論　文　の　要　旨

題 目　　A Study on Efficient GPU Implementations to Compute the Diameter of a Graph

(グラフの直径計算のための効率的な GPU 実装に関する研究)

氏 名　　髙 藤 大 介

複雑なネットワークの構造解析などのために点と辺で構成されたグラフにモデル化されており，グラフの全点対間の最短距離はよく使用されている．たとえば，全点対間の最短距離の中の最大値はグラフの直径と呼ばれ，ネットワークの通信遅延解析に応用がある．グラフの全点対間の最短距離を求めるアルゴリズムとしてフロイド・ワーシャルアルゴリズムがよく知られているが，計算量が非常に大きい．そのため，これを並列計算によって高速化するためにブロック化フロイド・ワーシャルアルゴリズムが提案され，多くの並列環境で実装されてきた．本論文は，まずブロック化フロイド・ワーシャルアルゴリズムの並列実装の高速化に関する成果を述べる．また，複数の異なる入力に対しても高速に計算する並列実装の成果も述べる．

本論文では，GPU (Graphics Processing Unit) と呼ばれる演算装置を使用した，効率的な並列計算に関する研究をおこなった．GPU は画像処理専用の演算装置として提供されていたが，コアと呼ばれる演算器が多く含まれており，これらのコアを使った汎用並列計算機として多くの研究が行われている．NVIDIA 社は自社が提供する GPU を効率的に計算させる並列計算アーキテクチャ CUDA を提供しており，この CUDA の利用者は非常に多く，本論文ではこの CUDA を用いて GPU 実装を行っている．

本論文は，グラフの直径計算に有向なブロック化フロイド・ワーシャルアルゴリズム

の GPU 実装とその他の並列計算環境での実装，複数の異なる入力に対し高速に計算する GPU 実装の研究成果を示すものである．申請論文の章構成は次の通りである．

第 1 章は概要，第 2 章は GPU アーキテクチャの説明である．

第 3 章では，ブロック化・フロイド・ワーシャルアルゴリズムの GPU 実装およびマルチコア実装について述べる．ブロック化フロイド・ワーシャルアルゴリズムは全点対の最短距離を計算する際に，いくつかの領域に分割し，領域ごとに並列計算を行うことにより高速化を図っているが，いくつかの領域計算には依存関係があるため，領域計算には計算順序が決められている．

既存の GPU 実装では計算順序を維持するために，分割された領域を依存関係に基づき次のようにグループに分ける：グループ内の異なる 2 つの領域は依存関係がなく，異なるグループの 2 つの領域には依存関係がある．これによりグループ内の領域を並列に計算しその終了後に同期処理を行った後で，依存関係のあるグループの計算を開始する．しかし，この実装では同期処理を行う回数が増え，計算時間の増加の原因になっている．

本論文では，グループごとの同期処理を減少させることにより計算時間を減少させた実装を行っている．依存関係を考慮して各領域に計算順序の番号を付けその小さい順に処理を行うが，依存関係のある領域には計算開始可能かどうかの制御のみを行う．これにより同期処理を減らすことができ，高速化を実現している．さらなる高速化のために，1 つの命令を同時に複数のデータに適用する並列化手法である SIMD 演算を実装し，提案実装に適用している．

ここまでは 1 つのグラフに対する全点対の最短距離を高速に計算することに着目したが，複数のグラフが与えられたときに，効率的に同時に計算することを目指した．このように複数の独立した入力に対し，順にまたは並列に計算することを特にバルク計算という．1 つのグラフに対する手法の番号付けを拡張したバルク計算の GPU 実装を示し，そ

の性能を示している．さらに，近年は複数の CPU コアを有する計算機が普及しているため，ブロック化フロイド・ワーシャルアルゴリズムをマルチコアプロセッサ上に実装し，その性能を明らかにしている．

第 4 章では，アルゴリズム実行中の各ステップにおいてアクセスされるアドレスが入力に依存しない逐次アルゴリズムを，オブリビアスなアルゴリズムという．まず，オブリビアスなアルゴリズムのバルク計算を行う CUDA C プログラムを自動生成するツールを開発している．このツールを使うと，オブリビアスなアルゴリズムの C 言語プログラムから，そのバルク計算を行う CUDA C プログラムに自動で変換することが可能である．生成された CUDA C プログラムは CUDA で動作可能な GPU で実行可能であり，CUDA C プログラムの初学者にとって有効なツールでもある．

GPU のメモリアクセスにおいて，連続するアドレスに保存されたデータを読み書きする場合，そうでないデータの読み書きに比べ高速に実行できることが知られている．この事実を基に，複数の異なる入力データを連続するアドレスに保存できるよう工夫することで，バルク計算を行う GPU 実装を高速化した．性能を評価するために，バイトニックソート，フロイド・ワーシャルアルゴリズム，モンゴメリ乗算アルゴリズムのバルク計算を GPU 上に実装し，逐次処理と比較しそれらの性能を評価した．

バルク計算において大量のデータを入力するとデータの転送時間が長くなるため，解を出力するまでの時間が長くなる．この解決のために，データ転送時間を計算時間で可能な限り隠蔽するような GPU 実装も提案している．

最後に，第 5 章で論文の結論を述べる．