Vol.2017-OS-141 No.23
2017/7/27

GPUミドルウェア “Victream” におけるI/Oとプロセッシングの連携スケジューラの性能評価

鈴木 順[1,2,a)]　林 佑樹[1]　荒木 拓也[1]　竹中 崇[1]　喜連川 優[2]

概要：CPU(Central Processing Unit) から GPU(Graphics Processing Unit) への計算オフロードによる高速化では，GPU へのデータ移動のための I/O(Input/Output) オーバヘッドを最小化することが高い高速化利得を得るために重要である．我々は [1] において複数の GPU を用いて GPU メモリの合計より大きなデータを処理する Out-of-Core 処理において GPU の I/O オーバヘッドを最小化する “Victream” ミドルウェアを提案した．本稿では GPU への I/O オーバヘッドを最小化する Victream スケジューラの詳細を述べ，State-of-the-Art の従来技術と性能の比較評価を行う．評価では従来手法に対し最大で 117%の性能向上が得られた．

Performance Evaluation of Cooperative Scheduling of Processing and I/O of Victream GPU Middleware

Jun Suzuki[1,2,a)]　Yuki Hayashi[1]　Takuya Araki[1]　Takashi Takenaka[1]　Masaru Kitsuregawa[2]

1. はじめに

GPU は CPU の計算をオフロードし，高速な計算処理を実現するデバイスとして注目されている．NVIDIA 社による CUDA の提供により，GPU は画像処理に留まらず，並列計算や機械学習処理へと適用範囲を拡大させている．近年ではより高速な計算処理を実現するために，複数の GPU を使用し，用いる GPU 数に対し計算性能をスケールさせることも重要になっている．

GPU への計算オフロードでは，CPU が処理データを保持するホストのメインメモリと GPU メモリの間でデータを移動させる I/O オーバヘッドに注意する必要がある．表 1 と表 2 に CPU と GPU の性能比較を示す．GPU は CPU に対し 10 倍程度の演算性能とメモリ帯域を保持する．しかし，メインメモリと GPU メモリの間でデータを移動する際に用いる I/O バスは PCIe 3.0 x 16 程度である．そしてその I/O 帯域は 16 GB/s であり，GPU メモリの帯域の 10 分の 1 以下である．これらの性能比は，CPU から GPU に計算をオフロードして高速な計算を実現するには，I/O バスを介して GPU に移動させたデータに対し GPU で I/O オーバヘッドを上回る十分な演算を行う必要があることを意味する．データに対する演算量が小さい場合，I/O オーバヘッドにより CPU だけで計算を行う場合と比較して大きな高速化効果が得られない場合がある．

また表 2 を参照すると，GPU が保持するメモリは 10GB 程度である．これは近年データセンターで一般に用いられているホストのメインメモリより 2 桁小さい．従って GPU を用いて大規模なデータ処理を行う場合，データを分割し，GPU メモリに入れ替えながら計算を行う Out-of-Core 処理となる．ここで Out-of-Core で高い計算性能を得るには，先に述べた GPU の大きい I/O オーバヘッドを抑制するため，GPU メモリにロードしたデータは可能な限りまとめて処理を行い，ホストメモリと GPU メモリのデータの入れ替えの I/O を削減することが重要である．

我々は [1] において複数の GPU を用いた Out-of-Core 処理で GPU の I/O オーバヘッドを最小化する”Victream”

[1] NEC　システムプラットフォーム研究所
System Platform Research Laboratories, NEC
[2] 東京大学生産技術研究所
Institute of Industrial Science, the University of Tokyo
[a)] j-suzuki@ax.jp.nec.com

表 1　Intel Xeon E7-8894 v2 の特性 [5].

Single-precision Floating Point Performance	1.8 Tflops
Memory Bandwidth	85 GB/s

表 2　NVIDIA Tesla P100 GPU の特性 [6].

Single-precision Floating Point Performance	9.3 Tflops
Memory Size	16 GB
Memory Bandwidth	732 GB/s
I/O Bus	PCIe 3.0 x 16

ミドルウェアを提案した．Victream はユーザプログラマにデータパラレルアプリケーションを作成するための API(Application Programming Interface) を提供する．ユーザプログラムから API が呼ばれることで，ミドルウェア内にアプリケーションの計算を示す DAG(Directed Acyclic Graph) が形成され，形成された DAG がミドルウェアにより遅延評価で実行される．Victream は複数の GPU を用いて DAG の計算を実行する際に，GPU の I/O オーバヘッドを最小化するように GPU のデータ処理と I/O のスケジューリングを行う．

従来からデータパラレル処理において DAG を作成し，DAG のノードであるタスクを複数の演算デバイスにスケジュールする手法が提案されてきた．ホストに関しては Dryad[2] や Spark[3] が知られている．また複数の GPU を用いた計算には PTask[4] がある．これらの従来手法に対し Victream では，特に複数の GPU を用いて大規模なデータを処理する Out-of-Core 処理に着目する．Victream は演算と I/O をソフトウェアから個別に制御する必要がある GPU において，I/O と演算を連携してスケジューリングする新しい手法により，GPU の I/O オーバヘッドを最小化する．具体的には，GPU の演算のスケジュールを I/O のスケジューリング時に決めるオンラインスケジューリングを特徴とする．

本稿では [1] で提案した Victream について，GPU の I/O オーバヘッドを最小化するスケジューラの原理の詳細を述べる．またその性能を State-of-the-Art の従来手法と比較する評価を行う．評価の結果，Victream は従来手法に対し最大で 117%の性能評価が得られることがわかった．

以下本稿では 2 節で Victream アーキテクチャ，3 節で Victream スケジューラ，4 節で Victream スケジューラの評価結果について述べ，最後に 5 節でまとめる．

2. Victream アーキテクチャ

Victream は Spark の API を拡張し，GPU による画像や行列等の異なるデータ形式のデータパラレル処理に対応する．Victream は CPU でのデータ処理は考慮せず，複数 GPU を用いたデータ処理に特化し C++で実装している．

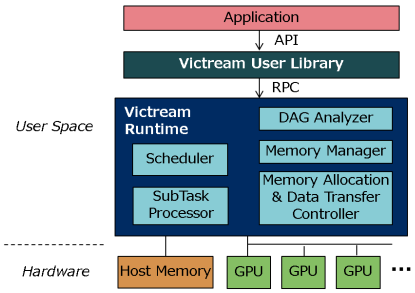


図 1　Victream のアーキテクチャ．

Victeam のアプリケーションは実行する処理を示す DAG を作成し Victream に実行させる．図 1 に Victream のアーキテクチャを示す．Victream はユーザライブラリとランタイムから構成される．アプリケーションプログラムは Victream ライブラリが提供する API を用いてコーディングされる．アプリケーションプログラム内でライブラリの API を呼ぶことで，ライブラリ内部に DAG が作成され，RPC(Remote Procedure Call) により DAG の実行がランタイムに要求される．DAG の処理はホストの I/O スロットに挿入された複数の GPU を用いて行われる．

DAG のノードにおいて UDF(User-Defined Function) に処理されるデータオブジェクトを GRDD(GPU Resilient Distributed Dataset) と呼ぶ．GRDD はランタイムにより分割され，各 GPU でパーティション毎に UDF を用いて処理が行われる．以下では分割した GRDD をデータパーティション，UDF で定義された GRDD に実行する処理をタスク，データパーティション毎に分割したタスクをサブタスクと呼ぶ．ランタイムが計算を実行する DAG はサブタスクをノードとし，サブタスク間で入出力されるデータパーティションがエッジである．GRDD のデータパーティションは GPU メモリに配置されるかホストのメインメモリにスワップアウトされる．Victream はサブタスクの演算と，データパーティションの GPU 間，あるいは GPU とホストのメインメモリ間の I/O を単位として GPU の演算と I/O のスケジューリングを行う．このスケジューリングは GPU の Out-of-Core 処理において最も I/O オーバヘッドを最小化するように実現される．このスケジューリングはユーザプログラムには透過である．

また Victream は GRDD にデータ形式の属性を付与することで，データ形式に応じた GRDD の管理を行う．現在データ形式として画像，密/疎行列，Key-Value ペアの 4 つをサポートしている．GPU の I/O と演算のスケジューリングはデータ形式によらない共通のレイヤで提供される．これにより様々なデータ形式の DAG 処理の実行を最適化できる．

Victream は使用する GPU のメモリの総和を超える GRDD を扱うことができる．今後の検討の 1 つに GRDD のパーティションを NVM(Nonvolatile Memory) Express

Card 等のストレージデバイスに配置する手法がある．

Victream のライブラリとランタイムの詳細は [1] で述べたため説明を省略する．

3. Victream スケジューラ

3.1 スケジューリング手法

Victream では，計算する DAG のノードであるサブタスクにおいて画像や行列など様々なデータ形式を扱う．Victream スケジューラはこれらのサブタスクの演算と入出力となるデータパーティションの I/O のスケジューリングをデータ形式に共通の方法で行う．

スケジューリングの目的は GPU を用いて最短時間の DAG 計算を実現することである．ここで，Victream では DAG がアプリケーションから与えられるため，DAG が含む各サブタスクの実行時間がわからずオフラインで最適スケジュールを決めることができない．このため Victream スケジューラは DAG 処理を実行しながらオンラインでスケジューリングを行う．

Victream では GPU の Out-of-Core 処理のボトルネックを GPU がホストのメインメモリと GPU メモリでデータを入れ替えるための I/O と想定している．そのため GPU が DAG 処理で実行する全ての I/O を完了する時間を最短化することが最短時間の DAG 処理を実現することと等価であると仮定する．従って Victream スケジューラの目的関数は，GPU で実行する総 I/O の完了時間の最短化である．より正確には，GPU は複数存在するため，それらの中で最も完了時間が遅い GPU の最短化である．

ここで Sundaram らは [7] において我々と同様に GPU の Out-of-Core 処理のボトルネックを I/O と想定し，GPU に対する I/O 量を最小化する問題を解いた．彼らは単一 GPU でかつ GPU が演算と I/O のオーバラップをサポートしない場合を対象としたため，GPU における全ての演算と I/O がシリアライズされ GPU の I/O 完了時間の最短化が I/O 量の最小化と等価となった．また問題を NP-hard である Pseudo-Boolean (PB) Optimization に定式化することができた．一方我々は，複数のサブタスクを異なる GPU で並列に処理するため，問題設定をシリアライズの性質を用いて PB Optimization に定式化できない．

Victream スケジューラでは GPU の I/O 完了時間の最短化をオンラインのヒューリスティックなスケジューリング手法で実現する．前提として DAG 内のサブタスクは実行するまで実行時間がわかない一方，サブタスクの入出力となるデータパーティションのサイズ (使用する GPU メモリ容量) はライブラリ API から与えられる情報や API のオペレータの性質からデータパーティションの I/O を行う時点でわかっているとする．このとき以下の 2 つを実現する．(1)GPU のリソースをアイドルにせずできるだけサブタスクの入出力となるデータパーティションの I/O を進めることで，ボトルネックとなる I/O のリソースを有効に活用する．(2)GPU が実行する総 I/O 量を最小化する．これら 2 つを DAG が規定するサブタスクの実行順の制約の中で実現する．

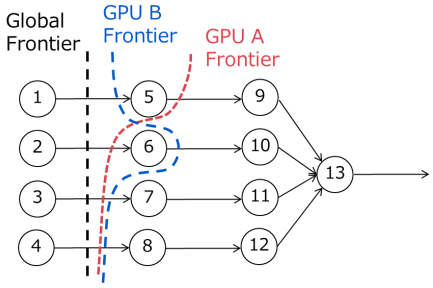


図 2　GPU 毎のフロンティア．

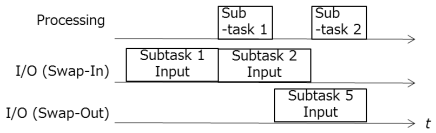


図 3　従来手法の DAG を用いたスケジューリング．

そこで Victream では GPU の Out-of-Core 処理の課題である (1) と (2) を解決するために I/O と演算の連携スケジューラを提案する．連携スケジューラの特徴は，オンラインスケジューリングで探索するスケジューリング候補のサブタスクを従来より拡張することである．従来の DAG 計算では，DAG において依存する全ての上流のサブタスクの実行が完了したサブタスクをスケジューリング対象としていた [4]．以下ではこの手法では Out-of-Core 処理において I/O オーバヘッドを最小化するスケジュールが得られないことを説明する．

図 2 と図 3 を用いて従来手法では I/O オーバヘッドを最小化できない例を説明する．説明のためそれぞれのサブタスクは番号付けし，図の DAG を 1 つの GPU で実行することとする．簡単のため，サブタスクの実行時間はサブタスクの入出力データパーティションを GPU 外にスワップする I/O 時間より短いとし，GPU メモリは 4 つのデータパーティションしか保持できないとする．また GPU メモリは使用量 3 以上でスワップアウトを開始する．従来手法では DAG で依存する上流のサブタスクが完了し実行可能となったサブタスクがスケジュールされる．よって最初にサブタスク 1-4 のいずれかが選択される．今，サブタスク 1 を選択し入力データパーティションを GPU にロードしたとする．入力をロード後，サブタスク 1 の演算を実行する．このとき GPU の I/O リソースがアイドルとなるため実行可能のサブタスク 2-4 のうち 2 のサブタスクの入力データのプリフェッチを開始する．この時点で GPU メモリはサブタスク 1 と 4 の入出力により 4 つ使用されているため，スワップアウトを行いたい．しかし全ての領域が使用中のためスワップアウトできない．スワップアウト可能

となるのはサブタスク 1 の完了後である．そのときサブタスク 1 の入力がその後の計算で不要であれば GPU メモリから消去される．また出力は GPU メモリ上に残る．結局 GPU メモリの使用量が 3 であるため，サブタスク 5 の入力であるサブタスク 1 の出力がスワップアウトされる．しかし，サブタスク 5 の入力は再度 GPU にロードする必要があるため GPU への I/O が増大する．もしサブタスク 1 と 5 を連続して実行していればこのような問題は生じない．

そこで Victream では，上記の問題が生じないようにスケジューリングにおけるサブタスクの探索範囲を拡張する．従来手法の問題はスケジュール候補を上流のサブタスクが完了したサブタスクに限定することで，GPU メモリが保持するデータパーティションを使用するサブタスクをスケジュールできず，データパーティションのスラッシングを起こすことである．そこで Victream では，それらのサブタスクを先にスケジューリングできるようにする．その上で (1) の課題は GPU における I/O と演算を非同期化し，演算の制約を受けずに将来のサブタスクの入力データのプリフェッチを進めることで解決する．また (2) は GPU に対する総 I/O 量を最小化する順でサブタスクの演算を実行することで解決する．

Victream のスケジューリングでは図 2 に示すように GPU 毎の異なるフロンティアを用いる．ここでは説明のため，用いる GPU の数は 2 つであるとする．グローバルフロンティアは演算が完了したサブタスクを示す．従ってサブタスクに入力する全てのエッジがグローバルフロンティアと交差するサブタスクは依存するサブタスクの演算が完了しており，どの GPU へも演算のスケジューリングが可能である．一方 Victream は図 2 に示すように GPU 毎にフロンティアを管理する．各 GPU は全ての入力エッジがグローバルフロンティアと交差するサブタスクの演算のスケジューリングが可能である．

グローバルフロンティアと GPU フロンティアの間のサブタスクは，GPU フロンティアが該当する GPU でサブタスクの演算が実行待ちかあるいは実行中で完了していないサブタスクを示す．例えば GPU A ではサブタスク 5 である．サブタスク 5 が GPU A のフロンティアに含まれるのは，サブタスク 5 の入力データパーティションが GPU にロードされ，演算が実行待ちになった時点である．この時点は，GPU の I/O リソースがアイドルになり，GPU A に次の I/O をスケジュールする時点でもある．

この時点で，GPU A では GPU フロンティアと交差する入力エッジを保持するサブタスク 9 について先に GPU A で演算の実行が決まっていたサブタスク (この場合はサブタスク 5) が先に実行されるという前提であれば，サブタスク 5 の次にその演算を実行するとして GPU の演算のスケジュールを決めても良い．この場合，サブタスク 9 の演算ではサブタスク 5 が GPU メモリ上に出力するデータパーティションをそのまま入力とすれば良い．また図には示していないが，サブタスク 9 が GPU メモリにロードされていない別の入力も必要とし，その入力の DAG の該当エッジがグローバルフロンティアと交差している場合，先と同様にサブタスク 9 の次にサブタスク 5 の演算を実行すると演算のスケジュールを決め，さらに GPU A がサブタスク 5 の演算を行っている間にロードされていない入力データパーティションを GPU A で演算と I/O をオーバラップさせてロードしても良い．つまり GPU A は I/O のスケジューリング時にサブタスク 9 の GPU I/O の要否によらず演算の実行のスケジュールを決め，かつサブタスク 9 が GPU I/O の入力データパーティションをロードする必要がある場合はその I/O をスケジュールできる．

上記を一般化して述べる．Victream スケジューラでは GPU の I/O のスケジュール時に演算のスケジュールを同時に決定する．演算のスケジュールの候補となるサブタスクは，全ての入力エッジがスケジューリングを行う GPU のフロンティアと交差するサブタスクである．選択したサブタスクが GPU へのデータロードを必要としない場合，選択したサブタスクの演算を GPU の演算のスケジュールを管理する FIFO(First-In First-Out) キューに追加し，選択したサブタスクを含むよう GPU フロンティアを更新し，次に演算のスケジュールに追加するサブタスクの選択に進む．FIFO キューに追加されたサブタスクの演算は GPU のプロセッサで順に実行される．一方選択したサブタスクがデータロードを必要とする場合，サブタスクを上記の演算の FIFO キューに追加することに加え，データロードのための I/O をスケジュールする．そしてその I/O の完了後，GPU フロンティアを更新し，次の I/O のスケジューリングに進む．I/O のスケジューリング時には，再びサブタスクの演算のスケジュールが決められる．また，サブタスクを実行しない GPU のフロンティアは，サブタスクの演算が完了した後，フロンティアに実行が完了したサブタスクを含むように更新される．ここで FIFO キューを用いてスケジュールが管理されたサブタスクの演算をシリアライズして順に実行することで，FIFO キューに追加された時点では DAG 内で依存する上流のサブタスクの演算が完了していなかったサブタスクも，その演算の実行時には依存するサブタスクが完了していることが保証される．

上記の手法により，GPU の I/O と演算は非同期に動作し，GPU は将来のサブタスクの入力データパーティションのプリフェッチを進めることができる．サブタスクのスケジュールは GPU I/O のスケジューリング時に決まるが，演算の実行は FIFO キューで管理され，GPU I/O と非同期で動作する．I/O は演算が実行されているサブタスクに関わらず, 次の I/O に進むことができる．また GPU フロンティアを用いることで DAG のサブタスク実行順の制約を守って I/O を進めることが可能である．ここで従来のミ

ドルウェアでも GPU へのプリフェッチは行われてきた．Victream がプリフェッチの観点で従来手法と異なる例は，上述した図 2 においてサブタスク 9 が別の入力データパーティションのロードを必要とする場合である．Victream ではサブタスク 9 は DAG で依存する上流のサブタスク 5 が未完了であるが，入力となるデータパーティションのプリフェッチを行える．一方従来手法ではサブタスク 5 の演算が完了するまでそのような I/O ができない．

次に Victream スケジューラが GPU に対する総 I/O 量を最小化する方法について述べる．これは GPU I/O のスケジューリング時に決定する演算のスケジュールによって実現する．Victream で演算のスケジュールに追加する候補となるサブタスクは，全ての入力データパーティションを示すエッジがスケジューリングする GPU のフロンティアと交差するサブタスクである．Victream はスケジューリング候補の中から貪欲法により最も GPU に対する I/O が少ないサブタスクを選択する．スケジューラはその時点で GPU メモリが保持するデータパーティションを確認し，スケジューリング候補の各サブタスクを選択した場合に必要となる GPU へのデータロードとホストのメインメモリへのスワップアウト量の和を比較することでサブタスクの選択を行う．このオンラインスケジューリングにより実現されるサブタスクの演算のスケジュールは，GPU の Out-of-Core 処理において GPU への I/O 量を最小化する (ヒューリスティックであるため最適解の保証はない)．

以上のように Victream スケジューラは複数 GPU を用いた Out-of-Core 処理で性能ボトルネックである I/O オーバヘッドを I/O リソースの効率的な利用と GPU に対する I/O 量の最小化によって最小化する．I/O リソースの効率的な利用は GPU において I/O と演算を非同期で実行し，データプリフェッチを進めることで実現する．また I/O 量の最小化は貪欲法によるサブタスク演算のスケジュール決定で実現する．Victream の特徴はこれら 2 点をオンラインスケジューリングで実現する場合，従来の DAG 計算が用いていた上流のサブタスクの実行が完了したサブタスクをスケジューリング候補としていた手法に対し，DAG が与えるサブタスクの実行順の制約を満たしながら上流のサブタスクが未完了のサブタスクまで探索するサブタスクの候補を拡張したことである．

3.2 スケジューラ実装

3.2.1 I/O と演算の連携スケジューリング

Victream は GPU の I/O のスケジューリング時に演算のスケジュールを同時に決定する．図 4 にスケジューラの構成を示す．前段の処理がサブタスクの I/O，後段が演算である．

演算のスケジューリング候補となるサブタスクはグローバルリストかローカルリストのいずれかが保持する．グローバルリストに記憶されるサブタスクは図 2 において全ての入力エッジがグローバルフロンティアと交差するサブタスクである．グローバルリストが保持するサブタスクは，DAG において依存する上流のサブタスクの演算が全て完了し，入力となるデータパーティションが全て出力された実行可能のサブタスクである．従ってグローバルリストのサブタスクはどの GPU でもスケジューリング可能である．一方 GPU ローカルリストは図 2 において該当する GPU フロンティアとグローバルフロンティアの間に含まれるサブタスクを保持する．GPU ローカルリストに保持されるサブタスクは DAG において全ての入力エッジがグローバルフロンティアか GPU ローカルフロンティアと交差し，少なくとも 1 つのエッジは GPU ローカルフロンティアと交差している．GPU ローカルリストが保持しているサブタスクの，DAG における上流の全ての未完了サブタスクは，GPU ローカルフロンティアが該当する GPU で演算待ちか演算の実行中である．前述のように GPU の演算のスケジュールに追加されているサブタスクは，シリアライズして実行されるため，GPU ローカルリストが保持するサブタスクは該当する GPU でスケジューリング可能である．

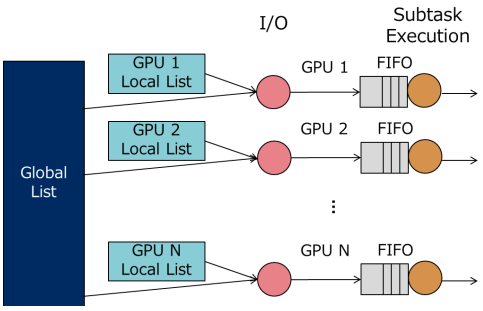


図 4 スケジューラの構成.

```
subtask get_next_subtask(gpu) {
  glob_min = iomin_subtask(global_list);
  local_min = iomin_subtask(local_list[gpu]);
  if(glob_min < local_min) {
    remove(global_list, glob_min);
    return glob_min;
  } else {
    remove(local_list[gpu], local_min);
    return local_min;
}}

void schedule() {
  foreach(g in available_gpu) {
    if(size(global_list) > 0
       || size(local_list[g]) > 0) {
      if(memory_use[g] < load_threashold) {
        st = get_next_subtask(g);
        pipeline_dispatch(st,g);
      }
}}}
```

図 5 スケジューリングアルゴリズムの疑似コード.

Victream スケジューラは各 GPU 毎にスケジューリングを行う．各 GPU の I/O のスケジューリングにおいてグローバルリストか GPU に該当するローカルリストのいずれかからサブタスクを選択し，図 4 の前段の I/O 処理に投入する．サブタスクが I/O 処理に入ると，プリフェッチにより入力データパーティションが GPU に準備され，出力メモリ領域が確保される．入出力メモリ領域はスケジュールされたサブタスクの演算が完了するまでロックされる．ここでスケジューラの実装では，全てのサブタスクを I/O 処理を通すことでスケジューラの構成を簡単化するため，入力データパーティションが既に GPU メモリに存在するサブタスクや，入力データパーティションを出力する DAG の上流のサブタスクが GPU で実行待ちあるいは実行中で GPU に入力データパーティションをロードする必要がないサブタスクも疑似的にデータ量がゼロバイトの I/O を伴うと疑似化し I/O 処理を通している．つまり GPU への I/O をスケジューリングしない場合もゼロバイトの I/O をスケジューリングしたと扱う．このとき I/O 処理では出力メモリの確保と入出力領域のロックだけが行われる．またこれはスケジューラ内の話であり実際に GPU にゼロバイトの I/O 命令が発行されるわけではない．

またスケジューラの後段は演算処理である．この処理に入るサブタスクは入力データパーティションの準備が完了し出力メモリ領域が確保されている．I/O 処理を出たサブタスクは，その順番通りにシリアライズされ演算が行われる．サブタスクの演算の完了後，入出力データパーティションのメモリ領域のロックが解除される．

Victream スケジューラはサブタスクが I/O 処理を抜け，演算処理に投入された時点で，グローバルリストと GPU ローカルリストのいずれかから I/O 処理に投入する次のサブタスクを選択する処理に進む．従って GPU の I/O は I/O を必要とするサブタスクが I/O 処理に投入されるまでスケジューリングされない．それまでは選択されたサブタスクの演算の実行が演算のスケジュールに追加されるだけである．

Victream スケジューラはサブタスクが I/O 処理を抜けた段階で，抜けたサブタスクに依存する DAG の下流のサブタスクが GPU が該当するローカルリストに登録される条件を満たせばリストを更新する．またサブタスクが演算処理を完了した段階で，完了したサブタスクに依存する DAG の下流のサブタスクがグローバルリストに登録される条件を満たせばリストを更新する．

また GPU のメモリ使用量が予め定めたデータロードのしきい値を超えた場合，Victream スケジューラはメモリ使用量がしきい値を下回るまで新たなサブタスクの I/O 処理への投入を停止する．一方 GPU のメモリ使用量がスワップアウトのしきい値を超えた場合，Victream スケジューラはバックグラウンドで LRU(Least Recently Used) アルゴリズムを用いて GPU メモリ上のロックされていないデータパーティションをメモリ使用量がしきい値以下になるまでスワップアウトする．通常，スワップアウトのしきい値はデータロードのしきい値より低い値に設定する．データロードとスワップアウトは I/O バスの逆方向の通信であり並列に行われる．

3.2.2 I/O と GPU の非同期パイプライン

Victream スケジューラは GPU の I/O 処理と演算処理を GPU 毎に非同期のパイプラインで管理する．図 4 の後段の演算処理の前には FIFO キューが配置され，I/O 処理と演算処理が非同期に行われる．I/O 処理では投入されたサブタスクが GPU I/O を必要としない場合，メモリを確保後直ちに演算処理に投入され次のサブタスクが I/O 処理に投入される．このような構成により，GPU で実行している演算の完了を待たずに将来演算を実行するサブタスクの入力データのプリフェッチを進めることが可能となる．

3.2.3 GPU に対する総 I/O 量の最小化

Victream スケジューラがグローバルリストと GPU が該当するローカルリストを探索して I/O 処理に投入するサブタスクを選択する場合，貪欲法により，選択時点で GPU メモリが保持する入力データパーティションの状態に対し，GPU への I/O が最も少ないサブタスクを選択する．GPU への I/O は GPU へのデータロードと必要となるスワップアウトの和として計算する．このため入力データパーティションが既に GPU メモリに存在しているサブタスクが優先して選択される．スケジューラのアルゴリズムを図 5 に示す．このオンラインスケジューリングにより選択されたサブタスクの演算のスケジュールが GPU への Out-of-Core 処理において GPU へのデータ I/O を最小化するスケジュールとなる．

3.3 現在の実装の限界

提案の I/O と演算の連携スケジューラは I/O を複数の GPU 間で分散させる．しかし，現在の実装は GPU 間で演算の負荷が偏る可能性を考慮していない．このような場合，対策としてある GPU の演算の FIFO キューで長い間待っているサブタスクの処理を他のアイドルな GPU に移す対処が考えられる．

4. 評価結果

4.1 評価構成

後述する 4 つのマイクロベンチマークの評価に用いた評価系と比較対象とした従来技術を述べる．

評価では 4 つの NVIDIA Tesla K20 GPU をホストの I/O スロットに挿入した．これはホストに挿入できる最大の GPU 数である．それぞれの GPU は 5GB のメモリを保持しており，単精度浮動小数点の演算スループットは 3.52 Tflops である．ホストは E5-2609 Xeon CPU を 2 つ

保持し，OS には Ubuntu 14.04 を用いた．DAG 計算の入出力データは RAMdisk のファイルとして保存した．現在の Victream のソースコードは C++と CUDA 7.5 を用いて実装しており，ユーザライブラリとランタイムを合わせて 10K ラインの規模である．

Victream の性能を従来手法と比較するため，FIFO スケジューラと PTask[4] で提案された Data-Aware スケジューラを実装した．FIFO スケジューラはサブタスクが依存する親タスクの演算が完了し実行可能になった順にスケジュールを行う．GPU の演算リソースを効率的に活用するために，実行可能のサブタスクを GPU に貪欲法で割り当てる．ただしそれに伴う GPU の I/O 量は考慮しない．PTask スケジューラも FIFO スケジューラど同様の動作を行うが，スケジュールする入力データパーティションを保持している GPU を考慮する．もしスケジュール対象のサブタスクの入力データパーティションを多数保持している GPU が，スケジュール対象となった GPU と異なれば，その GPU がスケジュール可能になる場合に備えて一定時間待機する．しかし，本稿が対象とする Out-of-Core 処理では，FIFO キューの先頭のサブタスクの入力データパーティションが全て GPU 外にスワップアウトされることが有り得る．この場合，キューの後ろで待っているサブタスクを先にスケジュールする方を考慮する必要があり，Data-Aware スケジューラでは GPU の I/O 量を最小化しない．なお，FIFO と PTask の両方のスケジューラでは I/O と演算のオーバラップを行っている．

4.2 評価結果

提案手法の評価にはロジスティック回帰，ソート，複数回の画像フィルタ (Blur フィルタ)，行列積の 4 つのマイクロベンチマークを用いた．それらを Victream の API を用いて実装した．これらの評価結果は Victream が Out-of-Core 計算において従来手法より優れた性能を提供することを示した．

今回，4 つのマイクロベンチマークを用いて 2 種類の評価を行った．最初の評価では計算に用いる GPU 数を変更しながら計算性能のスケーラビリティを評価した．評価では GPU あたりの演算量が一定となるようデータ量を調整し，処理が GPU 数を増加させても Out-of-Core となるようにした．つまり N 個の GPU を用いた測定では 1 つの GPU を用いた評価に対し演算量が N 倍となるようにした．一方 2 つ目の評価では処理データサイズを変更しながら計算時間の測定を行った．評価で用いた入力データパーティションのサイズは約 256MB である．マイクロベンチ間における入力データパーティションのサイズのわずかな差異は用いたデータのアラインメントによるものである．測定では GPU へのデータロードを中断するデータロードのメモリ使用量のしきい値を 70%に，スワップアウトを開始するしきい値を 50%に設定した．

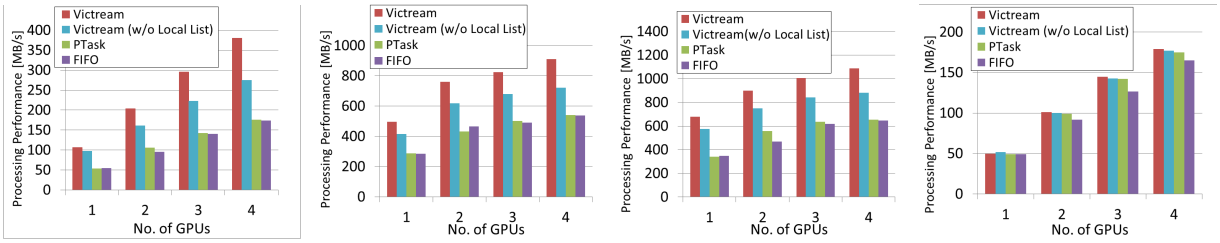
評価した計算性能のスケーラビリティを図 6 に示す．図の縦軸は計算性能を示し，入力データサイズと 1 秒当たりの計算回数の積である．従ってこの指標は 1 秒あたりに処理されるデータ量であり，GPU 数に対する計算性能を示す．

図 6 の評価結果は Victream が全てのベンチマークにおいて FIFO と PTask より良い性能を保持することを示している．性能差はロジスティック回帰で最も大きく，PTask より 92%-117%良い．ロジスティック回帰では GPU メモリに入りきらないデータに対する繰返し計算が行われる．それぞれの繰返し毎に Victream スケジューラはその前の回の計算で GPU メモリに残ったデータ群に対する計算から行い，また複数段の計算をまとめて行う．それに対し，FIFO スケジューラは入力データパーティションの状態を考慮せず全ての繰返し計算において同じ順番でサブタスクを実行する．一方 PTask スケジューラは入力データパーティションの場所を考慮するが，スケジューラのキューの先頭のサブタスクの入力データパーティションが GPU からスワップアウトされていない場合だけである．もし全てのデータパーティションがスワップアウトされていた場合，先頭のサブタスクはスケジュール可能になった GPU に割り当てられる．その場合，スケジューラのキューの後ろで待機している他のサブタスクの入力データパーティションをスワップアウトしてしまう．

図 6 はまた，図 4 のスケジューラの実装に示した GPU ローカルリストをスケジューラが用いない場合の結果も示している．この場合，スケジューリングで探索するサブタスクの候補は従来手法と同じである．この測定では，スケジューラはグローバルリストのから貪欲法を用いて最も I/O が小さいサブタスクをスケジュールする．従ってこの結果は従来の DAG 計算にプリフェッチと I/O 最小化を適用した場合の結果である．評価結果はサブタスクの探索空間の拡大が Out-of-Core 処理の性能向上に寄与することを示している．ロジスティック回帰における性能向上は 9%-38%である．

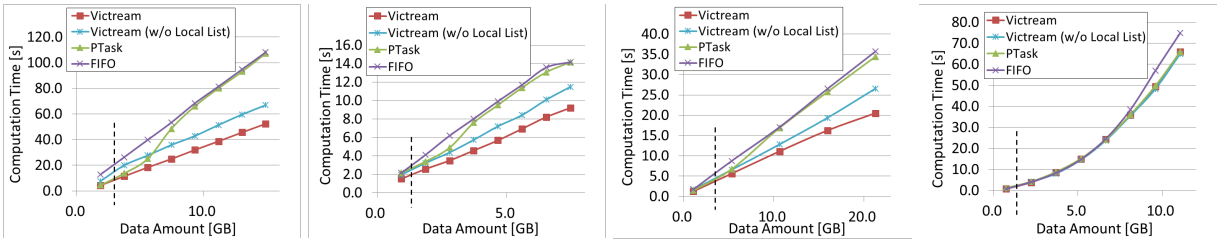
またソートと Blur フィルタにおける GPU 数に対する性能飽和は入出力データの保持に用いた RAMdisk の I/O 帯域が原因である．一方行列積では，異なるスケジュール手法の間で性能差がほとんど見られなかった．Victream は行列積でも I/O 量を削減したが，計算の性能ボトルネックが演算のため，4 つの手法の間の性能差は見られなかった．

図 7 は異なる入力データサイズに対する計算時間の評価結果である．それぞれのグラフの点線は計算が Out-of-Core 処理となる境界を示している．計算が Out-of-Core 処理でない場合，Victream と PTask の性能差は殆ど見られない．一方，データ量が増加し Out-of-Core となると Victream が良い性能を示している．また計算が Out-of-Core でない



(a) ロジスティック回帰　(b) ソート　(c) Blur フィルタ　(d) 行列積

図 6　計算性能のスケーラビリティ.



(a) ロジスティック回帰　(b) ソート　(c) Blur フィルタ　(d) 行列積

図 7　入力データサイズを変化させた場合の計算時間.

場合 Ptask の性能は FIFO より優れている．PTask スケジューラが入力データの場所を考慮することで GPU の I/O を FIFO より削減していると考えられる．一方行列積の計算時間は入力データサイズに対し非線形に増加している．これは行列積における演算量が入力データの 2 乗に比例するからである．

これらの評価結果は Victream が Out-of-Core 処理において GPU 数にスケールした計算性能を提供し，その性能は既存手法より優れていることを示している．サブタスクのスケジューリングを I/O のスケジューリングと同時に決定することでサブタスクのスケジューリング候補を拡張する手法は，従来の演算が実行可能になったサブタスクだけをスケジューリング候補とする手法に単に貪欲法を適用した場合より，優れた性能を示すこともわかった．

5. まとめ

本稿では [1] で提案した複数 GPU を用いた Out-of-Core 処理でボトルネックとなる I/O オーバヘッドを最小化する Victream ミドルウェアのスケジューラの詳細を述べ, State-of-the-Art の従来技術と比較評価を行った．Victream では従来上流のサブタスクが完了し実行可能となったサブタスクのみをスケジューリング対象としていた探索対象を拡張する．そのため Victream スケジューラは DAG を用いた計算において GPU の I/O のスケジュール時に演算のスケジュールを決定するという制約を導入した．その上で GPU の I/O リソースを効率的に用いるためのプリフェッチと GPU への I/O 量を最小化するサブタスクの実行順を実現した．それにより，State-of-the-Art の従来技術と比較して最大で 117%の性能向上が得られた．またスケジューリング対象の拡張による効果は 38% だった．

参考文献

[1] 鈴木順， 菅真樹， 林佑樹，荒木拓也，宮川伸也，喜連川優：リソース分離アーキテクチャのためのアクセラレータミドルウェア “Victream” の提案，2016 年並列/分散/共著処理に関する「松本」サマー・ワークショップ (SWoPP2016) (2016).

[2] Isard, M., Budiu, M., Yu, Y., Birrell, A. and Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks, *ACM SIGOPS Operating Systems Review*, Vol. 41, No. 3, ACM, pp. 59–72 (2007).

[3] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S. and Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, USENIX Association, pp. 2–2 (2012).

[4] Rossbach, C. J., Currey, J., Silberstein, M., Ray, B. and Witchel, E.: PTask: operating system abstractions to manage GPUs as compute devices, *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ACM, pp. 233–248 (2011).

[5] : インテル Xeon プロセッサー E7-8894 v4, , 入手先 ⟨https://ark.intel.com/ja/products/96900/Intel-Xeon-Processor-E7-8894-v4-60M-Cache-2$_4$0 − GHz⟩

[6] NVIDIA: NVIDIA TESLA P100 GPU ACCELERATOR, http://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-PCIe-datasheet.pdf.

[7] Sundaram, N., Raghunathan, A. and Chakradhar, S. T.: A framework for efficient and scalable execution of domain-specific templates on GPUs, *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, IEEE, pp. 1–12 (2009).