様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成 ２７ 年　 ６ 月　 ３ 日現在

機関番号：１２６０１
研究種目：基盤研究(A)
研究期間：2011～2014
課題番号：２３２４０００２

研究課題名（和文）メニーコア環境での汎用計算モデル開発

研究課題名（英文）Development of General Computation Models for Many-core Environments

研究代表者
定兼　邦彦（SADAKANE, Kunihiko）

東京大学・情報理工学系研究科・教授

研究者番号：２０３２３０９０

交付決定額（研究期間全体）：（直接経費）　　33,400,000円

研究成果の概要（和文）：GPU向けアルゴリズムを漸近解析するための並列計算モデルとしてAGPUモデルを提案した．アルゴリズムの正確な計算量はデバイス仕様に依存するが，AGPUモデル上で解析された計算量の高々定数倍である．AGPUモデルにより，GPUデバイスの仕様や入力データの値に依らない汎用的なアルゴリズムの性能を知ることができる．また，I/O計算量が最適となる比較ソートアルゴリズムを提案した．実際のGPUでの実験により，既存手法よりも1.9倍高速となった．

研究成果の概要（英文）：We have proposed a parallel computation model AGPU to analyze asymptotic behavior of GPU algorithms. Though precise complexities of algorithms depend on architectures of GPU devices, these are bounded from above by a constant factor. By using our AGPU model, we can estimate performance of algorithms which do not depend on specification of GPU devices nor input data. We have also proposed a comparison-based sort algorithm which has optimal I/O complexity. Experimental results on real GPUs show that our algorithm runs 1.9 times faster than an existing algorithm.

研究分野：アルゴリズムとデータ構造

キーワード：メニーコア　GPGPU　並列アルゴリズム　省スペースアルゴリズム

１．研究開始当初の背景

プロセッサの動作クロック周波数の向上は限界を迎えており，周波数向上に代わるパフォーマンス向上の手段として並列アーキテクチャが注目されている．GPU (Graphics Processing Unit)は元々はグラフィック処理のための専用プロセッサとして開発された．しかし，非常に高い並列性を持っていることから，グラフィック処理以外にも GPU が使われ始めている．汎用の処理に GPU を使用することは GPGPU (general-purpose GPU)と呼ばれており，安価に超並列環境が構築できることから注目されている．

GPU は多数のコアを用いて効率よく処理を行うため，特殊なアーキテクチャとなっている．GPU プログラミングにおいては，このアーキテクチャを適切に考慮する必要がある．NVIDIA 社は GPGPU のための開発環境として，CUDA を提供しており，CUDA 上で開発することにより，様々な GPU モデル上で動作するプログラムを実装することができる．しかし，最適なパフォーマンスを得るためには，GPU アーキテクチャを適切に考慮してアルゴリズムを設計する必要がある．

逐次アルゴリズムの評価では，RAM(Random Access Machine)モデル上での漸近解析が一般的に行われている．RAM モデルはすべての逐次実行マシンに対する抽象化となっており，RAM モデルを用いて漸近解析を行うことで，デバイスの仕様や入力データの値に依らない汎用的なアルゴリズムの性能を知ることができる．一方，並列実行マシンには，RAM モデルのような共通の抽象化が存在しない．並列アルゴリズムの漸近解析に一般的に使用されているモデルに PRAM モデルがあるが，PRAM モデルは GPU アーキテクチャとは大きく異なっており，GPU 向けアルゴリズムの性能を正しく評価できない．文献[1] では GPU における実際の計算実行時間を精度よくシミュレートすることについて検討されているが，
計算実行時間は GPU のモデルに大きく依存するため，GPU 向けアルゴリズムの汎用的な性能評価とならない．

２．研究の目的

GPU 向けアルゴリズムを漸近解析するための並列計算モデルとして AGPU モデルを提案する．アルゴリズムの正確な計算量はデバイス仕様に依存するが，AGPU モデル上で解析された計算量の高々定数倍である．AGPU モデルにより，GPU デバイスの仕様や入力データの値に依らない汎用的なアルゴリズムの性能を知ることができる．

３．研究の方法

まず GPU のための計算モデルを定義し，それを用いてアルゴリズムを設計する．計算機実験により，モデルの妥当性を検証する．

４．研究成果

AGPU モデルのアーキテクチャを図１に示す．AGPU モデルのアーキテクチャは並列計算を行うためのデバイス(GPU)とデバイスを制御するためのホスト(CPU)の異種混載システムとなっている．デバイスは p 個のコアを備えている．コアのワード長は w ビットであり，コアはワード単位でデータにアクセスする．また，デバイスは k 個のマルチプロセッサで構成されており，各マルチプロセッサは b 個のコアを備えている．すなわち p=kb である．

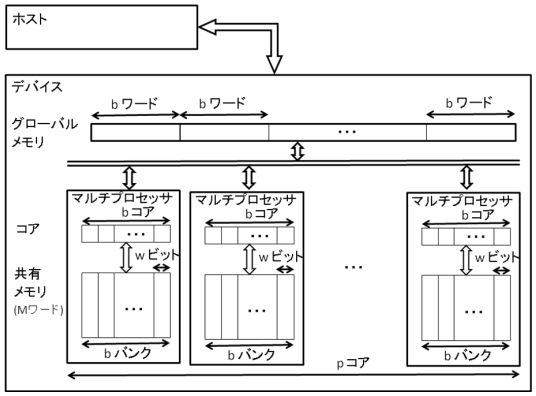

図 1 AGPU モデルのアーキテクチャ

マルチプロセッサはホストから起動されたプログラムを個別に実行する．すなわち，マルチプロセッサは他のマルチプロセッサとの通信手段および同期手段を持たない．ホストはすべてのマルチプロセッサがプログラム実行を完了するのを待つことにより，マルチプロセッサ間の同期を行うことができる．しかし，マルチプロセッサの処理完了時，共有メモリのデータはすべて削除される．共有メモリの必要なデータはマルチプロセッサの処理終了時にすべてグローバルメモリに書き込む必要がある．

マルチプロセッサ内のすべてのコアは常に同一の命令を実行する．ただし，オペランドに指定されるデータアドレスについてはコアごとに指定することができる．また，命令には実行条件を含めることができ，条件を満たすコアのみ命令を実行させることができる．

デバイスは 2 種類のメモリを備えている．1 つ目はグローバルメモリである．これは低速であるが大容量であり，すべてのマルチプロセッサおよびホストからアクセスすることができる．グローバルメモリは b ワードごとのブロックに分割されている．同一命令を実行するマルチプロセッサ内のコアが同一ブロックにアクセスする時，1 回のメモリアクセスで全コア分のデータにアクセス可能である．これはコアレッシングと呼ばれており，処理時間に大きな影響を与える．

2 つ目は共有メモリである．各マルチプロセッサは内部に容量 M ワード (bw ≤ M) の共有メモリを備えている．これは高速であるが低容量である．また，マルチプロセッサ内部のコアからのみアクセス可能である．共有メモリは b 個のバンクから構成される，同一命令を実行する b 個のコアのそれぞれが異なるバンクにアクセスする時，単位時間でデータにアクセスできる．複数のコアが同一のバンクにアクセスする時は，処理がシリアライズされる．これはバンクコンフリクトと呼ばれており，これも処理時間に大きな影響を与える．

以上で定義される計算モデルを AGPU(p,b,M,w)と記載する．ただし M,w については，省略される場合がある．

AGPU(p,b,M)上での比較ソートの時間計算量および I/O 計算量の下界について考察する．時間計算量については，並列処理を行わない場合の下界がΩ(n log n)であり，AGPU(p,b,M)のコア数は p なので，自明な下界はΩ((n/p) log n)となる．I/O 計算量については，I/O モデルでの計算量下界がΩ((n/b)log_(M/b) (n/b))であることが知られている[2]ことから，AGPU(p,b,M)での I/O 計算量下界もΩ((n/b)log_(M/b) (n/b))となる．

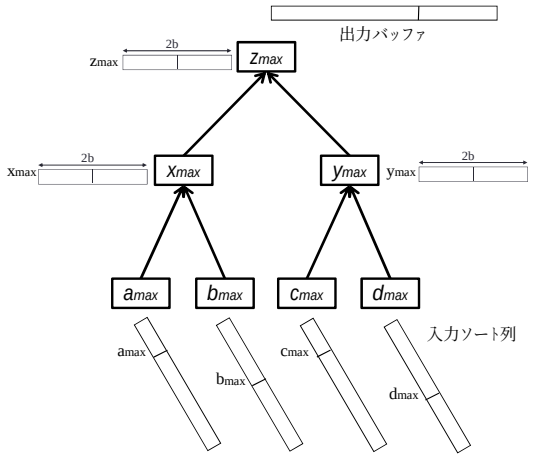

図 2 マージ処理に用いるヒープ

本研究では I/O 計算量が最適となるアルゴリズムを提案した．本アルゴリズムでは，まず入力列を大きさ b ごとに分割する．分割された入力列のそれぞれを基本ブロックと呼ぶ．次に各基本ブロックをソートする．次に各基本ブロックのマージする．マージしてできた列をサブアレイと呼ぶことにする．そして，サブアレイが 1 つになるまで，サブアレイのマージを繰り返す．マージの処理は 3 つのフェーズに分けられる．すなわち，サブアレイの個数が多い時は通常のマージを行う（この処理を「前半」の処理と呼ぶ）が，サブアレイの個数が少なくなったら，ピボット値によるデータの振り分けを行い（「データ振り分け」処理），振り分けられたデータごとにマージを行う（「後半」の処理）．

d 個のサブアレイをマージして 1 つのサブアレイを出力する処理について説明する．提案アルゴリズムでは，この処理を繰り返し行うことにより全入力に対するマージを行う．この処理は 1 つのマルチプロセッサを用いて行われる．

まず，この処理のために用いるヒープについて説明する．図 2 に d=8 の時のヒープの例を示す．ヒープは d 個の葉を持ち，それぞれは入力サブアレイへのポインタを保持している．葉は後述する規則で入力データを親に渡す．内部節点のそれぞれは共有メモリ中に大きさ 2b のバッファを持っている．内部節点の数は d-1 個なので，共有メモリ使用量の合計は 2b(d-1)である．バッファ内の要素は常にソートされた状態であるようにする．根を除く内部節点は後述する規則でバッファ内のデータを親に渡す．根は後述する規則でバッファ内のデータを出力サブアレイに書き込む．全節点はキーを持っている．キーの値は自身のバッファ(葉の場合は入力サブアレイ)から親(根の場合は出力サブアレイ)にデータを移動した際の最終要素の値とする．バッファ，サブアレイはソート済みなので，最終要素は移動するデータの最大値である．キーはヒープ条件を満たしている．また，ヒープの各節点には index が付けられており，根は 1，節点 i の左の子は 2i，右の子は 2i+1 である．

次に Heapify の処理について説明する．Heapify(i)は index i の節点のバッファに対し，子のバッファから b データを移動する処理である．節点 i のバッファに格納されているデータの要素数が b 以下の際に呼び出すことができる．節点 i はキーが小さい方の子からデータを移動させ，その子のキーの値を移動した最終要素の値に更新する．そして自バッファについて，元からあるデータと取得したデータをマージする．その後，同様の処理をデータ取得先の子に対しても行う．これを葉に到達するまで繰り返す．

次にマージの処理手順について説明する．まず，Heap の初期化として，index の大きい内部節点から順に Heapify を行う．そして，自節点のキーの値を INFINITY（正確には入力データのどの要素と比較してもそれより大きくなる値）に設定する．その後，根のノードから b データを出力サブアレイに書き込み，根に対して，Heapify を行う．これをすべてデータが出力し終わるまで繰り返す．

提案アルゴリズムの計算量は時間計算量が O((n/p) log b log (n/b))，I/O 計算量が O((n/b) log_(M/b) (n/b)) となる．I/O 計算量は下界と一致するため最適である．

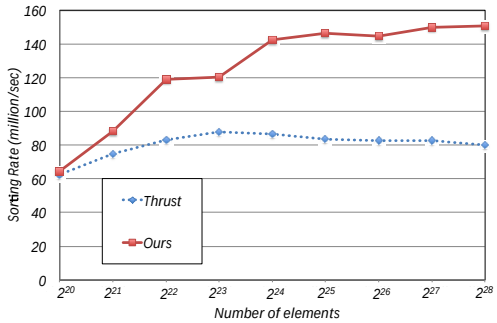

図 3 ソーティングレート

図 3 に実験結果を示す．これは 2^{20} 個から 2^{28} 個の整数値をソートする際のソーティングレート（ソートした値の個数を掛った時間で割った値）である．既存のアルゴリズムを実装した Thrust ライブラリと比較して，1.9 倍高速になっている．また，I/O 計算量は Thrust の 27% に削減されている．

以上のように，AGPU モデルを提案し，計算量を解析するとことで，既存のアルゴリズムが非効率である点を発見し，それを改善することが可能となった．最も基本的な問題であるソーティングにおいて I/O 計算量が最適なアルゴリズムを開発し，実際の GPU において既存手法よりも高速になることを示した．

＜引用文献＞

[1] Kothapalli, K., Mukherjee, R., Rehman, M., Patidar, S., Narayanan, P. and Srinathan, K.: A performance prediction model for the CUDA GPGPU platform, High Performance Computing (HiPC), 2009 International Conference on, pp. 463-472 , 2009. DOI:10.1109/HIPC.2009.5433179

[2] Aggarwal, A. and Vitter, Jeffrey, S.: The input/output complexity of sorting and related problems, Commun. ACM, Vol. 31, No. 9, pp. 1116-1127, 1988. DOI:10.1145/48529.48535

５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 10 件）

[1] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, Tak-Wah Lam. MEGAHIT: An ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph, Bioinformatics, 2015. DOI: 10.1093/bioinformatics/btv033

[2] Atsushi Koike, Kunihiko Sadakane. A Novel Computational Model for GPUs with Applications to Efficient Algorithms, International Journal of Networking and Computing, 5(1), 26-60, 2015.

[3] Wing-Kin Sung, Kunihiko Sadakane, Tetsuo Shibuya, Abha Belorkar, Iana Pyrogova. An O(m log m)-time algorithm for detecting superbubbles, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2015. DOI:10.1109/TCBB.2014.2385696

[4] Ryoji Tanabe, Alex Fukunaga. On the pathological behavior of adaptive differential evolution on hybrid objective functions, Proceedings of the ACM/SIGEVO Genetic and Evolutionary Computation Conference, pp. 71-78, 2014. DOI:10.1145/2576768.2598322

[5] Ryoji Tanabe, Alex Fukunaga. Improving the performance of SHADE using linear population size reduction, Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1658-1665, 2014. DOI: 10.1109/CEC.2014.6900380

[6] Ryoji Tanabe, Alex Fukunaga. Reevaluating Exponential Crossover in Differential Evolution, Proceedings of Parallel Problem Solving from Nature PPSNXIII, pp. 201-210, 2014. DOI: 10.1007/978-3-319-10762-2_20

[7] Erik D. Demaine, Yamming Huang, Chung-Shou Liao, Kunihiko Sadakane. Canadians Should Travel Randomly, Proceedings of ICALP, LNCS(8572), 380-391, 2014. DOI:10.1007/978-3-662-43948-7_32

[8] Shuhei Denzumi, Jun Kawahara, Koji Tsuda, Hiroki Arimura, Shin-Ichi Minato Kunihiko Sadakane. DenseZDD: A Compact and Fast Index for Families of Sets, Proceedings of SEA, LNCS(8504), 187-198, 2014. DOI:10.1007/978-3-319-07959-2_16

[9] Tomohiko Ohtsuki, Naoki Nariai, Kaname Kojima, Takahiro Mimori, Yukuto Sato, Yosuke Kawai, Yumi Yamaguchi-Kabata, Tetsuo Shibuya and Masao Nagasaki. SVEM: a Structural Variant Estimation Method using Multi-Mapped Reads on Breakpoints, 1st International Conference on Algorithms for Computational Biology (AlCoB), LNBI 8542, pp 208-219, 2014. DOI: 10.1007/978-3-319-07953-0_17

[10] Tatsuya Imai, Alex Fukunaga. A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning, Proceedings of the European Conference on Artificial Intelligence, 459-464, 2014.
DOI: 10.3233/978-1-61499-419-0-459

〔学会発表〕（計 1 件）

[1] Makoto Onizuka, Hiroyuki Kato, Soichiro Hidaka, Keisuke Nakano, Zhenjiang Hu. Optimization for Iterative Queries on MapReduce, 40th International Conference on Very Large Data Base (VLDB), Hanzhou, China, September 1-5, 2014.

６．研究組織

(1)研究代表者

定兼　邦彦 (SADAKANE, Kunihiko)
東京大学・大学院情報理工学系研究科・教授
研究者番号：２０３２３０９０

(2)研究分担者

渋谷　哲朗 (SHIBUYA, Tetsuo)
東京大学・医科学研究所・准教授
研究者番号：６０３９６８９３

福永　ALEX (FUKUNAGA, Alex)
東京大学・大学院総合文化研究科・准教授
研究者番号：９０４５２００２

胡　振江 (HU, Zhenjiang)
国立情報学研究所・アーキテクチャ科学研究系・教授
研究者番号：５０２９２７６９