生成AIを用いたプログラムの並列化の評価

立川崇之[1]

Evaluation of Parallelization of Programs Using Generative AI

Takayuki TATEKAWA

Abstract

The technology of generative AI, which produces output based on learned data patterns and relationships in response to natural language information input by the user, is rapidly developing. In this paper, we attempted to reduce the parallel programming effort by inputting computer programs that are considered to have high compatibility with generative AI and instructing them to parallelize based on a standardized specification. The results indicated that parallelization was possible to avoid incorrect processing for programs that were only practice problems, but for complex programs actually used in the natural sciences, parallelization caused a variety of errors. At present, parallelization of a program by a generative AI requires a human operator who is familiar with parallelization techniques to confirm the output results and modify them.

Keywords: Parallelization, Generative AI, Programming

1　はじめに

データのパターンや関係を学習し，新しい文章や画像を生成する「生成AI」の発展が目覚ましい．具体的なデータセットを学習して決められたルールに従い結果の出力を自動化する従来のAIとは異なり，非構造化データを学習し新たなものを生成することが特徴と考えられる．一方で，学習のために使用したデータの権利の取扱いや，教育現場における学生，生徒，児童の学習の放棄につながらないかという懸念もある[1),2)]．

本論文では，生成AIが得意としているプログラミングについて評価の一例を示す．研究分野によっては，大規模なシミュレーション，データ解析を必要とする場合がある．その際には処理を行うためのプログラムの性能が，処理時間に大きな影響を及ぼす．

特に本論文では並列処理に着目する．かつてはプロセッサ（CPU）が複数搭載され並列処理が可能な計算機は，スーパーコンピュータなどの高価な計算機に限られていた．ところが個人用PCにおいて，１つのプロセッサに複数のプロセッサ・コアを搭載した製品が出荷されるようになった．例えば2005年にAMD社がAthlon 64 X2を，Intel社がPentiumDを発表している．この発表以降，複数のプロセッサ・コアを搭載したプロセッサが個人用PCにも徐々に搭載されるようになった．つまり，個人レベルでも複数のプロセッサ・コアを用いて並列処理を行うことにより，処理の高速化の恩恵

[1] 高知工業高等専門学校　ソーシャルデザイン工学科　教授

を受けられる．

並列処理を行うためのプログラミングには様々な技法がある．高知高専では情報セキュリティコース５年に設置されている「ハイパフォーマンスコンピューティング」で半期をかけて背景となる知識やプログラミング技法を学習する．学習の際に,「並列処理による処理時間の短縮よりも，プログラムの並列化のコストが多大であるならば，並列処理のメリットはない」と述べている．並列処理に慣れない方にとっては，プログラムの並列化はハードルが高いと思われる．

そこで既存のプログラムを生成AIに並列化させることで，プログラムの並列化のコストを下げ，利用者が並列処理の恩恵を受けやすくなるかどうかを評価することが，本論文の目的である．既存のプログラムを並列化させる文章を生成AIに入力し，出力結果を実際に動作させ評価することで，生成AIによる並列化がどれくらい有効かを判断する．なお，本論文ではC言語のみ評価している．

2　並列化の手法

2.1 並列化可能なプログラムとは

プログラムにおける基本的な処理は，大きく３つに分けられる．その３つとは，処理を１つずつ順序立てて行う逐次処理，条件に応じて処理を変える分岐処理，処理を繰り返す反復処理である．これらの処理の中で，並列化を行う処理は反復処理である．同じ処理を多量のデータに繰り返し行う反復処理を並列化することにより，プログラム全体の実行時間を短縮できる．

ただし反復処理であっても,並列化できる場合と出来ない場合がある．並列化できる簡単な例は，巨大な行列の積の計算である．$N\times N$ 正方行列 A, B を考え，行列の i 行 j 列の成分をそれぞれ a_{ij}, b_{ij} と表すことにする．行列の積の計算

$$C = AB,$$

は $N\times N$ 正方行列 C の i 行 j 列の成分 c_{ij} を求める際に，以下の計算を行うことをなる．

$$c_{ij} = \sum_{k=1}^{N} a_{ik}\, b_{kj}$$

行列の成分を求めるには i, j それぞれについて１から N まで繰り返さなければならない．i, j に対する反復処理を並列化することで，行列の積の計算を高速化できる．

一方で，並列化できない処理の例として，フィボナッチ数列の漸化式の計算を考える．

$$a_{k+2} = a_{k+1} + a_k,\ a_1 = a_2 = 1$$

漸化式は,「前の計算の結果を用いて，次の計算を行う」ものである．このため，例えば a_{100} の値を知りたい場合には，a_{99} までの値を順次求めていかなければならず，並列化が出来ない．強引に並列化を行うと，未知の値を用いることになり，計算結果に誤りを生じる．

プログラムの並列化を行う場合には，まず上記のように並列化をして支障のないプログラムかどうかを考える必要がある．

2.2 OpenMP

OpenMPは，共有メモリ型の計算機で並列処理を行うプログラムを作成するための，標準化されたAPIである[3)]．並列化したい反復処理の部分に，指示文（ディレクティブ）を明示的に記載する．そして，OpenMPに対応したコンパイラでコンパイルする．Linuxなどで使用されるフリーのコンパイラであるGCCは，OpenMPに対応している．並列処理の際にはマルチスレッドで実行され，プログラムの同じ変数は基本的に同じメモリ領域に格納される．

```
#pragma omp parallel for
for (i=0; i<N; i++){
  /* 反復処理される命令一式 */
}
```

図１　OpenMP による並列化の指示文記載の例

OpenMPによる指示文記載の例を図１に示す．#pragma から始まる１行により，直後のfor文による反復処理が並列化される．実行の際に特に明示しなければ，計算機に搭載されているプロセッサ・コアの数と同じ並列度で実行される．たとえば４つのプロセッサ・コアを搭載するプロセッサを用いた計算機では，並列度４で実行される．

2.3 MPI

Message Passing Interface（MPI）は，分散メモリ型の計算機で並列処理を行うプログラムを作成するための，標準化されたAPIである[4),5)]．OpenMPとの違いは，ネットワークを介して接続された複数の計算機上で，一つのプログラムを並列実行できることである．MPIでは例えばオープンソースのOpenMPIが存在し，様々な計算機で使用することが出来る．

MPIの利点は，複数の計算機上での並列化が出来るため，大規模計算に非常に適しているという事である．その一方で，並列処理はマルチプロセスで動作し，プログラム作成時にはプロセス間のデータなどの通信処理を明示的に記述するため，OpenMPに比べて並列化プログラムの作成に手間がかかる．

3　生成AIによる並列化の試み

3.1 生成AIに対するプログラム変換の実行

本論文では，プログラムの変換にはOpenAI社のChatGPTを使用した[6)]．ChatGPTには自然言語処理のモデルGPT-3.5が搭載されており，対話形式での質疑応答が可能である．並列化に際しては「以下のプログラムを OpenMP による並列化で高速化してください.」のように文章を記述した後に，プログラムのソースコードを転載する．ChatGPTにおいては入力，出力の文字数に制限がある．このため，長いプログラムの並列化を行う場合には，入力の際はプログラムの一部分を切り出したり，主要部分だけを要約したりする必要がある．また，出力についても，文字数制限などで省略された部分について，重ねて質問する必要がある．

本論文では2023年９月から10月にかけてプログラムの並列化を試みた．その後のChatGPTの学習状況によっては，精度の高い並列化がなされる可能性があることに注意が必要である．

3.2 行列の積の例

まず，式(2)で示すような行列の積のプログラムについて，並列化を試みる．並列化前のプログラムにおいて，行列の積を計算する反復処理の行だけを図２で示す．

```
for (i=0; i<N; i++){
   for (j=0; j<N; j++){
      for (k=0; k<N; k++){
         c[i][j]+= a[i][k] * b[k][j];
      }
   }
}
```

図２　行列の積を計算する，並列化前の反復処理

OpenMPによる並列化を試みると，注意を払うべきところは払われたが，少々回りくどい変更がなされた．実行してみると， OpenMPによる並列化が成功している．OpenMPでは特に指示をしなければ，同じ変数は同じメモリ領域に格納される．ところが，一番外側の変数 i に関するループを並列化すると，並列化された各々のスレッド内で内部のループで現れる変数 j, k が使用される．各スレッドで j, k を別のものとして扱わないと，内部のループが正常に処理されない．このため，並列化の宣言では private というオプションで，各々の変数は独立とみなすプライベート変数であると明示する必要がある．また，行列の成分の計算において，足し上げの部分である変数 k のループを並列化して，各々のスレッドごとに足し上げを行い，最後に総和を行うreductionというオプションが追加されている．

```
#pragma omp parallel for private(i, j, k) shared(a, b, c)
   schedule(static)
   for (i = 0; i < N; i++) {
     for (j = 0; j < N; j++) {
         double temp = 0.0;
        #pragma   omp   parallel   for   reduction(+:temp)
   schedule(static)
         for (k = 0; k < N; k++) {
            temp += a[i][k] * b[k][j];
         }
         c[i][j] = temp;
      }
  }
```

図３　行列の積を計算する，OpenMPによる並列化後の反復処理

同様の並列化をMPIで行う場合についても実行した．MPIによる並列化も成功している．プログラムが長くなるので掲載は省略するが，「司令塔」となるランク０と名付けられたプロセスから，各プロセスに対し行列Ａ，Ｂのデータが配布される．各プロセスでは分割された範囲で行列の積の演算処理を行い，行列Ｃの部分的な結果をランク０のプロセスに返す．

3.3 数値積分の例

次に数値積分について並列化を行う．数値積分は，定積分

$$I = \int_a^b f(x)\,dx$$

を，積分範囲を n 個の等間隔の区間に細かく区切って足し合わせるシンプソンの公式を用いる[7]．

$$I = \frac{\Delta x}{3}\sum_{k=0}^{n/2}\langle f(x_{2k}) + 4\cdot f(x_{2k+1}) + f(x_{2k+2})\rangle$$

シンプソンの公式を用いた積分では，並列化において積分範囲をいくつかの区間に区切って，各々の区間での足し上げを行い，最後に取りまとめて総和を行う．演算は図４の様に行う．関数 f (x) は別途定義しておく．OpenMPではreduction，MPIではMPI_Reduceという命令を用いて指示すると，並列化して総和を行うことが出来る．

```
S=0.0;
for (i = 0; i < N/2; i++) {
  x=(2.0*(double)i)*dx;
  S+=(f(x)+4.0*f(x+dx)+f(x+2.0*dx))*dx/3.0;
}
```

図４　シンプソンの公式による数値積分を行うための反復処理

単純な一重積分の場合には，OpenMPでは図５のように正しい命令文が追加された．MPIでも，MPI_Reduceを用いたプログラムに正しく変換がなされた．

```
#pragma omp parallel for private(x) reduction(+:S)
```

図５　図４の反復処理をOpenMPで並列化した時に追加された行

ところが，以下のような二重積分を行う場合には問題が生じた．

$$\int_{-L}^{L} dx \int_{-L}^{L} dy\, f(x, y)$$

この積分を数値的に行うには，図６のような二重ループが必要となる．積分範囲が x, y それぞれで対称であることから，for文を用いたループを-NからNの範囲で記した．

```
for (i = -N; i < N; i++) {
  for (j = -N; j < N; j++) {
     /* 数値積分の演算 */
   }
}
```

図６　二重積分の数値積分を行うための反復処理

このプログラムの並列化を指示したところ，OpenMPの場合には j をプライベート変数に指定しないプログラムを出力した．また，MPIの場合には i の範囲を $0 \leq i < N$ と誤認識したプログラムを出力した．このため，出力されたプログラムを人間が確認して修正する必要があった．

3.4 自然科学のプログラムに対する例

最後に，自然科学の研究で用いるプログラムの並列化を試みる．宇宙に存在する様々な天体は，主に重力（万有引力）によって互いに引き合い集まって，現在のような姿をなしたと考えられる．そこで物質を質点の集団とみなし，互いに及ぼしあう重力によって，系全体がどのように進化するかというシミュレーションが長年なされている[8]．N個の質点を考えて番号を振る．これらの質点で，i 番の質点に及ぼされる重力は以下のように計算される．

$$\vec{F}_i = \sum_{j \neq i}^{N} G \frac{m_i m_j}{|\vec{r}_j - \vec{r}_i|^3} (\vec{r}_j - \vec{r}_i)$$

m_i は i 番の質点の質量，$\vec{r}_i$ は i 番の質点の位置ベクトル，Gは重力定数である．重力の計算は全ての質点同士でなされるので，$O(N^2)$ の計算量となる．アルゴリズムの工夫により$O(N \log N)$に計算量を減らすことが出来るが，本論文では $O(N^2)$ の計算量のアルゴリズムのまま，並列化がなされるかどうかを調査する．

やや長いが，具体的には図７のような反復処理の並列化を行う．

```
for(i=0;i<n;i++) {
    for(k=0;k<3;k++) a[i][k] = 0.0;
    pot[i] = 0.0;
    for (j=0;j<n;j++) {
      for(k=0;k<3;k++) dx[k] = x[j][k] - x[i][k];
      r2 = dx[0]*dx[0] + dx[1]*dx[1] + dx[2]*dx[2] + eps2;
      rinv = rsqrt(r2);
      mrinv = m[j]*rinv;
      mr3inv = mrinv*rinv*rinv;
      a[i][0] += mr3inv * dx[0];
      a[i][1] += mr3inv * dx[1];
      a[i][2] += mr3inv * dx[2];
      pot[i] -= mrinv;
    }
  }
  for(i=0;i<n;i++) pot[i] += m[i]/sqrt(eps2);
```

図７　万有引力の加速度と位置エネルギーを計算する反復処理

プログラム中のm，x，a はそれぞれ各質点の質量，位置，加速度を表す配列である．また，potは各質点が存在する場所における，単位質量当たりの位置エネルギーを表す．質点同士の距離が近づきすぎると重力が急激に大きくなるため，数値誤差回避のためにeps2という小さい値を導入している．また，エネルギー誤差評価のため，各質点における位置エネルギーも併せて計算する．数値誤差回避の小さい値を含めると，位置エネルギーは以下の式で表される．

$$V_i = -\sum_{j \neq i}^{N} G \frac{m_i m_j}{|\vec{r}_j - \vec{r}_i|^2 + \varepsilon^2}$$

この反復処理についてOpenMP, MPIによる並列化を試みる．まずOpenMPについては，プライベート変数の指定を間違えずになされた．具体的には，先頭のfor文の前に図８で示すような行が追加された．

```
#pragma omp parallel for private(i, j, k, dx, r2, rinv, mrinv,
  mr3inv) shared(a, pot, x, m, eps2, n)
```

図8　ChatGPTにより生成された図6の反復処理に対するOpenMPの指示文

また，引数 j のループの中で，図９のような if 文による条件分岐がなされた．実際には i=j の場合には重力を計算すると０になるので条件分岐は不要である．一方，図７で示したように，i=j の場合の位置エネルギーを余計に加算している分を修正する最後の行が残るので，位置エネルギーに誤りが生じる．

```
if (i != j) { // Exclude self-interaction
```

図9　ChatGPTにより生成された，誤った分岐処理の追加

並列化の出力結果を見て気づいたことは，並列化の際にChatGPTはプログラムの意図に気づき，「自己相互作用を除外」というコメントを追加していることである．本論文以外でプログラムをテストした時に，x や v を位置や速度を意味する変数と解釈したこともあった．MPIによる並列化を指示した場合には，データの分配，並列化，統合を行う関数を追記したが，OpenMPの場合と同様にif文による余計な条件分岐がなされた．

本プログラムの場合には，ChatGPTは部分的にはプログラムの意図するところまで解釈しているが，却って計算結果を誤らせるような並列化の処理を行っている．ChatGPTにおいて並列化を行うには，入力者がプログラムの意図や並列化で用いられる指示文，関数の意味を理解し，確認できることが必要である．

4　結論

本論文では，生成AIを用いて既存のプログラムを並列化することを試みた．共有メモリ型計算機にのみ適用できるOpenMP，分散メモリ型計算機にも適用できるMPIの２種類の手法を適用するように指示することで，並列化の手間を省くことが出来るのではないかと考えた．

適用結果は行列の積や一変数の数値積分のような，練習問題レベルではほぼ問題ない水準での並列化が出来た．しかしながら二変数の数値積分や，積分範囲の一端が０でないような数値積分では，プログラムの認識に誤りがあり，正しい並列化がなされなかった．自然科学のプログラムの例として万有引力の場合に適用した場合には，適用結果を改めて見直して修正したり，生成AIへの指示を細かく指定したりする必要があったりするなど，却って手間がかかった．現時点では生成AIはプログラムの並列化の補助には有用ではあるが，出力結果をそのまま用いるのではなく，人間の目で再度チェックする必要がある．

今後の応用として，グラフィックプロセッサ（GPU）へプログラムを対応させる高速化が考えられる．GPUへプログラムを対応させるためには，OpenMPと同様にGPUに処理させる反復処理を指示文で指定できる，OpenACCという規格がある[9]．また，GPUの製造元であるNVIDIA社による

CUDA に対応したプログラミングを行うこともある[10]．後者の方が高いパフォーマンスが得られるが，難度が上がる．また，GPUにプログラムを処理させる場合には，単にGPUでの処理を指示するだけでなく，計算機のメインメモリとGPU側のメモリとの通信を減らすことも重要である[11),12)]．現時点ではまだ人間の目によるチェックが不可欠であり，並列化の実現には手間がかかるが，将来は生成AIの技術の進化により，難度の高い並列化も容易に実現可能になることが期待できる．

本論文ではプログラムの並列化にのみ着目したが，広い視野で考えると生成AIはソフトウェア開発のあり方すら変える可能性が考えられる[13)]．また，本論文ではプログラムを理解できる人間の目が必要であると述べたが，そのような人間を育てるプログラミング教育の方法を大きく変える可能性も考えられる[14)]．生成AIがソフトウェア開発とプログラミング教育にもたらす可能性は非常に大きく，我々は有用な活用方法を今後検討していく必要がある．

謝辞

本論文について，高知高専専攻科の濵田幸希氏には注意深く読んでいただき，誤りを指摘していただきました．ここに感謝の意を表します．

参考文献

1）文部科学省：初等中等教育段階における生成AIの利用に関する暫定的なガイドライン（令和５年７月４日）．令和５年８月31日閲覧.
https://www.mext.go.jp/content/20230710-mxt_shuukyo02-000030823_003.pdf
2）文化庁著作権課：令和５年度著作権セミナー「AIと著作権」（令和５年６月）．令和５年８月31日閲覧
https://www.bunka.go.jp/seisaku/chosakuken/pdf/93903601_01.pdf
3）OpenMP 公式サイト．令和５年９月13日閲覧.
https://www.openmp.org/
4）MPICH 公式サイト．令和５年９月４日閲覧.
https://www.mpich.org/
5）Open MPI公式サイト．令和５年９月４日閲覧.
https://www.open-mpi.org/
6）ChatGPT公式サイト．令和５年９月８日閲覧.
https://chat.openai.com/
7）W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, “Numerical Recipes 3rd Edition: The Art of Scientific Computing”, Cambridge University Press, Cambridge, 2007.
8）R. W. Hockney and J. W. Eastwood, “Computer Simulation Using Particles”, Tailor & Francis, London, 1988.
9）OpenACC公式サイト．令和５年９月４日閲覧.
https://www.openacc.org/
10）NVIDIA CUDA Toolkit．令和５年９月４日閲覧.
https://developer.nvidia.com/cuda-toolkit
11）S. Portegies Zwart, R. Belleman, P. Geldof, “High-performance direct gravitational N-body simulations on graphics processing units”, New Astronomy, 12, 641-650 (2007).
12）T. Hamada and T. Iitaka, “The Chamomile Scheme: An Optimized Algorithm for N-body simulations on

Programmable Graphics Processing Units", arXiv:astro-ph/0703100
13）小野哲, "ソフトウェア開発にChatGPTは使えるのか？", 技術評論社, 2023.
14）三澤拓真, 福岡伊織, " フローチャッピー－理解するプログラミング－", 第34回 全国高等専門学校 プログラミングコンテスト自由部門応募作品, 2023.

受理日：2023年10月16日