［研究成果］

チップマルチベクトルプロセッサのためのプログラム最適化技術

佐藤義永*1　撫佐昭裕*2　江川隆輔*3
滝沢寛之*1　岡部公起*3　小林広明*3
*1 東北大学大学院情報科学研究科
*2 日本電気株式会社
*3 東北大学サイバーサイエンスセンター

ベクトルプロセッサのさらなる高性能化、高効率化を実現するために、チップマルチベクトルプロセッサ(CMVP)が提案されている。1 チップに複数の演算コアを搭載することにより、高い演算性能が実現できる一方で、ピーク演算性能に対するメモリバンド幅（Bytes/Flop、以下、B/F）は減少すると考えられる。このため CMVP 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。したがって、CMVP の性能ポテンシャルを引き出すには、従来のベクトルプロセッサ向けのプログラム最適化に加えて、キャッシュメモリを活用する最適化も考慮する必要がある。これまで本研究では、キャッシュメモリを有するベクトルプロセッサを対象として、ループアンローリングとキャッシュブロッキングの二つの最適化手法を効果的に組み合わせるプログラム最適化戦略を提案してきた。本稿では、この最適化戦略を CMVP に適用し、その有効性を実アプリケーションによる性能評価に基づき考察する。

1. 緒言

過去 30 年間、ベクトル型計算機は先端科学技術計算分野の発展に多大な貢献をしており、今後の更なる性能向上が期待されている。計算機の性能は、搭載されるプロセッサの性能に大きく依存する。科学技術計算におけるプロセッサの性能とは、実効性能（単位時間当たりに処理可能な浮動小数点演算数、Flop/s）の高さで議論することができる。半導体加工技術の進歩により、プロセッサ 1 チップ辺りに搭載されるトランジスタ数は年々増加し、より多くの演算器が搭載可能となる。しかし、単に演算器の数を増やしただけでは性能向上は得られない。なぜならば、大量の演算器を同時に稼働させるよう命令を発行する必要があるが、並列実行可能な命令数には限界があるためである。そこで、近年では 1 チップに複数のプロセッサコアを搭載するチップマルチコアプロセッサ（CMPs）が一般的になってきている。CMPs とは、従来のプロセッサを演算コアとみなし、1 チップに複数の演算コアを搭載するプロセッサである。CMPs では、命令単位よりも粗い並列性を持つスレッド単位で並列処理を行うことで、高性能化を実現している[1]。ベクトルプロセッサにおいても、科学技術計算の主要カーネル部のマルチスレッドが容易であることから、CMPs の適用が検討されてきている。本研究室では、次世代のベクトルプロセッサとして、チップマルチベクトルプロセッサ（CMVP）を提案しており、本研究の目的は CMVP の実効性能を高めるプログラム最適化手法の検討である。

CMVP では、1 チップに従来のベクトルプロセッサを複数搭載することから、演算性能の飛躍的な向上が期待できる。しかし、演算性能と比べて、チップとメインメモリ間のメモリバンド幅の向上は緩やかである。これは、メインメモリと通信するためにチップに搭載される I/O ピンの数は物理的制約により、これ以上増やせないためである。したがって、チップに搭載するコア数が増えれば増えるほど、演算性能とメモリバンド幅の性能差が拡大する。演算性能とメモリバンド幅のバランスである Bytes/Flop (B/F) に着目すると、B/F が半減することで理論最大性能に対する実効性能が半減するという報告がなされている[2]。不足するメモリバンド幅を補うために、CMVP には共有ベクトルキャッシュと呼ばれるキャッシュ機構が搭載されており、共有ベクトルキ

ャッシュの利用方法が性能向上の鍵となる。

従来のベクトルプロセッサと CMVP 向けの最適化の大きな違いは、最内ループ長の扱い方である。従来のベクトルプロセッサでは、最内ループ長を伸ばして、一度のベクトル命令で処理可能なベクトル長を長くすることにより、データ転送の遅延時間を隠ぺいしている。しかし、CMVP ではキャッシュが存在することから、ただループ長を伸ばしただけでは性能が低下する恐れがある。ループ長が長くなればなるほど、メモリアクセスするデータ範囲が大きくなるため、データ参照における時間的局所性が小さくなる可能性があり、その結果キャッシュヒット率が低下する。したがって、CMVP ではキャッシュブロッキングと呼ばれるループ長を変化させる最適化手法を用いて、適切なループ長を設定する必要がある。また、ループアンローリングもベクトルプロセッサで頻繁に利用される最適化であるが、ループを展開することにより参照の局所性が変化するため、キャッシュヒット率に影響する。したがって、CMVP において最適化を行う場合には、ループアンローリングとキャッシュブロッキングを適切に組み合わせる必要がある。このように、CMVP では、複数の最適化のパラメータとその組み合わせ方を検討しなければならず、容易に最適化を適用することは困難である。

そこで本稿では、CMVP においてループアンローリングとキャッシュブロッキングを効果的に組み合わせる最適化戦略を提案する。本戦略では、プログラムにおける性能のボトルネックに着目し、ボトルネックの解消に有効な最適化を優先的に適用することにより、最適化の効果を最大限に引き出す。また本稿では、この最適化戦略を適用した実アプリケーションによる性能評価も行い、その有効性を検証する。

2. チップマルチベクトルプロセッサ

CMVP のブロック図を図 1 に示す。CMVP は従来のベクトルプロセッサをベクトルコアとして複数備え、加えて共有ベクトルキャッシュを有している。ベクトルコアは 5 種類のベクトル演算パイプラインセットを備えると共に、連続する演算命令を処理する際に、一方の演算パイプラインの結果を次の演算パイプラインの入力として並列処理を行うベクトルチェイニングが可能である。共有ベクトルキャッシュは、32 個のサブキャッシュから構成されている。これは、メインメモリとプロセッサ間のインターリーブ転送をキャッシュ機構にも適用するためである。メインメモリの各メモリバンクから、共有ベクトルキャッシュのサブキャッシュに独立して転送可能とすることにより、各サブキャッシュから各演算コア内のベクトルレジスタへデータを並列に転送できる。これにより、高い実行メモリバンド幅を実現している。

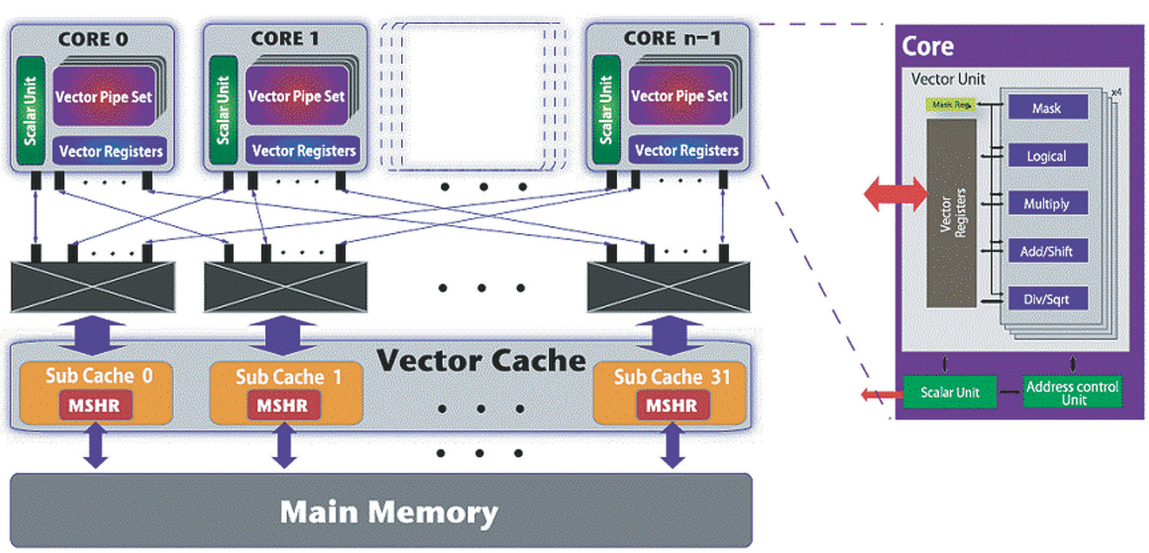


図 1 CMVP のブロック図

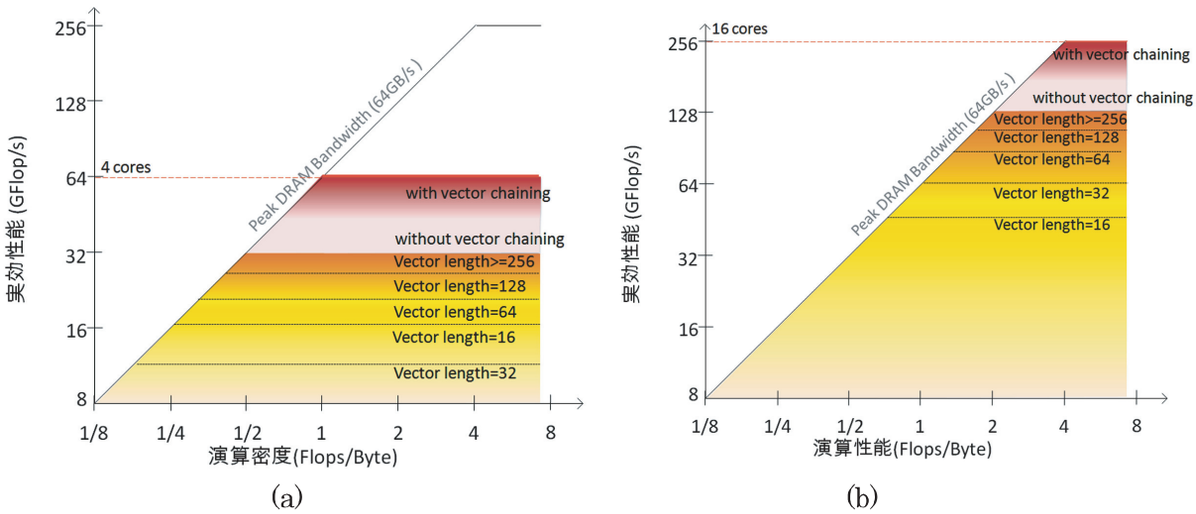


図 2 (a) 4 コア CMVP のルーフラインモデル、(b) 16 コア CMVP のルーフラインモデル

3. CMVP のためのプログラム最適化手法

3.1. ルーフラインモデルを用いた性能解析

CMVP の性能を活かすには、従来のベクトルプロセッサ向けの最適化に加えて、ベクトルキャッシュ向けの最適化が必要となり、複数の最適化手法を検討する必要がある。本稿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最適化手法の組み合わせを検討するが、双方の最適化が影響しあうため、最善の最適化パラメータを探索するのは困難である。そこで、プログラムのボトルネック解析に基づき、プログラム最適化手法を組み合わせる方法を提案する。ボトルネック解析ではルーフラインモデル[3]と呼ばれる性能モデルを用いる。

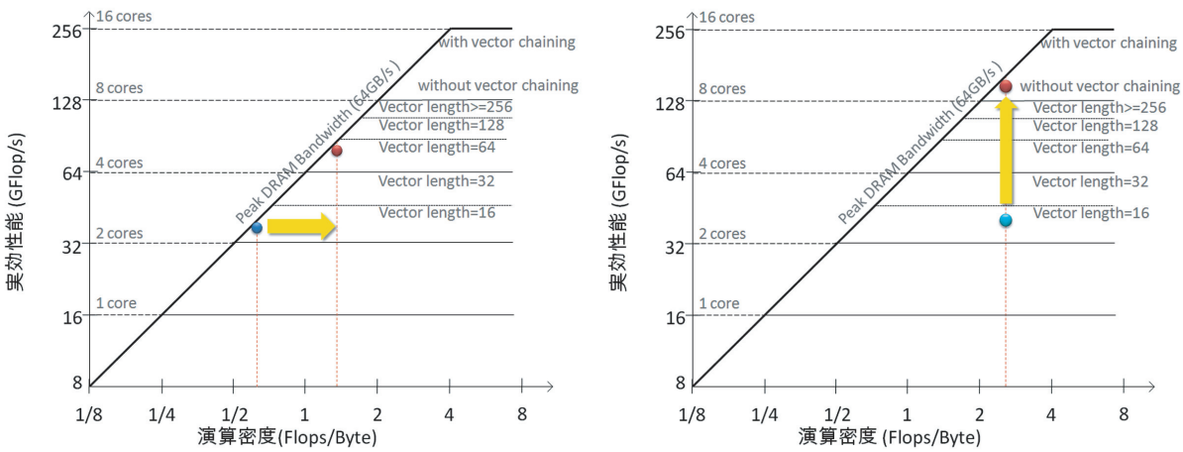
ベクトルプロセッサで利用される科学技術アプリケーションでは、プロセッサとメインメモリ間におけるデータ転送時間が実行時間の大部分を占める。そのため、システムの実効性能はプロセッサとメインメモリ間のデータ転送性能に大きく左右される。このことから、CMVP の性能モデルは CMVP の演算性能だけではなくデータ転送性能も考慮する必要がある。そこで、ルーフラインモデルを CMVP の性能モデルとして採用する。ルーフラインモデルは演算密度を性能指標として用いている。演算密度とは、アプリケーションに含まれる演算量と、演算に必要なデータ転送量(プロセッサ-メインメモリ間)の比であり、これを利用することによりシステムとアプリケーションの双方の特徴から性能を考慮することができる。なお、演算密度の単位として、Flops per Byte が一般的に用いられる。これにより、アプリケーション毎に性能のボトルネックの解析が可能となり、アプリケーションとシステムの特徴を踏まえたプログラム最適化の指針を導くことができる。

図 2 に CMVP のルーフラインモデルを示す。なお、ピーク演算性能は、コア当たりの性能(16 GFlop/s)とコア数の積である。4 コアと 16 コアの CMVP におけるルーフラインを比較すると、ピーク演算性能は 16 コアの方が高い。しかし、ピーク演算性能が得られる演算密度をみると、低い演算密度では、4 コアよりも 16 コアの方が、メモリバンド幅がボトルネックになりやすいことがわかる。したがって、このように CMVP で利用するコア数を考慮したルーフラインモデルを構築することにより、コア数に応じて性能のボトルネックを解析することが可能となる。

図3にCMVPのルーフラインモデルを用いたボトルネック解析を示す。図3 (a)は実効性能がメモリバンド幅により律速される場合である。特に、積層コア数が多い場合には、メモリバンド幅がボトルネックになりやすいと考えられる。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、実効性能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3(b) に示す。演算密度が高くとも、ベクトル長が短い、並列演算パイプラインを十分に利用できない場合は実効性能が低くなる。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。

ルーフラインモデルを用いて性能のボトルネックを解析することにより、性能改善に効果的なプログラム最適化手法を選択することが可能となる。本研究では、これを最適化の組み合わせに応用する。次節において、組み合わせる最適化手法、および性能解析に基づく最適化戦略について詳説する。



(a)　　(b)

図 3 (a)メモリバンド幅ネックの場合のルーフラインモデル、
(b)演算ネックの場合のルーフラインモデル

3.2.　ルーフラインモデルに基づくプログラム最適化戦略

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、効果的な組み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ロッキングの性質について述べる。

3.2.1. 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。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、アンローリング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。

3.2.2. 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が存在する。

3.2.3. プログラム最適化戦略

アンローリングやブロッキングを適用することにより、それぞれの最適化に応じて参照の局所性が変化するため、両最適化を同時に適用する際には互いに影響を及ぼす。したがって、最適化パラメータにはトレードオフの関係があると言える。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。そこで本手法では、グリーディサーチアルゴリズム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。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。本最適化戦略では、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最適化のパラメータをグリーディサーチにより探索して適用する。その後、他方の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。

4.　性能評価

本章では、実アプリケーションに提案戦略を適用し、CMVP における実効性能を評価することにより、提案戦略の有効性を検証する。

4.1.　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トレースドリヴンシミュレータを用いる。本評価で用いるシミュレータは、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。さらに、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。また、CMVPの構成は、ベクトルキャッシュの容量を1 MBとし、メモリバンド幅はコア数によらずメインメモリ-キャッシュ間を64GB/sとし、一方でキャッシュから各コアへは常に64GB/s/coreで提供できるものとする。したがって、ベクトルコアのピーク性能を16 GFlop/sと仮定すると、メインメモリのB/Fはコア数が増加するに従い、1コア時 4 B/Fから16コア時 0.25 B/Fまで減少する。

4.2. 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4種類の実アプリケーションを用いる。表1に実アプリケーションの詳細について述べる。なお、アプリケーションの並列化は、最外DOループを均一なスレッドに分割するようマイクロタスキングを用いて行い、各スレッドを一つのコアで実行する。

表 1 評価アプリケーション

評価アプリケーション	計算手法	問題サイズ	ベクトル化率	ベクトル長
Matrix Multiply	-	1024×1024×1024	99.5%	256
7-pt Stencil	-	256×256×256	99.7%	256
27-pt Stencil	-	256×256×256	99.7%	256
Earthquake [4]	Friction Law	2047×2047×257	99.5%	256
Land Mine [5]	FDTD	1500×1500×50	99.7%	250
Turbulent Flow [6]	DNS	512×512×256	99.9%	256
Antenna [7]	FDTD	252756×9×138	99.5%	256

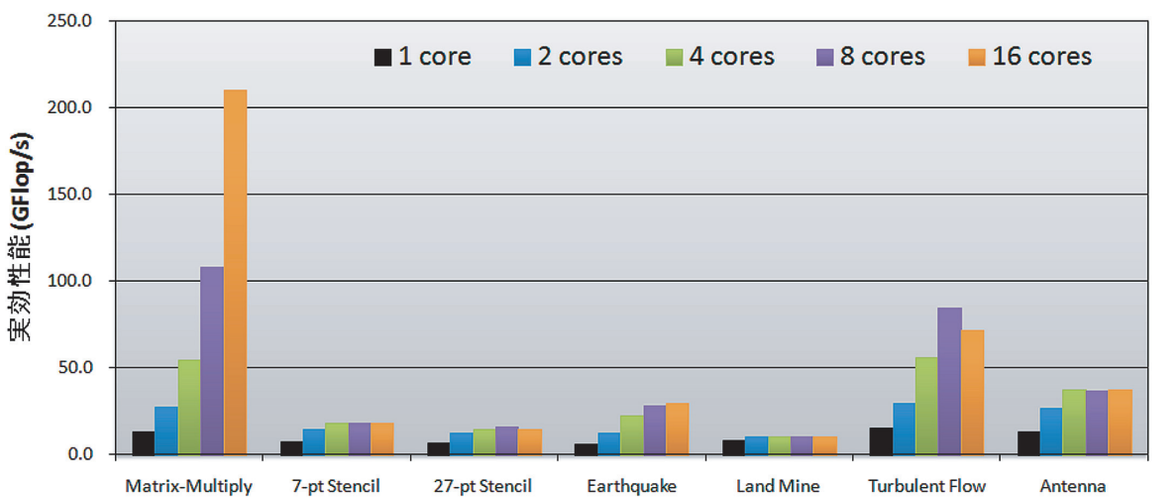


図 4 評価結果 (最適化戦略適用前)

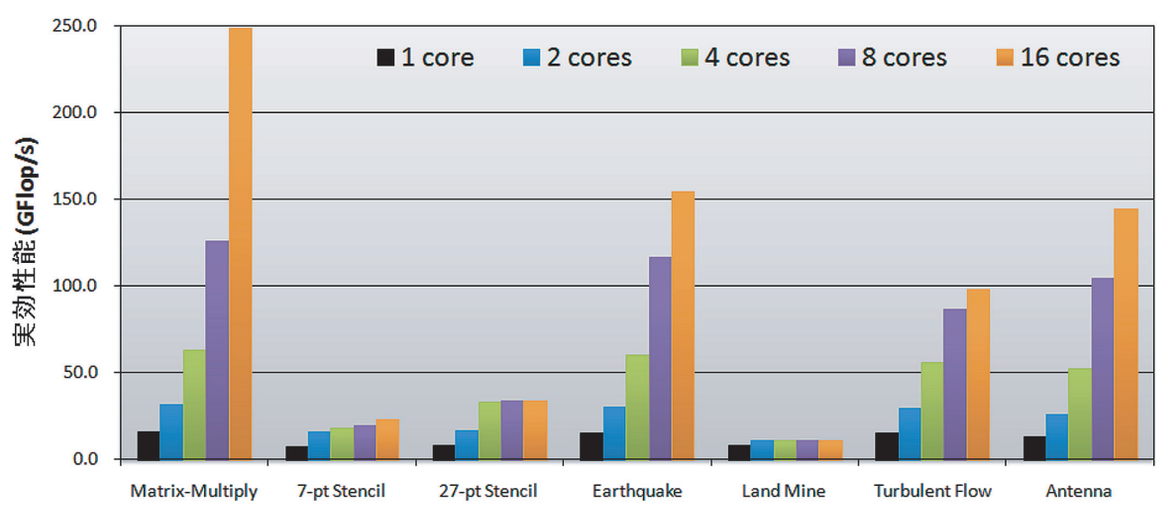


図 5 評価結果 (最適化戦略適用後)

4.3.　評価結果

図 4 に最適化戦略適用前の評価結果を示す。全体としてみると、1 コアから 8 コアまでコア数が増加する間では、ほとんどのベンチマークにおいて性能が向上する。しかし 16 コア時では Matrix-Multiply を除くすべてのベンチマークは性能が向上しなくなる。これは。コア数の増加に伴いコア当たりに供給できるメモリバンド幅が低下するためである。

Matrix-Multiply、Earthquake そして Turbulent Flow ではコア数が増加するに従い、他のベンチマークと比較して高い性能向上が得られることが分かる。これは、各コアでデータをベクトルキャッシュで共有することにより、メインメモリアクセス数を削減できるためである。そのため、コア数増加に伴う B/F 低下の影響を受けにくい。しかし、16 コア時までコア数を増やすと、B/F 低下の影響が表れ、性能向上は得にくくなる。さらに Turbulent Flow では、16 コアでスレッドを並列処理すると、各スレッドが利用するキャッシュ容量が、ベクトルキャッシュ容量を超えてしまうためキャッシュヒット率が減少し、結果として演算性能が低下する。同様に、Antenna においてもキャッシュ容量が不足するため、性能の向上幅が低下する。したがって、これらのアプリケーションではキャッシュブロッキングを適用する必要がある。

一方で、Stencil と Land Mine では、ほとんど実効性能は向上していない。これは、これらのベンチマークはメモリアクセスが他のアプリケーションと比較して多いためである。特に、コア数が増加するに伴いコア当たりのメモリバンド幅は低下するため、コア数増加に伴うスケーラビリティを得ることは困難である。したがって、CMVP では実効メモリバンド幅を向上させるキャッシュブロッキングによる最適化は不可欠であると言える。

図 5 に最適化戦略適用後の評価結果を、図 6 にルーフラインモデルにおける最適化戦略適用前後の演算性能と演算密度の関係を示す。図 5 より、最適化戦略を適用することにより、全てのベンチマークにおいて実効性能が向上していることがわかる。なお、図 6 のルーフラインモデルによる解析より、16 コア時においては Matrix-Multiply と Earthquake が演算ネック、それ以外がメモリネックであると判断できる。これに基づき最適化戦略を適用することにより、最大で 5 倍の性能向上が得られた。

最適化戦略により実効性能の飛躍的な向上を達成したが、Earthquake や Turbulent Flow、Antenna では、さらなる性能向上の余地がある。また、メモリネックのアプリケーションでは 16 コア時には最適化の効果は得難い。そこで今後は、アンローリングやブロッキング以外の最適化の適用も検討し、最適化が与える効果や最適化間の影響について評価、および解析に取り組む予定でいる。

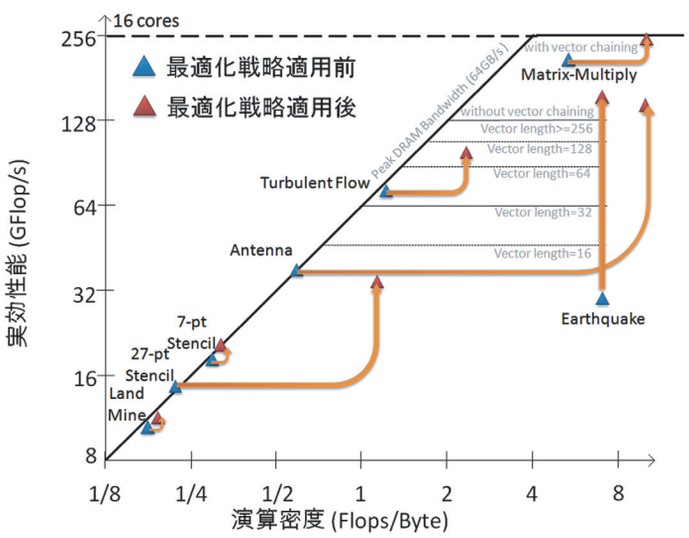


図 6 最適化適用前後のルーフラインモデル(16 コア CMVP)

5. 結言

本研究の目的は、次世代ベクトルプロセッサとして本研究室で提案しているCMVP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。そこで本稿では、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。提案する最適化戦略では、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5倍の性能向上を達成した。本提案戦略を用いることで、CMVPにおいて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。

今後は、CMVP向けのルーフラインモデルの改善、および新たなプログラム最適化手法の検討を行う予定である。最適化後においても性能向上の余地がまだ残っており、現在の性能を決定づける要因についてルーフラインモデルで解析可能となるよう、アプリケーションの詳細な解析を行う。また、アンローリングやブロッキング以外の最適化手法も最適化戦略に取り込み、さらなる性能向上が可能な最適化戦略へと発展させる予定である。

謝辞

本研究は、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を利用することで実現することが出来た。また、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] Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun., "A Single-Chip Multiprocessor," Computer, pp.79-85, September, 1997.

[2] Hiroaki Kobayashi., “Implication of Memory Performance in Vector-Parallel and Scalar-Parallel HEC Systems”, High Performance Computing on Vector Systems 2006, Springer-Verlag, pp.22-50, 2006.

[3] Samuel Williams, Andrew Waterman, and David Patterson., “Roofline: an Insightful Visual Performance Model for Multicore Architecture”, Communications of the ACM, Vol. 52, No. 4, pp.65-76, 2009.

[4] Keisuke Ariyoshi, Toru Matsuzawa, and Akira Hasegawa., “The key frictional parameters controlling spatial variations in the speed of postseismic-slip propagation on a subductionplate boundary”, In: Earth and Planetary Science Letters, Vol. 256, pp.136-146, 2007.

[5] Takeo Kobayashi, Motoyuki Sato., “FDTD simulation on array antenna SAR-GPR for landmine detection”, In: Proceedings of SSR2003, pp.279-283, 2003.

[6] 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.

[7] Yukiko Takagi, Hiroyasu Sato, Yoshihiko Wagatsuma, and Kunio Sawamura., Study of High Gain and Broadband Antipodal Fermi Antenna with Corrugation. Proceedings of 2004 International Symposium on Antennas and Propagation, Vol. 1, pp. 69-72, 2004.