Vol.2021-DPS-186 No.15
Vol.2021-CSEC-92 No.15
2021/3/15

DPDKの超高速通信を活用した、緩和法解析の分散処理に関する研究

伴野 隼一[1]　矢吹 道郎[2]

概要：1992 年に上智大学 小石昇らによって、多元連立一次方程式の緩和法解析の分散処理に関する研究が行われた。この研究では UDP によるブロードキャストを用いており、パケットの遅延、消失した場合に置いても、計算が収束することを実証していた。当時の実験環境は 10Mbps のネットワークで、12〜40MB のメモリを搭載した計算機であった。現在、10Gbps のネットワーク、GB 単位のメモリになっている。さらに、汎用 OS でのパケット処理のインタラプト処理のオーバーヘッドを改善した Data Plane Development Kit(DPDK) という技術が誕生した。DPDK は、CPU コアを占有しポーリングで NIC にアクセスし、インタラプト処理がなく、標準的な OS のシステムコールよりも通信オーバーヘッドが少ない開発キットである。そこで、多元連立一次方程式の緩和法解析のネットワークを使用した分散処理に、DPDK を用いることで、速くなったネットワークを活用し、計算の高速化を目標にした。結果的に DPDK を用いた緩和法解析の分散処理アプリケーションは、UDP を用いた分散処理よりも高速化することに成功した。通信オーバーヘッドの影響が最も発生する一度に送る共有データの要素数が 1 個の時、DPDK は UDP に比べて、40.1%高速化された。このことから、DPDK により UDP による通信オーバーヘッドを改善し、緩和法解析の分散処理アプリケーションにおいて効果が確認できたと考えられる。

A distributed processing technique for relaxation method analysis using DPDK communication

1. 序論

1992 年に上智大学 小石昇らによって、多元連立一次方程式の緩和法解析の分散処理に関する研究 [6] が行われた(以下、小石らの研究と言う)。この研究では UDP によるブロードキャストを用いており、パケットの遅延、消失が発生した場合においても、計算が収束することを実証していた。

当時の実験環境は MTU1500byte の 10Mbps のネットワークで、12〜40MB のメモリを搭載した計算機であった。現在、10Gbps のネットワーク、GB 単位のメモリを搭載したコンピュータが普及してきた [4]。

汎用 OS でのパケット処理のインタラプト処理のオーバーヘッドを改善した Data Plane Development Kit(DPDK)[1] という技術が誕生した。DPDK は、CPU コアを占有しポーリングで NIC にアクセスし、インタラプト処理がなく、標準的な OS のシステムコールよりも通信オーバーヘッドが少ない開発キットである。

この DPDK を 10Gbps のネットワーク環境で、複数のコンピュータ間で計算データを大量に送受信する多元連立一次方程式の緩和法解析の分散処理に用いる。

[1] 明星大学情報学研究科情報学専攻
Hino, Tokyo 191–8506, Japan
[2] 明星大学情報学部情報学科
Hino, Tokyo 191–8506, Japan

2. 事前知識

2.1 DPDK

DPDK(Data Plane Development Kit) はユーザー空間上に構築されており、標準的な OS のシステムコールよりもオーバーヘッドが少なく、OS が提供できる以上のパフォーマンスを備えている [10]。DPDK ではカーネルバイパスを Polling Mode Driver(PMD) により行うため、ユーザーアプリケーションがネットワークデバイスを直接アクセスできる仕組みを提供している。ポーリングを行うために、CPU コアを送信と受信にそれぞれ占有する使い方をする。

DPDK はカーネルバイパスを行っているため、プロトコルスタックがない。また、DPDK が提供しているライブラリ関数は、イーサネットレベルの単純な単機能の関数に限られている。

2.2 多元連立一次方程式の緩和法解析

線形方程式を

$$Ax = b, A = L + D + U \tag{1}$$

ただし、

$$A = [a_{ij}],$$

$$L = [l_{ij}], l_{ij} = \begin{cases} a_{ij} & \texttt{if } (i > j) \\ 0 & \text{otherwise} \end{cases}$$

$$U = [u_{ij}], u_{ij} = \begin{cases} a_{ij} & \texttt{if } (i > j) \\ 0 & \text{otherwise} \end{cases}$$

$$D = [d_{ij}], d_{ij} = \begin{cases} a_{ij} & \texttt{if } (i > j) \\ 0 & \text{otherwise} \end{cases}$$

とするとき、ガウス・ヤコビ法における x についての反復式は

$$x^{(k+1)} = D^{-1}(b - Lx^{(k)} - Ux^{(k)}) \tag{2}$$

と表わされる。ガウス・ザイデル法における x についての反復式は、

$$x^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)}) \tag{3}$$

と表わされる。ガウス・ザイデル法に注目すると、(3) 式の反復式が有効になるのは x_i の添字の小さい方から順に更新値を求める場合のみである。一方その逆順で反復式を求めると、(2) 式のガウス・ヤコビ法と等価になる。それ以外の順序で (3) 式の反復式を計算する時、$Lx^{(k+1)}$ の項で使用する $x^{(k+1)}$ は $x_i^{(k+1)}$ と $x_j^{(k)}$ の混在したものとなる。この場合の収束速度は、ガウス・ヤコビ法より加速されており、ガウス・ザイデル法より減速されることになる。

3. 関連研究について

3.1 小石らの研究のメカニズム

3.1.1 1 台での緩和法解析

(1) 式を空間分割すると図 1 になる。例えば、図 1 の x_2 を求める場合、a_2 と b_2、x_1〜x_n を使い計算する。この x_2 の値が更新されることで、他の x の計算結果も変わってくることになる。収束するまで計算を繰り返し行うことから、反復計算という。収束は x の値の更新前と更新後の差を見ることで判定することができる。

3.1.2 緩和法解析の分散処理

分散処理を行うホストについて、収束判定を行うホストを収束判定ホスト、反復計算を行うホストを反復計算ホストと呼ぶ。各ホストの接続及び保持しているデータを図 2 に示す。分散処理では x のデータを全てのホストが保有かつ共有していることから、この x のデータを共有データと呼んでいる。反復計算ホストは更新された共有データの一部を UDP のブロードキャスト送信することで他の全てのホストの共有データを更新する。

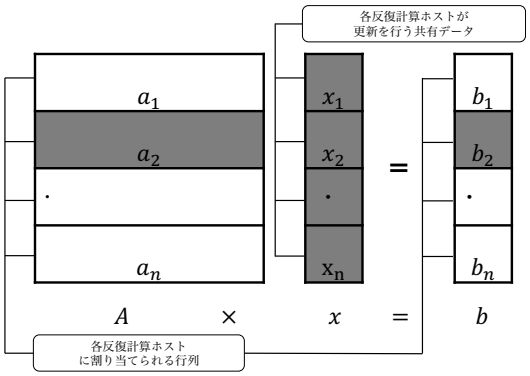

図 1 $A \times x = b$ の空間分割

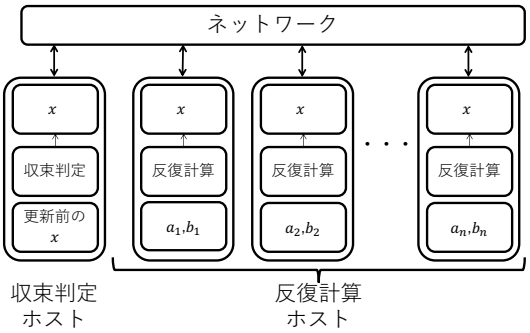

図 2 収束判定ホストと反復計算ホストの接続及び保持データ

図 1 の x である共有データは、全てのホストで必要になる。収束判定ホストでは収束判定のために、x の更新前と更新後のデータを保有する必要がある。反復計算ホストでは、図 1 の担当する A 及び b 範囲のみ必要となる。分散処理により、A 及び b に必要なメモリは反復計算ホスト 1 台あたりに必要なメモリ量が削減できることで、1 台のみではオンメモリで作業できない計算も行えるようになる [7]。

各ホストの役割を以下に示す。

収束判定ホスト 反復計算ホストからの更新値を元に、収束判定を行う。収束したと判定すると、反復計算ホストに対して計算終了を伝える。

反復計算ホスト 図 1 の A 及び b の担当するデータと、その時点で最新の x を使い、緩和法解析の反復計算を行う。一定の計算が進むと、更新値を全てのホストに対してブロードキャストで送信する。他の反復計算ホストが担当する範囲の x の更新値を受信し、更新を行う。更新値の受信に失敗しても、反復計算を継続して行うことができる。

収束したと判定され、反復計算が終了すると、収束判定ホストは検算を行う。検算の結果、収束していなければ、反復計算を再開し、再び収束するまで計算を反復する。

更新値をブロードキャストする際の通信オーバーヘッドを考慮して、x の更新値をまとめて送信する必要がある [8]。一つのパケットで送信する更新値の個数を要素数と呼ぶ。要素数は、1500byte の MTU から UDP ヘッダ及び分散処理に必要なデータを除くと、181 個以下となる。要素数を多くすることは x の更新が遅れることで結果的に古いデータを使うことにより、収束が遅れることにつながる。反対に要素数を少なくすることで、更新値が速く得られるが、パケット数の増加により、通信オーバーヘッドの影響を受けることになる。

3.2 小石らの研究結果

この研究では緩和法解析の分散処理で、計算データの更新に TCP ではなく、UDP によるブロードキャストを用いることで通信オーバーヘッドを抑えていた。さらに計算データの更新による通信オーバーヘッドを削減するために、要素数を MTU の限界まで溜めて送信を行っていた。緩和法解析の分散処理で、データの更新に失敗または、遅れや消失が生じても致命的な影響を与えることがなく処理を続けられることが実証された。反復計算ホストを増やすことで計算時間が短縮される台数効果が得られていた。

4. DPDK を用いた緩和法解析の分散処理アプリケーション

4.1 DPDK 用関数

プロトコルスタックがなく、DPDK が提供するライブラリ関数が単純な単機能の関数の集合であることにより、アプリケーションの作成が煩雑になる [3]。LightWeight TCP/IP(LwIP) と DPDK を統合し、TCP/IP を DPDK で使えるようにする研究 [11] があるが、まだ一般的に使えるものではない。これらにより、緩和法解析の分散処理用の DPDK を用いた通信ライブラリを作成してから、分散処理アプリケーションの作成を行う。

DPDK を用いる際の初期化やメモリに関する関数の作成を行い、UDP 通信を行うシステムコールの sendto 関数、recvfromn 関数に対応するイーサネットレベルの関数を作成した。通信オーバーヘッドを極力削減することを最優先にしているため、Ether ヘッダと分散処理を行う際に必要なヘッダに限り、L2 レベルの通信を行うデータグラムを作成した。

DPDK の初期化に関する関数を以下に示す。

dpdk_setup 関数　DPDK の初期化の設定をする
make_mempool 関数　メモリプールの作成をする
config_port 関数　NIC の設定及び起動を行う
change_mtu 関数　MTU の変更を行う

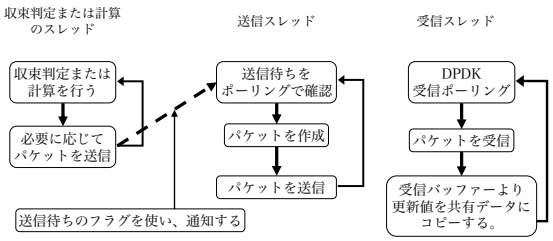

図 3　DPDK の共有データに関する流れについて

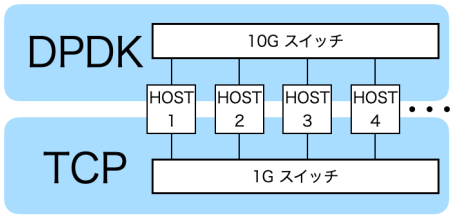

図 4　ネットワーク構成

link_up_wait 関数　NIC のリンクが up になるまで監視する

通信に関する関数を以下に示す。

dpdk_recvfrom 関数　DPDK で受信を行う (複数パケットの同時受信に対応している)
dpdk_sendto 関数　DPDK で送信を行う
dpdk_set_info 関数　ヘッダを作成するための情報を作る

4.2 スレッドによる受信

DPDK はユーザ空間で動いているため、小石らの研究で利用していた OS によるパケット受信のシグナルを利用できない [9]。また、DPDK は NIC にポーリングでアクセスすることで高速化を実現している [12]。そのため、DPDK での共有データの受信には、スレッドによる受信とすることとした。

DPDK を用いた分散処理アプリケーションでの共有データに関する流れを図 3 に示す。共有データは DPDK を使い、ブロードキャストされている。

4.3 ネットワークの構成

SSH による実験機の操作等のために信頼性のある通信が必要なため、TCP/IP プロトコルスタックを持ったネットワークと DPDK のためのネットワークの二重化を行った。ネットワークの構成を図 4 に示す。

4.4 アプリケーション

アプリケーションの流れについて説明する。小石らの UDP を用いた分散処理と基本は同じである。本研究では、ブロードキャストに UDP ではなく DPDK を使い、通信オーバーヘッドを削減することを目的としているためである。反復計算ホストは全てのホストに対して、計算した更新値を DPDK を用いてブロードキャストする。全てのホ

表 1 「16GB のメモリを搭載した実験機での行列サイズと計算時間の関係」

一辺の行列サイズ	計算時間 (sec)
10000	2.14
20000	8.58
30000	19.35
40000	2984.46
50000	malloc のエラー

ストは DPDK による受信ポーリングが行われているため、NIC にパケットが到着すると DPDK による高速なパケット処理が行われる。

5. 実験と考察

5.1 実験機器

実験機は、Ubuntu 14.04、DPDK 2.1.0 を用いた。収束判定ホスト 1 台には 64GB メモリと Core i7-7700、反復計算ホスト 7 台には 16GB のメモリと Core i5-9600K と i5-8400 が搭載されている。

5.2 1 台の計測結果

分散処理の実験と比較・検証を行うために、先に 1 台での緩和法解析の実験を行った。16GB のメモリを搭載した実験機では一辺の行列サイズを 40000 にすると、メモリが足らないためスワップが発生し大幅に遅くなった。表 1 の 1 辺の行列サイズ 30000 までは、行列サイズの大きさに比例する計算量に比例して計算時間が増加している。

1 台での実験を全ての反復計算ホストを対象に行った。その結果、反復計算ホストのうち、4 台に比べ、3 台は約 9%遅い結果となった。

5.3 UDP を用いた分散処理

5.3.1 UDP を用いた分散処理の目的

DPDK を用いた分散処理の効果を計るために、比較対象として小石らの方法で、MTU9000byte の 10Gbps のネットワークで UDP を用いた分散処理の実験を行った。本実験では、以下の

- 1 つのデータグラムで送る共有データの要素数
- 反復計算ホストの台数
- 行列サイズ

の観点から、比較を行った。

5.3.2 1 つのデータグラムで送る共有データの要素数

小石らの研究では MTU の都合で 1 つのデータグラムで送る共有データの要素数を 181 個以下としていた。本研究では MTU が 9000byte になったため、要素数を 1117 個以下まで送ることができる。UDP を用いた分散処理アプリケーションで要素数と計算時間の関係を図 5 に示す。

図 5 から、要素数 100 以上で、収束時間が増加していることがわかる。要素数が多くなり、更新が遅くなり、古いデータを使うことで収束に遅れが生じる。

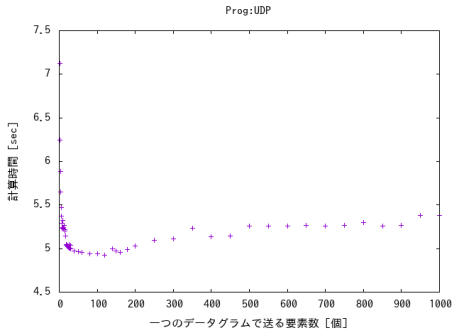

図 5 「要素数と計算時間との関係」(UDP、反復計算 4、一辺の行列サイズ 30000、5 回実行した平均値)

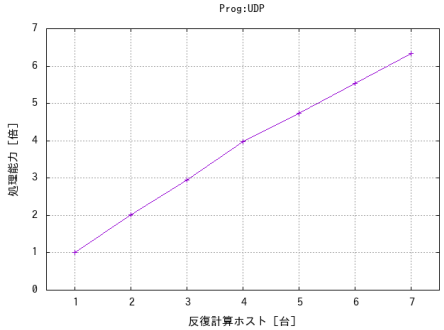

図 6 「反復計算ホストの台数効果による高速化」(UDP、一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値)

要素数を少なくすると送信間隔が短くなることで更新が速くなり、収束が速くなることが期待される。しかしながら、要素数を減らしていくと、計算時間が大幅に長くなっていることが図 5 からわかる。これは要素数を少なくしたことで、頻繁に送信を行うことになり、パケット数が増加し、通信オーバーヘッドの影響を受けていると考える。UDP では、共有データの更新が頻繁に起こることによる計算の加速を上回る通信オーバーヘッドが発生したことで、計算時間が大幅に長くなることがわかる。

5.3.3 反復計算ホストの台数効果

小石らの研究においても、良好な台数効果が得られていた。UDP を用いた分散処理アプリケーションで反復計算ホストの台数効果を、反復計算ホスト 1 台の時の計算時間を基準とした比として、図 6 に示す。図 6 から、反復計算ホストが 1〜4 台目は実験機の構成が共通のため、比例している。反復計算ホストが 5〜7 台目は、1〜4 台目に比べて実験機のスペックが低いため高速化が緩やかになった。上記より、実験機のスペックが異なる場合でも台数効果により、計算が高速化されていることがわかる。

5.3.4 行列サイズ

UDP を用いた分散処理アプリケーションで行列サイズと計算時間の関係を対数スケールで描いたグラフを図 7 に示す。図 7 から、1 台ではスワップの発生があった 1 辺の行列サイズ 40000 の計算が分散処理により行えている [5]。

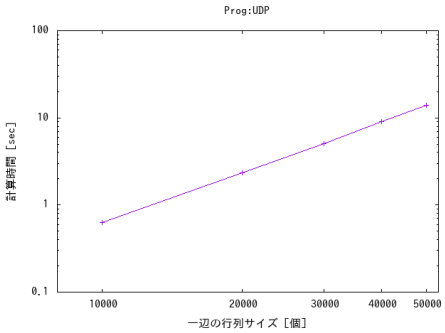

図 7 「行列サイズと計算時間の関係」(UDP、対数スケール表示、反復計算 4、要素数 20 個、5 回実行した平均値)

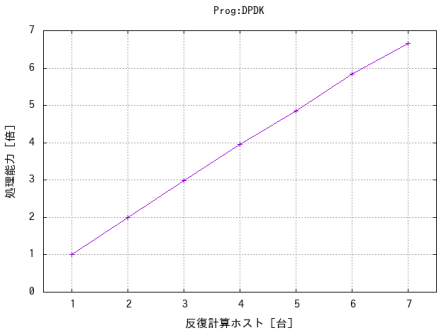

図 9 「反復計算ホストの台数効果による高速化」(DPDK、一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値)

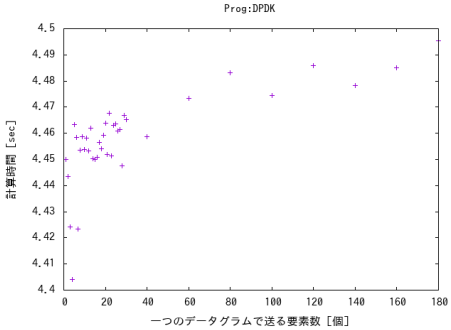

図 8 「要素数と計算時間の関係」(DPDK、反復計算 4、一辺の行列サイズ 30000、5 回実行した平均値)

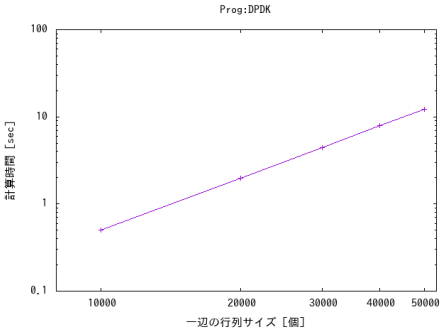

図 10 「行列サイズと計算時間の関係」(対数スケール表示、DPDK、反復計算 4、要素数 20 個、5 回実行した平均値)

図 7 のグラフが直線であることから、分散処理による他のオーバーヘッド影響を受けていないことが分かる。

5.4 DPDK を用いた分散処理

5.4.1 1 つのデータグラムで送る共有データの要素数

UDP と同様に DPDK で実験を行った。DPDK を用いた場合の要素数と計算時間の関係を図 8 に示す。DPDK による高速化に焦点を当てるため、パケット数が多くなる要素数 180 以下のみを示している。要素数 180 以上では緩やかに収束に時間がかかるようになった。要素数が 30 個以下では、計算時間にバラツキがある。DPDK の超高速通信により、一度に送る共有データの要素数を少なくし、共有データの更新による計算の加速で、速く収束することを期待していたが、安定的な結果が得られなかった。しかしながら、UDP を用いた場合のように極端に計算時間が遅くなることはなかった。

5.4.2 反復計算ホストの台数効果

DPDK を用いた場合の反復計算ホストの台数効果を、反復計算ホスト 1 台の時の計算時間を基準とした比として、図 9 に示す。UDP の場合と同様に、反復計算ホストが 1〜4 台目は実験機の構成が共通のため比例し、反復計算ホストが 5〜7 台目は実験機のスペックが低いため高速化が緩やかになった。

5.4.3 行列サイズ

一辺の行列サイズの違いによる計算時間について検証した。 DPDK を用いた分散処理アプリケーションで行列サイズを変えたときの計算時間の関係を対数スケールで描いたグラフを図 10 に示す。図 10 のグラフが、直線になることから、分散処理による他のオーバーヘッド影響を受けていないことが分かる。

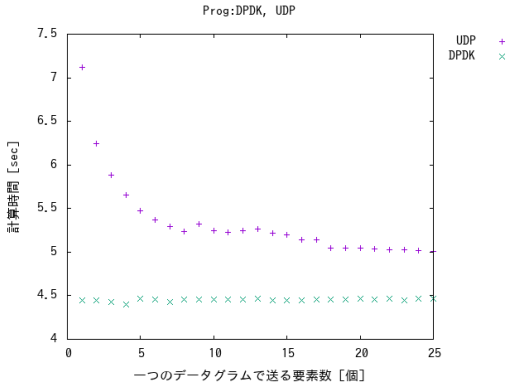

図 11 「DPDK と UDP の要素数と計算時間の関係」(一辺の行列サイズ 30000、反復計算 4、5 回実行した平均値)

6. 比較と評価

6.1 一つのデータグラムで送る要素数による比較

DPDK を用いた場合と、UDP を用いた場合で、要素数を変えた計測結果を図 11 に示す。DPDK を用いた場合に比べて UDP を用いた場合は、5.3.2 に示したように通信オーバーヘッドが上回り、その計算時間の差が顕著になって行

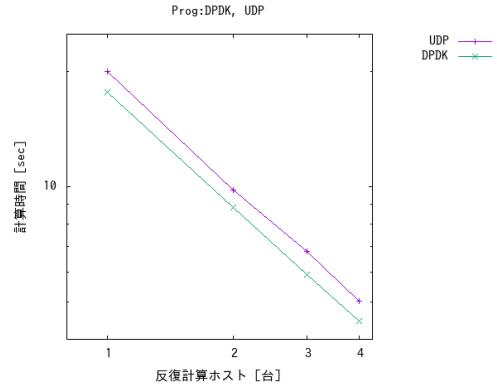

図 12 「DPDK と UDP の反復計算ホスト台数と計算時間の関係」（対数スケール、一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値）

く。DPDK を用いた場合に要素数の減少が UDP の場合のような計算時間の増加に結びつかなかったものの、期待されるような計算時間の減少には至らなかった。通信オーバーヘッドの増加と速い更新による加速が同程度となっていたためと考えられるが、更なる検証が必要である。

DPDK を用いた場合では UDP を用いた場合に比べて、要素数がいかなる場合でも速く計算が終わった。一辺の行列サイズを 30000 で反復計算ホストを 4 台にした場合の UDP で最も速い時の要素数と、DPDK で最も速い時の要素数を比較した。UDP を用いた場合は要素数が 120 個の時に、4.92389 秒であった。DPDK を用いた場合は要素数 4 個の時に、4.40406 秒であった。DPDK は UDP に比べて、10.5%高速化された。通信オーバーヘッドの影響が最も発生する要素数が 1 個の時は、UDP は 7.12187 秒、DPDK は 4.44995 秒であり、DPDK は UDP に比べて、37.5%高速化された。要素数が 1 個の時に、一辺の行列サイズが 50000 で反復計算ホストを 7 台にすると、UDP は 12.21012 秒、DPDK は 7.31087 秒であり、DPDK は UDP に比べて、40.1%高速化された。 これらから、DPDK により UDP による通信オーバーヘッドを改善し、緩和法解析の分散処理アプリケーションで通信による影響の多くを削減することが出来、計算の加速にも寄与することができたと言える。

6.2 反復計算ホストの台数効果による比較

DPDK を用いた場合と、UDP を用いた場合で、反復計算ホストの台数を変えた計測結果を対数スケールで描いたグラフを図 12 に示す。図 12 から、最小二乗法を用いて、台数効果によるグラフの傾きを求めた。この傾きが、−1 になった時、オーバーヘッドが全くない理想となる。UDP を用いた場合のグラフの傾きは-0.985 であった。 DPDK を用いた場合のグラフの傾きは-0.993 であった。 このことから、DPDK はより理想的な台数効果が得られていることがわかる。反復計算ホストの台数が増加すると、それに伴うパケット量増加のために台数効果を打ち消すことが考えられる。しかし、本研究では、資材の関係から、台数の限界を検証するには至らなかった。

表 2 「DPDK と UDP の反復計算ホストでの反復計算の周回回数」（一辺の行列サイズ 50000、5 回実行した平均値）

一つのデータグラムで送る共有データの要素数	反復計算ホストの台数	DPDK の周回回数 (回)	UDP の周回回数 (回)
1	4	21.0	22.5
1	7	21.4	23.4
20	4	21.3	23.0
20	7	21.4	23.2

6.3 DPDK による高速化

DPDK を用いた場合と UDP を用いた場合で、反復計算ホストでの反復計算の周回回数の確認を行った。反復計算ホストでの反復計算の周回回数とは、反復計算ホストが担当する範囲の計算を何周行ったかである。反復計算の周回回数が増加すると計算量も増加する。DPDK を用いた場合と UDP を用いた場合で、反復計算ホストでの反復計算の周回回数を表 2 に示す。UDP では通信オーバーヘッドにより更新が遅れることで、計算量が増加していることが周回回数からも検証された。

7. まとめ

本研究では、多元連立一次方程式の緩和法解析の分散処理アプリケーションに DPDK を用いることで、通信オーバーヘッドの削減による計算の高速化を行った。

前研究では緩和法解析の分散処理で行われる反復計算の更新値を UDP によるブロードキャストを用いて、データの共有を行っていた。本研究では UDP を用いることによるインタラプトによるオーバーヘッドを削減できる DPDK を用いることで、通信オーバーヘッドを削減し計算を高速化することを目標とした。

DPDK はプロトコルスタックがなく、DPDK が提供するライブラリ関数が単純な単機能の関数の集合であることにより、アプリケーションの作成が煩雑になる。緩和法解析の分散処理用の DPDK を用いた通信ライブラリを作成してから、分散処理アプリケーションの作成を行った。

上智大学で行われた UDP を用いた緩和法解析の分散処理アプリケーションと同等の DPDK を用いた緩和法解析の分散処理アプリケーションを作成して実験を行った。DPDK を用いた緩和法解析の分散処理アプリケーションでは、通信オーバーヘッドを極力削減することを最優先にしている。そのため、Ether ヘッダと分散処理を行う際に必要なヘッダに限り、L2 レベルの通信を行うデータグラムを作成した。

UDP を用いた場合と DPDK を用いた場合とで、同じ条件での実験を行った。この実験では、以下の

- 1 つのデータグラムで送る要素数
- 反復計算ホストの台数
- 行列サイズ

の観点から、比較を行った。

DPDK を用いた場合では、UDP を用いた場合よりも高速化することに成功した。DPDK では UDP を用いた場合に比べて、要素数がいずれの場合でも速く計算が終わった。例えば、一辺の行列サイズを 50000 で反復計算ホストを 7 台にした場合に、通信オーバーヘッドの影響が最も発生する一度に送る共有データの要素数が 1 個の時は、UDP は 12.21012 秒、DPDK は 7.31087 秒であり、DPDK は UDP に比べて、40.1%高速化された。 上記のことから、DPDK により UDP による通信オーバーヘッドを改善し、緩和法解析の分散処理アプリケーションで通信による影響の多くを削減することが出来、計算の加速にも寄与することができた。

今後の課題としては、

- 実験機の台数を増やし、パケット数が増えた環境での通信オーバーヘッドの影響
- 本研究では反復計算ホストで計算に使用した CPU コアは 1 コアである。反復計算ホストでの計算を高速化させるために、反復計算ホストで計算に使用する CPU コアを増やした場合

について調べるべきである。

謝辞

本研究をすすめるにあたり、御指導御鞭撻を賜った矢吹道郎准教授に深く感謝の意を表します。

参考文献

[1] DPDK Project: Data Plane Development Kit, DPDK Project (online), available from ⟨https://www.dpdk.org⟩ (accessed 2021-02-15).
[2] Hans, W.: DPDK-based Implementation of Application-tailored Networks on End User Nodes, *2014 International Conference and Workshop on the Network of the Future (NOF)*, IEEE, pp. 1–5 (2014).
[3] 板垣寛久：DPDK による 10Gb ネットワーク通信高速化に関する研究, 明星大学情報学部情報学科 (2015).
[4] 伊藤　大：10GbNIC を用いたルーティングにおける通信速度に関する研究, 明星大学情報学部情報学科 (2014).
[5] 吉敷一将，田村是慶，松谷宏紀：パラメータサーバのための FPGA および DPDK による通信高効率化手法, 電子情報通信学会技術研究報告，Vol. 117, No. 459, pp. 99–104 (2017).
[6] 小石　昇：分散処理におけるブロードキャストを利用した共有データの参照および更新に関する研究，修士論文，上智大学理工学研究科・電気電子工学専攻 (1992).
[7] 小石昇，千種康民，矢吹道郎，孫堅：ブロードキャストを利用した大規模行列の緩和法解析の分散処理，全国大会講演論文集第 45 回，pp. 47–48 (1992).
[8] 久保真一郎：分散処理のためのマルチキャストを利用した共有データの参照及び更新に関する研究, 明星大学情報学部電子情報学科 (1998).
[9] 前田宗則，田邨優人，松尾勇気，佐藤充，中島耕太：カーネルバイパスと軽量スレッドを用いた高速通信方式の実装，電子情報通信学会技術研究報告，Vol. 116, No. 177, pp. 187–191 (2016).
[10] ペングカザベッキヤソフィー，中村幸平，吉敷一将，松谷宏紀：変化点検出のネットワークによる最適化の性能評価，電子情報通信学会技術研究報告，Vol. 2017, No. 36, pp. 1–6 (2017).
[11] Rajesh, R., Ramia, K. B. and Kulkarni, M.: Integration of LwIP stack over Intel DPDK for high throughput packet delivery to applications, *2014 Fifth International Symposium on Electronic System Design*, IEEE, pp. 130–134 (2014).
[12] 澤崎文彦，堀米紀貴，高田直樹：データバッファあふれを考慮した SPP 設計の考察：NetroSphere 構想実現に向けて，電子情報通信学会技術研究報告， Vol. 117, No. 459, pp. 23–25 (2018).