Vol.2019-ARC-236 No.3
2019/6/11

データ圧縮に基づくGPU向け高性能キャッシュアーキテクチャの提案

岡 慶太郎[1,a)]　川上 哲志[2,b)]　谷本 輝夫[3,c)]　小野 貴継[4,d)]　井上 弘士[4,e)]

概要：Graphic Processsing Unit（GPU）は多数のプロセッサコアによる並列処理により高性能を達成する．一方で，GPU は多数のプロセッサコアが少量の L1 キャッシュを共有するため，プログラムによっては競合性ミスが頻発する．この問題へのアプローチの一つとして面積を増加させることなくキャッシュの実効容量を増加させるデータ圧縮に基づくキャッシュメモリが挙げられる．しかしながら，既存手法はキャッシュライン圧縮効果が低い場合がある点や復元レイテンシが長くなりやすいため，GPU における性能向上が十分でない．そこで，本研究はまず，同一の静的命令が参照するキャッシュライン間ではデータ値の局所性が高い場合があるという性質を利用して，既存手法の圧縮効果を改善する．つぎに，我々は復元時に不必要なデータへのアクセスを抑制することで，復元レイテンシを削減する手法を提案する．評価の結果，全ての提案手法を包括的に適用した場合に従来型キャッシュに対して，平均 8.7 ポイントの性能向上を達成した．また，アプリケーションごとに適切な手法を選択する場合では従来型キャッシュと比較して平均 14.2 ポイントの性能向上を達成する可能性があることが明らかになった．

1. はじめに

Graphics Processing Unit (GPU) は多数のプロセッサ・コアを搭載することで高い性能を達成する．たとえば，2560 個のコアを搭載する GTX 1080[1] は 8873 GFLOPS の理論ピーク性能を有する．GPU では命令はワープという複数のスレッド単位で，発行，実行，コミットが行われる．GPU プログラム実行性能を低下させる要因の一つとして L1 キャッシュにおける競合性ミスの多発が知られている [2], [3], [4]．これは GPU で多数のコアが L1 キャッシュを共有することに起因する．

この問題を緩和させるためにデータ圧縮に基づく GPU 向けキャッシュアーキテクチャ [5] が存在する．この手法は，ワープ内のスレッド間でメモリアクセス先には値局所性があるという性質を利用する．さらに，ワープ内のスレッドのメモリアクセス先が同一のラインに存在しやすい．したがって，ライン内のワード間で値局所性が高くなりやすいため，ライン内で圧縮を施す．圧縮ではライン左端のワードの値をベースとし，それ以外のワードはベースの値との差分を用いて圧縮される．圧縮されたラインは 1 つのベースと複数の差分から成る．既存手法ではライン毎にベースを圧縮ラインに含めるため，ライン間のワードにて値局所性が高い場合には，ベースの値が重複することがあり得る．そのため，ライン間の値局所性を利用する圧縮手法ではラインをより小さなサイズに圧縮できる可能性がある．

ライン間の値局所性を利用してデータ圧縮を行うキャッシュ・アーキテクチャが提案されている [6], [7]．しかしながら，これらの手法は全て汎用 CPU を対象としており，GPU 向けの技術は存在しない．そこで本研究では，ライン圧縮効果の高い CPU 向けキャッシュ・アーキテクチャ MORC [6] を GPU 向けに改良する．MORC は複数のラインで辞書を共有することで，ライン内だけでなくライン間のワードに対して，値局所性の利用を可能とする．ただし，圧縮と復元には，辞書を共有する複数のラインにアクセスするために，長いレイテンシを要する．

本研究は以下４つの改良を行う．第一に，ライン圧縮サイズの縮小ために，静的命令が参照するラインの集合 (静的命令参照ライン) の値局所性を利用する．静的命令参照ラインで辞書を共有させることで，ラインの圧縮サイズを縮小できる場合がある．第二に，圧縮や復元時の辞書を共

[1] 株式会社ソシオネクスト
[2] 九州大学大学院システム情報科学研究院 I & E ビジョナリー特別部門
[3] 九州大学情報基盤研究開発センター 学習環境デザイン研究部門
[4] 九州大学大学院システム情報科学研究院 情報知能工学部門
[a)] oka.keitaro@socionext.com
[b)] satoshi.kawakami@cpc.ait.kyushu-u.ac.jp
[c)] tteruo@kyudai.jp
[d)] takatsugu.ono@cpc.ait.kyushu-u.ac.jp
[e)] inoue@ait.kyushu-u.ac.jp

有するラインへのアクセスにおいて，過去の履歴を利用して一部のアクセスを省くことで，圧縮・復元レイテンシを削減する．第三に，アクセス対象のラインが非圧縮ならば，復元処理を省くことで，復元レイテンシを削減する．第四に，辞書を共有するライン数を制限することで，圧縮・復元レイテンシに上限を設ける．

性能評価の結果，全ての手法を適用した場合は，MORC と従来型キャッシュそれぞれと比較して平均実行時間をそれぞれ 3 ポイント，8.7 ポイント削減可能であることが明らかになった．また，ベンチマークに応じて適切な手法を選択することで，従来型キャッシュと MORC の実行時間をそれぞれ平均 8.6 ポイント，14.2 ポイント削減できる可能性があることがわかった．

2. 汎用 CPU 向けキャッシュデータ圧縮技術 MORC

2.1 GPU との高い親和性

GPU のキャッシュデータ圧縮では，ある程度の長い圧縮・復元レイテンシは許容される．なぜなら，GPU ではメモリアクセスレイテンシの増加に対して実行時間が増加しにくい傾向にあるためである．これは，GPU が実行対象ワープを命令毎で切り替えるマルチスレッディングをサポートすることに起因する．スケジューリング可能なワープが多数存在するならば，マルチスレッディングによりメモリレイテンシの多くを隠蔽できる．**図 1** に GPU の L1 キャッシュの読出・書込レイテンシを増加させた場合の実行時間を示す．レイテンシを 1 サイクルから 32 サイクルに増加した場合であっても，実行時間の増加率は平均 3% である．一方，128 サイクルに増加する場合では，実行時間は平均 13%低下する．

また，GPU におけるキャッシュデータ圧縮ではキャッシュラインをより小さく圧縮することが求められる．GPU にて L1 キャッシュコンフリクトを削減するためには，多大な L1 キャッシュ容量を要する場合があるためである．GPU の L1 キャッシュサイズを増加させた場合の実行時間を**図 2** に示す．L1 キャッシュサイズのベースラインは GTX480 に搭載されている 16KB とする．たとえば，BFS-R はベースラインの 4 倍の L1 キャッシュ容量で得られる性能向上率は 4% だが，ベースラインの 4 倍から 8 倍に L1 キャッシュ容量を増加させると 8%の性能向上が得られる．

以上から，GPU に適用するデータ圧縮に基づくキャッシュには，圧縮・復元レイテンシはある程度長いとしても，圧縮効果の高いものが望ましい．これまで提案されてきた代表的な既存手法 [6], [7], [8], [9], [10], [11] を圧縮・復元レイテンシと圧縮効果の程度により分類した結果を**図 3** に示す．既存研究は 3 つのグループに分類される．第一に，圧縮効果が低いが，圧縮・復元レイテンシが短いものである [8], [9], [10]．第二に，圧縮効果が中程度かつ圧縮・復元レイテンシが短いものである [7], [11]．第三に，圧縮効果と圧縮・復元レイテンシが中程度から高程度となる MORC[6] である．本研究では，GPU と親和性の高い特徴を包含する MORC[6] を対象とする．さらに，MORC の圧縮効果の向上と圧縮・復元レイテンシを削減するための改良を行う．

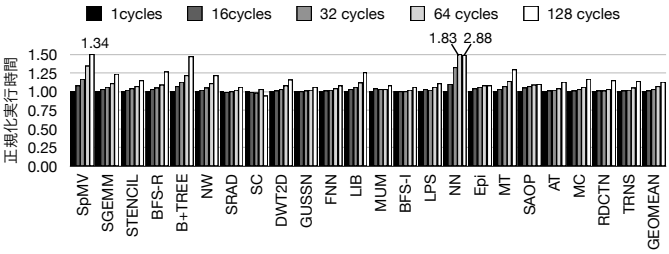

図 1 L1 キャッシュアクセスレイテンシの性能センシティビティ

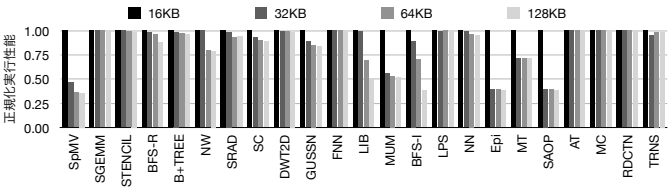

図 2 L1 キャッシュサイズの性能センシティビティ

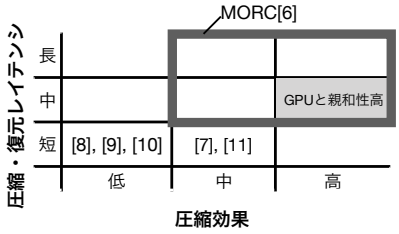

図 3 データ圧縮に基づくキャッシュアーキテクチャの分類

2.2 構造

MORC の構造を**図 4** に示す．MORC はログマップテーブル (Log Map Table: LMT)，データログアレイ，タグアレイ，圧縮・復元器，アクティブ・ログ・セレクタから構成される．

データログアレイは複数のログから成る．ログは圧縮を施したラインの格納先である．MORC は辞書ベースの圧縮方式を採っており，ログごとに辞書を管理する．各ログには ID（以下，ログ ID と呼称）が割当てられる．

LMT はメモリアドレスとラインの格納先ログのマッピング情報とラインステータスを記録する．メモリアドレスの一部でのインデッキシングにて対応する LMT エントリには，ラインの格納されたログ ID を記録する．LMT エントリは複数のメモリアドレス間で共有される．さらに，LMT はメモリアドレスのタグ情報を保持しない．そのため，LMT に記録されたログ ID，ラインステータス情報は偽陽性をもつ．

タグアレイは LMT が指すログの内部に参照対象のラインが存在するかを調査するために，用いられる．各ログにタグ格納領域（タグエントリ）がある．タグエントリにはログ内の全てのラインのタグとステータスを記録する．また，タグは差分を用いた圧縮 BDI [10] により圧縮されて格納される．

圧縮・復元器は一時辞書から構成される．一時辞書は，あるログにおいて，そのログ内の全ラインのユニークなワード値と辞書のインデックス（辞書 ID）のマッピングの生成（以下，辞書生成と呼称）を行うために用いる．生成された辞書はラインの圧縮と復元に使用される．

アクティブ・ログ・セレクタはラインの格納先のログを選択するために用いる．アクティブ・ログ・セレクタには二種類の手法が存在する．第一の手法は，ログに空き領域がある限り同一ログに書き込む同一ログ選択方式である．第二にラインを最も小さいサイズに圧縮するログを優先的に選択する値考慮方式である．

2.3 動作

圧縮と書込動作: MORC はアクティブ・ログ・セレクタが選択したログに対して一時辞書を生成し，その辞書を用いてラインを圧縮する．たとえば，図 4 において 4 つのワード値（b，c，b，b）から成るライン B を圧縮する場合を想定する．アクティブ・ログ・レジスタがログ ID 0 を指すため，ログ ID 0 に対して一時辞書を生成する．まず，**図 5** の左側にようにログ ID 0 の全てのラインにアクセスして，ユニークなワード値（ワード値 a, b）が一時辞書に登録される．つぎに，ライン圧縮時にはラインの各ワードに対して辞書存在有無がチェックされる．存在する場合（以降，辞書ヒットと呼称）は，辞書 ID を用いてワードが圧縮される．一致するワード値が辞書に存在しない場合（以降，辞書ミスと呼称）は，当該ワード値は一時辞書に登録され，そのワードは圧縮されない．さらに，辞書ヒット，辞書ミスいずれの場合でも，そのワードの圧縮サイズ情報を圧縮後のワードに含める．対象ライン B が一時辞書により圧縮される仕組みの図 5 の右側に示す．ライン内の全ワードの圧縮結果を結合したものが圧縮ラインとなる．その後，圧縮したラインをログに書込みと，タグエントリ，LMT エントリを更新する．図 5 ではログ ID 0 の空き領域の先頭に圧縮されたライン B が書込まれる．つぎに，ライン B のメモリアドレス J に対応する LMT エントリ更新する．最後に，ログ ID 0 に対応するタグエントリにタグを追加する．

読出し動作: 読出し時には MORC はまず，メモリアドレスのインデッキシングに基づいて，対応する LMT にアクセスし，ラインステータスとログ ID を読出す．ラインステータスが無効の場合は，キャッシュミスとなる．ステータスが有効の場合，ログ ID の示すタグエントリの全タグに

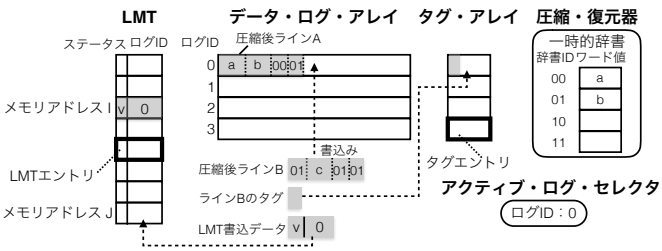

図 4 MORC の構造

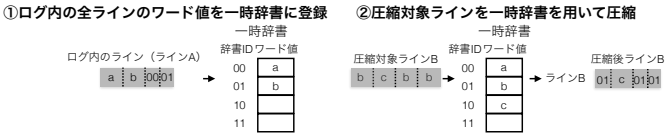

図 5 MORC における圧縮方式

対してタグ比較を行う．全て不一致の場合はキャッシュミスとなる．一致する場合（以降，タグヒットと呼称），タグエントリの先頭から何番目のタグで一致したか記録する．ここで，ログ ID x に対応するタグエントリにて先頭から n 番目でタグヒットする場合を想定する．ログ ID x のログの先頭から n 番目のラインが復元対象である．まず，ログ ID x の 1 番目から n-1 番目のラインに対して辞書生成する．つぎに，n 番目のラインを読出す．ラインのログ上の格納位置は，それより前方のワードの圧縮サイズを累計することで特定される．読出したラインの各ワードに対して，圧縮されているならば，辞書を参照して辞書 ID に対応するワード値で置き換える．非圧縮であれば，ワード値そのものを用いる．これらを結合した値を復元されたラインとして出力する．

3. 提案キャッシュ

3.1 MORC の問題点とモチベーション

MORC には，ラインの圧縮サイズの観点と圧縮・復元レイテンシの観点でそれぞれ問題がある．

辞書データ重複問題：アクティブ・ログ・セレクタの同一ログ選択方式ではログに空き領域があるかぎり，前回の書込で選択したログを選ぶ．そのため，同一ワード値が，時間的に疎らに書込まれると，それらのワード値は異なるログに格納され得る．MORC はログ内のライン間でのみ辞書を共有するため，異なるログに配置された同一のワード値は圧縮されない．複数のログ間に同一のワード値が格納される状況を**図 6** に示す．ライン P とライン S は共通のワード値 A，B を有するが，異なるログに格納されるため，ライン P とライン S のワード値 A，B は圧縮されない．一方，値考慮方式はログ間の非圧縮ワード値の重複を削減できるが，複数のログに対して辞書生成するため，同一ログ選択方式と比較しての数倍の圧縮レイテンシと圧縮エネルギーを要する．

複数のログで辞書生成することなく，ログ間の非圧縮

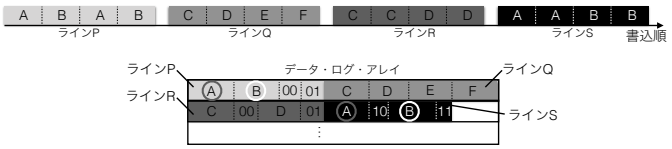

図 6 ログ間に重複ワード値が格納される仕組み

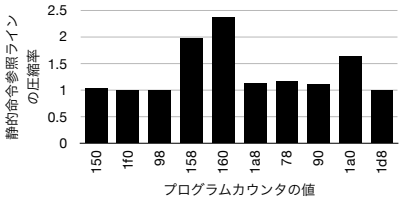

図 7 SpMV における静的命令参照ラインにおける圧縮率

ワードの重複を削減するために，本研究は静的命令参照ライン間で値の局所性が高い場合があるという性質に注目する．ベンチマークプログラム SpMV における，静的参照命令の値局所性を**図 7** に示す．静的命令参照ラインの圧縮率は式（1）のように定義される．この値が高い程，対応する静的命令参照ライン間は，値局所性が高いことを意味する．

$$\text{静的命令参照ラインの圧縮率} = \frac{\text{静的命令参照ライン数} \times \text{ラインサイズ}}{\text{ユニークなワード値の数} \times \text{ワードサイズ}} \quad (1)$$

図 7 によれば静的命令に応じて，値局所性に偏りがあることがわかる．そこで，静的命令参照ラインを同一のログに集約することで辞書の重複の削減を期待できる．

圧縮・復元に長いレイテンシを要する問題：MORC では読出しと書込には長いレイテンシを要する場合がある．MORC は読出し対象ライン前方の全てのワードにアクセスして辞書生成とワード圧縮サイズ累計によるライン格納位置の特定を行う．書込みではログの末尾にラインを書込むため，ログ内の全てのワードを読出すことになる．そのため，読出対象ラインがログの後方に位置する場合や書込にてログに多数のラインが既に格納されている場合には圧縮・復元レイテンシが長くなる．**図 14** の MORC_CONV は MORC における平均復元・圧縮レイテンシを表す．BFS-R では平均 356 サイクルを圧縮・復元に要する．図 1 によると BFS-R では 128 サイクルにて 25%以上実行時間が増加するため，BFS-R では MORC によるレイテンシの増加が性能向上を阻害する．

辞書生成とラインの格納位置特定にてアクセスするライン数を削減できる 2 つのケースがある．第一に，読出しアクセスにて辞書生成とワード圧縮サイズの累計が不要となる場合である．参照データの値局所性が低いアプリケーションでは，**図 8** に示すようにログ内の全てのラインが非圧縮となる場合がある．非圧縮ライン読出し時にはラインの全ワードが辞書ヒットしないことが明らかであるため，辞書生成が不要となる．また，ログ内の全ラインのサイズが固定であるため，前方ワードの圧縮サイズを累計することなく，ログ内のラインの格納位置を特定できる．図 8 で

図 8 読出対象の前方のラインへのアクセスが不要となるケース

はライン Q を読出す際に，ライン P へのアクセスが不要となる．第二に，辞書生成とワードの圧縮サイズ累計処理の一部を省略できる場合がある．MORC はキャッシュアクセスの度に一時辞書のデータをクリアする．しかしながら，連続するキャッシュアクセスの参照先が同一ログに属する場合では，直前に生成した辞書を再利用できる．また，ラインの格納位置は，前回のキャッシュアクセスで得た位置情報を利用することで，ワードの圧縮サイズの累計処理の一部を省略できる．

3.2 問題に対する対策

3.2.1 静的命令に基づくログ割当て

静的命令参照ライン間の値局所性を利用するために提案手法は静的命令参照ラインを可能な限り同一のログに格納させる．このログ割当てにより，ログ間の辞書の重複が削減される例を**図 9** に示す．ライン P とライン S は静的命令 X による参照ライン，ライン Q とライン R は静的命令 Y による参照ラインである．このとき，ライン P とライン S，ライン Q とライン R はそれぞれ同一のログに格納する．その結果，同一のワード値を有するラインが同じログに集約されるため，辞書の重複が削減される．

提案手法はログに静的命令のプログラムカウンタ（PC）を割り当てる．割り当てた PC をログ毎に記録するために**図 10** のログ・ステータス・テーブルの第二フィールドを用いる．静的命令参照ラインが一つのログに収まらない場合は，複数のログに同一の PC が割り当てられる．それらのログから書込対象ログを区別するためにログにアクティブステータスを持たせる．同一の PC を割り当てられた複数のログの内，アクティブステータスが有効となるログとは唯一つである．そのログが対応する PC に対する書込み対象のログとなる．

アクティブ・ログ・セレクタは，ログ・ステータス・アレイの全エントリから書込みやフィルする命令の PC を検索する．しかしながら，ログ・ステータス・アレイの全エントリにアクセスするため，検索には長いレイテンシを要する．そこで，PC とログ ID のマッピングをキャッシングした PC・マップ・キャッシュを設ける．PC・マップ・キャッシュには，アクティブステータスが有効なログに限り PC とログ ID のペアが記録される．

提案するアクティブ・ログ・セレクタの動作を**図 11** に示す．アクティブ・ログ・セレクタは，まず，PC・マップ・キャッシュの全エントリに対して，書込・フィルする命令の PC と比較する．一致する PC が存在する場合には対応するログ ID を出力する．一致する PC が存在しない

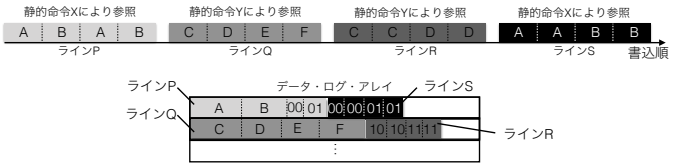

図 9 静的命令参照ライン間の値局所性の利用

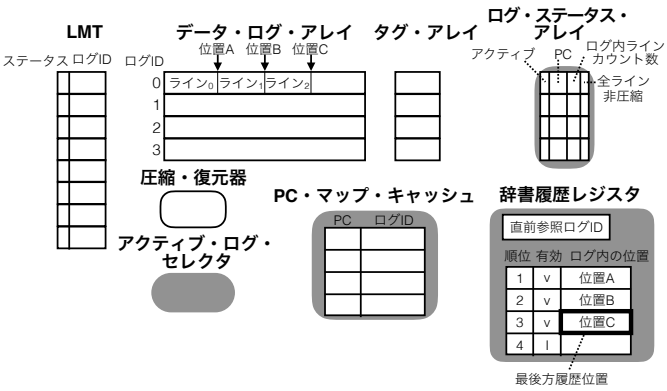

図 10 提案型 MORC の構造

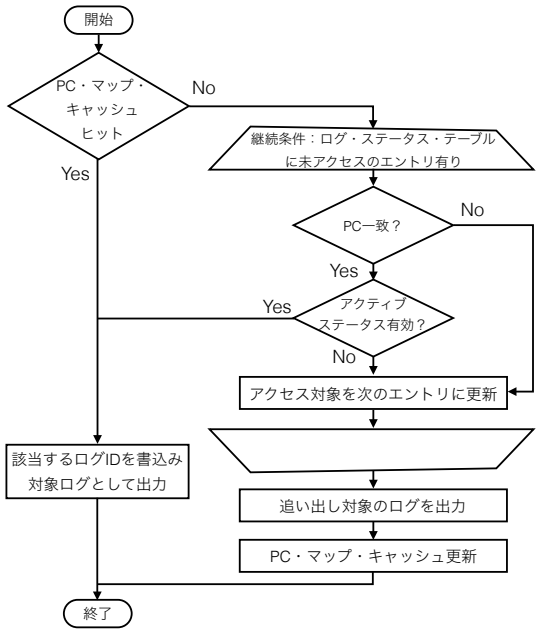

図 11 提案するアクティブ・ログ・セレクタの動作

場合，ログ・ステータス・テーブルを検索する．ログ・ステータス・テーブル内の PC 検索では，テーブルの各エントリを読出して，その PC 部分と書込みやフィルを行う命令の PC を比較する．PC が一致かつログがアクティブとなるエントリが存在すれば，そのログ ID を出力する．条件を満たすものが存在しない場合，新たなログを用意して，そのログに検索している PC を割当てて出力する．さらに，この時その PC と新たなログ ID のペアを PC・マップ・キャッシュに登録する．古い情報を更新するために，PC・マップ・キャッシュから PC を検索する．既に PC が登録されているならば，ログ ID 部分を新しいログ ID に更新する．PC が未登録であるならば，新しいエントリを割り当てて，PC と新たなログ ID のペアを書込む．PC・マップ・キャッシュの更新は，この場合に加えてログやタグエントリに空き領域がなく，新たなログに置換えられた場合にも行われる．

3.2.2 ログ内最大ライン数制限

ログに格納可能なライン数に制限を設ける．本手法をログ内ライン数制限と呼ぶ．アクセスする前方ラインの数を抑制することで，アクセスレイテンシが低減する．この手法の実現のために，ログ・ステータス・アレイの第 3 フィールドに，ログ内のラインの数を記録する．ラインをログに格納する度に，該当するログのカウント値がインクリメントされる．同時に，そのカウント値が閾値を超えるかを判定する．閾値を超えた場合，ラインの格納先には新しいログを割り当てる．

3.2.3 非圧縮ラインアクセス低遅延化

ログ内の全ラインが非圧縮の場合，前方ラインへのアクセスを省略して，読出し対象ラインにアクセスする．ログ内の全ラインが非圧縮であるかを知るために，ログ・ステータス・アレイの第 4 フィールドに，ログ内の全ラインが非圧縮かを記録する．読出時に，ログ・ステータス・アレイのログに該当するエントリにアクセスして，全ラインが非圧縮かを調査する．全ラインが非圧縮の場合は，読出対象ラインだけにアクセスする．読出対象ラインのログ内格納位置は，(ログ内ライン格納順位-1) × 非圧縮ラインサイズにより求められる．ここで，ログ内ライン格納順位は，当該タグのタグエントリ内の格納順位と対応するため，タグ比較により定まる．

3.2.4 辞書再利用

本研究は連続するキャッシュアクセスの参照先が同一ログに属する場合に，直前に生成した辞書を再利用することで，辞書生成の処理を間引く．これに加えて，直前のキャッシュアクセスで判明したログ内のライン格納位置を利用することで，直後のキャッシュアクセスにおけるライン格納位置特定の処理の一部を省略する．

これを実現するためのハードウェアを図 10 の辞書履歴レジスタに示す．同一ログ ID に連続アクセスを行ったかを判定するために，直前に参照したラインが属するログ ID を記録するためのレジスタを設ける．さらに，判明した全てのライン格納位置を記録するためのテーブルを用意する．テーブルにはログ先頭からのラインの格納順位とそのラインのログ上の格納位置を格納するための記憶領域を用意する．テーブルの各エントリには有効ビットを設ける．テーブルのエントリ数は，ログ内最大ライン数制限によって定められた値となる．テーブルの有効エントリの内，順位の値が最も大きいエントリのログ内位置情報を最後方履歴位置と呼ぶ．

本手法適用時のライン復元の動作を**図 12** に示す．まず，アクティブ・ログ・セレクタにより選択されたログ ID と

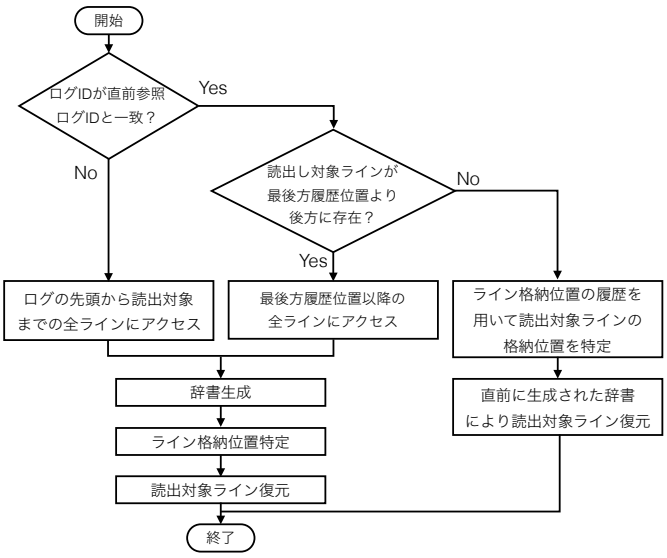

図 12 辞書再利用適用時のライン復元動作

直前参照ログ ID を比較する．ログ ID が一致しない場合は，従来型 MORC と同様に，ログの先頭から読出し対象ラインまでアクセスする．ログ ID が一致する場合は，読出し対象ラインの格納位置が，最後方履歴位置より後方にあるかに依存して動作が異なる．この条件が成り立つ場合は，最後方履歴位置以降の全ラインにアクセスして直前に生成された辞書に，未登録のユニークな値を追加する．さらに，最後方履歴位置を初期値として，これ以降の全ワードの圧縮サイズを累計することでライン格納位置を特定する．上の条件が成り立たない場合は，辞書生成が不要で，かつ，読出し対象ラインのログ内格納位置は辞書履歴レジスタから得られる．そのため，ログの前方ラインへのアクセスが不要となる．

3.3 対策の利点と欠点

静的命令に基づくログ割当により，ログ間の辞書データの重複を削減できる場合がある．しかしながら，PC の検索のために圧縮レイテンシが増加する．さらに，本手法は静的命令の数が多いプログラムでは，ログの置換えが頻発することで，キャッシュミス回数が増加する恐れがある．ログ内最大ライン数制限と辞書再利用は圧縮・復元レイテンシの削減を可能にする．しかしながら，ログ内ライン数制限は，ログの置換え頻度を増加させる可能がある．また，非圧縮ラインアクセス低遅延可により復元レイテンシが削減される．

4. 性能評価

4.1 実験方法

提案手法の性能向上の度合いを明らかにする．MORC に 4 つの対策を施したものを GPU タイミング・シミュレータ (GPGPU-Sim) [12] に実装し，プログラム実行クロックサイクル数を評価する．本評価では**表 1** に示す 8 つを評価対象とする．

表 1 評価対象

評価対象	説明
CONV_16KB	16KB の従来型 L1 キャッシュ
MORC_CONV	従来型 MORC
MORC_PROP	4 つの対策を全て適用
SINST_BASE_MGMT	静的命令に基づくログ割当だけ適用
MAX_LINES	ログ内最大ライン数制限だけ適用
DIC_REUSE	辞書再利用だけ適用
UNCOMP_MGMT	非圧縮ラインアクセス低遅延化だけ適用
MORC_PROP_ADAP	4 手法の内，最も性能向上に貢献するものだけ適用

表 2 MORC のパラメータ設定

パラメータ	設定値
LMT セット数	256
LMT 連想度	4
ログエントリ数	124
ログサイズ	1088 bit
タグエントリ数	124
タグエントリサイズ	352 bit
PC・マップ・キャッシュエントリ数	32
ログ内ライン数の最大値	5

表 3 MORC の各ハードウェアコンポーネントの容量とベースラインに対する面積オーバーヘッド

要素	サイズ
LMT	1.13 KB
一時辞書	136 B
データ・ログ・アレイ	17 KB
タグ・アレイ	5.37 KB
ログ・ステータス・アレイ	592 B
PC・マップ・キャッシュ	156 B
辞書履歴レジスタ	10.25 B
MORC と提案手法による面積オーバーヘッド	7.81 KB

4.2 評価環境

GPGPU-Sim のパラメータは GTX480 [13] に基づいて**表 5** のように設定する．また, GTX480 の 16KB L1 キャッシュの構成をベースラインとする．

ベンチマークセットには Parboil [14]，ISPASS09 [12], Rodinia [15], CUDA-SDK [16] を用いる．L1 キャッシュ容量の増減に対して性能の変化幅が微少なプログラムではデータ圧縮に基づくキャッシュがほとんど性能向上に寄与しないため，そのようなプログラムは，実験対象プログラムから除外される．L1 キャッシュ容量が性能向上に寄与するかは，L1 キャッシュサイズを 16 KB から 128 KB に増加させた場合にベースラインに対する実行サイクル削減率が 0.01 以上となるかにより判断する．実験に用いたプログラムを**表 4** に示す．プログラムを実行命令数が 1 億に達するまで実行する．命令数が 1 億命令に満たないプログラムは，終端まで実行する．

MORC の設定パラメータ値を**表 2** に示す．また，アク

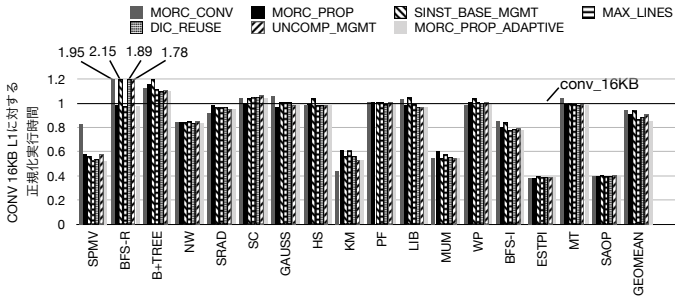

図 13 提案手法の実行性能

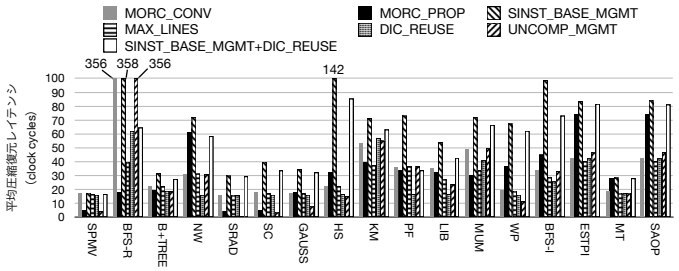

図 14 提案手法における平均復元レイテンシ

ティブ・ログ・セレクタには同一ログ選択方式を用いる．MORC および提案手法の各ハードウェアコンポーネントの容量とベースラインに対する面積オーバーヘッドを**表 3** に示す．この内，提案手法による面積オーバヘッドは，ログ・ステータス・アレイ，PC・マップ・キャッシュ，辞書履歴レジスタの合計値の 0.74 KB である．

静的命令に基づくログ選択における PC 検索では，PC・マップ・キャッシュやログ・ステータス・アレイの複数のエントリにアクセスするため長いレイテンシが要する．そこで，本評価では提案手法のレイテンシオーバヘッドとして PC 検索を考慮する．PC・ログ・マップキャッシュおよびログ・ステータス・アレイの 1 エントリの読出しサイクル数は，CACTI 6.5[17] によって得られた 1 サイクルとする．

4.3 性能評価結果

実行時間の評価結果を**図 13** に示す．これらの値は従来型キャッシュ 16KB 搭載時の実行時間（CONV_16KB）で正規化されている．分析のために，各評価指標の圧縮・復元レイテンシを図 14 に示す．改良型 MORC は，MORC，ベースラインそれぞれに対して，実行時間の幾何平均を 3 ポイント，8.7 ポイント削減する．MORC_CONV から MORC_PROP の正規化実行時間を減算した値に基づきベンチマークプログラムを**表 6** のように 3 種類に分類する．

グループ I：NW，PF，ESTPI，SAOP では，4 つの手法を個別に適用した場合の実行時間が MORC_CONV のものと同程度となり，手法の効果が得られない．その結果，MORC_PROP では MORC_ORG に対して性能向上および性能効果が見られない．一方, HS は, SINST_BASE_MGMT 単体では性能低下し, 他の手法による性能向上効果が見られないにも関わらず，MORC_PROP と MORC_ORG と同程度実行時間となる．これは, 複数の手法を組合わせることで, SINST_BASE_MGMT による性能低下を打ち消す性能向上効果が存在するためである．図 14 に示すように，HS では DIC_REUSE 単体では, レイテンシ削減効果が低い．しかしながら, DIC_REUSE と SINST_BASE_MGMT を組み合わせることで, 図 14 の SINST_BASE_MGMT+DIC_REUSE に示すようにレイテンシが 142 サイクルから 80 サイクルに削減される．

グループ II：SpMV，GAUSS，MT ではいずれの手法

表 4 実験に用いたベンチマークプログラム

プログラム名	説明
SpMV	疎行列ベクトル積
BFS-R	幅優先探索 (Rodinia ベンチマーク・セット)
B+TREE	B+木の探索
NW	Needleman-Wunsch
SRAD	Speckle Reducing Anisotropic Diffusion
SC	データ・ストリーム・クラスタリング
GAUSS	ガウスの消去法
HS	プロセッサの熱シミュレーション
KM	k 平均法
PF	Path finder
LIB	LIBOR マーケット・モデル
MUM	DNA シーケンス・アライメント
WP	数値気象予報
BFS-I	幅優先探索 (ISPASS ベンチマーク・セット)
LPS	離散的ラプラス変換
EPi	円周率の近似計算
MT	メルセンヌ・ツイスタによる疑似乱数生成
SAOP	ヨーロピアン・アジアンオプションの価格付け

表 5 GPGPU-SIM の設定値

コア数	15 SMs, 480 CUDA cores, 12 sub partitions
CUDA core	700 MHz, SIMT width=32, in-order
Resources / Core	Max. 1536 threads (48 warps, 32 threads / warp), 48 KB shared memory
Caches / Core	16 KB, 4way, and 1 cycle latency L1 cache, 12 KB texture cache, 8 KB constant cache, 128 B line size
L2 Cache	64 KB 8way per sub partition, 2 sub partitions per a memory controllers, 128 B line size, total 768 KB
Warp Scheduling	Two-level warp scheduling [18]
Memory Model	6 memory controllers (MC), FR-FCFS scheduling, 16 DRAM-banks/MC, 4 KB row size, 924 MHz memory clock
GDDR Timing	Based on hynix H5GQ1H24AFR [19] (tCL=16, tRP=12, tRC=40, tRAS=28, tRCD=12, tRRD=6, tCDLR=5, tWR=12)

表 6 ベンチマークプログラムの分類

グループ名	基準	ベンチマークプログラム
グループ I	減算値の絶対値が 1 ポイント未満かつ，性能低下 1%未満	NW，HS，PF，ESTPI，SAOP
グループ II	減算値が 1 ポイント以上	SPMV，BFS-R，SC，GAUSS，LIB，BFS-I，MT
グループ III	減算値が-1 ポイント以上	B+TREE，SRAD，KM，MUM，WP

でも性能向上するため，MORC_PROP においても性能向上を達成する．BFS-R では MAX_LINES が性能向上に貢献する．BFS ではラインは平均 50 分の 1 程度に圧縮されため，ログに多数のラインが存在する．このとき，図 14 の MORC_CONV に示すように BFS-R の平均圧縮・復元レイテンシは 356 サイクル要するが，MAX_LINES により 40 サイクルにまで削減される．LIB と BFS-I では SINST_BASE_MGMT による性能向上は得られないが，その他の手法で性能向上が得られる．SC はいずれに手法でも単体では性能向上が得られないが，MORC_PROP は MORC_CONV に対して性能向上する．この理由については今後調査する．

グループ Ⅲ：B+TREE と WP においては SINST_BASE_MGMT にて性能が低下し，他の手法単体では MORC_CONV と同程度の性能となるため，MORC_PROP の性能が低下する．MUM は MAX_LINES を単体で適用すると性能が低下する．これは，MUM は MAX_LINES の L1 レイテンシ削減効果による性能向上が得られにくく，格納ライン制限により性能低下しやすいためである．図 1 と図 2 に示すように，MUM は L1 サイズに対して性能がセンシティブに対して，L1 レイテンシを変化させも性能が変化しにくい．SRAD と KM はいずれの手法においても性能低下するため，MORC_PROP の性能も MORC_CONV に劣る．

4.4 最も効果の高い手法のみ適用時の性能評価

プログラム実行前に 4 つの手法から実行時間を最小にする手法が既知と仮定し，実行時間最小となる手法を選択したときの性能を評価する．MORC_PROP_ADAPTIVE は，MORC，ベースラインそれぞれに対して，実行時間の幾何平均をそれぞれ 8.6 ポイント，14.2 ポイント削減する．MORC_PROP_ADAPTIVE は，欠点の影響が大きい手法を，適用しないことで，MORC_PROP よりも高い性能向上を示した．手法選択の方法は今後の課題である．

5. おわりに

GPU の性能阻害要因の一つに L1 キャッシュにおける競合の多発がある．本研究はこの問題を緩和するために，CPU 向けデータ圧縮に基づくキャッシュアーキテクチャ MORC を改良する．静的命令が参照するライン間の値局所性を利用したライン配置を行うことで，ログ間の辞書データの重複を削減する．さらに，辞書生成の一部または全ての処理を省くことで圧縮・復元に要するレイテンシを削減する．評価の結果，提案手法は従来型キャッシュの正規化実行時間を平均 8.7 ポイント削減することが明らかになった．さらに，適切な手法をアプリケーションに応じて選択することで従来型キャッシュの正規化実行時間を平均 14.2 ポイント削減できる可能性があることが明らかになった．

参考文献

[1] NVIDIA: GeForce GTX 1080 Specifications, https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-1080/specifications.
[2] Rogers, T. G., OConnor, M. and Aamodt, T. M.: Cache-Conscious Wavefront Scheduling, *2012 45th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 72–83 (2012).
[3] Xie, X., Liang, Y., Wang, Y., Sun, G. and Wang, T.: Coordinated static and dynamic cache bypassing for GPUs, *2015 21th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 76–88 (2015).
[4] Duong, N., Zhao, D., Kim, T., Cammarota, R., Valero, M. and Veidenbaum, A. V.: Improving Cache Management Policies Using Dynamic Reuse Distances, *2012 45th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 389–400 (2012).
[5] Atoofian, E.: Compressed L1 data cache and L2 cache in GPGPUs, *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–8 (2016).
[6] Nguyen, T. M. and Wentzlaff, D.: MORC: A manycore-oriented compressed cache, *2015 48th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 76–88 (2015).
[7] Panda, B. and Seznec, A.: Dictionary sharing: An efficient cache compression scheme for compressed caches, *2016 49th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12 (online), DOI: 10.1109/MICRO.2016.7783704 (2016).
[8] Atoofian, E.: Compressed L1 data cache and L2 cache in GPGPUs, *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–8 (2016).
[9] Sardashti, S., Seznec, A. and Wood, D. A.: Yet Another Compressed Cache: A Low-Cost Yet Effective Compressed Cache, *ACM Trans. Archit. Code Optim.*, Vol. 13, No. 3, pp. 27:1–27:25 (2016).
[10] Pekhimenko, G., Seshadri, V., Mutlu, O., Kozuch, M. A., Gibbons, P. B. and Mowry, T. C.: Base-delta-immediate compression: Practical data compression for on-chip caches, *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 377–388 (2012).
[11] Arelakis, A. and Stenstrom, P.: SC2: A statistical compression cache scheme, *2014 41th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pp. 145–156 (2014).
[12] Bakhoda, A., Yuan, G. L., Fung, W. W. L., Wong, H. and Aamodt, T. M.: Analyzing CUDA workloads using a detailed GPU simulator, *2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 163–174 (2009).
[13] NVIDIA: GeForce GTX 480 Architecture, https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/architecture.
[14] Stratton, J. A., Rodrigues, C. I., Sung, I., Obeid, N., Chang, L., Anssari, N., Liu, D. and Hwu, W.: Parboil: A revised benchmark suite for scientific and commercial throughput computing, Technical report, Center for Reliable and High-Performance Computing (2012).
[15] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S. H. and Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing, *2009 IEEE In-*

ternational Symposium on Workload Characterization (IISWC), pp. 44–54 (2009).
[16] NVIDIA: CUDA C/C++ SDK Code Samples (2011).
[17] Muralimanohar, N., Balasubramonian, R. and Jouppi, N. P.: Cacti 6.0: A tool to model large caches (2009).
[18] Narasiman, V., Shebanow, M., Lee, C. J., Miftakhutdinov, R., Mutlu, O. and Patt, Y. N.: Improving GPU performance via large warps and two-level warp scheduling, *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 308–317 (2011).
[19] Hynix Semiconductor: *1Gb (32Mx32) GDDR5 SGRAM H5GQ1H24AFR* (2009).