

GPUプログラムにおける 静的参照関係を表すハイパーグラフの分割を用いた 参照効率のよいデータ配置

比嘉 慎哉^{1,a)} 置田 真生^{2,b)} 萩原 兼一^{2,c)} 伊野 文彦^{2,d)}

概要：本報告では，GPU プログラムを対象に，メモリ参照効率化を目的としたデータ配置の最適化問題をヒューリスティックに解く手法を提案する．具体的には，グローバルメモリ上の領域を L1 キャッシュラインと同程度の大きさのチャンクに分割し，プログラムの静的なアクセスパターンを入力として，各スレッドブロックおよびワープがアクセスするチャンク数を最小化するデータ配置を求める．この問題をハイパーグラフの分割問題に変換し，それを効率的に解く分割アルゴリズムを提案する．評価実験の結果，連続アクセスの増大に特化したデータ配置と比較して，L1 キャッシュミス数を 70%に削減し，最大 1.28 倍の速度向上を達成した．分割に要する時間は，頂点数が 2000 万を越える大規模なハイパーグラフに対して 38 分であった．提案手法は，静的グラフ処理のように，不規則なアクセスパターンを含む GPU プログラムの高速化に有用である．

1. はじめに

GPU は広いメモリ帯域幅を持ち，メモリアクセス律速なアプリケーションの高速化に有用なアクセラレータである．ただし，グローバルメモリ（以降，GM）のレイテンシが比較的大きいため，アクセスパターンに応じて GM 上のデータ配置を変更することで実効メモリ性能を向上するメモリ参照の効率化が重要である．

一般に，任意のアクセスパターンに対する効率的なデータ配置の決定は組み合わせ最適化問題を解く必要がある．行列演算およびステンシル計算などアクセスパターンが規則的な場合には，データ配置の最適化を効率的に解ける [1]．しかし，不規則なアクセスパターンを持つ計算を対象とする場合は難しい．例えば，データ配置の入れ替えとワープの再構成だけで非コアレスアクセスを最小化することは，NP 完全 [2] である．

これまでに，不規則なアクセスパターンを持つ GPU プログラムを対象としてメモリ参照を効率化する研究が存在する [2-5]．Yang ら [5] は，スレッドおよびスレッドブロック（以降，TB）を併合することで GM へのアクセス

を削減する手法を提案した．また，Zhang ら [3] はデータを複製することでアクセスパターンを単純化し，非コアレスアクセスを 0 回に削減した．さらに Wu ら [2] は unnecessary 複製を避けることで複製するデータ量を削減した．

データを複製せずにアクセスパターンを改善する研究に Flint [6, 7] がある．Flint は，有向非循環グラフで定義可能な参照関係を持つ連立微分方程式を入力として，それを解析的に解く CUDA プログラムあるいは OpenMP プログラムを自動生成する．生成の過程では，計算順序の交換，データ配置の変更，数式の統合，および冗長計算の挿入などの最適化を出力プログラムに施す．角田ら [7] は，生成されるプログラムのアクセスパターンを改善し，参照局所性を向上する手法を提案した．この手法は，プログラム全体を通して連続アクセスが増大するようにデータ配置と計算順序を入れ替える．OpenMP プログラムだけでなく，CUDA プログラムの L2 キャッシュ効率を改善できることを示した．ただし，角田らの手法は参照局所性を定量的に評価した全体最適化を行わない．

そこで本報告では，GPU プログラムの空間的局所性を定量化した評価モデルを作成し，定量的予測に基づいてデータ配置を最適化する手法を提案する．具体的には，GM 上の領域を L1 キャッシュラインと同程度の大きさのチャンクに分割し，各 TB およびワープがアクセスするチャンク数を最小化するデータ配置を求める．提案手法は，プログ

¹ 大阪大学 基礎工学部
² 大阪大学 大学院情報科学研究科
a) s-higa@ics.es.osaka-u.ac.jp
b) okita@ist.osaka-u.ac.jp
c) hagihara@ist.osaka-u.ac.jp
d) ino@ist.osaka-u.ac.jp

ラムを静的に解析して得られるアクセスパターンを入力として、データ配置を出力する。データの複製および計算順序の変更は行わない。提案手法は、TB ごとに L1 キャッシュ（以降、FLC）が独立する GPU の特徴を利用するため、CPU プログラムは対象外である。また、ハードウェアキャッシュを持つ Fermi 以降の NVIDIA GPU を用いる場合に適用可能である。

提案手法は、データ配置の最適化問題をハイパーグラフの分割問題に変換して近似解を求める。この分割問題では、頂点クラスタの大きさを一定以下に制限する必要がある。既存の著名なハイパーグラフ分割ライブラリに hMETIS [8] があるが、hMETIS はクラスタの大きさに関して保証のないアルゴリズムであるため、今回の目的には利用できない。そこで、クラスタの大きさの上限を保証したうえで切断数を削減する新たなハイパーグラフ分割アルゴリズムを提案する。なお、提案アルゴリズムは貪欲法に類似したヒューリスティックな手法であり、厳密解を保証しない。

以降では、2 節で関連研究を紹介し、3 節でハイパーグラフに関する用語を定義する。4 節では取り組む問題を定義し、5 節で提案手法を説明する。6 節では提案手法の効果を評価する。最後に 7 節で本報告についてまとめる。

2. 関連研究

Yang らは、GPU プログラムの静的なデータ依存を解析して、メモリ参照効率のよいプログラムに変換する手法 [5] を提案した。複数回アクセスするデータを共有メモリに格納し、さらに同一データにアクセスするスレッドおよび TB を 1 つの TB に併合することで、GM から共有メモリへのコピー回数を削減する。提案手法と比較すると、データアクセスを併合して GM へのアクセス回数を削減する点に類似がある。ただし、Yang らの手法は併合の対象を隣接する TB に限定する点に対し、提案手法は評価値に基づいて全体から選択する点に相違がある。

また、動的に変化するアクセスパターンを持つ GPU プログラムを対象に、データの複製を利用してメモリ参照を効率化する研究 [2,3] がある。Zhang らの手法 [3] は、複数回アクセスするデータに対して、アクセス回数に等しい数の複製を作成する。アクセスごとに異なる複製を参照することで、全アクセスのコアレスニングを可能にした。さらに、Wu ら [2] は TB ごとに同一データの複製を 1 つに削減することで、全アクセスのコアレスニングを保ったまま冗長な複製を排除する方法を提案した。これらの手法は非コアレスアクセスを必ず最小化できる点で有用であるが、本来のメモリ使用量が VRAM の容量に近い大規模なプログラムへの適用は難しい。一方、提案手法はデータを複製しないため、Wu らの手法と比較してより大規模なプログラムに適用可能である。

3. ハイパーグラフに関する諸定義

ハイパーグラフ G を、頂点の集合 V および辺の集合 E を用いて $G = (V, E)$ と表す。辺 $e \in E$ を V の空でない部分集合として表し、集合の大きさ $|e|$ を辺の次数と呼ぶ。なお、本研究では辺の重複を許容し、 E を多重集合とする。また、頂点 v を含む辺を v の接続辺と呼び、その集合 $\mathcal{E}(v)$ を次の式 (1) で定義する。

$$\mathcal{E}(v) = \{e \mid e \in E, v \in e\} \quad (1)$$

接続辺の数 $|\mathcal{E}(v)|$ を頂点の次数と呼ぶ。

本研究では、ハイパーグラフ $G = (V, E)$ の頂点集合の分割のみを扱う。 V の分割 V' に対して、 V' の元をクラスタと呼び、頂点 $v \in V$ を含むクラスタを $C_{V'}(v)$ と表す。また、辺 $e \in E$ が架かるクラスタの数をスパンと呼び、式 (2) で定義する。

$$q_{V'}(e) = |\{X \in V' \mid X \cap e \neq \emptyset\}| \quad (2)$$

全ての辺に対するスパンの総和を $Q_{V'}(E)$ で表す。

$$Q_{V'}(E) = \sum_{e \in E} q_{V'}(e) \quad (3)$$

ハイパーグラフ分割問題は、各クラスタの大きさに関する条件を満たし、かつ辺に関する関数を最適化する分割を求める問題である。本報告では、前者の条件を分割の制約と呼び、後者の関数最適化を分割の要件と呼ぶ。典型的な条件はクラスタの大きさの均衡である。また、典型的な要件は切断辺すなわち $|q(e)| > 1$ となる辺の数の最小化である。他にも多様な要件が存在し、SOED (the Sum Of External Degrees), Scaled Cost および Absorption など [9] がある。

一般に、ハイパーグラフの分割は NP 困難 [10] であり、いくつかのヒューリスティックアルゴリズムが存在する。例えば、Karypis らの multilevel k -way 分割 [11] は、式 (4) の制約の下で切断辺あるいは SOED を最適化する k 個の分割を求めるヒューリスティックスである (c は正の実数)。

$$\forall X \in V', \frac{|V|}{ck} \leq |X| \leq \frac{c|V|}{k} \quad (4)$$

なお、 k -way 分割の実装の 1 つ hMETIS は、必ず k 個のクラスタに分割するため、入力グラフによっては式 (4) を充足できない可能性がある。

4. 問題の定義

GPU プログラムのアクセスパターンを $P = (V, T, B)$ と表現する。まず、 V はプログラムがアクセスする GM 上のデータ集合を表し、 $v \in V$ はデータを保持する 1 つの変数を表す。簡単のため、全変数は同じ大きさのデータ (double 型) を保持すると仮定する。次に、 T はタスクの集

合を表す. $t \in T$ は1つのスレッドが実行する計算に対応し, $t = (S, L)$ と定義する. S および L は V の部分集合であり, それぞれ t がストアおよびロードの対象とするデータ群を表す. なお, T は多重集合であり, アクセスパターンが重複するタスク t_1 および t_2 ($t_1 = t_2$) を含む可能性がある. 最後に B は TB の集合を表す. すなわち, B は T の分割である. ただし, 全ての $b \in B$ について, $|b|$ はプログラマが指定した上限値 β 以下である. また, $b \in B$ の実行においてストアおよびロードの対象となるデータの集合をそれぞれ $V_S(b)$ および $V_L(b)$ (式 (5) および (6)) とする.

$$V_S(b) = \{v \mid (S, L) \in b, v \in S\} \quad (5)$$

$$V_L(b) = \{v \mid (S, L) \in b, v \in L\} \quad (6)$$

本研究は, アクセスパターン P が静的に決まる場合のみを対象とする. すなわち, P が実行中に変化するプログラムを考慮しない. また, タスク $t = (S, L)$ が分岐を含む場合, S および L は条件の真偽に関わらず, アクセスする可能性のある全ての変数を含む.

データの配置を, V の各元に対して GM における相対位置を指定する単射な写像 f で定義する (式 (7))

$$f: V \rightarrow \mathbb{W}_n \quad (7)$$

\mathbb{W}_n は n 以下の非負整数であり, $n \geq |V|$ を満たす. 簡単のため, GM の容量は $|V|$ より十分大きいと仮定する. 式 (7) が示すように, V の全元を連続的に配置する必要はない.

本研究で扱う問題は, 与えられたアクセスパターン P に対する, FLC のミス率が低いデータ配置 f の決定である. FLC と下位メモリ間のデータ転送 (以降, TFL) の回数を表す関数 F を用いて, この問題を式 (8) に示す最小化問題として定義する.

$$\arg \min_f F(P, f) \quad (8)$$

5. 提案手法

本報告では, FLC の容量が十分大きいと仮定した場合に, 式 (8) の近似解を得るヒューリスティック手法を提案する. まず, $F(P, f)$ の下限を最小化する問題をハイパーグラフの分割に帰着し, 制約および要件を明らかにする. 次に, 制約を満たしかつ要件に近い分割を得るためのハイパーグラフ分割アルゴリズムを提案する. 最後に, 提案アルゴリズムと組み合わせ, GPU のワーブ単位の局所性を重視した分割を求める手法を提案する. 提案手法の分割結果に従って f を定めることで, プログラムの高速化を目指す.

5.1 ハイパーグラフ分割問題への帰着

まず, FLC のキャッシュラインを考慮し, GM 上の領域

を分割する. キャッシュラインの大きさ γ ($\gamma \ll n$) を用いて, GM 上の V が存在する領域 $[0, n]$ を大きさ γ の領域 (以下, チャンク) に区分する. TFL はチャンク単位で起こるため, チャンク内のデータの順序は TFL の回数と無関係である. したがって, 全ての $v \in V$ について順序を決定する必要はなく, チャンクへの割当を決定すればよい. そこで, 全ての $X \in V'$ について $|X| \leq \gamma$ を満たす V の分割 V' を求め, 各クラス X を1つのチャンクに割り当てるようにデータ配置を決定する.

次に V' に従うデータ配置 g を定義する. g は次の条件 (9) および (10) を満たす.

$$g \in \text{Map}(V, \mathbb{W}_n) \quad (9)$$

$$\forall X \in V', v \in X \wedge u \in X \rightarrow \lfloor \frac{g(v)}{\gamma} \rfloor = \lfloor \frac{g(u)}{\gamma} \rfloor \quad (10)$$

以降, クラスタとチャンクの対応が1対1であることを前提とする.

さらに, V' を用いて TFL 回数の下限 $F_l(P, g)$ を定義する. ロードとストアでは独立して TFL が起こると仮定する. ある TB b の実行中にキャッシュ置換が発生しない場合, TFL 回数は b がアクセスするチャンクの総数に等しい. したがって, $F_l(P, g)$ は以下の式 (11) で求まる.

$$F_l(P, g) = \sum_{b \in B} (|\{X \in V' \mid X \cap V_S(b) \neq \emptyset\}| + |\{X \in V' \mid X \cap V_L(b) \neq \emptyset\}|) \quad (11)$$

キャッシュ置換によって TFL 回数が増加する可能性があるため, 式 (12) が成り立つ.

$$F(P, g) \geq F_l(P, g) \quad (12)$$

本報告では, 式 (8) の代替として式 (13) の最小化問題を解く手法を提案する.

$$\arg \min_{V'} F_l(P, g) \quad (13)$$

提案手法は, キャッシュ置換が発生しない場合, すなわち SM (Streaming Multiprocessor) が同時に実行する TB の集合 $B_1 \subseteq B$ について $\sum_{b \in B_1} |V_S(b) \cup V_L(b)|$ が FLC の容量より小さい場合に有用である. キャッシュ置換を考慮して式 (8) を解く方法は, 今後の課題である.

式 (13) はハイパーグラフの分割に帰着する. 頂点集合 V および, 辺集合 $E = \{V_S(b) \mid b \in B\} \cup \{V_L(b) \mid b \in B\}$ からなるハイパーグラフを $G = (V, E)$ とする. 図 1 に示すように, G の分割における辺 $e \in E$ のスパンは, e に対応する TB がロード (あるいはストア) 時にアクセスするチャンク数に等しい. すなわち, $F_l(P, g) = Q_{V'}(E)$ である.

したがって, 式 (13) は, 以下の制約および要件を満たす G の分割 V' の決定と等価である.

$$\text{制約 (C): } \forall X \in V', |X| \leq \gamma$$

$$\text{要件 (R): } Q_{V'}(E) \text{ の最小化}$$

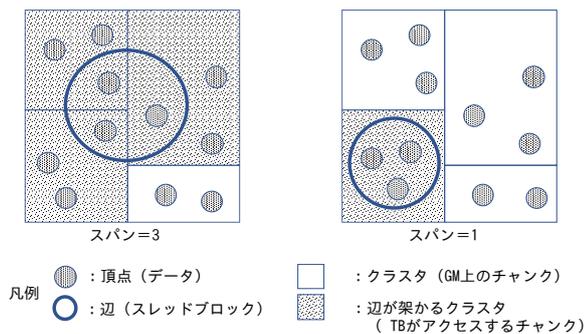


図 1 スパンと TB がアクセスするチャンク数の対応

5.2 ハイパーグラフ分割アルゴリズム

ハイパーグラフ $G = (V, E)$ およびクラスタの大きさの上限 γ が与えられて、制約 (C) の下で要件 (R) を目指す分割 V' を求めるアルゴリズム γ -制限分割を提案する。さらに、 G が疎グラフである場合に有用な γ -制限分割の高速化を提案する。

5.2.1 γ -制限分割アルゴリズム

γ -制限分割は、粒度の小さい分割 V'_0 から始めて、スパンの減少量の下限が大きい順にクラスタを併合する貪欲法の一種である。最も単純な V'_0 として、各頂点 $v \in V$ を 1 つずつ含むクラスタの集合がある。 V'_0 の詳細は 5.3 節で後述する。

異なるクラスタに属する頂点 u および v について、 $C_{V'}(u)$ および $C_{V'}(v)$ を併合すると、 v および u を結ぶ辺のスパンが 1 減少する。 v および u を結ぶ辺の集合を共有辺 $E_c(u, v)$ と呼び、式 (14) で定義する。

$$E_c(u, v) = \mathcal{E}(u) \cap \mathcal{E}(v) \quad (14)$$

$C_{V'}(u)$ および $C_{V'}(v)$ の併合によって、スパンの和 $Q_{V'}(E)$ は少なくとも $|E_c(u, v)|$ 減少する。

したがって、提案手法は $|E_c(u, v)|$ の大きい頂点对を基準にクラスタを併合する。ただし、同一の辺集合 $Y \subseteq E$ を共有辺とする頂点对 $\{u, v\}$ は複数存在する。局所解を避けるため、それらの頂点对は一括して併合パターンを求める必要がある。

概要をアルゴリズム 1 に示す。あらかじめ全ての共有辺を列挙し、共有辺を大きさの降順に並べ替える (2~3 行目)。次に、共有辺ごとにクラスタを併合する (4~11 行目)。同一の共有辺をもつ頂点对を平坦化した集合 U をもとに、 U の元を含むクラスタ集合 C について併合パターンを求める。併合パターンの計算は、 C の分割 C' の計算に帰着する。制約 (C) より、全ての $X \in C'$ について $|X| \leq \gamma$ を満たす必要がある。また、要件 (R) より、 $|C'|$ の最小化が望ましい。これは NP 困難問題の 1 つビンパッキング問題 [12] と等価である。そこで、First-Fit 減少アルゴリズム [12] と呼ばれる $\frac{3}{2}$ -近似アルゴリズムを用いて C' を求める (7 行目)。

アルゴリズム 1 γ -bound partitioning

Input: ハイパーグラフ (V, E) , 制約 γ , 初期分割 V'_0

Output: 分割 V'

```

1:  $V' \leftarrow V'_0$ 
2:  $I \leftarrow \{Y \subseteq E \mid (u, v) \in V^2, C_{V'}(u) \neq C_{V'}(v), E_c(u, v) = Y\} \triangleright$ 
   共有辺の列挙
3:  $(I_1, \dots, I_{|I|}) \leftarrow \text{sort } I \text{ by size in descending order}$ 
4: for  $i = 1, \dots, |I|$  do
5:    $U \leftarrow \{v \in V \mid \exists u \in V, E_c(u, v) = I_i\} \triangleright$  頂点对の平坦化
6:    $C \leftarrow \{C(v) \mid v \in U\} \triangleright C \subseteq V'$ 
7:    $C' \leftarrow \text{bin\_packing}(C, \gamma) \triangleright C'$  は  $C$  の分割
8:   for all  $X \in C'$  do
9:      $V' \leftarrow (V' \setminus X) \cup \{v \mid Z \in X, v \in Z\}$ 
10:  end for
11: end for
12: return  $V'$ 

```

5.2.2 共有辺の列挙の高速化

γ -制限分割における計算量のボトルネックは、共有辺の列挙 (アルゴリズム 1 内 2 行目) である。単純に全ての頂点の組 $(u, v) \in V^2$ に対して共有辺を求める場合 (M1) の最悪計算量は、頂点次数 $|\mathcal{E}(v)|$ の平均 d_{avg} を用いて $\mathcal{O}(|V|^2 d_{\text{avg}})$ である。本報告で実験の対象とするプログラムは $|V| \geq 10^8$ であり、短時間に解を得ることが難しい。そこで、(M2) 接続辺に基づく探索範囲の限定および (M3) 頂点次数に基づく 2 段階列挙の 2 つの高速化を行う。

まず、(M2) について説明する。 $E_c(x, y)$ の要素が存在する場合、 $y \in e$ となる辺 $e \in \mathcal{E}(x)$ が存在する。したがって、ある $v \in V$ に注目すると、全ての $u \in V (u \neq v)$ との組に対して共有辺を列挙する必要はない。接続辺 $e \in \mathcal{E}(v)$ のそれぞれについて、全ての $w \in e$ との組 (v, w) に対して列挙すれば十分である。(M2) の計算量は、頂点の次数の最大値 d_{max} と $|e|$ の二乗平均 m を用いて $\mathcal{O}(\sum_{e \in E} \sum_{v \in e} |e| |\mathcal{E}(v)|) = \mathcal{O}(|E| m d_{\text{max}})$ である。 $|E| m d_{\text{max}} < d_{\text{avg}} |V|^2$ であるようなハイパーグラフに対してこのアルゴリズムは高速に動作する。

次に、(M3) について説明する。任意の $(u, v) \in V^2$ について、 $E_c(u, v) \in 2^{\mathcal{E}(v)} \cap 2^{\mathcal{E}(u)}$ が成り立つ。つまり、頂点次数 $|\mathcal{E}(v)|$ が十分小さい場合には、あらかじめ頂点 v ごとに独立して接続辺のべき集合 $2^{\mathcal{E}(v)}$ を求め、これらのべき集合を組み合わせることで高速に共有辺を得られる。したがって、任意の正整数 h を基準として、以下のように 2 段階に分けて共有辺を列挙する。まず、 $|\mathcal{E}(v)| \leq h \wedge |\mathcal{E}(u)| \leq h$ をみたす組 (u, v) に対して、接続辺のべき集合を利用して共有辺を列挙する。次に、 $|\mathcal{E}(v)| > h$ となる頂点 v に対して (M2) を利用して共有辺を列挙する。本報告の実験で用いる GPU プログラムでは、ハイパーグラフにおいて 99% 以上の頂点 $v \in V$ に対して $|\mathcal{E}(v)| \leq 10$ であるため、 $h = 10$ として (M3) を用いることで高速化を期待できる。

Algorithm 2 ワープ内の局所性を重視した初期分割

Input: ハイパーグラフ (V, E) , 制約 γ , ワープごとの参照 W^+
Output: 分割 V'_2
1: $V'_2 \leftarrow \{\{v\} \mid v \in V\}$
2: $H \leftarrow \{U \subseteq V \mid u \in U \wedge v \in U \rightarrow \mathcal{E}(u) = \mathcal{E}(v)\} \triangleright H$ は V の分割
3: **for all** $U \in H$ **do**
4: $W^- \leftarrow \{Y \cap U \mid Y \in W^+, Y \cap U \neq \emptyset\} \triangleright U$ 外の頂点を除外
5: $(W_1^-, \dots, W_{|W^-|}^-) \leftarrow \text{sort } W^- \text{ by size in ascending order}$
6: **for** $i = 1, \dots, |W^-|$ **do**
7: $D \leftarrow \{C_{V'}(v) \mid v \in W_i^-\}$
8: $D' \leftarrow \text{bin_packing}(D, \gamma)$
9: **for all** $X \in D'$ **do**
10: $V'_2 \leftarrow (V'_2 \setminus X) \cup \{v \mid Z \in X, v \in Z\}$
11: **end for**
12: **end for**
13: **end for**
14: **return** V'_2

5.3 ワープ内の局所性を重視した初期分割

γ -制限分割の解 V' は, 初期分割 V'_0 に依存する. 本報告では, 単純な初期分割 V'_1 に加えて, GPU のワープを意識した初期分割 V'_2 を提案する.

まず, V'_1 は頂点を 1 つずつ含むクラスタの集合とし, 次式で定義する.

$$V'_1 = \{\{v\} \mid v \in V\} \quad (15)$$

次に, V'_2 はワープごとの参照局所性に基づく初期分割である. P におけるワープの集合 W は, TB 集合 B の細分である. ただし, 任意のワープ $w \in W$ について原則として $|w| = k$ である (一般には $k = 32$). $W^+ = \{\{v \mid (S, L) \in w, v \in S \cup L\} \mid w \in W\}$ と定義する. TB よりも小さいワープ単位の参照 W^+ を用いてあらかじめクラスタを併合することで, ワープの参照局所性を向上しつつスパンの和 $Q_{V'}(E)$ を削減した分割を期待する.

V'_2 を求める手順をアルゴリズム 2 に示す. 初期クラスタを V'_1 とし, 段階的にクラスタを併合する. まず, $v \in V$ を接続辺で分類し, 分割 H を求める (2 行目). 以降は区分 $U \in H$ ごとに独立して, U の内部にあるクラスタを併合する (3~13 行目). U 内では, ワープの参照 $W_i^- \in W^-$ ごとに, W_i^- と関連するクラスタを併合する (7~8 行目). ただし, W^- の要素は互いに素でないため, ワープを選択する順に依存して結果が異なる. 提案手法では, 参照するデータが少ないワープから順に選択する (5~6 行目).

6. 評価

以下の観点から, 提案手法を評価する.

- ハイパーグラフ分割におけるスパンの削減
- スパンと FLC ミスの相関
- GPU プログラムの実行時間

表 1 実行環境

CPU	Intel Xeon E5-4640v2
コア数	10
動作周波数	2.2GHz
ソケット数	4
主記憶	1TB DDR3 SDRAM
GPU	Tesla V100
L1 キャッシュ	最大 128 KB
L2 キャッシュ	6144 KB
ビデオメモリ	16 GB
TB サイズの上限	1024
コンパイラ	gcc 5.4.0 nvcc 9.2.148 rustc 1.32.0
プロファイラ	nvprof

● ハイパーグラフ分割の実行性能

実験の対象には, Flint [6] を利用して生体モデルから生成した 4 つの CUDA プログラムを用いた. プログラムの特徴を表 2 に示す. B0 は心房を表す生体モデルから生成し, その他は視覚野を表すモデルから生成した. R0, R1 および R2 は計算の規模が異なり, ハイパーグラフの頂点数および辺数が異なる. 各プログラムは複数のカーネルから構成されており, カーネル間でアクセスする GM 上の領域に重複がある. そこで本報告では, プログラム内の全カーネルをまとめて 1 つのアクセスパターンを抽出し, 提案手法を適用した.

TB の大きさ β に依存してアクセスパターンが変化し, ハイパーグラフの辺数および辺次数が増減する. 辺次数は β に概ね比例し, 辺数は概ね反比例する. スパンの和および提案手法の計算量は辺に依存するため, β を 32 から 1024 の範囲で変えながら実験を行った. また, チャンクの大きさ γ も変えながら実験を行った. 本報告では, 計測した範囲で GPU プログラムの実行時間が最短となった $\gamma = 16$ の結果を示す.

比較のため, 既存手法 [7] および 2 通りの初期分割を用いた提案手法の計 3 通りで性能を計測した. 既存手法を用いた場合を EX と表し, 提案手法に初期分割 V'_1 および V'_2 を用いた場合をそれぞれ PN および PW と表す.

実験に用いた環境を表 1 に示す. 提案手法の実装には Rust を使用し, データ並列ライブラリ Rayon [13] を利用してループ並列化を行った. ループ並列化の並列度は, CPU スレッド数と同じ 80 とした.

6.1 スパンの削減

R2 に対して提案手法を適用した場合のスパンの総和 $Q_{V'}(E)$ を図 2 に記す. 図 2 が示す EX の値は, EX のデータ配置に g^{-1} (式 10) を適用して求めたクラスタを基準に算出した.

提案手法と既存手法を比較すると, β が大きいほどスパ

表 2 実験に用いた GPU プログラムの規模およびハイパーグラフ指標 ($\beta = 1024$)

プログラム	総タスク数	頂点数 $ V $	辺数 $ E $	頂点次数		辺次数 二乗平均	共有辺の総数
				平均	最大		
B0	8,904,569	11,465,254	17,400	3.34	54	8,419,697	6,172,266
R0	4,301,813	6,065,010	8,420	3.10	50	8,098,569	12,719,420
R1	8,479,680	11,840,818	16,584	3.12	71	8,114,624	36,295,436
R2	17,238,752	24,235,362	33,690	3.13	137	8,385,613	126,593,567

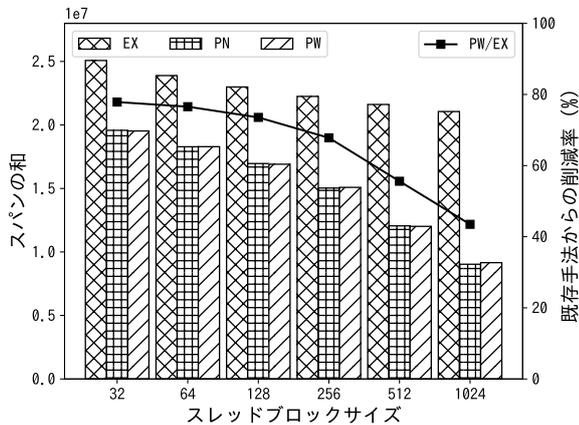


図 2 スパンの総和 $Q_{V'}(E)$ の比較 (対象: R2)

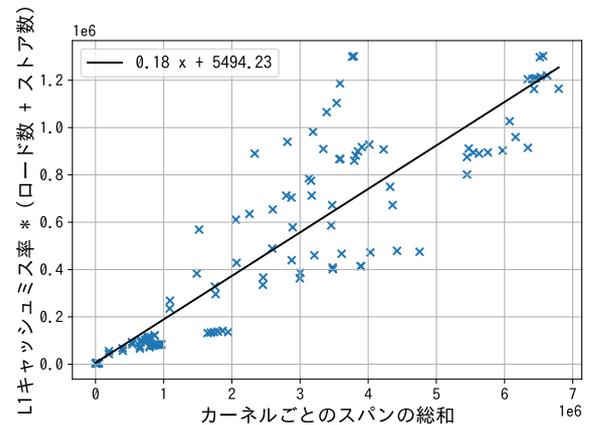


図 3 スパンと FLC ミスの相関 (対象: R2)

ンの削減率が大きい。 β が最大の 1024 の場合、スパンを 43% に削減できた。 PN と PW を比較すると β に関わらずスパンの削減率はほぼ等しく、差は高々 1% であった。 R2 以外のプログラムでも同様の結果が得られ、既存手法と比較したスパンの削減率は $\beta = 1024$ の場合で 34% - 43% だった。

6.2 スパンと L1 キャッシュミスの相関

要件 (R) の妥当性を評価するため、スパンの総和 $Q_{V'}(E)$ と FLC ミス数の相関を確認する。 FLC ミス数は、プロファイラ nvprof の測定値 `global_hit_rate`, `global_store_requests`, および `global_load_requests` をもとに算出した。 なお、 `global_atomic_requests` および `global_reduction_requests` は 0 であったため、無視した。

R0 を対象に、 $\beta = 32, 64, 128, 256, 512, 1024$ と 3 通りの手法の組み合わせについて、カーネルごとの $Q_{V'}(E)$ および FLC ミス数の関係を図 3 の散布図に示す。 相関係数は 0.902 であり、かなり強い正の相関を示した。 したがって、 FLC ミスの削減に関して $Q_{V'}(E)$ すなわち $F_l(P, g)$ は有用な評価関数である。 なお、 $\beta = 1024$ の場合、 EX と比較して PN は実際のキャッシュミス数を 54% に削減し、 PW は 70% に削減した。

6.3 GPU プログラムの実行時間

GPU プログラムの実行時間を、既存手法および提案手法で比較した結果を図 4 に示す。 実行時間は、 CPU の処理

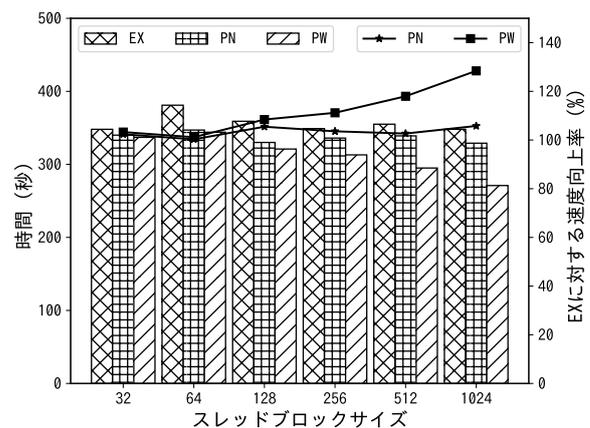


図 4 GPU プログラムの実行時間の比較 (対象: R2)

時間を排除し、カーネルの実行時間のみを計測した値である。 PW を用いて $\beta = 1024$ の場合、実行時間が最短となり、 EX の最短の場合と比較して約 1.28 倍の速度向上を得た。 全ての場合で提案手法は既存手法より高速であった。

さらに、 FLC ミス率と実行時間の相関係数を表 4 に示す。 PN と EX においては負の相関と正の相関が混在した。 PW においては全てのカーネルにおいて高い相関があった。 よって、速度向上のためには FLC ミス率のみを最小化しても不十分である。

6.4 ハイパーグラフ分割の実行性能

提案手法 PN および PW について、ハイパーグラフ分割に要する時間および主記憶使用量を比較する。 測定結果

表 3 ハイパーグラフ分割に要する処理時間および主記憶使用量

β	処理時間 (分)								メモリ使用量 (GB)							
	B0		R0		R1		R2		B0		R0		R1		R2	
	PN	PW	PN	PW	PN	PW	PN	PW	PN	PW	PN	PW	PN	PW	PN	PW
32	3.59	3.36	1.29	1.38	3.89	4.02	27.14	24.12	21.98	22.99	13.72	13.97	24.51	24.69	55.90	56.16
64	3.14	3.28	1.66	1.69	6.36	6.56	32.08	37.56	21.37	21.74	14.14	15.46	27.76	28.51	63.18	63.08
128	2.91	2.91	1.78	1.82	8.66	7.69	32.53	29.96	21.98	23.38	15.11	14.77	29.22	29.66	75.91	76.24
256	2.51	2.58	1.87	1.87	7.89	8.14	33.80	31.78	24.65	25.24	16.91	17.14	35.65	35.78	87.27	88.62
512	2.24	2.19	2.42	2.57	7.05	6.87	26.60	27.41	30.73	30.92	21.97	23.20	46.18	44.24	117.94	118.80
1024	2.19	2.17	3.52	3.16	8.13	8.41	27.11	27.66	46.55	47.27	33.84	33.19	62.92	61.09	148.94	149.59

表 4 FLC ミス率と実行時間の相関係数 (対象: R2)

カーネル	EX	PW	PN	全体
0	0.056	0.647	-0.736	-0.805
1	-0.624	0.994	-0.495	0.677
2	0.854	0.911	0.770	0.296
3	0.867	0.787	-0.633	0.326
4	0.900	0.980	0.906	0.166
5	0.033	0.995	0.979	0.827
6	0.843	0.985	0.291	0.175

表 5 ハイパーグラフ分割に要する処理時間の内訳 (対象: R2, 単位: 分)

	PN		PW	
	$\beta = 32$	$\beta = 1024$	$\beta = 32$	$\beta = 1024$
初期分割の決定	0.00	0.00	0.27	0.27
共有辺の列挙	22.95	20.75	19.40	20.16
クラスタの併合	3.06	5.41	3.17	6.13
その他	1.12	0.94	1.27	1.10
合計	27.14	27.11	24.12	27.66

を表 3 に示す。PN および PW の実行時間の差は-11.2%–17.0%であった。さらに、最も大規模な R2 を対象にした場合の処理時間の内訳を表 5 に示す。PN および PW の主な違いは初期分割の決定方法にあるが、PW における初期値決定の時間は全体の 2%以下と小さく、全体の実行時間に影響しなかった。

主記憶使用量の差は-9.3%–4.2%であった。また、PN および PW の両方において、 β が増大すると主記憶使用量が増大する傾向があった。この理由は、共有辺の集合 I (アルゴリズム 1 の 2 行目) を格納する領域の増大である。提案手法の実装では I を列挙すると同時に、各 $Y \in I$ について、全ての $e \in Y$ に含まれる頂点の集合 U (アルゴリズム 1 の 5 行目) を求め、 $Y \mapsto U$ を連想配列に格納する。この連想配列の空間計算量は $\mathcal{O}(\sum_{v \in V} |\{e(v) \cap \mathcal{E}(u) \mid e \in \mathcal{E}(v), u \in e\}|)$ である。 β の増大は TB の併合、すなわち辺の併合を意味するため、 β の増大に従って辺次数が増大し、任意の v に対して $\{u \mid e \in \mathcal{E}(v), u \in e\}$ の要素数が増大する。

7. 終わりに

本報告では静的なアクセスパターンを持つ GPU プログラムを対象に、L1 キャッシュヒットミス数の削減をハイ

パーグラフの分割問題としてモデル化し、その問題を解くヒューリスティックなアルゴリズムを 2 種類提案した。一方のアルゴリズムは L1 キャッシュミス数のみを削減する。他方はワーブごとの局所性と L1 キャッシュミス数を同時に改善する。

実験の結果、モデル上のキャッシュミス数と実際のキャッシュミス数の相関係数は 0.902 となった。また、既存手法と比較して、モデル上でのキャッシュミス数を最大 66%削減した。結果、ワーブごとの局所性と L1 キャッシュミス数を同時に改善するアルゴリズムで最大 1.28 倍の速度向上を達成した。

今後の課題は、ワーブごとの局所性を考慮したモデルにおける最適化と、キャッシュ置換を考慮したモデル化である。

謝辞 本研究の一部は、科学研究費補助金 (基盤研究 (A) JP15H01687 の支援による。

参考文献

- [1] Ryoo, S., Rodrigues, C. I., Baghsorkhi, S. S., Stone, S. S., Kirk, D. B. and Hwu, W. W.: Optimization Principles and Application Performance Evaluation of a Multi-threaded GPU Using CUDA, *In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming (PPoPP '08)*, pp. 73–82 (2008).
- [2] Wu, B., Zhao, Z., Zhang, E. Z., Jiang, Y. and Shen, X.: Complexity analysis and algorithm design for reorganizing data to minimize non-coalesced memory accesses on GPU, *ACM SIGPLAN Notices*, Vol. 48, No. 8, pp. 57–68 (2013).
- [3] Zhang, E. Z., Jiang, Y., Guo, Z., Tian, K. and Shen, X.: On-the-fly elimination of dynamic irregularities for GPU computing, *In Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS XVI)*, pp. 369–380 (2011).
- [4] Baskran, M. M., Bondhugula, U., Krishnamoorthy, S., Ramanujam, J., Rountev, A. and Sadayappan, P.: A Compiler Framework for Optimization of Affine Loop Nests for GPGPUs, *In Proceedings of the 22nd annual international conference on Supercomputing (ISC '08)*, pp. 225–234 (2008).
- [5] Yang, Y., Xian, P., Kong, J. and Zhou, H.: A GPGPU Compiler for Memory Optimization and Parallelism Management, *In Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '10)*, pp. 86–97 (2010).

- [6] Okuyama, T., Okita, M., Abe, T., Asai, Y., Kitano, H., Nomura, T. and Hagihara, K.: Accelerating ODE-based Simulation of General and Heterogeneous Biophysical Models using a GPU, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 8, pp. 1966–1975 (2014).
- [7] 角田優貴, 置田真生, 萩原兼一: 数式の依存グラフを仕様とする OpenMP プログラムのメモリ参照効率向上, 情報処理学会研究報告, 2016-HPC-155, 9 pages (2016).
- [8] Karypis, G.: hMETIS - Hypergraph & Circuit Partitioning, <http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview> (accessed on 2019-02-02).
- [9] Alpert, C. J. and Kahng, A. B.: Recent directions in netlist partitioning, *Integration, the VLSI*, Vol. 19, No. 1-2, pp. 1–81 (1995).
- [10] Diestel, R.: グラフ理論, 丸善出版 (2012).
- [11] Karypis, G. and Kumar, V.: Multilevel k-way hypergraph partitioning, *VLSI design*, Vol. 11, No. 3, pp. 285–300 (2000).
- [12] Korte, B. and Vygen, J.: 組合せ最適化: 理論とアルゴリズム, 丸善出版 (2009).
- [13] Blandy, J. and Orendorff, J.: プログラミング Rust, オライリー・ジャパン (2018).