2014 年 4 月 2 日 情報システム基盤学専攻修士論文中間発表会発表資料

大規模並列計算機システムの電力効率向上に関する研究

1353002 大坂 隼平
高性能コンピューティング学講座　本多研究室
主任指導教員： 本多弘樹

1　はじめに

今日のスーパーコンピュータでは、並列処理により複数のノードを同時に使用することで、高い性能を得ている。

しかし、この膨大な数のノードを用いた並列処理を行うシステムは現在電力の消費が激しく、例えばスーパーコンピュータの一つである理化学研究所に設置された京コンピュータの消費電力は 12.7MW と一般家庭での使用電力量のおよそ 3 万世帯分に相当する。よって、これ以上ノードの数を増やすことによる性能向上が難しいという問題がある。

この問題を解決する方法の一つとして、電力を考慮したプログラムの最適化がある。スーパーコンピュータのシステムのプログラムや、実行させるアプリケーションのプログラムを電力効率が向上するようなプログラムに書き換えることによって消費電力を削減し、結果的にスーパーコンピュータのノードを増やすことが可能となり、更なる性能向上を見込むことができる。

本研究では、スーパーコンピュータで実行させるアプリケーションのプログラムを対象とする。このアプリケーションのプログラムに対して解析を行い、電力効率を向上させるようにプログラムをチューニングするツールの開発を行う。

2　研究背景と目的

並列処理のプログラミングには MPI が広く用いられている。この MPI を用いた並列アプリケーションを解析するためのツールとして、Scalasca[1] や TAU[2] などが知られている。これら並列性能解析ツールは、大規模並列システムにおいて並列アプリケーションの解析を行う。解析結果から処理効率を改善させる手法を示すことで性能向上に貢献している。現在の HPC システムにおいて、アプリケーションの性能を詳細まで理解し最適化を行うためには、プログラマの労力が大きく並列性能解析ツールは必要不可欠である。

図 1 に並列プログラムの解析結果を表したグラフの一例として MPI プログラムを 16 プロセスで並列に実行した場合の結果を示す。縦軸はプロセス番号、横軸はプログラムを動かした際にかかった時間と内訳を示している。図

図 1: 解析を行った際の評価結果の一例
出典 [TAU-Tuning and Analysys, TAU uses Vampir, a commercial trace visualization tool fromPallas, GmbH.]

より、MPI_Alltoall 関数の実行に最も時間を費やしたことが分かる。

本研究では、この並列性能解析ツールを応用して並列プログラムの電力効率が最適になるようにプログラムを改良させるツールを開発する。これにより、電力効率の向上に貢献することを最終的な目標として研究を行う。

3　提案手法

並列アプリケーションの性能解析ツールは、図 1 の例のように、主にプロセスごとの処理時間の解析により、演算速度の向上させることを目的としたものが殆どで、現在に至るまで省電力化や電力効率向上を目的としたものは少ない。

そこで、オープンソースである並列性能解析ツールに電力消費状況を視覚的に把握できる機能を構築・実装する方法を提案する。並列プログラムのどの関数がどこのプロセスでどれだけ電力を消費しているか既存の処理時間の解析手法を応用し、解析できるようにすることで、プログラムのどの箇所を書き換えると電力効率が向上するか、電力最適化を行う際の指針となる。具体的には以下のような手順で開発を進める。

1. 対象となるベンチマークプログラムに電力解析を行うコードを手動で実装し結果を表示させる。

2. 並列性能解析ツール TAU に電力解析結果を表示する機能を組み込む。

3. 電力解析結果から、電力効率が最適になるような並列プログラムの検討を行う。その後、プログラムを書き換え再度解析・比較を行い、電力効率向上を図る。

4. 電力効率を自動的に最適化出来るスクリプトを開発する。

ステップ 1,2 に関しては、図 2 に示すように各関数の始めと終わりに電力解析を行うコードを実装することで、各関数の電力消費状況を解析する。また、プログラムを実行後にその解析結果をファイルとして出力させるようコードを改良する。ステップ 3,4 に関しては未だ具体的な提案手法は考案できていないため、実アプリケーションの解析結果を考慮しつつ、関連論文を読み知識を深め検討・実装していきたい。

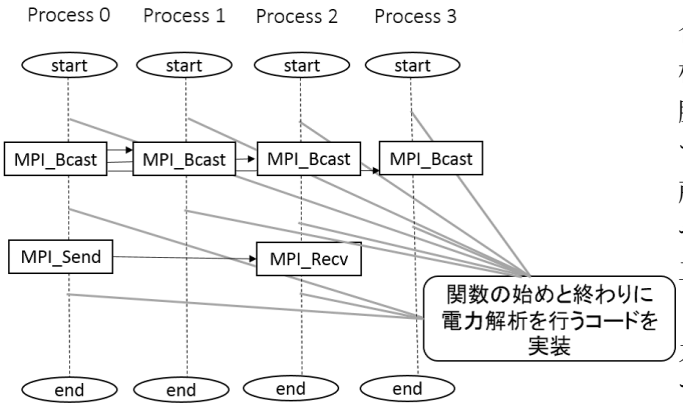

図 2: 電力消費状況の解析手法

4 進捗状況

実際に並列解析ツールを使用して解析を行うためにサーバの設定を行った。最大 32 プロセスまで並列化できる並列環境を構築した。また、MPI プログラムを実行させるために必要な MPICH2 と並列解析ツールの一つである TAU のインストールを行い、簡単な MPI プログラムを解析し、解析結果の検討を行った。

更に、より複雑なプログラムの解析を行うために姫野ベンチマーク [3] の解析を行った。姫野ベンチマークはポアッソン方程式解法をヤコビの反復法で解くプログラムである。図 3 に姫野ベンチマークの MPI プログラムをノード数 2、入力データサイズ m で実行した際のプログラムを TAU で解析した結果の一部を示す。これは、全体の実行時間に対する関数毎の実行時間の割合を表した図である。ヤコビ法を計算する関数で全体の 99.7%もの時間を費やしていることが分かる。現在はこのベンチマークに電力消費状況を表示させるコードを書き足し、まずは手動で解析結果の取得を試みている。

```
FUNCTION SUMMARY (total):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0       0.442     2:00.077           4          39   30019495 int main(int, char **) C
 99.7    1:58.065     1:59.733           4        1924   29933410 float jacobi(int) C
  1.1       0.684        1,374         962         962       1428 void sendp(int, int, int) C
  1.1           2        1,373         962        4810       1428 void sendp3() C
  1.0       1,224        1,224         962           0       1273 MPI_Waitall()
  0.2         294          294         966           0        305 MPI_Allreduce()
  0.2         271          271           2           0     135890 void initmt(int, int) C
  0.1         141          141        1924           0         74 MPI_Isend()
  0.0          57           57           4           0      14300 MPI_Init()
  0.0           8            8           4           0       2046 MPI_Finalize()
  0.0           3            3        1924           0          2 MPI_Irecv()
  0.0       0.047            2           2           6       1455 void initcomm(int, int, int) C
  0.0           2            2           2           0       1412 MPI_Cart_create()
  0.0           2            2           4           0        526 MPI_Barrier()
  0.0           1            1           2           0        715 MPI_Send()
  0.0       0.013        0.083           2           4         42 int initmax(int, int, int) C
  0.0       0.041        0.041           2           0         20 MPI_Type_commit()
  0.0       0.029        0.029           2           0         14 MPI_Type_vector()
  0.0       0.026        0.026           2           0         13 MPI_Cart_get()
  0.0       0.013        0.013           2           0          6 MPI_Cart_shift()
  0.0       0.002        0.002           3           0          1 double mflops(int, double, double) C
  0.0       0.001        0.001           4           0          0 MPI_Comm_rank()
  0.0       0.001        0.001           2           0          0 MPI_Comm_size()
  0.0       0.001        0.001           2           0          0 double fflop(int, int, int) C
```

図 3: ベンチマークの解析結果

5 課題と今後の予定

提案手法のステップ 3 においてプロセス数が少ない場合は、解析から判明したノード毎の消費電力量を確認・分析を行うことは容易であるが、スーパーコンピュータなど膨大な数のノードを用いたプログラムの解析結果を分析することは非常に困難である。よって、電力効率の悪い箇所を視覚的にわかりやすく表示させるために、実行時間や電力消費が少なく効果の小さい箇所を省略するなどの工夫が必要となる。

今後の予定としては、提案手法のステップ 2 である電力を解析することができる機能を実装させる方法について、具体的な提案が出来るよう検討を行う。その後、複雑なプログラムの解析を行いプログラムの電力消費状況が正しく解析できているかどうか実験・評価を行う。また、自動的に最適化を行うことができるシステムを構築・実装し、最適化手法の有効性について評価し改善を行う。

参考文献

[1] Markus Geimer, Felix Wolf, Brian J.N. Wylie Erika Abraham, Daniel Beeker, Brend Mohr:The SCALASCA performance toolset architecture: Concurrency and Computation:Practice & Experience - Scalable Tools for High-End Computing, Volume 22 Issue 6, April 2010, pp702-719.

[2] Sameer S. Shende, Allen D. Malony,THE TAU PARALLEL PERFORMANCE SYSTEM: The International Journal of High Performance Computing Applications,Volume 20, No.2,Summer 2006, pp.287311.

[3] 理化学研究所 HPC(High Performance Computing) グループ
http://accc.riken.jp/2145.htm