることができない．一方，キャッシュブロッキングを初めに適用する場合では，ブロッキングによる性能向上を活かした上でアンローリングの検討が可能である．このことから，ボトルネック解析を用いることにより，最初に効果的な最適化を選択することができ，結果として最良の最適化パラメータを得ることが出来る．

5.5　消費エネルギの評価

ベクトルプロセッサにおける多大な消費電力や，それに伴う発熱を抑制するために，消費エネルギの削減は重要な課題である. そこで，本節では消費エネルギの観点から，プログラム最適化戦略の有効性を評価する．評価方法は，メインメモリとキャッシュのアクセス数をシミュレーションにより計測し，得られた結果からそれぞれで消費される消費エネルギを算出する．また，メモリアクセスにかかる消費エネルギを 100 μJ

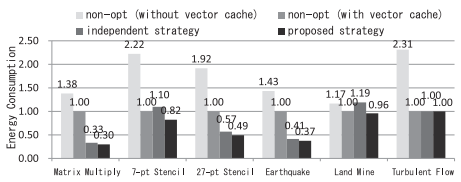

図 12　消費エネルギの削減

として評価した結果を用いる．なお，キャッシュメモリの消費エネルギは CACTI5.1[22] を用いて算出する．

各アプリケーション実行時における消費エネルギを図 12 に示す．なお，評価結果は最適化を用いない場合の結果で正規化する. 評価より，Matrix Multiply や Earthquake では提案戦略を用いることで消費エネルギが 7 割削減可能であることがわかる. これらのアプリケーションでは，アンローリングによりメモリアクセス回数が削減可能なことに加え，残りのメモリアクセスの多くもキャッシュから転送することができるため，メモリアクセスに関わる消費エネルギが大幅に削減される．このように，消費エネルギの大きいメインメモリへのアクセスが，消費エネルギの小さいキャッシュへのアクセスに置き換わることで，アプリケーション全体の実行に要する消費エネルギの削減が可能となる. また，Stencil や Turbulent Flow ではキャッシュを活用することで実行時間の短縮の効果もあり 6 割の消費エネルギ削減が達成できる. Land Mine でも同様にキャッシュを利用することで消費エネルギが削減されるが，他アプリケーションと比較してキャッシュヒット率が低いため，消費エネルギの削減は 4%に留まる.

以上の結果から，キャッシュを用いることで消費エネルギを削減することが可能であり，さらに最適化によりキャッシュを活用することで消費エネルギ削減の効果はより大きくなることが示された.

6. ま と め

本研究の目的は，キャッシュメモリを有するベクトルプロセッサにおいて，その性能を最大限に活用するためのプログラム最適化戦略の確立である．そこで，ルーフラインモデルを用いたボトルネック解析に基づいてプログラム最適化を行う戦略を提案した. 本最適化戦略では，複数の最適化の適用順序を決定するために，ルーフラインモデルを用いて性能のボトルネックの解析を行う. そして，解析結果からボトルネックの解消に有利な最適化を選択し，効果的にプログラム最適化手法を組み合わせることができる.

実アプリケーションを用いて，実効性能と消費エネルギを評価した結果，実効性能では最適化適用前と比較して最大で約 3 倍の性能向上が得られ，さらに消費エネルギでは約 7 割の削減を達成した. また，互いに与える影響を考慮せずに二種類の最適化を適用する場合では，各最適化が競合し合い性能が低下するが，本提案戦略では最適化の順序付け及びパラメータ探索を行うことにより効果的に最適化が組み合わせ可能であることが明らかになった.

本論文ではルーフラインモデルの制約条件はベクトルアーキテクチャで広く用いられるシステムで与えており，対象とするプログラム最適化についてもベクトルアーキテクチャに対して与える影響について議論した．したがって，本論文では NEC ベクトルアーキテクチャに基づいて提案戦略の評価を行ったが，Cray などの他のベクトルアーキテクチャに対しても提案戦略は適用可能であると言える.

今後の課題として，高速な最適化パラメータの探索が可能である探索アルゴリズムの検討が挙げられる．考慮するプログラム最適化が増えるにつれ，最適化パラメータの探索に要する時間はその組み合わせの数だけ増加するため，高速な探索アルゴリズムが必要になると考えられる．そこで，複数の探索アルゴリズムを検討し，最適化パラメータの探索時間や，得られる最適解について定量的に評価する必要がある．

謝辞 本研究の一部は文部科学省基盤研究 (S) 課題番号 21226018 の研究プロジェクト「ペタフロップス級計算機に向けた次世代 CFD の研究開発」による．

参 考 文 献

1) Leonid Oliker, Jonathan Carter, Michael Wehner, Andrew Canning, Stephane Ethier, Art Mirin, David Parks, Patrick Worley, Shigemune Kitawaki, and Yoshinori Tsuda. Leading Computational Methods on Scalar and Vector HEC Platforms. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.
2) Hiroaki Kobayashi. Implication of Memory Performance in Vector-Parallel and Scalar-Parallel HEC Systems. In Michael Resch, Thomas Bönisch, Sunil Tiyyagura, Toshiyuki Furui, Yoshiki Seo, and Wolfgang Bez, editors, *High Performance Computing on Vector Systems 2006*, pp. 21–50. Springer Berlin Heidelberg, 2007.
3) Hiroaki Kobayashi, Akihiko Musa, Yoshiei Sato, Hiroyuki Takizawa, and Koki Okabe. The Potential of On-Chip Memory Systems for Future Vector Architectures. In Michael Resch, Sabine Roller, Peter Lammers, Toshiyuki Furui, Martin Galle, and Wolfgang Bez, editors, *High Performance Computing on Vector Systems 2007*, pp. 247–264. Springer Berlin Heidelberg, 2008.
4) Thomas H. Dunigan Jr., Jeffrey S. Vetter, James B. White III, and Patrick H. Worley. Performance Evaluation of the Cray X1 Distributed Shared-Memory Architecture. *IEEE Micro*, Vol. 25, pp. 30–40, January 2005.
5) Hiroaki Kobayashi, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, Akihiko Musa, Takashi Soga, and Yoichi Shimomura. First Experiences with NEC SX-9. In Michael Resch, Sabine Roller, Katharina Benkert, Martin Galle, Wolfgang Bez, Hiroaki Kobayashi, and Toshio Hirayama, editors, *High Performance Computing on Vector Systems 2008*, pp. 3–11. Springer Berlin Heidelberg, 2009.
6) Michael E. Wolf, Dror E. Maydan, and Ding-Kai Chen. Combining Loop Transformations Considering Caches and Scheduling. *International Journal of Parallel Programming*, Vol. 26, No. 4, 1998.
7) Dennis Gannon, William Jalby, and Kyle Gallivan. Strategies for Cache and Local Memory Management by Global Program Transformation. *Journal of Parallel and Distributed Computing*, Vol. 5, No. 5, 1988.
8) Kathryn S. McKinley, Steve Carr, and Chau-Wen Tseng. Improving Data Locality with Loop Transformations. *ACM Trans. Program. Lang. Syst.*, Vol. 18, No. 4, pp. 424–453, 1996.
9) NEC. Fortran90/sx programmer's guide. 2008.
10) Tadashi Watanabe. *Instruction Set Architecture for a Series of Vector Processors and Their Performance Evaluations*. PhD thesis, Tohoku University, 2005.
11) Akihiro Musa, Yoshiei Sato, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, and Hiroaki Kobayashi. An On-Chip Cache Design for Vector Processors. *MEDEA Workshop*, 2007.
12) David Kroft. Lockup-Free Instruction Fetch/Prefetch Cache Organization. *ISCA*, pp. 81–88, 1981.
13) Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, Vol. 52, No. 4, pp. 65–76, 2009.
14) Roger W. Hockney and Chris R. Jesshope. Parallel computers 2; architecture, programming, and algorithms. 1988.
15) Steve Carr and Ken Kennedy. Improving the Ratio of Memory Operations to Floating-Point Operations in Loops. *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 6, 1994.
16) Yoshiei Sato, Ryuichi Nagaoka, Akihiro Musa, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, and Hiroaki Kobayashi. Performance Tuning and Analysis of Future Vector Processors Based on the Roofline Model. *MEDEA Workshop*, 2009.
17) Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, pp. 127–136, 1971.
18) Samuel Williams Kaushik Datta, Vasily Volkov, Jonathan Carter, Leonid Oliker, John Shalf, and Katherine A. Yelick. Auto-tuning the 27-point Stencil for Multicore. *International Workshop on Automatic Performance Tuning*, 2009.
19) Keisuke Ariyoshi, Toru Matsuzawa, and Akira Hasegawa. The Key Frictional Parameters Controlling Spatial Variations in the Speed of Postseismic-slip Propagation on a Subduction Plate Boundary. *Earth and Planetary Science Letters*, 2007.
20) Takeo Kobayashi and Motoyuki Sato. FDTD Simulation on Array Antenna SAR-GPR for Land Mine Detection. *Proceedings of SSR2003*, 2003.
21) Takahiro Tsukahara, Kaoru Iwamoto, and Hiroshi Kawamura. Evolution of Material Line in Turbulent Channel Flow. *Proceedings of the Fifth International Symposium on Turbulence and Shear Flow Phenomena*, pp. 549–554, 2007.
22) Shamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P. Jouppi. Cacti 5.1. *HP Laboratories*, 2008.