Vol.2018-HPC-165 No.5
2018/7/30

プロセッサシミュレータの多数並列実行環境による性能最適化への応用

小田嶋 哲哉[1,a)]　児玉 祐悦[1]　佐藤 三久[1]

概要：コデザインは，アプリケーションの特性をアーキテクチャに反映することや，アーキテクチャの特性に合わせたアプリケーションプログラムの最適化を行う双方向な取り組みであり，システムの設計において重要視されている．本研究では，コデザインの取り組みを支援するためのシミュレータ多数並列実行環境を提案している．これによって，ソフトウェアを最適化するためのパラメータとハードウェアのパラメータの組み合わせを多数並列に実行することができ，コデザインのための評価を加速することを期待している．本稿では，この環境を用いることで，ハードウェアのパラメータは固定し，ソフトウェアの最適化に関するパラメータサーチを行う．行列積のアンロール数を最適化パラメータとして設定し，シミュレータの多数並列実行を行ったところ，ナイーブな実装に対して速度向上を達成することを確認した．さらに，シミュレータが提供する詳細な情報から，アプリケーションの特性を評価することも可能であることを確認した．

1. はじめに

HPC (High Performance Computing) 向けの高性能・低電力システムの設計において，コデザインの重要性について指摘されている．コデザインは，アプリケーションの特性を解析しアーキテクチャに反映するとともに，アーキテクチャの特性に合わせたアプリケーションプログラムのコード最適化を行うなどの双方向的な取り組みである．本稿では，このコデザインの双方向の取り組みを支援するシミュレータ多数並列実行環境を提案する．この環境は，プログラムの様々なパラメータとハードウェアのパラメータを変化させて，プロセッサシミュレータを多数同時実行することを支援するものである．プロセッサシミュレータにおいて，サイクルレベルのシミュレーションを行うには非常に長い時間かかり，実際の実行時間としては短いプログラムに限られるが，実際のプロセッサでは得られない詳細な情報が得られる．また，この環境においてはソフトウェアだけでなく，ハードウェアのパラメータを変えることができ，ソフトウェアに適したハードウェアの探索にも利用可能である．

アプリケーションの性能最適化を行うにあたって，プロファイラなどを使用してプログラム実行時の各種情報を取得することは重要である．オープンソースでは PAPI (Performance Application Programming Interface) [1] が広く利用されている．これは，CPU が持つハードウェアカウンタを必要な区間だけ取得することが可能であり，キャッシュのアクセス数/ミス数，命令数などの情報を取得することができる．プログラムのパラメータとして，多重ループのブロック化のサイズなど変えて多数実行し，最適化を行う手法は，オートチューニングの 1 つとして Atlas [2] などが知られている．

本研究では，汎用 CPU シミュレータとして広く利用されている gem5 [3] を用いる．シミュレータを利用することのメリットとして，正確な実行時間とハードウェアカウンタの取得が可能なことが挙げられる．クロックごとのプロセッサの内部動作をシミュレートするサイクルレベルシミュレータでは，タイマーの精度を考慮する必要が無いため，短いイテレーションのループでも正確な実行時間を取得することができる．gem5 では，PAPI では得られないような詳細な情報だけでなく，ユーザの要求に応じて gem5 の内部の情報を取得するなどの拡張することで，性能最適化に必要なデータを取得することが可能である．しかし，サイクルレベルシミュレータは，実行時間が非常に長くなることが問題である．特に，アプリケーションの性能最適化におけるパラメータサーチでは，複数のパラメータの組み合わせを実行する必要があり，膨大な時間が必要である．また，シミュレータが提供する各種情報から，必要な情報を抽出することも，経験の少ないユーザには困難である．

[1] 理化学研究所 計算科学研究センター
[a)] tetsuya.odajima@riken.jp

© 2018 Information Processing Society of Japan

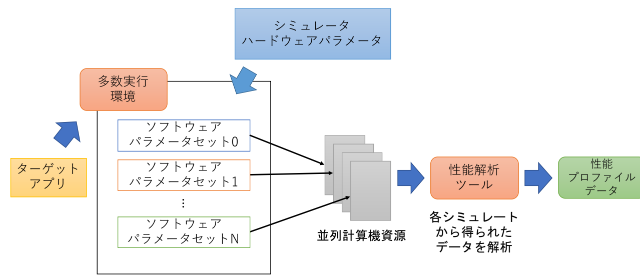

図 1　並列計算機環境におけるシミュレータ多数実行

そこで，本研究では，サイクルレベルシミュレータである gem5 を多数並列実行し，得られた情報を整形するシステムを提案する．本稿では，行列積のアンロール数に対するパラメータサーチを例に，シミュレータの多数実行を行い，性能最適化を行うことを試みる．

2. プロセッサシミュレータ多数並列実行環境

本章では，プロセッサシミュレータの多数並列実行環境について提案を行う．本評価に用いたシミュレータの概要とそのパラメータ設定，シミュレータ多数実行環境について説明する．

2.1 シミュレータ多数並列実行環境の提案

図 1 に，本稿で提案するシミュレータ多数並列実行環境の概略を示す． 2.2 節で述べる gem5 シミュレータを用い，これを多数並列実行し，シミュレート結果を取得する．サイクルレベルシミュレータである gem5 の O3 モードは，ハイエンドの CPU を用いたとしても，たかだか十数万命令/秒のオーダーでしかシミュレートを行うことができない．一例として， 図 2 では，8.738μ 秒のシミュレート結果に対して，シミュレータ自体の実行時間は 101.18 秒と約 1 万倍もの差がある．一方で，サイクルレベルシミュレータはクロックごとのプロセッサの内部動作をシミュレートするという特徴から，シミュレータ自体を並列化することは困難である．

そこで，本研究では 図 1 に示すように，シミュレータを PC クラスタなどの並列環境を用いて多数並列に実行するシステムを構築し，その結果を集約する環境を提供し，プログラムの最適化への応用を提案する．これによって，特にプログラムのパラメータサーチにおいて，シミュレータを何度も実行する時間を削減し，最適なパラメータを選択できる．

Slurm [4] などのジョブスケジューラを用いて，並列計算機資源を活用したシミュレータの多数実行が最終的な目標であるが，本稿ではその予備評価として，ノード内の複数 CPU コアを使用して多数実行の評価を行う．

2.2 gem5 シミュレータ

本研究では，ARM 社が提供する ARMv8-A AArch64 命令セットの HPC 向け SIMD 拡張である Scalable Vector Extension (SVE) [5] を用いる．2018 年 7 月現在，SVE を実装している計算機はプロダクトとしては存在していない．そこで，我々は SVE を用いた評価のために汎用プロセッサシミュレータである gem5 [3], [6] を用いる．gem5 には，命令レベルシミュレータである “Atomic モード” と，Out-of-Order のパイプラインをシミュレートして，正確な実行サイクル数を見積もることが可能なサイクルレベルシミュレータである “O3 モード” がある．また，gem5 は ARM 以外にも Alpha，SPARC，x86 など多くのプロセッサのシミュレーションにも対応している．

さらに，gem5 はフルシステムモード (fs モード) とシステムエミュレーションモード (se モード) を提供している．本研究では，システムコールをソフトウェアによるエミュレートする “se モード” を用いて評価を行う．

現在公開されている gem5 は ARM SVE に対応していない．そこで我々は，ARM 社から提供された Atomic モードのみに対応した gem5 に対して，O3 モードへの拡張を行っている．本稿ではこの実装のことを “gem5-sve” と呼ぶ．

```
1  sim_seconds 0.008738
2  host_seconds 101.18
3  sim_insts 19945316
4  ...
5  system.cpu.dcache.ReadReq_miss_rate::total 0.379413
6  system.cpu.dcache.WriteReq_miss_rate::total 0.024971
7  system.l2.ReadSharedReq_miss_rate::cpu.data 0.000122
8  system.l2.WriteSharedReq_miss_rate::cpu.data 0.0
9  ...
10 system.cpu.rename.ROBFullEvents 0
11 system.cpu.rename.ROBFullEvents 138488
12 system.cpu.rename.LQFullEvents 2726714
13 system.cpu.rename.SQFullEvents 2082
14 system.cpu.rename.FullRegisterEvents 0
15 ...
16 system.cpu.vector_ext_num_insts 12713986
17 system.cpu.vector_ext_num_mem_insts 6356992
18 system.cpu.vector_ext_num_loads 4259840
19 system.cpu.vector_ext_num_stores 2097152
20 ...
```

図 2　gem5 が提供するシミュレート統計情報

図 2 に，gem5 の実行によって得られるプログラムの統計情報から一部を抜粋したものを示す．“sim_seconds” はシミュレートによって得られたプログラムの実行時間 [秒] である．“host_seconds” は，gem5 を実行するためにかかった時間 [秒] を示している．また，L1 キャッシュ，L2 共有キャッシュの Read/Write ミス率や，パイプライン実行において Out-of-Order 資源が不足したことによって

表 1　シミュレータパラメータセット

ハードウェアパラメータ	
周波数	2.0 GHz
ベクトル幅	512 bit
L1 Dcache, Icache	
Size	32 kB
Associate	4
L2 Cache	
Size	2 MB
Associate	16
演算器	
整数パイプライン	2
浮動小数点数パイプライン	2
ロードユニット	1
ストアユニット	1
同時命令 fetch, decode, rename 幅	3
同時命令 dispatch 幅	6
同時命令 issue, WB, commit 幅	8
Out-of-Order リソースパラメータ	
IQ (Reservation Station)	64
ROB (Re-order Buffer)	64
LQ (Load Queue)	16
SQ (Store Queue)	16
Physical Vector Register	96 (=32+64)

“Renaming” ができなかった回数，実行した SVE 命令の総数やメモリ操作命令数とその内訳など，詳細なデータが含まれている．このようにシミュレータを用いることで，Out-of-Order に関するデータなど，プロファイラを用いたとしても得ることができない情報を取得することが可能である．一方で，多数の統計情報からユーザが性能最適化のために必要なデータを抽出することは，シミュレータに精通していないユーザには難しいと考えられる．

2.3 gem5 の命令パイプライン

gem5 の O3 モードで用いる Out-of-Order 実行は，RISC マイクロプロセッサである Alpha21264 をベースとしている．これは，“Fetch”，“Decode”，“Rename”，“Issue”，“Execute”，“Write Back”，“Commit” の 7 ステージから構成される．gem5 は，各ステージのレイテンシ (cycle 数) や同時実行幅，演算器のリソース量，各命令クラスのレイテンシを容易に変更することができる．

2.4 gem5-sve のパラメータ設定

表 1 に，gem5-sve のハードウェアおよび Out-of-Order リソースのパラメータを示す．これらのパラメータは，ARM 社より提供された gem5 のデフォルトパラメータセットに準拠している．これは，ARMv7-A アーキテクチャのハイエンドシリーズである Cortex-A15 [7] に類似している．これは従来，組み込み向けのプロセッサであるため，HPC およびサーバ用としては Out-of-Order リソース量が少ないと思われるが，本評価ではできるだけ同じ値を使用する．主な変更点は，以下のとおりである．

- SVE 向けの命令レイテンシを ARMv7-A NEON のレイテンシ準拠として追加
- 物理ベクトルレジスタ数は，論理ベクトルレジスタ数と合計して 96 と定義
- IQ および ROB がそれぞれ 32，40 と少なすぎたため，両リソースを 64 と定義
- 整数演算器および整数乗算/除算演算器を統合し，整数演算器 2 つとして定義

```
1 for (int i = 0; i < N; i++)
2   for (int j = 0; j < N; j++)
3     for (int k = 0; k < N; k++)
4       C[i][j] += A[i][k] * B[k][j];
```

図 3　評価プログラム: 行列積

3. 評価

本稿では，行列積を用いてシミュレータ多数実行の評価を行う．行列積の性能最適化についてはこれまで数多くの手法が提案されてきたが，本稿ではループの入れ替えとループアンローリング数を評価パラメータとする．

図 3 に行列積のナイーブな実装を示す．これに対して，性能最適化を行う場合，まず考えられるのがループの入れ替えである．Row-major order な配列に対して 図 3 は，配列 C および A は連続方向へのアクセスになるが，配列 B はストライドアクセスとなり，ロードの効率が低下してしまう．そこで，最内の k ループと真ん中の j ループを入れ替えることで，すべての配列に対して連続アクセスとなるため，性能が向上することが期待される．本稿では， 図 3 に示すループを “ijk ループ”，k ループと j ループを入れ替えたループを “ikj ループ” と定義する．ループアンローリングは，for 文を展開することで並列実行可能な命令を増やし，同時に分岐命令数を削減することで高速化を図るという手法である．さらに，ループ演算中にレジスタを再利用することで，配列ロード/ストアのコストを削減する，いわゆるレジスタブロッキングの効果も生まれる．昨今の高性能なコンパイラは，ループの入れ替えやループアンローリングによる最適化を自動的に適用するものもあるが，依然として手動での性能最適化が必要なことも多い．

3.1 評価環境

評価環境として，使用するサイクルレベルシミュレータは gem5-sve とし， 表 1 に示すパラメータを使用する．本シミュレータの理論ピーク性能は $2.0[GHz] \times 2[Unit] \times 2[FMA] \times 8[SIMD] = 64.0$GFLOPS（倍精度浮動小数点数）となる．使用するコンパイラは，ARM 社が開発して

© 2018 Information Processing Society of Japan

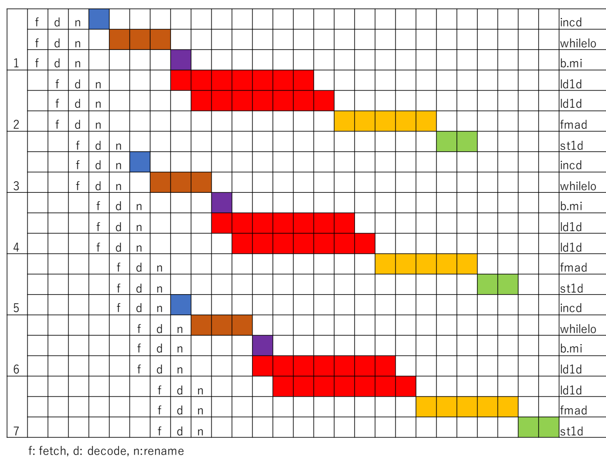

図 4　ナイーブな実装におけるパイプラインの状態

いる Arm Compiler for HPC 18.2 [8] を使用する．行列積の問題サイズは $N = 256$ の正方行列とする．

3.2　ナイーブな実装

まず，ijk ループ，ikj ループのナイーブな実装の評価を行う．**表 2** に 2 つのループの実行性能およびピーク性能に対する比率を示す．

表 2　ナイーブな実装の性能

	GFLOPS	ピーク性能比
ijk ループ	0.38	0.59 %
ikj ループ	6.16	9.63 %

表 2 より，ijk ループの性能が非常に低いことがわかる．この原因として，ARM コンパイラが出力する Gather Load 命令が影響していると考えられる．ARM SVE から，不連続なアクセスを一度にロードすることが可能な Gather Load 命令が導入されており，行列積では配列 B のアクセスがこれに当たる．ハードウェアが Gather Load に最適化されている場合，効率的なロードが可能であるが，現在の gem5-sve の実装では，ベクトルの要素数だけベクトルサイズの連続ロードを発行するナイーブな実装となっているため，非常にコストが大きい．つまり，ijk ループの最内ループではロード：演算の比が $9(= 1 + 8) : 1$ となり，演算効率が非常に低い．一方，ikj ループは ijk ループと比較すると性能は高いが，ピーク性能比は約 10%にとどまる．ikj ループの最内ループについて，Out-of-Order リソースが理想状態で最大効率の動作を仮定すると，**図 4** のようなパイプラインになる．赤で示す命令がロード（LD1D），緑で示す命令がストア（ST1D）であり，ロードユニット/ストアユニットが独立して動作をする．黄で示した命令は FMA 演算（FMAD）が浮動小数点数演算器で実行され，その他の色は整数演算器で実行される．今回使用した gem5-sve のパラメータは，表 1 にあるように 3 命令同時 fetch が可能であるため，図 4 のようなパイプライン実行で律速となり，7 サイクルで 3 つの演算が実行される．今回用いた CPU モデルには，2 つの浮動小数点数演算器があるため $3/7 \times 1/2 = 21.4\%$ がピーク実行効率となる．さらに，L1 キャッシュミス率が約 30%と高いことからも，ループアンローリングによる性能最適化が必要である．

```
1 for (int i = 0; i < N; i+=BI)
2   for (int j = 0; j < N; j+=BJ)
3     for (int k = 0; k < N; k++)
4 #pragma clang loop unroll(enable)
5       for (int ii = 0; ii < BI; ii++)
6 #pragma clang loop unroll(enable)
7         for (int jj = 0; jj < BJ; jj++)
8           C[i+ii][j+jj] += A[i+ii][k] * B[k][j+jj];
```

図 5　行列積プログラム自動生成例

3.3　シミュレータ多数実行による評価

前節のナイーブな実装の結果を踏まえ，ijk，ikj ループに対してアンロール数を $\{0, 2, 4, 8\}$ [*1] と設定し，シミュレータを多数実行することでパラメータサーチを行う．**図 6** は ijk ループの結果であり，4 つのグラフは最外ループである “i ループ” のアンロール数，各グラフの横軸は真ん中のループである “j ループ” のアンロール数，各棒グラフのバーは最内ループである “k ループ” のアンロール数ごとのピーク性能に対する比率を示している．一方，**図 7** は ikj ループの結果であり，4 つのグラフは最外ループである “i ループ” のアンロール数，各グラフの横軸は真ん中のループである “k ループ” のアンロール数，各棒グラフのバーは最内ループである “j ループ” のアンロール数ごとのピーク性能比を示している．それぞれのグラフの縦軸のスケールが異なっていることに注意されたい．

これらより，最内である k ループまたは j ループをアンロールしてしまうと性能が低下していることがわかる．当初，最内ループのアクセスは SIMD による並列化が行われるため，このループをアンロールしてしまうとコンパイラの SIMD 化が阻害されてしまうと考えていた．しかし，生成したコードのアセンブリを確認すると，SIMD 化はされているがレジスタブロッキングをしている配列以外はストライド方向の Gather Load 命令が生成されていることがわかった．本評価では，**図 5** のように，ナイーブな実装で示したループの内側に，アンロール数を設定したループを挿入し，コンパイラにアンロールしたコードを生成させて評価を行っており，図 5 の場合，コンパイル後のアセンブリでは $BI \times BJ$ 回アンロールされている．アンロールしたコードにおいて新たに最内となった k ループについて，ARM コンパイラは配列 B のアクセスに対してストライド

[*1] 0 はアンロールしない

© 2018 Information Processing Society of Japan

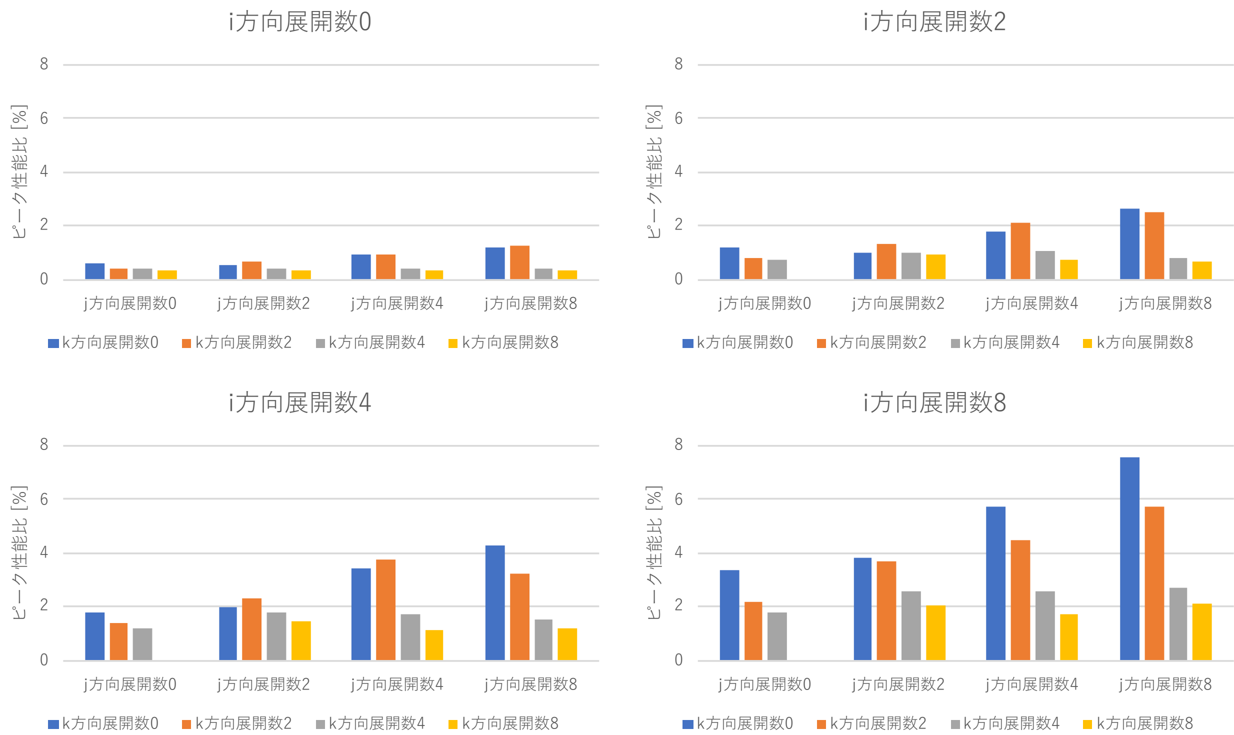

図 6　ijk ループのピーク性能比

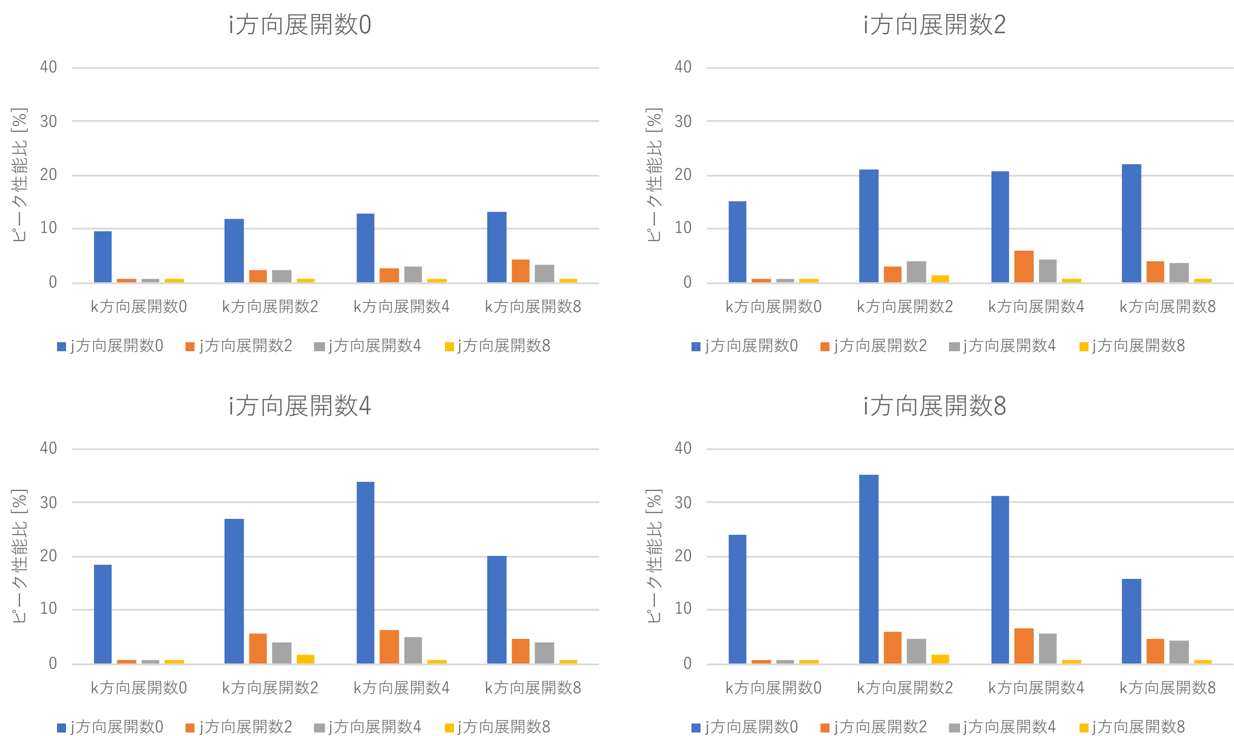

図 7　ikj ループのピーク性能比

方向の並列性を検出し，それに最適なコードを生成していると考えられる．3.2 節でも示したとおり，Gather Load に最適化されたハードウェアであれば高い性能を得ることが期待できるが，本シミュレータの実装では，複数の連続ロードに分割されて実行されるため，ロードの負荷が著しく大きくなってしまう．これによって，性能が低下したと考えている．これを検証するために，シミュレータを改良し，Gather Load などの命令を一度に実行できる環境における評価は今後の課題としたい．

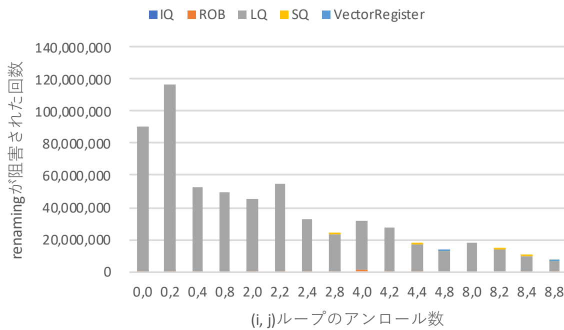

図 8　ijk ループの Renaming 実行が阻害された回数とその内訳

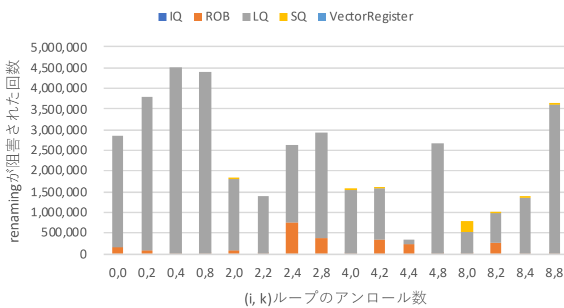

図 9　ikj ループの Renaming 実行が阻害された回数とその内訳

3.4　ijk ループの評価

ijk ループでは，アンロール数 (i, j, k) = (8, 8, 0) のときに 4.83GFLOPS とピーク性能に対して 7.55%を達成した．ナイーブな実装に対して，性能は向上しているが依然として絶対性能が低いことがわかる．この原因について，シミュレータの結果から得られる Out-of-Order に関するパラメータを用いて検証する．図 **8** に，ijk ループにおいて Out-of-Order の Renaming がリソース不足により実行できなかった回数と阻害されたリソースの内訳を示している．これより，LQ (Load Queue) の不足により Renaming ができなかったことがわかる．これは，今回使用したシミュレータのパラメータにおいて LQ の設定数が少なかったことも原因の一つであるが，Gather Load によるロードの負荷増大が主な原因と考えられる．アンロール数が少ない場合，最内ループにおけるロードの比率が増大するため特に影響が大きい．一方で，i および j ループをアンロールすることで，最内ループにおいて演算の密度が増加し，相対的にロードの負荷が低減したことで，Renaming が阻害される回数が低下したと考えられる．これは，図 8 の値に反比例して実行性能が向上していることからも言える．

3.5　ikj ループの評価

ikj ループでは，アンロール数 (i, k, j) = (8, 2, 0) のときに 22.62GFLOPS とピーク性能に対して 35.35%を達成した．ijk ループと比較して，非常に性能が高いことがわかる．ikj ループの最内ループをアンロールしなければ，配

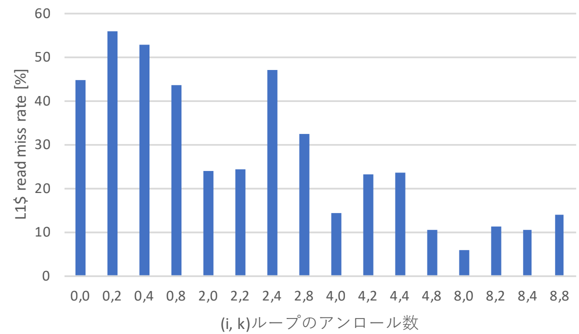

図 **10**　ikj ループの L1 キャッシュ Read ミス率

列のアクセスが連続方向に統一され，ロード/ストアの効率が向上したことが要因として挙げられる．また，ループアンロールによって最内ループの演算密度が増加したことも要因として考えられる．(8, 2, 0) のとき，最内ループの最大演算効率は図 **??**と同様のパイプライン実行を仮定すると，38 サイクルで FMA 命令が 48 個発行されることから，$48/32 \times 1/2 = 63.16\%$ と見積もられる．これより，ナイーブな実装よりも演算効率が高いため，性能が向上したと言える．ijk ループと同様に，Out-of-Order に関するパラメータについて検証する．図 **9** に，ikj ループにおいて Out-of-Order の Renaming がリソース不足により実行できなかった回数と阻害されたリソースの内訳を示す．ijk ループでは，アンロール数を増やしていくと，演算の密度が増加するため，LQ のプレッシャーが減少していたが，ikj ループでは逆に増加している．ikj ループでは，i 方向へのアンロールをした場合，独立した FMA 演算が増加し，かつ配列 B が再利用可能になるため，LQ のリソースプレッシャーは減少する．一方，k 方向にアンロールを行うと，配列 C のみがレジスタブロッキングの対象になり，配列 A および B はアンロールした数だけロードが必要になる．つまり，n 回アンロールした場合，$C[i][j] = C[i][j] + A[i][k+0] * B[k+0][j] + ... + A[i][k+(n-1)] * B[k+(n-1)][j];$ と配列 C への書き込みが直列になる．このため，これまでの研究 [9] でも言及していたように，最内のループボディーが長くなると，配列 C へのストアが commit されるまで Out-of-Order リソースが開放されないため，リソースが枯渇し，Renaming が阻害される回数が増加したと考えられる．ikj ループで最大性能を達成したパラメータは (i, k, j) = (8, 2, 0) であるが，Out-of-Order のリソースだけに注目すると (4, 4, 0) というパラメータが最も良いという結果であった．性能はほぼ同等であるが，この性能差は図 **10** に示す L1 キャッシュの Read ミス率が原因と考えられる．(i, k, j) パラメータ (8, 2, 0) のミス率は 11.14%であるが，(4, 4, 0) のミス率は 23.52%と増加している．これは，先程も述べたように，i 方向のアンロールによるレジスタブロッキングの効果と k 方向にアンロールしたことによるロード

© 2018 Information Processing Society of Japan

の増加が原因と考えられる．今回のハードウェアパラメータでは，Out-of-Order のリソースだけで見れば，(4, 4, 0) というパラメータが最もバランスしている．例えば，L1 キャッシュサイズが大きいパラメータでは，キャッシュミス率が低下し，Out-of-Order の効率が良い (4, 4, 0) というパラメータを用いることでより高い性能を期待することができると考えられる．

3.6 シミュレータによる性能解析

これまで，シミュレータを多数並列に実行することで，行列積の ijk ループおよび ikj ループそれぞれのアンロール数について評価を行ってきた．これより，一般的なプロファイラからは得られない Out-of-Order リソース量を比較することで，プログラムの内部動作について検討することが容易になった．今回は行列積という非常にわかりやすいプログラムを対象にしたが，ループボディがより複雑なプログラムにおいても性能最適化が比較的容易に行えるようになると考えている．例えば，今回は ikj ループにおいて，k 方向にアンロールすることでループボディが長くなり，Out-of-Order リソースが枯渇し，パイプラインが効率的に動作しなかったということがあった．この場合，ループをあえて分割することで，パイプラインが効率的に動作し，結果として性能向上につながる可能性があると考えられる．このような比較について，シミュレータを用いることで定量的に評価することが可能である．

4. おわりに

本稿では，サイクルレベルシミュレータである gem5-sve を多数並列に実行することで，行列積の性能最適化パラメータである「ループの入れ替え」と「ループアンロール数」について最適な組み合わせを評価してきた．これより，単純にキャッシュヒット率を用いた比較だけではなく，シミュレータを用いるからこそ得られた Out-of-Order リソース量を用いることで，プログラムの特性を検証し，最適なパラメータを選択することができることを確認した．

今後の課題として，gem5-sve の Gather Load の実装を改良し，不連続アクセスに最適化されたハードウェアが出てくることで，プログラムの実行効率がどうなるかについて検証を行う．また，よりループボディが複雑なプログラムに対しても，シミュレータの多数実行により性能最適化が可能かどうかを検証していく．本稿では，ソフトウェアのパラメータを探索することにとどまったが，ハードウェアのパラメータも同時に変更することで，より高い演算性能を達成することができるかについて検討を行う．

謝辞 本稿の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等)」で実施された内容に基づくものである．

参考文献

[1] PAPI. http://icl.cs.utk.edu/papi.
[2] Automatically Tuned Linear Algebra Software (ATLAS). http://math-atlas.sourceforge.net.
[3] The gem5 Simulator - A modular platform for computer-system architecture research. http://gem5.org/.
[4] Slurm Workload Manager. https://slurm.schedmd.com.
[5] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu, A. Reid, A. Rico, and P. Walker. The ARM Scalable Vector Extension. *IEEE Micro*, Vol. 37, No. 2, pp. 26–39, Mar 2017.
[6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 Simulator. *ACM SIGARCH Computer Architecture News*, Vol. 39, No. 2, pp. 1–7, May 2011.
[7] Cortex-A15 Overview. https://developer.arm.com/products/processors/cortex-a/cortex-a15.
[8] Arm HPC tools and libraries. https://developer.arm.com/products/software-development-tools/hpc/arm-compiler-for-hpc.
[9] 小田嶋哲哉, 児玉祐悦, 松田元彦, 李珍泌, 辻美和子, 佐藤三久. ベクトル長を可変とする SVE アーキテクチャの評価. 情報処理学会研究報告 (ハイパフォーマンスコンピューティング), Vol. 2017-HPC-160, No. 11, pp. 1–7, Jul. 2017.

© 2018 Information Processing Society of Japan