6A-05

Java で実装した空間応用一般均衡分析モデルでの OpenMP と CUDA の利用

久保　裕也†　　熊谷　聡‡
千葉商科大学†　　日本貿易振興機構アジア経済研究所‡

1. IDE-GSM とは何か

空間経済学は、スタンダードな経済学の中で必ずしも上手く扱われてこなかった経済活動の地理的な側面に注目した新しい分野で、経済活動が特定地域に集まろうとする「集積力」と、周辺地域に分散しようとする「分散力」の相互作用によって、経済活動の地理的分布が決定されると考える [1]。我々はこのモデルをさらに拡張して計算可能一般均衡モデル（CGE）とし、国や地域間のさまざまな貿易・交通円滑化政策の影響を分析することを目的に、アジア経済研究所・経済地理シミュレーションモデル（Institute of Developing Economies-Geographical Simulation Model: IDE-GSM）を開発している[2][3]。このモデルに投入するデータセットを、世界全体を概ね網羅した内容として、185 の国、それぞれの国を区分・詳細化したものとしての計 3,288 地域、さらにその各地域について 8 産業（農業、自動車、電子・電機、繊維・衣料、食品加工、その他製造業、サービス業、鉱業）の GDP データで構成している。また、それぞれの地域を代表する都市間での航路や道路などを、実際の交通網を反映した形で、11,129 の頂点と 40,190 本の辺による有向グラフで表現している。有向グラフの各辺には輸送方法（道路・海路・空路・鉄道・高速鉄道）ごとに算出された輸送費が重みとして付与されている。

2. IDE-GSM の実装

IDE-GSM のモデルおよびデータセットを用いて、経済シミュレーションを行うことができる。プログラムの主な部分は Java 仮想マシン上で動作し、ボトルネック部分については Java Native Access (JNA) を介して共有ライブラリ内の関数を呼び出す構成にしている。実行ファイルをビルドするツールに Gradle と Make を用いており、OpenMP や CUDA などの動作環境に応じた実装を選択的に組み込めるようにしている。

IDE-GSM のプログラムにおける処理のボトルネックは大きく 2 つある。1 つめは、交通網を表現した有向グラフをもとに、全都市間での最適な輸送ルートを求める際の、全点対最短経路計算である。2 つめは、全都市・全産業ごとの市場の相互依存関係を連立方程式としたものについて、その解を求めること、すなわち一般均衡価格を求めることである。これらのボトルネックについて、動作環境に応じた並列化技術を適切に選んで組み込むことが課題となる。

本稿における IDE-GSM の実験に用いる動作環境の、OS、CPU、GPU の組み合わせを表 1 に示す。

環境	OS	CPU	GPU
①	Ubuntu Linux	Intel Core™ i7-10700 2.90GHz(8 コア 16 スレッド)	nVidia GeForce RTX 3060
②	macOS	Intel Xeon®W-3254 3.20GHz(16 コア 32 スレッド)	AMD Radeon HD5770
③	macOS	Apple M1 Max(高性能 8 コア+高効率 2 コア)	

表 1: 動作環境

3. 全点対最短経路計算に用いる実装の選択

IDE-GSM による経済シミュレーションの過程で、全都市間での最適な輸送ルートを求める際に、全点対最短経路計算を行う。計算対象となる重み付き有向グラフは隣接リストとして定義され、頂点 V=11,129 と辺 E=40,190 となる大きさで、各辺に非負の実数値で重みが付与されている。一般に、全点対最短経路計算に用いられるアルゴリズムには、Floyd-Warshall 法と Johnson 法がある。その計算量は、前者は $O(V^3)$、後者は $O((E+V)\log V)$ である。本件の計算対象は、頂点数がやや多めであるのに対して辺の数が少ない、極めて疎なグラフであるので、Johnson 法の計算量のほうが大幅に小さくなると予想される。ただし先行研究[4]によれば、Floyd-Warshall 法のアルゴリズムは、ベクトル化、ブロック化、並列化を組み合わせて、GPGPU での効率的な演算が可能であるとされる。逆に Johnson 法には並列化が可能なものの、メモリをランダムアクセスする際にキャッシュミスが起こりやすく、ベクトル化には不向きであるとされている。そこで我々は、ブロック化された Floyd-Warshall 法と Johnson 法のそれぞれについて OpenMP と CUDA とを選択的に利用可能な共有ライブラリとすることで、IDE-GSM に組み込んで比較することにした。表 2 は、IDE-GSM のデータセットを表 1 の動作環境①を用いて各方式で計算した所要時間を各 5 回計測し、その中央値を示したものである。この結果では、Johnson 法を CPU 上で OpenMP により並列処理する組み合わせが、それ以外の組み合わせ(GPU での Floyd-Warshall 法を 32 並列で処理する場合を含む)と比べ

Using OpenMP and CUDA in Spatial Computable General Equilibrium Analysis implemented with Java
† Hiroya Kubo, Chiba University of Commerce
‡ Satoru Kumagai, IDE-JETRO

て最も速い。頂点数が多い疎なグラフは CPU コアごとの大きめのキャッシュを活用しながら並列化することで効率的な計算が可能であることが見て取れる。

方式	Floyd-Warshall 法	Johnson 法
CPU(逐次処理)	2232.2sec	11.9sec
CPU(OpenMP 並列数 16)	376.3sec	**1.7sec**
GPU(CUDA)	7.7sec	7.6sec

表 2: 全点対最短距離計算の所要時間

4. 一般均衡価格計算に用いる実装の選択

IDE-GSM のシミュレーションでは、一般均衡価格を求める処理として、輸送費を考慮に入れながら世界の全ての都市の GDP・賃金水準・財価格などを相互に参照し、各都市・産業ごとの価格指数を算出する。具体的には、配列長 3,288 の double 型配列 16 種のデータ(計 421KB)を与え、double 型配列 1 個を返す関数としたものを、3,288 都市×8 産業の並列度で実行する形を 1 セットとし、複数セット繰り返すものである。これをスカラー演算で実行する方式としていたものについて、ベクトル化を試みた。その際、次の 2 点が問題となった。1 点目は、この関数内で、底および冪指数が double 値であるような冪乗が計 11 回行われるというものである。動作環境により SIMD 命令セットの対応状況、数学ライブラリやコンパイラによる支援状況がまちまちであり、ベクトル化による性能向上が期待できない場合があると考えられた。2 点目は、本件の計算に必要となるデータサイズがやや大きく、データ構造の変換とメモリ転送に要するオーバーヘッドが生じること、データがキャッシュに乗り切らないことなどにより、ベクトル化の効率が低下する可能性があるということである。これらの問題に対して、動作環境ごとに最適化した実装を行う以前に、より抽象度の高い実装方法として、Java 19 (openjdk 19.0.1 2022-10-18) における Vector API を用いたベクトル化を行うこととした。Vector API は、x86 および AArch64 の CPU を対象として、JIT コンパイルにより実行時に最適な SIMD 命令を組み立てて演算をすることができる。また Vector API では、Java 仮想マシンが直接管理するメモリ上のデータでベクトル処理を行えるため、共有ライブラリ側にデータを転送する必要がない。総じて、CPU の種類によらない単一のコードベースでの開発をしやすくなり、メモリ操作に起因する性能低下を避けることもできるだろうと考えた。

以上の方針で、Java Stream API により並列化されたスカラー演算をする実装をベクトル化した。表 3 に、その 1 セット分のシミュレーションの所要時間の比較を示す。動作環境①②③は CPU アーキテクチャや対応する命令セットが異なるが、いずれにおいても、スカラー演算よりベクトル演算の所要時間のほうがやや長い程度で、大きな違いは見られなかった。Java 仮想マシンがスカラー演算についてもともと非常に高度な最適化を行っていること、その一方で Vector API によるベクトル演算について動作環境の本来の性能を活かせるような最適化がまだ十分に行われていない可能性があることが推察された。

方式	スカラー演算	ベクトル演算
動作環境① x86 AVX2	**54.1sec**	56.7sec
動作環境② x86 AVX-512	**44.0sec**	45.2sec
動作環境③ AArch64 NEON	**46.8sec**	49.8sec

表 3: 一般均衡価格計算の所要時間

5. まとめと今後の課題

空間応用一般均衡分析モデルを用いたシミュレーションで必要とされる、全点対最短経路計算と一般均衡価格計算のそれぞれについて、複数の実装を行い、比較した。全点対最短経路計算では、CUDA を用いる実装や Floyd-Warshall 法による実装よりも、CPU で OpenMP により並列化した Johnson 法による実装が最も性能が高かった。一般均衡価格計算では、スカラー演算による実装とベクトル演算による実装について比較したところ、現段階では Java Vector API によるベクトル化では期待したような性能向上が見られず、通常のスカラー演算による実装がやや勝る結果となった。

今後の課題は、一般均衡価格計算について、CUDA を用いたベクトル化での実装をすることである。この計算は 3,288 都市×8 産業の並列度で実行できる内容なので、GPU 環境によっては、CPU よりも GPU でのほうが有利となる場合があると考えられるためである。

参考文献

[1] Fujita, M., Krugman, P. R., and Venables, A. *The spatial economy: Cities, regions, and international trade*, MIT Press, (1999).

[2] Kumagai, S., K. Hayakawa, I. Isono, S. Keola, and Tsubota, K. “Geographical simulation analysis for logistics enhancement in Asia”, *Economic Modelling*, 34, 145-153. (2013)

[3] Isono, I. ed. “The Need to Upgrade the Geographical Simulation Model (IDE-GSM)”, *ERIA Collaborative/Support Research Report,* IDE-JETRO, (2021)

[4] Moore, J. and Kalapos, J. “Floyd-Warshall vs Johnson: Solving All Pairs Shortest Paths in Parallel”, https://moorejs.github.io/APSP-in-parallel/ (2023 年 1 月 4 日閲覧)