修士論文

DPDKの超高速通信を活用した、計算の分散処理アプリケーション作成に関する研究

2020年度

伴野 隼一

明星大学大学院
情報学研究科情報学専攻
19MJ-002

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 (以下、小石らの研究という) が行われた。この研究での分散処理では、一ヶ所のメモリのデータを利用するのではなく、個々のコンピュータのメモリのデータを更新する仕組みを採用していた。データの更新には 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 10Mbps のネットワークであったが、現在では 10Gbps のネットワークになっている。大幅に高速化されたネットワークでは、パケット処理による通信オーバーヘッドが大きくなる。緩和法解析の分散処理においても、通信オーバーヘッドの削減は計算速度の向上に重要であると考えた。汎用 OS でパケット処理のインタラプト処理のオーバーヘッドを、CPU コアを占有しポーリングで NIC にアクセスし、インタラプト処理行わないことで、改善した Data Plane Development Kit(DPDK) が誕生した。この DPDK を現在の 10Gbps 環境のネットワークで、小石昇らの多元連立一次方程式の緩和法解析の分散処理の改良に用いることで、計算の高速化を目標とした。

アプリケーションの流れは、小石らの UDP を用いた分散処理と基本は同じである。ただし、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。そのため、DPDK での更新値の受信には、スレッドによる受信とすることとした。反復計算を行うホストを反復計算ホスト、収束判定を行うホストを収束判定ホストと呼ぶ。全てのホストは 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。

小石らの UDP を用いたアプリケーション及び DPDK を用いたアプリケーションを、10Gbps 環境のネットワークで比較実験を行った。多元連立一次方程式の緩和法解析の分散処理では、更新値が共有する必要があり、更新が頻繁に行えるとより速く収束することができる。ただし、更新を頻繁に行うとパケット数が多くなり、通信オーバーヘッドの影響が大きくなる。DPDK を用いた場合に比べて UDP を用いた場合は、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く。DPDK を用いた場合に頻繁な更新が 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。

反復計算ホストの台数を変えた実験で対数スケールのグラフの傾きを最小二乗法を用いて求めた。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であり、UDP を用いた場合は-0.985、DPDK を用いた場合は-0.993 であり、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。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。

本研究では DPDK の超高速通信を用いた計算の分散処理アプリケーションを目的として、多元連立一次方程式の分散処理アプリケーションを題材に、計算の高

速化を目標とした。今回の分散処理の結果から緩和法解析のように収束問題であり、非常に更新の多い計算問題においては、非常に DPDK の超高速通信が有効に活用できた。今後その他の収束問題により広く活用できる枠組みを考えていきたい。また、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について調べるべきである。

目 次

第1章　序論

マルチコアプロセッサ上の汎用OSでのパケット処理は、ネットワークプロセッサ等の専用のパケット処理のパフォーマンスに比べてパフォーマンスが大幅に低い。一般的なパケット処理のオーバーヘッドが大きいことにより、ハードウェアの能力と比較してシステム性能を低下させている。Data Plane Development Kit(DPDK)[1]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。

DPDKを使ったアプリケーションを作成するにあたり、課題となることがある。それは、DPDKがTCP/IPのようなプロトコルスタックを提供していないことである。また、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[2]があるが、まだ一般的に使えるものではない。そのため、Etherフレームを直接扱わなくてはならないため、OSのシステムコールやライブラリ関数を使った通信プログラミングが行えない。

本研究ではDPDKの超高速通信を利用した、複数のコンピュータ間で大量のデータを送受信する計算の分散処理の高速化を目標とした。今回は、このような計算の中から、多元連立一次方程式の緩和法解析を題材として研究を行った。回路方程式を解くのに使われる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過去に行われている。1992年に上智大学小石らにより、多元連立一次方程式の緩和法解析の分散処理における共有データの参照および更新についての研究[3]が行なわれた。この研究ではガウス・ザイデル法という反復計算により近似を求めて行く計算を分散処理で行った。そこで、多元連立一次方程式の緩和法解析の分散処理ではデータは常に最新である必要はないメカニズムが提案された。1台で行っていたガウス・ザイデル法の反復計算を分散して行う場合に、一ヶ所のメモリのデータを利用するのではなく、個々のコンピュータのメモリのデータを更新しながら行えることが分かっている。この分散処理では、計算の収束を判定するホストと、反復計算を行うホストがある。一定量の計算が終わると反復計算を行うホストは、更新値を他のホストと共有する。この共有にUDPのブロードキャストで送信することで、擬似的にメモリの共有を実現している。UDPを利用しているためパケットを落とすことがあるが、収束が少し遅れる影響のみであることが実証された。

1992年に上智大学小石らにより行われた多元連立一次方程式の緩和法解析の分散処理は10Mbps環境下で行われていた。現在では、10Gbpsに対応したNICが普及してきている。大量のパケットを送受信する計算に加えて、単位時間あたりに

届くパケット数が増加していることから、高速なパケット処理が求められる。そのため、本研究ではDPDKを用いることで通信オーバーヘッドを削減し、高速化できないかを検討した。

第2章　DPDK

DPDK とは Data Plane Development Kit という高速なパケット処理が行える開発キットである。DPDK 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 [5]。通常の通信を行うアプリケーションは、カーネル環境で動作する。これに対して、DPDK を用いたアプリケーションは、カーネルバイパスにて実行する。カーネルバイパスを行うため、DPDK は NIC などのアーキテクチャ/プラットフォーム固有の環境抽象レイヤ (EAL) の作成を通じて、特定環境用のライブラリを作成している。環境抽象レイヤ (EAL) は、アプリケーションやライブラリから詳細を隠し、ユーザー空間とカーネル空間の間のインターフェースとして機能する。EAL はハードウェアやメモリースペースなどの低レベルのリソースへのアクセスを担い、DPDK ライブラリを使用したアプリケーション開発に必要なレイヤである。

DPDK はマルチコア CPU を利用することが前提にある。これは、CPU コアをパケットの送受信に占有させて、ポーリングにより NIC へのアクセスを行うことで高速化を実現しているためである。DPDK は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、Polling Mode Driver(PMD) によるカーネルバイパスとして提供している。

また、DPDK は hugepage という大きい単位でメモリを扱っており、ダイレクトにメモリにアクセスできることから高速に動作できる。この hugepage を mempool という対象のアプリケーションが使えるメモリ空間を確保する。mempool から必要なメモリをリスト形式の mbuf というバッファーとして取り出し利用している。

DPDK にはプロトコルスタックがなく、プリミティブな Ether フレームを扱うライブラリとなっている。また、NIC やメモリを高速に扱うための低レイヤ向け関数という構成になっている。このことに加えて、カーネルバイパスにより実行される DPDK では OS が提供していた機能に使えないものがある。このような手間がかかっても DPDK を用いたアプリケーションが作成されるのは、速度至上主義な考え方が元になっているためである。

ユーザがカーネル環境で使う UDP レベルの通信と、DPDK のようにプロトコルスタックのない Ether レベルの通信の通信速度の比較を行った。パケットサイズを変えたカーネル環境下での UDP を用いた転送速度を表 2.1 に示す。パケットサイズを変えた DPDK 環境下での転送速度を表 2.2 に示す。DPDK はプロトコルスタックがなく、PMD の効果もあることにより、いずれのパケットサイズの通信でも、カーネル環境下での UDP 通信に比べ、DPDK 環境下の方が速い結果となった。

表 2.1: カーネル環境下での転送速度

パケットサイズ (byte)	転送速度 (Mbps)
200	850
300	1122
1000	3437
1400	4940
8900	7059

表 2.2: DPDK 環境下での転送速度

パケットサイズ (byte)	転送速度 (Mbps)
200	2857
300	4152
1000	9184
1400	9477
9000	9511

第3章　関連研究と新技術を用いた改良

3.1　1992年の研究で発案されたメカニズム

1992年、以下のような条件の下に上智大学 小石らにより分散処理における共有データの参照および更新についての研究 [3] が行なわれた。

- 分散処理が、標準的なLAN環境であるイーサネット上で行なわれる。
- 各分散プロセスが異なるホスト上に存在し、共有データを使用する。
- 共有データへの参照および更新が頻繁に生じる。

このメカニズムに対するアプリケーションの例として緩和法解析を利用した場合の動作を説明する。

3.1.1　共有データの更新

分散処理における共有データの参照及び更新のメカニズムには次のような種類がある。

- 集中型共有データ
  この場合、分散プロセスから共有データへアクセスがあった時にネットワークを介せねばならないので、データアクセスが頻繁におこるようなアプリケーションの場合には全体の処理時間に大きな影響を与える可能性がある。
- 分散型共有データ
  分散型共有データは各ホストそれぞれに共有データが存在し各分散プロセスは共有データをアクセスするためにローカルにある共有データにアクセスする。共有データに、ある分散プロセスが更新を加えた場合、他の分散プロセスが保持している共有データにも更新をしなければならない。よって他のプロセスへ更新値を送信する事が必要になってくる。

前研究では共有データの更新及び参照が頻繁に起こるようなアプリケーションを対象にしているので、共有データにアクセスする際、ローカルにあった方がよい。よって、前研究では各ホストそれぞれが共有データを保持する分散型共有データを採用している。

異るホストに共有データが存在する場合、以下の事を考えなければならない。

- 共有データの一貫性
- 更新値を送信するための通信に使用するプロトコル
- 更新の際、どのくらいの量を更新するか。

以上の事をふまえ、前研究で共有データの更新及び参照のメカニズムが作成された。以下にこの 3 つの点について詳しく述べる。

共有データの一貫性

以前、上智大学で行なわれた研究では共有データへのアクセスが頻繁に起こるようなアプリケーションを対象としていたので、各分散プロセス間全てに共有データを配置する分散型共有データの形をとるものである。この場合、あるホストで共有データの更新が行なわれた場合はその更新値を他のホストに送信しなければならない。しかし、各ホスト上の共有データ間で内容に時間的なずれが生じる。アプリケーションがデータアクセスの際に最新のデータを必要とする場合には、それを保証するメカニズムが必要となるが、前研究ではそのようなアプリケーションは対象としておらず、参照する共有データが他の分散プロセスによって更新中か否かに関わらず、参照した時点のデータを使用できるようなアプリケーションを対象した。よって、この研究で作成されたメカニズムは各ホスト上の共有データの内容を最新に保つようなメカニズムではない [6]。

更新値の通信に使用するプロトコル

プロセス間通信を行なう際、通信に使用するプロトコルを選択する必要がある。前節で説明したような環境では、通信プロトコルとして TCP または UDP を用いる事ができる。共有データの更新があるホストで行なわれた場合、その更新値は他の共有データを使用している全てのホストへ送信される必要がある。この際 TCP を用いて送信するとなると、送信するホストと受信するホストと TCP コネクションを確立する必要が出て来る。また、更新値を受信すべきホストと同じ数だけ同じ内容がネットワーク上に転送される事になる。一方 UDP を用いた場合はイーサネットの機能を利用したブロードキャスト通信を使用する事ができ、同一のネットワーク上のホストであればホストの数に関わらず、一度の送信で同じデータを全てのホストへ送信する事ができる。共有データの更新が頻繁に起こる事を考慮すると、TCP を用いた場合ネットワークに大きな負荷を与え、更に通信の実現のためのオーバヘッドも非常に大きなものとなってしまう。よって前研究では UDP のブロードキャスト通信を利用して共有データの更新を行なっている。

通信プロトコルとして UDP を使用した場合、UDP では通信の信頼性を保証しておらず、データが相手に届かなかったり、送信した順番と異なる順番で受信される可能性がある。そこで、通信の信頼性について考えなければならないが、こ

の研究の対象となる緩和方解析のアプリケーションは、分散プロセスが参照を行なった時点での共有データを使用して処理を進める事が出来るようなものであり、データの更新に失敗したとしても、それは単に更新の遅れとみなす事ができ、そのまま処理を続行することが可能である。

送信データサイズ

緩和法解析の分散処理において、本来は共有データの更新が速いことが望まれる。しかし、速い更新を実現するために、頻繁に共有データの送受信を行うと通信オーバーヘッドが大きくなってしまう。すなわち、データ更新の遅れと、通信オーバーヘッドはトレードオフの関係になる。前研究での CPU 速度とネットワーク環境では、データ更新の遅れに対して通信オーバーヘッドが大きかった。そのため、極力データを貯めることで通信オーバーヘッドを抑えることが適していた。UDP のブロードキャスト通信は、イーサネットのブロードキャスト機能を利用しており、イーサネットにおける最大送信データサイズ以上のデータを送信する事ができない。したがって送信データの最大サイズは、イーサネットの最大送信サイズである 1500byte から UDP ヘッダの部分を除いた 1472byte であった。

3.1.2 非同期受信

緩和法解析の各分散プロセスでは共有データの更新が頻繁に行なわれるため、更新値はネットワークを介して頻繁に送られる。また、分散ホストからいつ更新値が送られて来るかわからない。このような時、非同期受信による受信が必要になってくる。前研究では非同期受信するため、パケットが送られて来たらシグナルが発生する事を利用し、これを実現している。

3.1.3 大規模多元連立一次方程式の解法

大規模多元連立一次方程式は回路方程式などを解く際に用いる。多元連立一次方程式の解法の一例を以下に示す。

- 基本の解法
  - ガウスの掃き出し法
    一次方程式を加減法を使って解く最も単純な解法である。
- 近似を用いた解法
  - 最小の 2 乗法
    最も近い直線を考え、近似的な直線からできたいくつかのペアになった数の組をもとに、近似的に関数を求める方法である。

- ヤコビ法
  三角行列のアルゴリズムが元である。関係式に初期値 0 を入れて計算し、新しい値を求めることを反復させる。
- ガウス・ザイデル法
  ヤコビ法では、新しい値を都度求めていたが、直前に計算した値を利用する方法である。

多元連立一次方程式の解法の説明をする。線形方程式を

$$Ax = b, A = L + D + U \tag{3.1}$$

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

$$x^{(k+1)} = D^{-1}(b - Lx^{(k)} - Ux^{(k)}) \tag{3.2}$$

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、

$$x^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)}) \tag{3.3}$$

と表わされる。ガウス・ザイデル法に注目すると、(3.3) 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、(3.2) 式のガウス・ヤコビ法と等価になる。それ以外の順序で (3.3) 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。

3.1.4 緩和法解析の分散処理

前研究では対象とするアプリケーションの例として大規模行列の緩和法解析を利用し実験した。このアプリケーションは共有データの更新及び、参照が頻繁に

起こり、また、参照する共有データも最新のものである必要がない。なぜなら、答えは反復計算により共有データを更新しながら収束値を求めて行くものであり、データの更新に失敗また、遅れが生じたとしても致命的な影響を与える事がなく処理を続ける事ができる。よって前研究では、対象となるアプリケーションとしてこの大規模行列の緩和法解析を収束速度の速いガウス・ザイデル法を例に挙げる事にした。

3.1.5 緩和法解析の分散処理モデル

前研究で作成された共有データ及び更新のメカニズムのアプリケーション例として緩和法解析を利用した。この時このメカニズムで分散処理させた時、どのように処理が行なわれるか以下に述べる。

緩和法アルゴリズムでは求める値が収束するまで $x^{(k)}$ の値を更新しながら反復計算を行なう。そこで負荷分散させるために (3.1) 式を空間分割し、複数のホストを用いて分散処理させるものである。

計算の流れ

まず、分散処理を行なう際、初期設定のために必要な情報を各分散プロセスへ送るホストを 1 台用意する。この送信は高い信頼性が要求されるので TCP による通信を用いる。また、このホストを利用して緩和法解析の収束判定も行なう。以後この収束を判定するホストを収束判定ホスト、反復計算を行なうホストを反復計算クライアントと呼ぶ。この収束判定ホストは共有データが収束したと判断すると各反復計算クライアントへ計算の中止を要求するメッセージを送信する。

(3.1) 式において、各反復計算プロセスが計算を行なうために必要とするデータは x の全データ、b の一部のデータ、A の一部のデータである。従って、反復計算プロセスが (3.1) に含まれるすべてのデータを持つ必要はない。よって、1 台のホストで反復計算を行なうよりも、複数のホストで計算を行なった方が各々の持つデータ領域を縮小させる事ができる。

(3.1) 式が空間的にどの様に分散されるかを図 3.1 に示す。n 台の反復計算プロセスに分散する時、それぞれの反復計算クライアントに、式 (3.1) の A は $a_1, a_2, ...a_n$ と割り当てられる。それぞれのクライアントは共有データ x_m と b_m を用いて新しい x_m の値を求める。そして、それぞれの反復計算クライアントは割り当てられた部分のみを計算し、共有データである x の $x_1, x_2, ...x_n$ を更新する。この計算を反復計算して収束値を求めて答えを導き出す。

収束の判定

計算を終了させるためには共有データである x が収束したかを調べる必要がある。一回の反復計算が終了した時に、反復後の値と反復前の値を比べれば値が収束しているかの判断ができる。そのためには収束判定ホストは、一回の反復計算が終了した事を知る必要がある。そこで各反復計算クライアントより収束判定ホストへ、一回の反復計算が終った事を収束判定ホストに伝えている。そして、すべての反復計算クライアントより、この情報が送られたなら、収束判定ホストは一回の反復計算が終ったと判断し、収束したかの判定に移る。前研究に於いて、この通信がどのように行なわれているか図 3.2 に示す。この図 3.2 に示す通り、前研究では、この一回の反復計算が終った事を知らせる通信は、収束判定ホストが一回の反復計算が終ったら必ず収束したかの判定に移れるように通信の信頼性を保証した TCP を用いている。

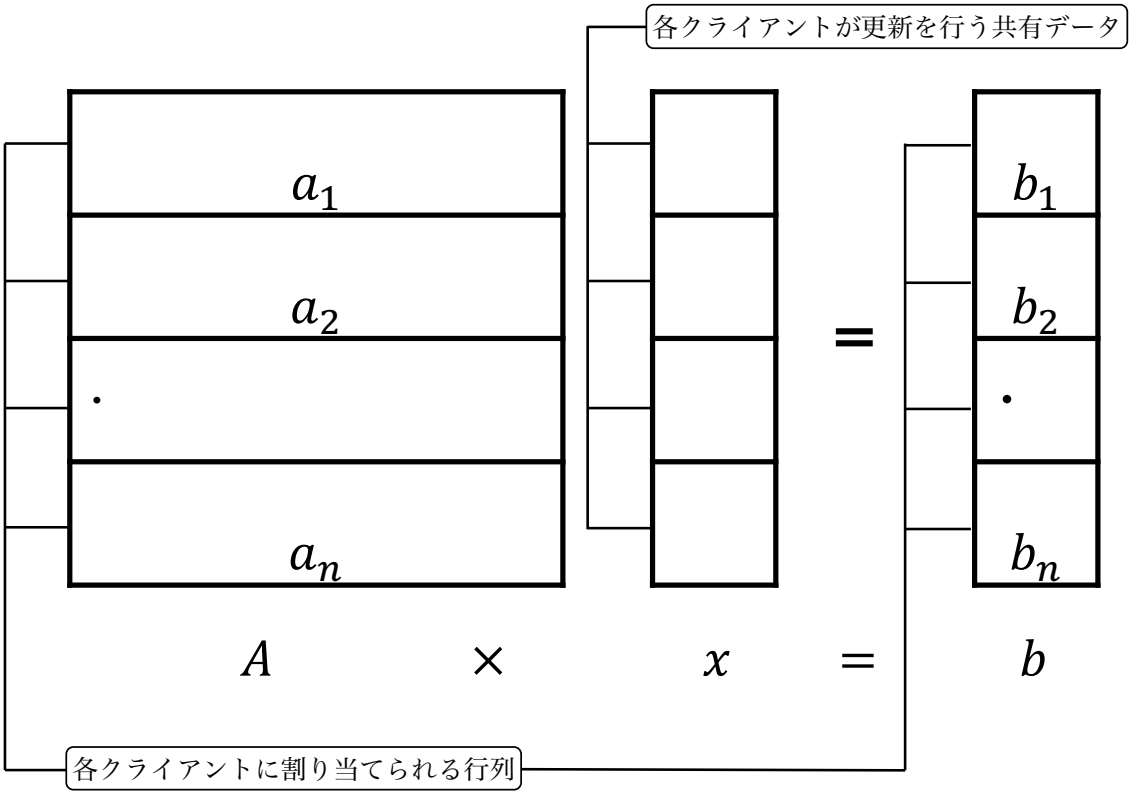

図 3.1: $A * x = b$ の空間分割

3.1.6 メモリの分散

緩和法解析の計算では担当する範囲の全てのメモリーに頻繁にアクセスするため、行列データはオンメモリーである必要がある。メモリーの量は有限であるため、大きな行列を計算する場合には、メモリーのスワップが発生することがある。

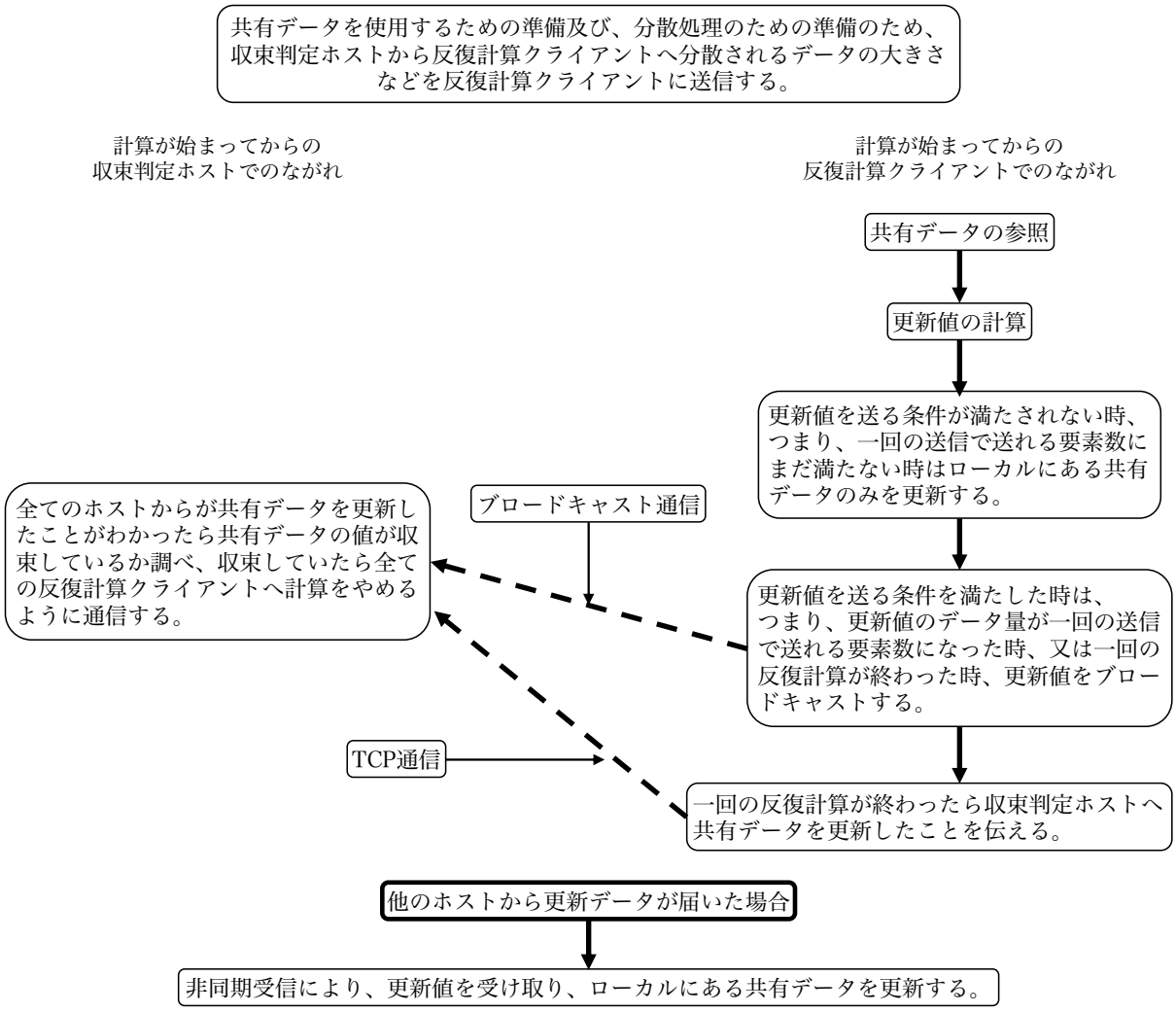

図 3.2: 収束判定ホストと反復計算クライアントでの流れ

頻繁に全てのメモリーにアクセスする場合、スワップの発生は計算時間を大幅に遅くしてしまう。

分散処理では 3.1.5 節の $A * b = x$ の空間分割した図 3.1 のように、計算を担当する範囲を分ける。図 3.3 は反復計算クライアント 1 台が必要とするメモリの箇所に色を塗った図である。

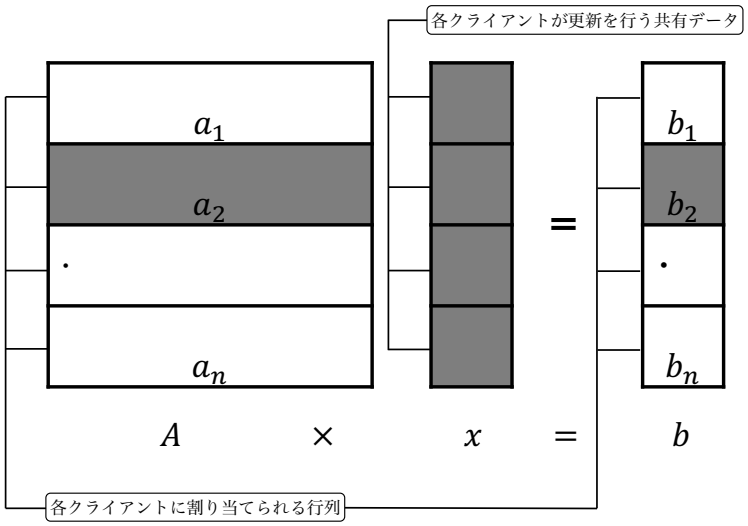

図 3.3: 各反復計算クライアントに必要な行列データ

図 3.3 のように、反復計算クライアントは A と b は担当する範囲しか参照する必要がない。データ量のほとんどは A のデータである。そのため、分散処理で反復計算クライアント 1 台当たりに必要なメモリ量は、1 台のみの場合に比べて反復計算クライアントの台数で割った量で良いことになる。A の行列の一辺の大きさと必要なメモリ量の関係を表 3.1 に示す。

表 3.1: 行列の大きさとデータ量の関係

一辺の行列サイズ	必要なメモリ量 (GB)
10000	約 0.7
20000	約 2.9
30000	約 6.7
40000	約 11.9
50000	約 18.6

表 3.1 の一辺の行列サイズが 50000 で約 18.6GB の時に、4 台の反復計算クライアントに分散させた場合、反復計算クライアント 1 台あたりの計算に必要なメモリ量は約 4.65GB となる。このように分散処理により、メモリを分散して持つことが可能なことから、スワップが発生する可能性を抑える事ができている。また、1 台あたりに搭載するメモリ量が少ない場合にも分散処理が有効であると言える。

収束判定ホストでは、収束判定時に必要なメモリは x の配列が 2 つになり、A 一辺の行列サイズが 10000 の場合、156KB になる。収束判定ホストは、収束したと

判定した後に検算を行わせている。本来、検算は不要ではあるが計算結果の信頼性の確認のために行っている。検算時には全てのデータが必要になるが、A と b は一回ずつしか使わないため、メモリが少なくても検算を行うことは可能である。

3.2 1998年に行われた研究

1998年に、明星大学で改めて実験が行われた。1998年と1992年での環境では大きく異なる。1998年の環境で実験した結果、次のような問題点を発見した。

- 台数を多くすると収束判定ホストで共有データの更新のためのパケットを落し、そのため、共有データの更新が遅れ、処理時間に影響を与えていた。
- ある仕事を複数のホストで分散処理させた場合、台数を増やせば増やすほど、処理時間は短縮する筈であるが、共有データは分散され、それを更新するためのネットワークの負荷が増える。そしてある台数を越えると処理時間が逆に増加する可能性がある。しかし、させたい仕事が何台まで有効なのかわからない。

1998年の研究[7]では以上の問題点に対し、それを解決するための方法を検討した。前研究では収束判定ホストでのみ、共有データの更新のためのパケットを落としていたことから、各反復計算クライアントが一回の反復計算が終わったことを収束判定ホストへ知らせる通信のオーバーヘッドが原因だとしていた。この通信は収束判定ホストが一回の反復計算が終わったら、必ず収束の判定をするように通信の信頼性を保証したTCPを用いており、TCPによるオーバーヘッドが原因としていた。この通信をUDPに変更した実験で、パケットを落とさないようになった。よって、TCPを用いた通信よりもUDPを用いた通信の方が処理時間が短縮できた。

TCPを使った通信をUDPを使った通信に変更したため、通信の信頼性は保証されなくなった。そこでパケットの損失について考えなければならない。収束判定ホストは全てのクライアントより一回の反復計算が終った事が知らされると答えが収束したかの判定に移っている。もし、あるホストよりパケットを落す事により、一回の反復計算が終った事が知らされなかったら、例え他のホストより一回の反復計算が終ったと知らされても、収束判定ホストでは全てのホストで一回の反復計算が終っていないと判断し答えが収束したかの判定に移る事ができない。しかし、もう一度そのホストが反復計算をし、一回の反復計算が終った事を収束判定ホストに知らせる事ができたら収束判定ホストでは全てのホストで収束判定が終ったと判断し、答えが収束したかの判定に移る事ができる。よって、ある特定のクライアントから、連続的に幾つも落すような環境でなければ数回の反復計算が増えるだけで済むといえる。

3.3 新しい技術の利用

現在の環境と前研究での環境では大きく異なる。1998年の研究では100Mbps対応のNICを利用していた。本研究では、10Gbps対応のNICの利用ができる時代となった。通信速度が向上することは、通信のインタラプトの間隔が小さくなる。

この結果、パケットの送受信に関わるインタラプト処理による通信オーバーヘッドがより大きくなると考ええられる。

現代では CPU はシングルコアからマルチコアになり、特定の用途に CPU コアを占有させる使い方が可能になった。それにより、前研究にはなかった、CPU コアをパケットの送受信に占有する DPDK が誕生した。10Gbps の高速なネットワークになったことで、通信オーバーヘッドに対処する必要が出てきた。そこで、3.1 節と 3.2 節で説明した研究に、この DPDK を 10Gbps ネットワークに活用しより高速な計算アプリケーションを実現することを目標とした。

第4章　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

4.1.1　設計

DPDKが提供するライブラリは、NICなどを高速に扱うための低レイヤ向け関数が細分化されている[8]。さらに、プリミティブなEtherフレームを扱うことから、アプリケーションを作成するためのプログラミングに時間がかかる。そのため、パケットを送信する、受信するという時に、システムコールが使えない。プログラミング時にヘッダーの作成や、Etherフレームからのデータの書き込み、読み出しまで面倒を見る必要がある。

DPDKが提供するライブラリ関数は、シンプルな単機能な関数の集合である。例えば、UDPパケットの送信を行うsendtoというシステムコールがある。DPDKで同様にUDPパケットを送信する場合、以下の処理が必要である。

- 送信用メモリの確保（hugepageの領域）
- ヘッダーごとにポインタを探す関数を使う（2つ目移行はポインタの移動でもよい）
- ヘッダーを作れる関数がないため、ヘッダー内の項目毎に作成し、送信用メモリに書き込み
- データ部のコピー
- NICの送信キューにパケットを渡す
- NICのキャパシティを越えている場合は再送処理を行う
- 送信に失敗した場合は、送信用に確保したメモリを解放する

上記の処理を行わないとUDPパケットすら送信することができない。OSが自動的に行っている処理などもすべて、DPDKライブラリを組み合わせて作らなければならない。これらの処理を分散処理など、アプリケーションプログラムの作成で書くのはとても効率が悪い。そのため、DPDKを用いたライブラリ関数を作成することとした。

本研究で行う分散処理は、ローカル環境の通信が行えれば十分である。汎用のスイッチを用いるため、L2レベルのヘッダは正しく作成しないと、スイッチングされない。

4.1.2 関数

DPDK を使うアプリケーションが実行可能な状態にコンピュータの初期化する関数を作成した。NIC を使える状態にするためには、以下の処理を行う。

1. 必要に応じて、MTU の設定を行う。
2. hugepage より、必要な量のメモリプールを作成する。
3. メモリプールより、送信バッファを作成する。
4. 送信バッファと対象の NIC を紐づける。
5. メモリプールより、受信バッファを作成する。
6. 受信バッファと対象の NIC を紐づける。
7. 対象の NIC を起動する。
8. NIC のリンクステータスが up になるのを待機する。

1 の MTU の変更は、必須ではない。デフォルトの MTU から変更しなくてよい場合は、設定をする必要はない。ジャンボフレームを使う場合、MTU を大きくしなくてはならないため、用意している。hugepage のメモリ領域を、NIC と組み合わせて効率よく使う設定を、2〜6 で行う。7 の NIC を起動させる DPDK のライブラリ関数が、成功の戻り値を返したとしてもすぐに通信を始められない。そこで、8 で NIC のリンクステータスが up になるまで待たせることで、通信ができないにも関わらずアプリケーションが始まることを防止する。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

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

DPDK 通信を行う通信関連の関数を作成した。C 言語の UDP 通信を行うライブラリ関数の sendto 関数、recvfrom 関数を意識した関数作成を行っている。ただし、通信オーバーヘッドを極力削減することを目標としているため、緩和方解析の分散処理アプリケーションを実現するために不必要なヘッダーは排除する方針とした。

DPDK にはプロトコルスタックがないため、使用していないプロトコルのヘッダーを使用する必要はない。本研究での環境では、L2 レベルの通信が行えれば通信が成立する。そのため、Ether ヘッダまでは、既存のプロトコルに合わせた設計としている。図 4.1 は DPDK を用いた緩和方解析の分散処理アプリケーションで使用するデータグラム構造である。

図 4.1: DPDK を用いた緩和方解析の分散処理アプリケーションでデータグラム構造

分散処理専用ヘッダーの構成を図 4.2 に示す。

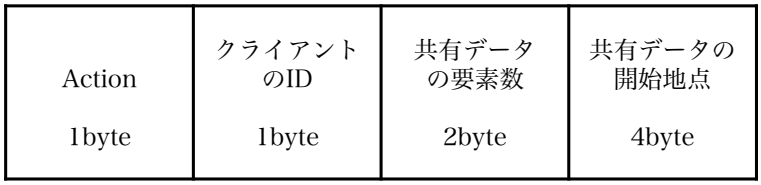

図 4.2: 分散処理専用ヘッダーについて

分散処理専用ヘッダーの詳細を以下に示す。

- Action
  受信側での処理方法を示しており、以下の種類がある。
  - データの共有
  - 計算の開始
  - 計算の中断
  - 計算の再開
  - 計算の終了
  - スイッチに MAC アドレスを学習させるダミーフレーム

  上記の計算の中断と再開は、収束値の検算を行う場合のみ用いる。検算を行った結果、収束値が収束していないとなった場合は、反復計算を再度行う必要がある。ただし、反復計算を検算中も進めることは可能ではあるが、計算時間に検算時間を入れていないため、計算を中断させている。検算で収束していないと判断した場合は、検算時間を計算時間には含めている。
  　スイッチに MAC アドレスを学習させるダミーフレームとは、UTP ケーブルや光ファイバーを NIC に接続しただけでは、スイッチはデータグラムの転送が行えない。そのため、DPDK を使用している NIC から何らかのデータを送信することで、スイッチに MAC アドレスを学習させている。ただし、スイッチが転送先が分かっている場合は、このフレームが他のコンピュータに届いてしまうため、受信した場合に無視できるように、Action を定義している。
- クライアントの ID
  送信元の特定ために用いる。Ether ヘッダの送信元 MAC アドレスから送信元を特定することは可能である。しかし、収束判定ホスト及び、反復計算クライアントに該当する MAC アドレスを順に比較することになり、避ける

方法があるなら避けたいことである。また、メモリの 4byte 境界というものがある。CPU アーキテクチャの違いによる影響を避けるために、構造体内で 4byte ごとの境界を不用意に跨がないようにする。そのため、「Action」と「共有データの要素数」で 3byte 使用しており、端数の 1byte を使わないのはもったいないためである。

- 共有データの要素数
  共有データのコピーする要素数である。データグラムの大きさから求めることも可能ではあるが、共有データの要素数が少ない場合に Ether フレームのペイロードを考慮し、ペイロードをダミーデータで埋めた場合に問題がある。そのため、データグラムに含まれる要素数を送信している。
- 共有データの開始地点
  共有データをコピーする際に、配列のコピーを開始する地点の値である。今日データの配列は一次元配列であり、行列データの一辺のサイズが最大値となる。符号なし short(2byte) の場合、0〜65,535 のため、行列が大きくなった場合に問題とならないように 4byte とした。

通信に関する関数を以下に示す。

dpdk_recvfrom 関数 DPDK でデータグラムの受信を行う (複数データグラムの同時受信に対応している)

dpdk_sendto 関数 DPDK でデータグラムの送信を行う

dpdk_set_info 関数 ヘッダを作成するための情報を作る

データグラムの送信を行う dpdk_sendto 関数を作成した。DPDK でデータグラムやパケットを送信する場合、メモリプールから mbuf を確保する。宛先 MAC アドレスや分散処理専用ヘッダーを含む構造体、送信データのポインタ、送信サイズを引数を使い受け取る。引数で受け取った情報をもとに、データグラムを作成する。そして、送信するために NIC へ渡される。

DPDK では NIC の処理能力を上回る速度でパケットを送信しようとすることがある。NIC の処理能力を上回った場合、パケットの送信に失敗する。パケット送信に失敗した場合に、一定時間待ってから、再送する仕組みを dpdk_sendto 関数に用意した。また、再送処理を行っても送信に成功できなかった場合は、送信時に確保した mbuf を解法する必要がある。

宛先 MAC アドレスは、DPDK 用の構造体に入っている。宛先 MAC アドレスを 16 進数表記で指定することも可能ではあったが、宛先ホスト名を用いた方がプログラミングが便利な場合がある。そのため、`/etc/ethers` に MAC アドレスとホスト名を紐づけるリストが OS で用意されている。このリストから、MAC アドレスをホスト名から解決して、DPDK 用の構造体に保存した。

データグラムの受信を行う dpdk_recvfrom 関数を作成した。DPDK でデータグラムやパケットを受信する場合、NIC から受信を担う DPDK のライブラリ関数が

自動的に、メモリプールから mbuf を確保する。このために、NIC から DPDK の受信を担うライブラリ関数にメモリプールのポインタを指定している。NIC を受信ポーリングを行う方法は、NIC から受信受信を担う DPDK のライブラリ関数を、ポーリングで呼び出すことで実現している。NIC から DPDK の受信を担うライブラリ関数が、パケットを受け取るってからデータのコピーを 1 回にする方法を実現している。NIC にデータグラムが到着すると、DPDK の受信を担うライブラリ関数が、受信バッファに直接書き込む。ここから、データコピーを 1 回のみに抑えた。dpdk_recvfrom 関数で共有データへのコピーを行わせることにした。分散処理専用ヘッダーの情報を元に、共有データのコピーを行っている。

NIC から受信を担う DPDK のライブラリ関数が、mbuf に受信データグラムを入れている。mbuf は 1 つのバッファではなく、複数のバッファがリンクでつながっている。データのコピーを行う際は、mbuf の仕組みを考慮したコピーが必要となっている。

NIC に届くデータグラムは、常に 1 つであるわけではない。複数のデータグラムが同時に届くこともある。DPDK では送信または受信を担うライブラリ関数に同時に送受信するパケット数の上限を指定することができる。緩和法解析の分散処理では、パケット送信をする時に少しでも速く送信したいため、パケットを貯めない方針にした。そのため、同時送信パケット数を 1 に設定している。次に、パケットを受信する時は複数台からのパケットを受信しているため、パケットの到着タイミングが被ることがある。そのため、同時受信パケット数を dpdk_recvfrom 関数の引数でで指定できるようにした。

4.2 スレッドによる受信

UDP を用いた分散処理アプリケーションでは、共有データは非同期受信にて行っていた。しかし、DPDK では OS のシグナルによる割り込み処理での受信の実装を行わない。DPDK はユーザ空間で動いているため、OS によるパケット受信のシグナルを利用できない [9]。DPDK の特徴が、CPU コアを占有した NIC のポーリングにより、インタラプトを行わないことによる高速化 [10] しているためである。これらから、DPDK での共有データの受信には、スレッドによる受信により実現することを考えた。共有データをメモリに置いているため、グローバルポインタによるメモリアクセスによりスレッドでの書き換えができる。

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 4.3 に示す。

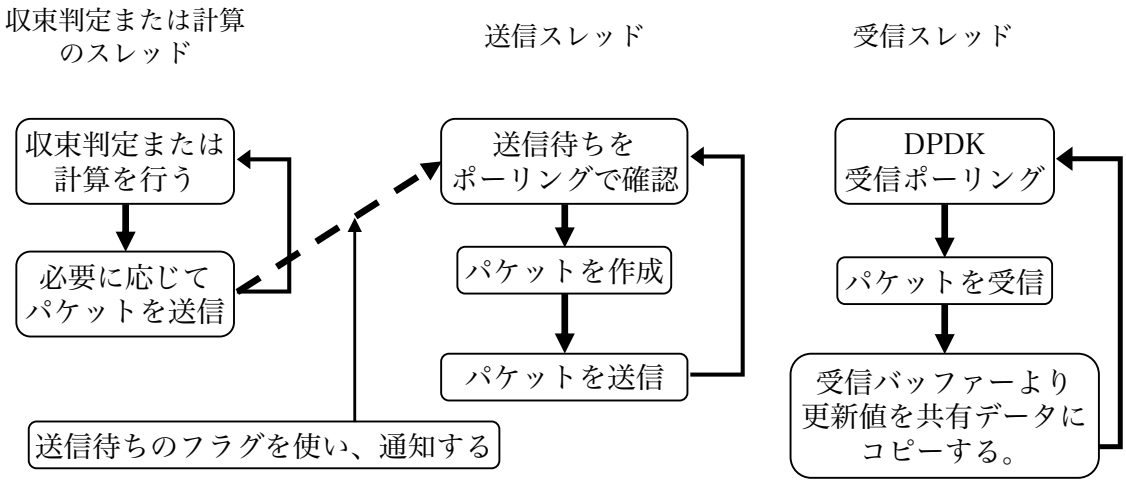

図 4.3: DPDK の共有データに関する流れについて

4.3 ネットワークの構成

前述した通り、DPDK には TCP/IP のように信頼性のある通信が提供されていない。DPDK はカーネルが用意するようなプロトコルスタックを提供していないためである。本研究で扱う大規模行列の緩和法解析は、共有データの更新及び、参照が頻繁に起こり、また参照する共有データも最新のものである必要がない。そのため、共有データの通信は、高速性のみを追求してよい通信である。ただし、分散処理の計算には信頼性の必要な通信はないが以下の目的のために、TCP 通信を用いている。

- 計算対象の行列データを配布するため。
- 計算結果等の統計データを収集するため。
- リモートで操作用の ssh のため。

UDP を用いた分散処理アプリケーションでは、UDP のため同じネットワークで TCP を使えた。しかし、DPDK では TCP のような信頼性のある通信がないため、通信の信頼性を確保するための仕組みが必要である。そこでネットワークを 2 重化することにした。ネットワークを 2 重化することで、DPDK と従来のカーネル環境下の TCP 通信を併用することで、信頼性を維持し超高速通信によるデータの共有が行える。DPDK を用いた分散処理アプリケーションの実験環境を図 4.4 に示す。

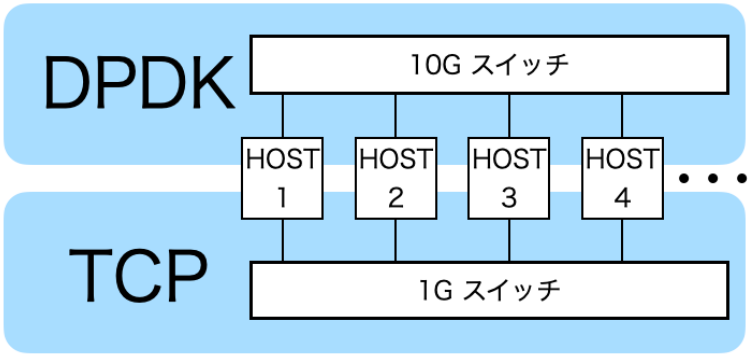

図 4.4: DPDK を用いた分散処理アプリケーションの接続イメージ

図 4.4 の上段は 10G 対応スイッチを用いた DPDK 通信を表している。DPDK 通信は、共有データの通信に用いる。正確性は求められていないが、通信量があるため高速通信を必要としているためである。図 4.4 の下段は 1G 対応のスイッチを用いたカーネル環境下での TCP 通信を表している。TCP 通信は通信の信頼性が必要な通信である、計算対象の方程式のデータに用いる。

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。1992 年の研究の 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。

第5章　結果と考察

5.1　実験機器

実験機の構成は以下である。

- 実験機 1, 実験機 2, 実験機 3, 実験機 4
  - CPU : Intel Core i5-9600K CPU @ 3.70GHz
  - real memory: 16GB
  - OS : Ubuntu 14.04 LTS
  - DPDK : DPDK 2.1.0
  - NIC(DPDK 用): Intel Corporation Ethernet Controller 10-Gigabit X540-AT2
  - NIC(ssh 用) : Intel Corporation 82574L Gigabit Network Connection
- 実験機 5, 実験機 6
  - CPU : Intel Core i5-9600K CPU @ 3.70GHz
  - real memory: 16GB
  - OS : Ubuntu 14.04 LTS
  - DPDK : DPDK 2.1.0
  - NIC(DPDK 用): Intel Corporation Ethernet Controller 10-Gigabit X540-AT2
  - NIC(ssh 用) : Intel Corporation Device 15bc

  ※ M/B と memory が実験機 1〜4 と異なる
- 実験機 7
  - CPU : Intel Core i5-8400 CPU @ 2.80GHz
  - real memory: 16GB
  - OS : Ubuntu 14.04.1 LTS
  - DPDK : DPDK 2.1.0
  - NIC(DPDK 用): Intel Corporation Ethernet Controller 10-Gigabit X540-AT2
  - NIC(ssh 用) : Intel Corporation 82572EI Gigabit Ethernet Controller
- 実験機 8
  - CPU : Intel Core i7-7700 CPU @ 3.60GHz
  - real memory: 64GB
  - OS : Ubuntu 14.04.1 LTS

- DPDK : DPDK 2.1.0
- NIC(DPDK 用): Intel Corporation Ethernet Controller 10-Gigabit X540-AT2
- NIC(ssh 用) : Ethernet controller: Intel Corporation 82571EB Gigabit Ethernet Controller

特に記述が無い場合は、収束判定ホストに実験機 8 を使用している。また、計算クライアントは実験機番号の若い番号から使用した。

5.2 1 台での計測結果

分散処理を行う前に、1 台のコンピュータで行える大きさの行列（多元連立一次方程式）を対象とした計算時間の計測をした。

本研究では計算時のネットワークのボトルネックの解消を目的にしている。そのため、図 5.1 に示す計算時間のみを計測している。

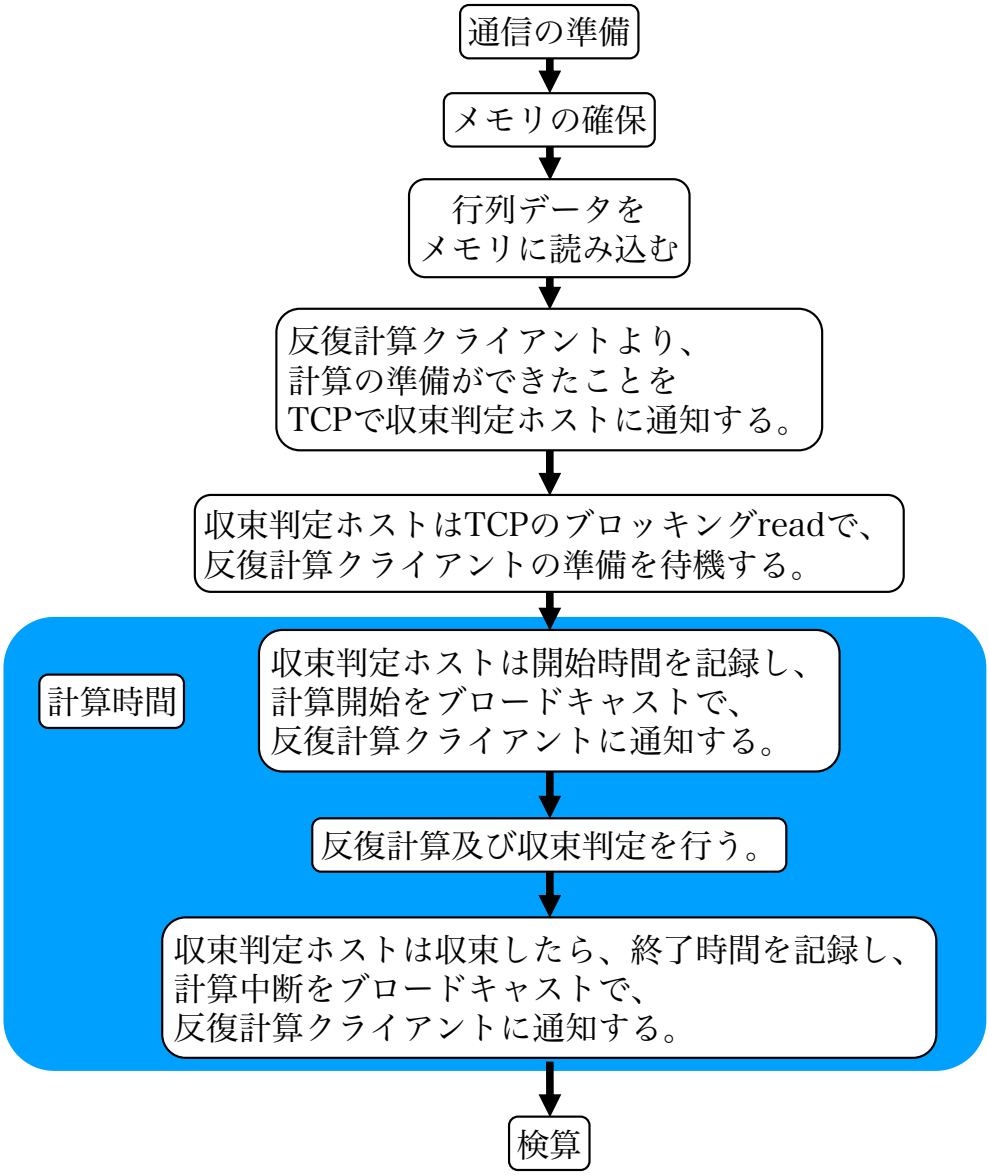

図 5.1: 計算時間の定義について

16GB のメモリを搭載した実験機 1 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 5.1 にスワップによる影響を示す。表 5.1 の一辺の行列サイズ 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。しかし、一辺の行列サイズ 40000 ではスワップが発生し、計算量の増加量を超えて計算が非常に遅くなった。さらに、一辺の行列サイズ 50000 では、malloc でのメモリ確保を行うことができなかった。そのため、スワップが起きない範囲での計算を行うことになる。

表 5.1: 「16GB のメモリを搭載した実験機 1 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	2.147
20000	8.589
30000	19.350
40000	2984.467
50000	malloc のエラー

5.3 1 台でのスレッドによる分散処理の計測結果

ネットワークを利用した分散処理行うにあたり、1 台（実験機 1 ）のコンピュータで複数のコアに分散させた実験を行った。実験機 1 は 6 コアの CPU である。カーネル等が利用するコアを考慮し、収束判定を 1 スレッド、反復計算を 4 スレッドの計 5 スレッドでの実験を行った。収束判定を 1 スレッド、反復計算を 4 スレッドとした 1 台でのスレッド化による行列サイズと計算時間の関係を図 5.2 のグラフに示す。収束判定及び反復計算をスレッド化により分散させることで、分散させ

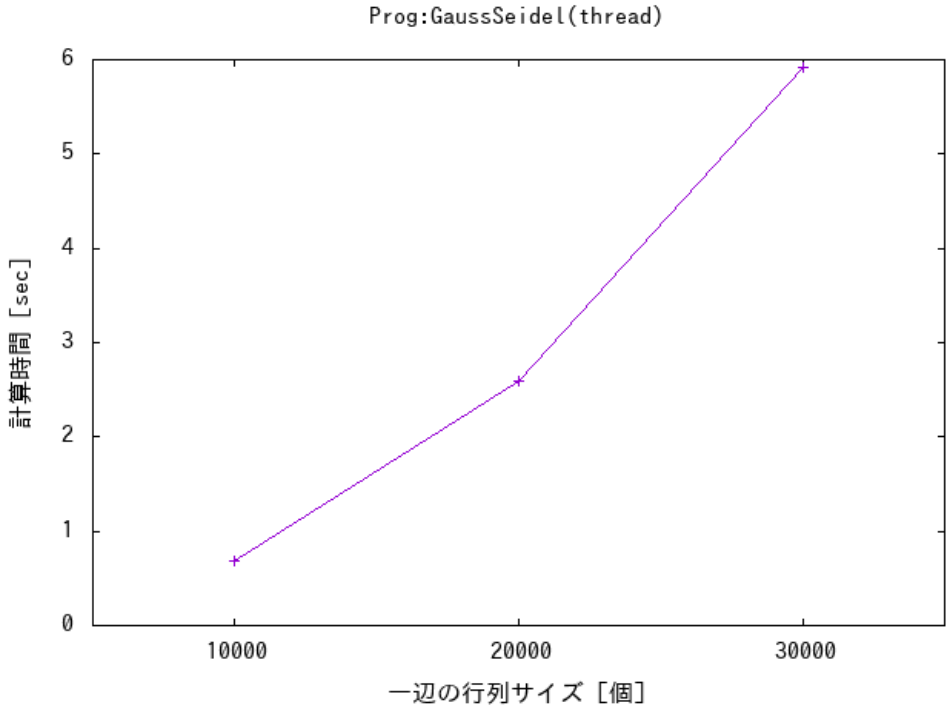

図 5.2: 「1 台でのスレッド化による行列サイズと計算時間の関係」（収束判定 1、反復計算 4、5 回実行した平均値）

る前に比べて高速化された。図 5.2 のグラフから、分させる前と同様に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。ただし、1 台

で行う以上、メモリスワップ発生による問題があるため、図 5.2 のグラフは一辺の行列サイズを 30000 までしか描いていない。

5.4　UDP を用いた分散処理の結果

5.4.1　1 つのデータグラムで送る共有データの要素数

前研究での UDP を用いた分散処理では通信オーバーヘッドを考慮して、通信回数を減らすために共有データの送信を MTU 近くの要素数まで貯めていた。前研究では MTU は 1500byte であり、行列の 1 要素は 8byte であった。MTU の 1500byte から、UDP ヘッダ及び共有データを送信するのに必要な情報を引くと、1448byte になる。行列の 1 要素は 8byte である。共有データの一回の送信が 1 つのデータグラムに収めるためには、要素数は 181 個以下になる。

本研究では MTU は最大 9000byte まで対応している NIC 及びスイッチを使用した。MTU の 9000byte から、UDP ヘッダ及び共有データを送信するのに必要な情報を引くと、8942byte になる。共有データの一回の送信が 1 つのデータグラムに収めるためには、要素数は 1117 個以下になる。UDP を用いた分散処理アプリケーションで 1 つのデータグラムで送る共有データの要素数を変えたときの計算時間の関係を図 5.3 のグラフに示す。

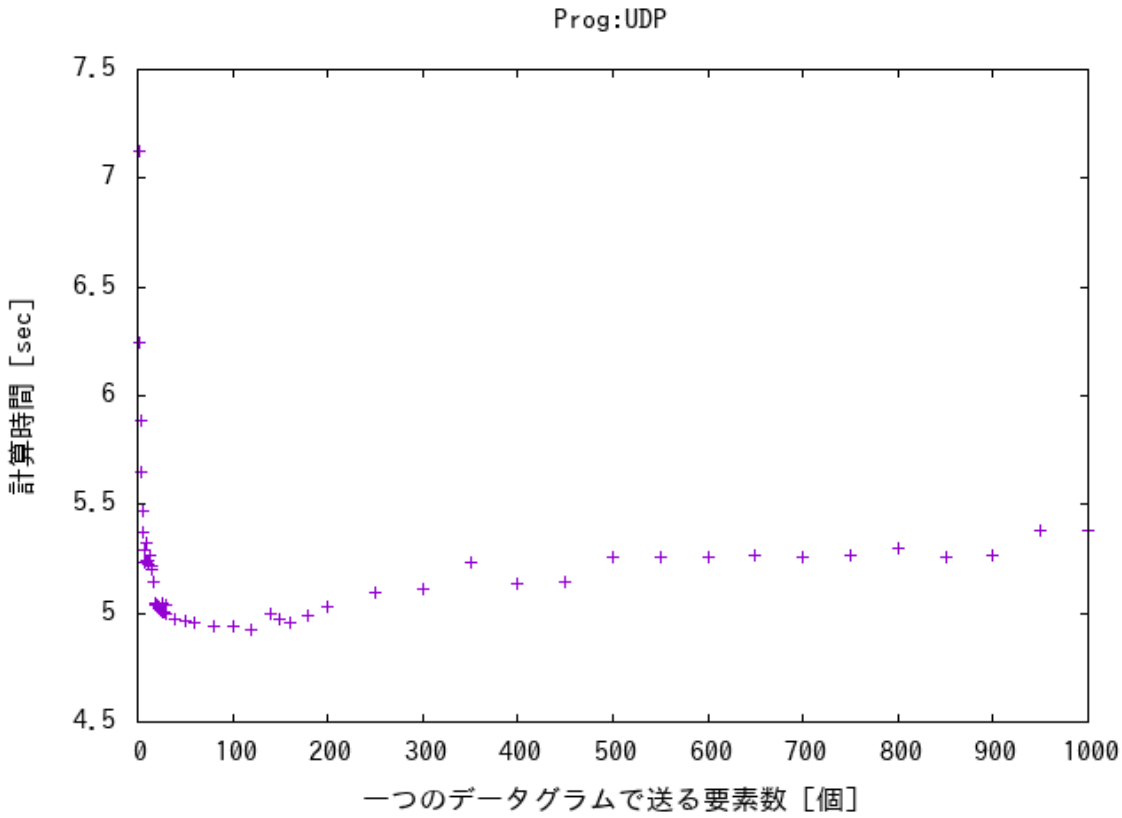

図 5.3:　「一つのデータグラムで送る要素数を変えた場合の計算時間との関係」（UDP、反復計算 4、一辺の行列サイズ 30000、5 回実行した平均値）

図 5.3 の右側である、1 つのデータグラムで送る共有データの要素数を増やしていくと多少、収束に時間がかかるようになった。1 つのデータグラムで送る共有

データの要素数を多くなると、共有データの更新が遅れることとなり、収束が遅くなる。ただし、MTU が 1500byte だった時は、1 つのデータグラムで送る共有データの要素数を 181 要素以下で実験を行っていた。1 つのデータグラムで送る共有データの要素数を少なくなると、共有データの更新が速くなり、収束が速くなることが期待される。

図 5.3 の左側である、1 つのデータグラムで送る共有データの要素数を減らしていくと、計算時間が大幅に長くなった。1 つのデータグラムで送る共有データの要素数を少なくしたことで、頻繁に共有データの送信を行うことになり、パケット数が増加している。パケット数が増加したことにより、通信オーバーヘッドの影響を受けていることがいえる。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。

図 5.3 の左側である、1 つのデータグラムで送る共有データの要素数を 180 以下で書き直したグラフを図 5.4 に示す。図 5.4 の左側である、1 つのデータグラムで

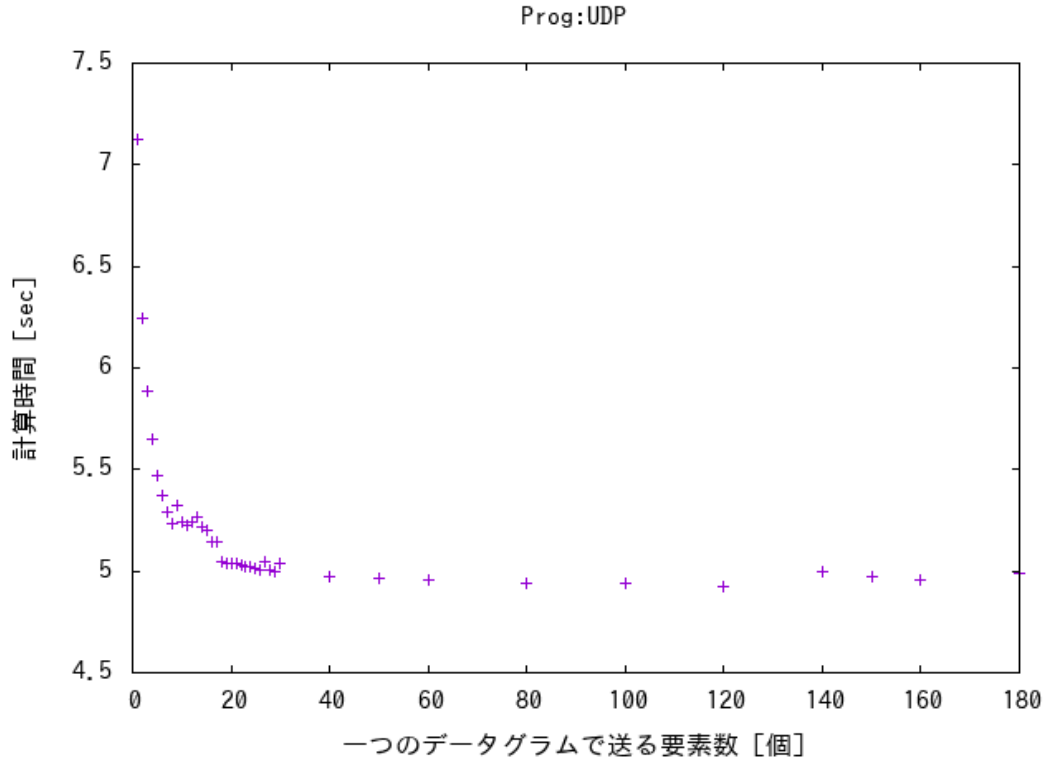

図 5.4: 「一つのデータグラムで送る要素数を少ない場合の計算時間との関係」（UDP、反復計算 4、一辺の行列サイズ 30000、5 回実行した平均値）

送る共有データの要素数が 20〜30 個ほどで計算時間の上限に概ね達している。そのため、要素数を 20 個を比較の基本として扱うこととする。

5.4.2　反復計算クライアントの台数

反復計算クライアントの台数による計算時間について検証した。前研究でも台数を増やすことで、1 台あたりが担当する計算量が減り、収束までの計算時間が短くなるとされていた。台数により計算時間が短くなることを、台数効果という。UDP を用いた分散処理アプリケーションで反復計算クライアントの台数を変えたときの計算時間の関係を図 5.5 のグラフに示す。図 5.5 から、台数が増えることで計算時間が短縮されていることが分かる。図 5.5 の結果を元に、台数効果により計算が高速化されていることを、反復計算クライアント 1 台の時を基準とし、計算時間の比を図 5.6 に示す。

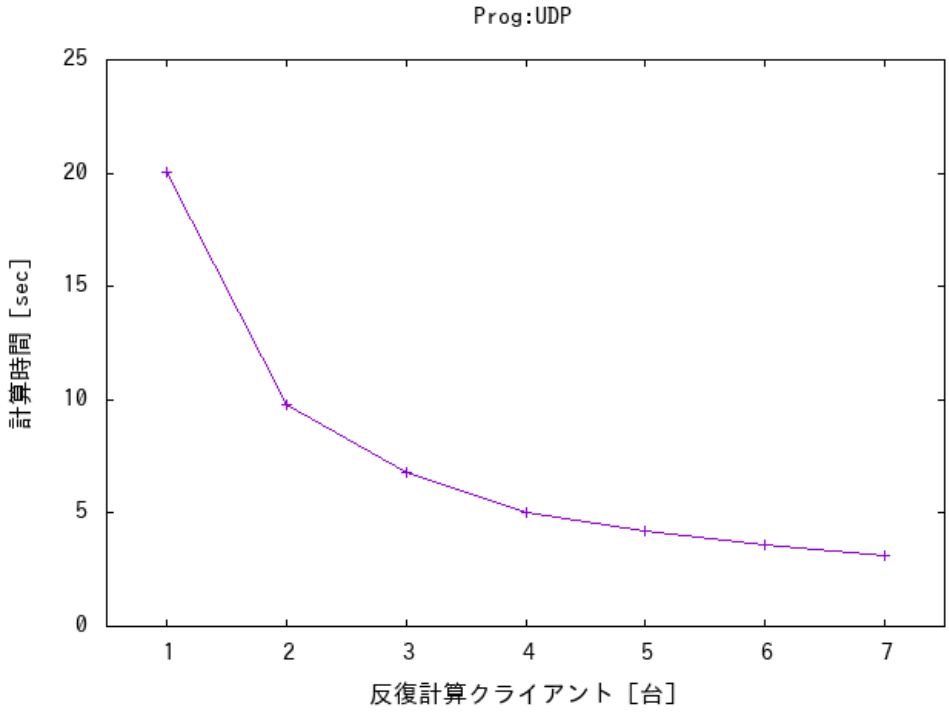

図 5.5:「反復計算クライアントの台数を変えた場合の計算時間の関係」（UDP、一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値）

図 5.6 から、反復計算クライアントが 1〜4 台目は実験機の構成が共通のため、比例している。反復計算クライアントが 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。

5.4.3　行列サイズ

行列サイズの違いによる計算時間について検証した。UDP 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図 5.7 のグラフに示す。図 5.7 からは、行列サイズの大きさの増加に伴い、計算時間が増加していることがわかる。計算量に比例して計算時間が増加するのは当然のことである。図

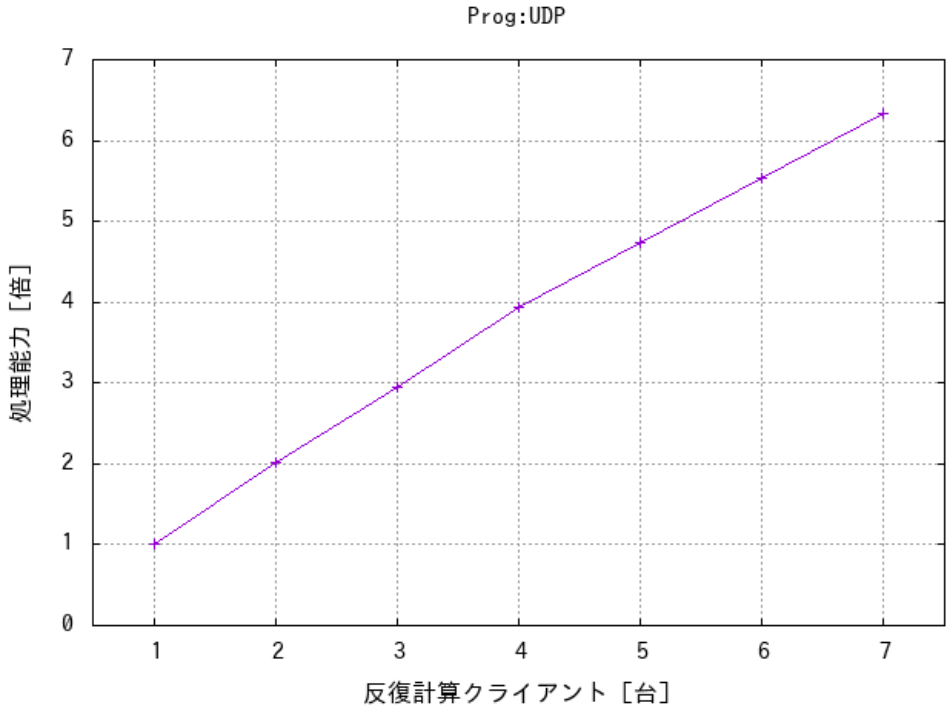

図 5.6: 「反復計算クライアントの台数を変えた場合の台数効果による計算の高速化」（UDP、一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値）

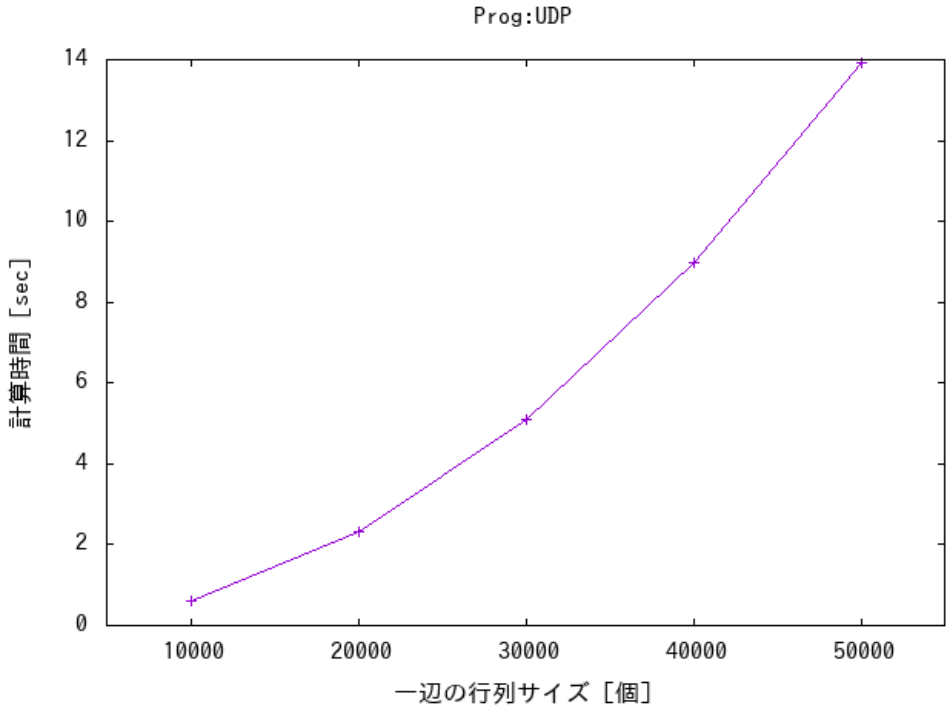

図 5.7: 「行列サイズを変えたときの計算時間の関係」（UDP、反復計算 4、要素数 20 個、5 回実行した平均値）

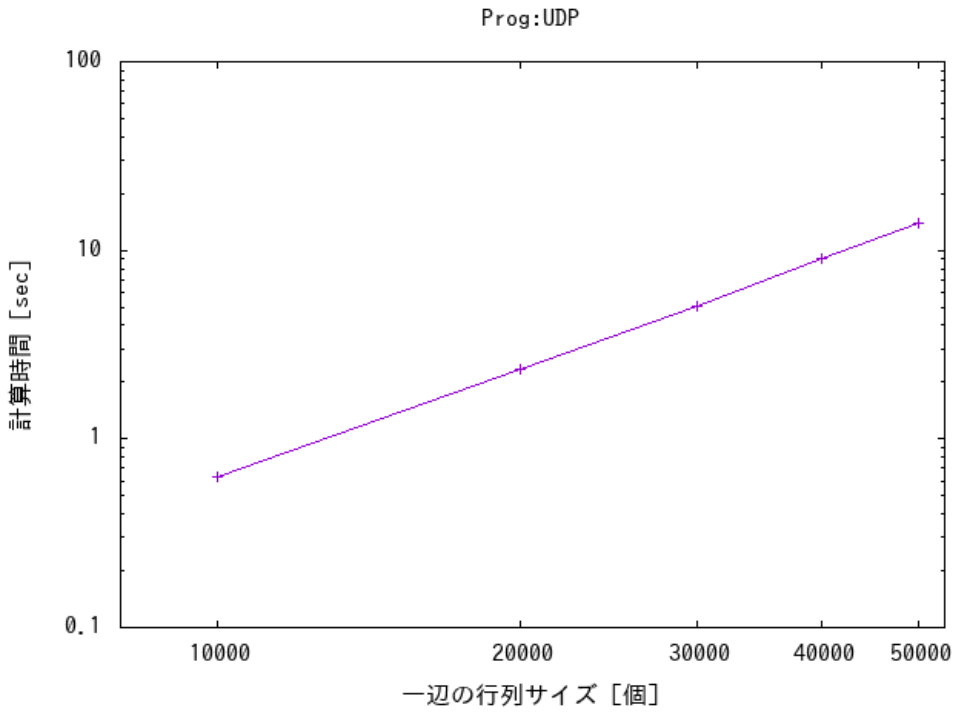

図 5.8: 「行列サイズを変えたときの計算時間の関係」（対数スケール表示、反復計算 4、要素数 20 個、5 回実行した平均値）

5.7 を対数スケールで書き直したグラフを図 5.8 に示す。分散処理によるオーバーヘッド影響を受ける場合は、図 5.8 の右側に進むにつれて、右上がりに曲がっていく。図 5.8 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。

5.5 DPDK を用いた分散処理の計測結果

MTU の 1500byte から、UDP ヘッダ及び共有データを送信するのに必要な情報を引くと、1478byte になる。行列の 1 要素は 8byte である。共有データの一回の送信が 1 パケットに収めるためには、要素数は 184 個以下になる。

5.5.1 1 つのデータグラムで送る共有データの要素数

DPDK を用いた分散処理アプリケーションで 1 つのデータグラムで送る共有データの要素数を変えたときの計算時間の関係を図 5.9 のグラフに示す。図 5.9 の右側

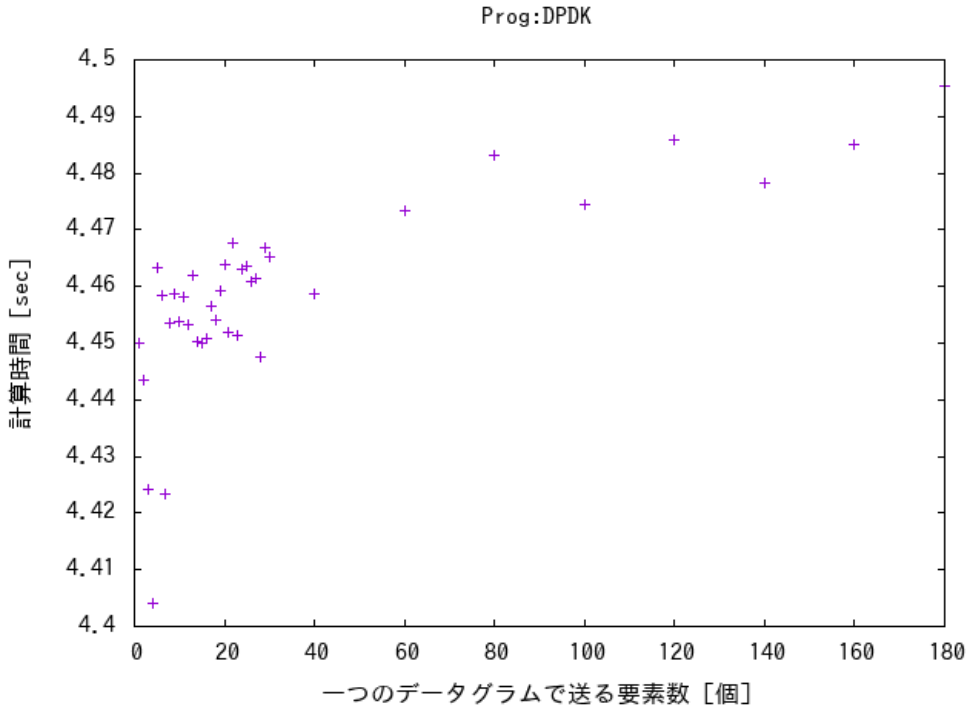

図 5.9: 「一つのデータグラムで送る要素数を少ない場合の計算時間との関係」（DPDK、反復計算 4、一辺の行列サイズ 30000、5 回実行した平均値）

である、1 つのデータグラムで送る共有データの要素数を増やしていくと緩やかに、収束に時間がかかるようになった。図 5.9 の左側である、1 つのデータグラムで送る共有データの要素数を 30 個以下にするでは、計算時間にバラツキがある。DPDK の超高速通信により、通信オーバーヘッドの影響が削減されることで、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。しかし、1 つのデータグラムで送る共有データの要素数を減らしても、極端に速くも遅くもならなかった。DPDK を用いて直接データグラムを作成する場合でも、通信オーバーヘッドはある [11]。ただし、1 つのデータグラムで送る共有データの要素数を減らしていったとしても、DPDK は UDP のようには遅くはならなかった。図 5.9 の結果から、反復計算クライアン

ト 4 台の時は、1 つのデータグラムで送る共有データの要素数が 20 前後くらいが、速く収束すると考えられる。

5.5.2 反復計算クライアントの台数

反復計算クライアントの台数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反復計算クライアントの台数を変えたときの計算時間の関係を図 5.10 のグラフに示す。図 5.10 から、台数が増えることで計算時間

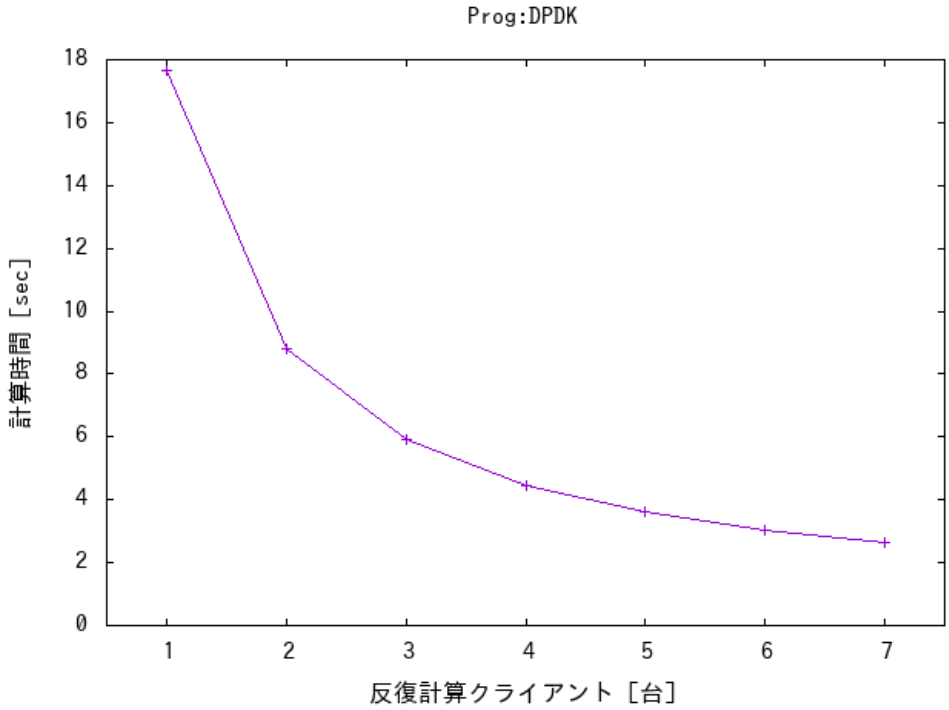

図 5.10:「反復計算クライアントの台数を変えた場合の計算時間の関係」（DPDK、一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値）

が短縮されていることが分かる。図 5.10 の結果を元に、台数効果により計算が高速化されていることを、反復計算クライアント 1 台の時を基準とし、計算時間の比を図 5.11 に示す。図 5.11 から、反復計算クライアントが 1〜4 台目は実験機の構成が共通のため、比例している。反復計算クライアントが 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。分散処理で台数効果による高速化は、反復計算クライアントが 1 台の時に比べて、2 台で 2 倍、3 台で 3 倍になるのが理想論である。2 台で 2 倍、3 台で 3 倍になる状況というのは、分散処理によるオーバーヘッドが発生していない状況になる。図 5.11 では、2 台で 2.02 倍、3 台で 2.95 倍、4 台で 3.97 倍となっており、分散処理によるオーバーヘッドがほとんど発生していないことがわかる。

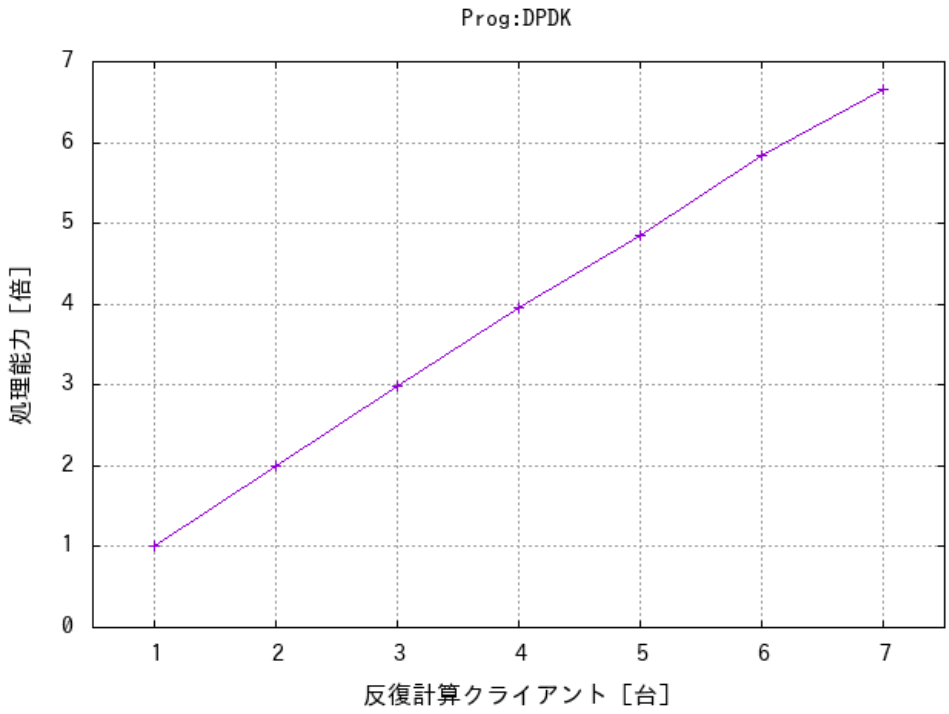

図 5.11: 「反復計算クライアントの台数を変えた場合の台数効果による計算の高速化」（DPDK、一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値）

5.5.3 行列サイズ

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図 5.12 のグラフに示す。図 5.12 からは、行列サイズの大きさの増加に伴い、計算時間が増加していることがわかる。計算量に比例して計算時間が増加するのは当然のことである。図 5.12 を対数スケールで書き直したグラフを図 5.13 に示す。分散処理によるオーバーヘッド影響を受ける場合は、図 5.12 の右側に進むにつれて、右上がりに曲がっていく。図 5.12 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。

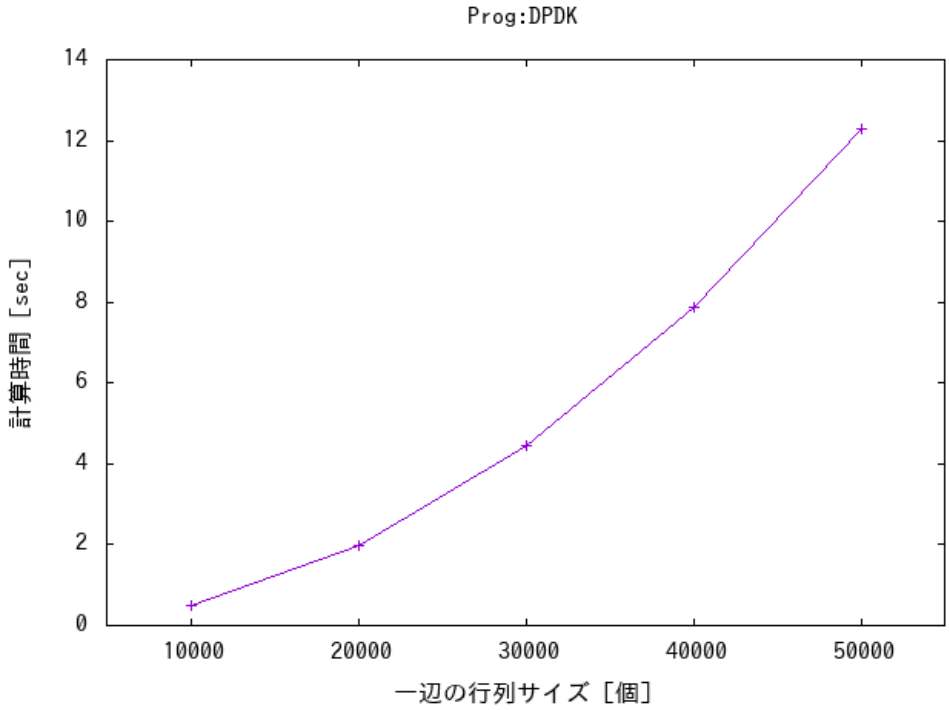

図 5.12:「行列サイズを変えた場合の計算時間の関係」（DPDK、反復計算 4、要素数 20 個、5 回実行した平均値）

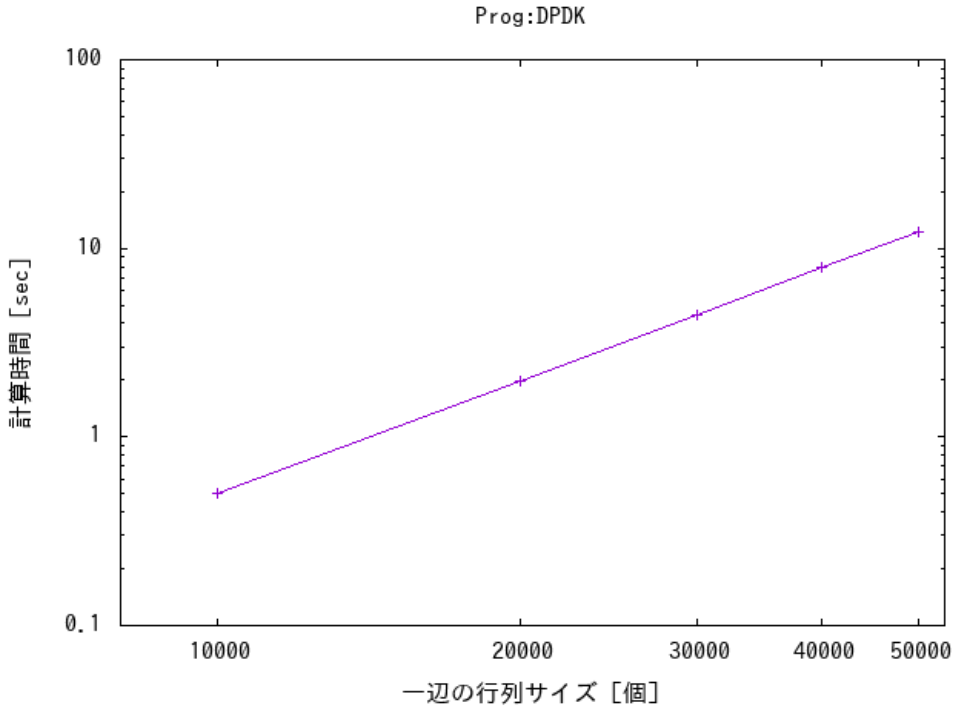

図 5.13:「行列サイズを変えた場合の計算時間の関係」（対数スケール表示、DPDK、反復計算 4、要素数 20 個、5 回実行した平均値）

第6章　比較と評価

6.1　通信速度の比較

DPDK を用いた緩和法解析の分散処理アプリケーションと、UDP を用いた緩和法解析の分散処理アプリケーションで、ネットワークの転送速度の確認を行った。DPDK と UDP それぞれのアプリケーションで、計算を行っている時の転送速度を表 6.1 に示す。

表 6.1:「DPDK を用いた緩和法解析の分散処理アプリケーションと、UDP を用いた緩和法解析の分散処理アプリケーションで計算を行っている時の転送速度」（5 回実行した平均値）

1 つのデータグラムで送る共有データの要素数	一辺の行列サイズ	反復計算クライアントの台数	DPDK の転送速度 (Mbps)	UDP の転送速度 (Mbps)
1	30000	4	67	37
1	30000	7	110	63
1	50000	4	40	30
1	50000	7	66	44
20	30000	4	10	11
20	30000	7	16	17
20	50000	4	6	6
20	50000	7	10	10

表 6.1 から、10Gbps のネットワーク上で 10〜100Mbps 程度でしかパケットが流れていないため、輻輳はおきていない。1 つのデータグラムで送る共有データの要素数が少なくなると、DPDK と UDP の転送速度の差が出てくる。ネットワークの転送速度が 10Gbps のネットワーク上で 10〜100Mbps 程度というのは、ネットワークがスカスカの状態であるといえる。ネットワークがスカスカの状態で、DPDK と UDP の転送速度に差が出たということは、パケット処理のオーバーヘッドの影響が考えられる。

6.2　一つのデータグラムで送る要素数による比較

1 つのデータグラムで送る共有データの要素数を多くなると、共有データの更新が遅れることとなり、収束が遅くなる。反対に、1 つのデータグラムで送る共有データの要素数を少なくなると、共有データの更新が速くなり、収束が速くなることが期待される。UDP を用いた緩和法解析の分散処理アプリケーションは、一度に送る共有データの要素数を減らすと、計算時間が大幅に長くなる。1 つのデータグラムで送る共有データの要素数を少なくしたことで、頻繁に共有データの送信を行うことになり、パケット数が増加している。パケット数が増加したことにより、通信オーバーヘッドの影響を受けていることがいえる。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。

DPDK を用いた緩和法解析の分散処理アプリケーションと、UDP を用いた緩和法解析の分散処理アプリケーションで、1 つのデータグラムで送る共有データの要素数を変えた計測結果を図 6.1 に示す。この図 6.1 は、先に示した 1 つのデータグラムで送る共有データの要素数を 25 個以下にした場合の DPDK のグラフと UDP のグラフを組み合わせたグラフである。DPDK を用いた緩和法解析の分散処理ア

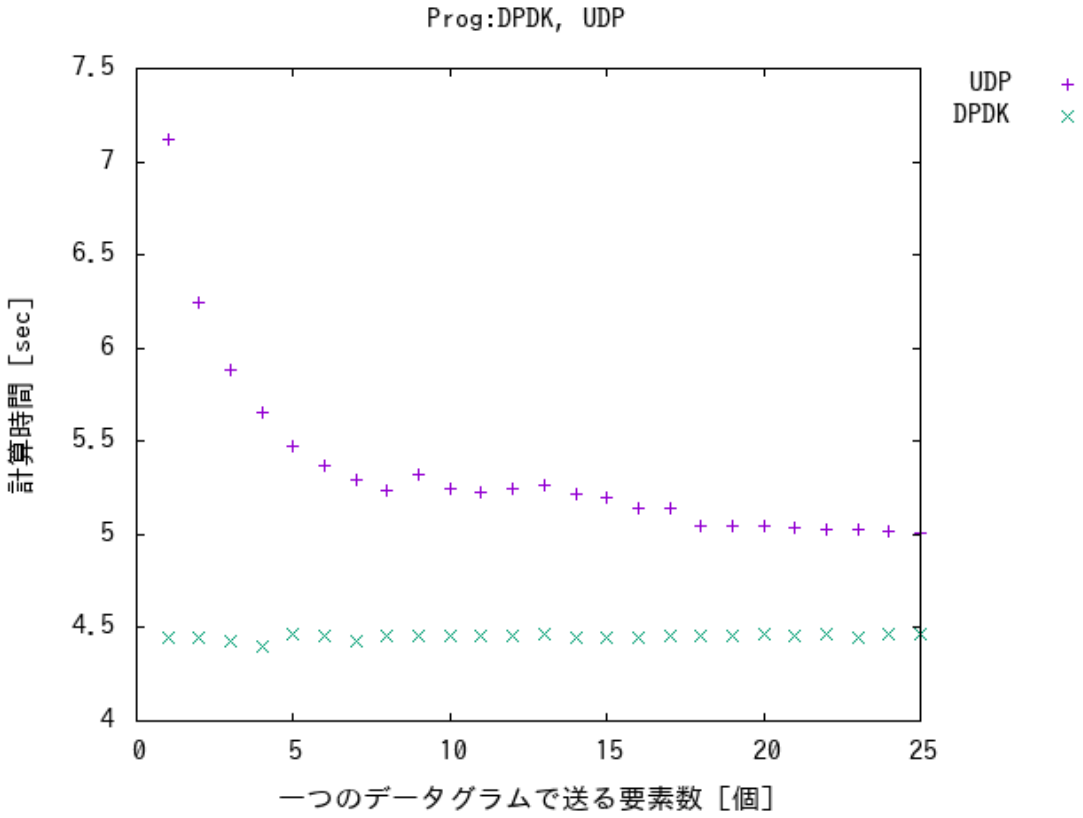

図 6.1: 「DPDK を用いた緩和法解析の分散処理アプリケーションと UDP を用いた緩和法解析の分散処理アプリケーションで一つのデータグラムで送る要素数と計算時間の関係」（一辺の行列サイズ 30000、反復計算 4、5 回実行した平均値）

プリケーションでは通信オーバーヘッドの改善により、1 つのデータグラムで送

る共有データの要素数を減らした場合に共有データの更新が速くなることで収束も速くなり、計算時間が短縮されることが予想された。しかし、図 6.1 から UDP の計算時間に比べると、DPDK の計算時間はほぼ横ばいになる。

DPDK では UDP を用いた場合に比べて、一つのデータグラムで送る要素数をいくつにした場合でも速く計算が終わった。一辺の行列サイズは 30000 で反復計算クライアントを 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 では一度に送る共有データの要素数が 120 個の時に、4.92389 秒であった。DPDK では一度に送る共有データの要素数 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一度に送る共有データの要素数が 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。

6.3 行列サイズによる比較

DPDK を用いた緩和法解析の分散処理アプリケーションと、UDP を用いた緩和法解析の分散処理アプリケーションで、行列サイズを変えた計測結果を図 6.2 のグラフに示す。図 6.2 から行列サイズが大きくなるにつれて、計算時間が長くなっ

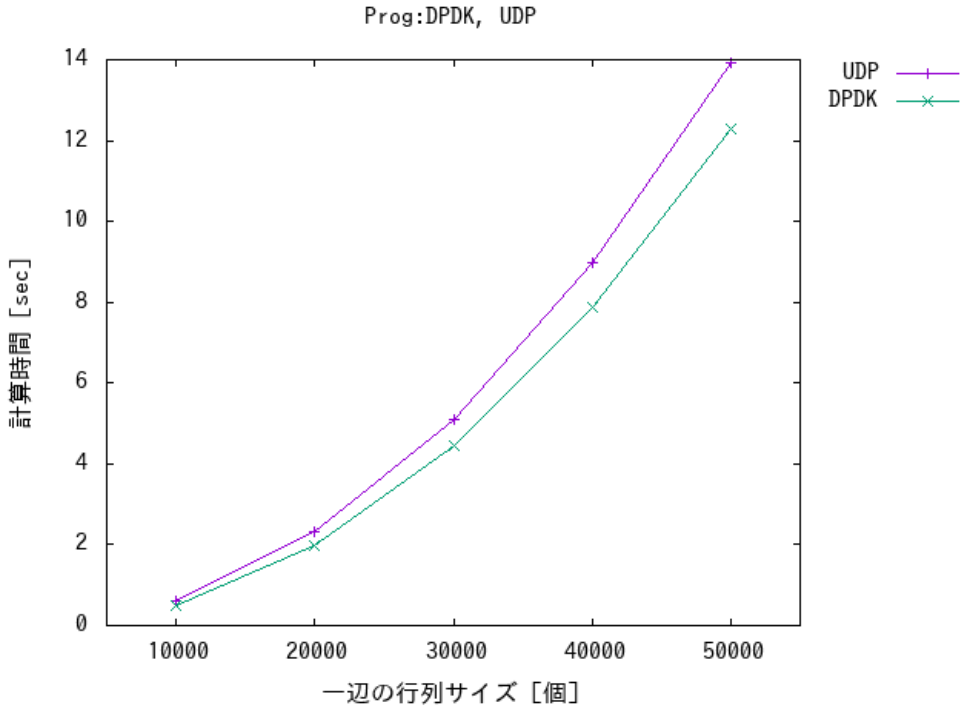

図 6.2: 「DPDK を用いた緩和法解析の分散処理アプリケーションと UDP を用いた緩和法解析の分散処理アプリケーションでの行列サイズと計算時間の関係」（反復計算 4、要素数 20 個、5 回実行した平均値）

ていることが確認できる。UDP を用いた緩和法解析の分散処理アプリケーション

と DPDK を用いた緩和法解析の分散処理アプリケーションでの計算時間の差が、行列サイズが大きくなるにつれて広がっている。そのため、行列サイズを大きくしたとしても DPDK が UDP に比べて速いことが予想される。

6.4 反復計算クライアント台数による比較

DPDK を用いた緩和法解析の分散処理アプリケーションと、UDP を用いた緩和法解析の分散処理アプリケーションで、反復計算クライアン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 6.3 に示す。図 6.3 から、最小二乗法を

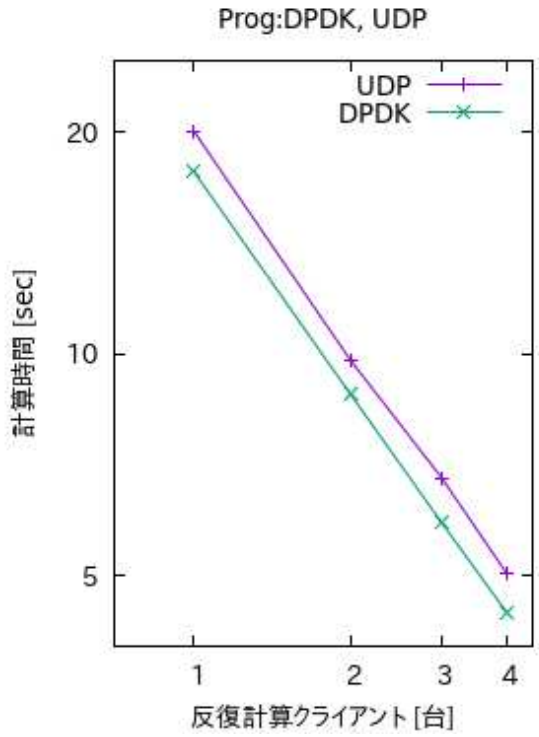

図 6.3: 「DPDK を用いた緩和法解析の分散処理アプリケーションと UDP を用いた緩和法解析の分散処理アプリケーションで反復計算クライアント台数と計算時間の関係」（対数スケール、一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値）

用いて、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 を用いた緩和法解析の分散処理アプリケーションで、台数効果によるグラフの傾きは、-0.985 であった。DPDK を用いた緩和法解析の分散処理アプリケーションで、台数効果によるグラフの傾きは、-0.993 であった。DPDK の方が UDP より、理想的な傾きであることから、DPDK はオーバーヘッドが小さいといえる。そのため、反復計算クライアント 4 台まではグラフの傾きに直進性があることから、DPDK が UDP に比べて速いことが予想される。反復計算クライアント台数を増やしていくことで、一定時間に流れるパケット数が多くなることが予想できる。この一定時間に流れるパケット数が多くなる予想から、通

信オーバーヘッドの影響が台数効果を打ち消すか上回るようになることが予想できる。ただし、本研究で用意可能な実験機の台数では、通信オーバーヘッドの限界に達する実験を行うことは出来なかった。

スペックが異なる実験機を含めた実験結果の比較を行った。DPDK を用いた緩和法解析の分散処理アプリケーションと UDP を用いた緩和法解析の分散処理アプリケーションのそれぞれの反復計算クライアントが 1 台の時を基準に、台数を増やすことで何倍の処理能力が得られたかのグラフを図 6.4 に示す。

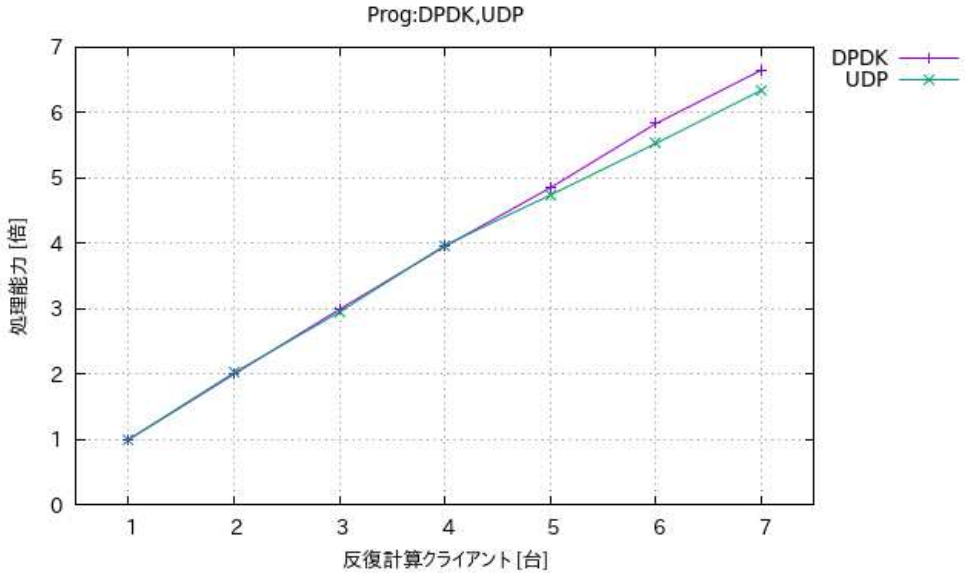

図 6.4: 「DPDK を用いた緩和法解析の分散処理アプリケーションと UDP を用いた緩和法解析の分散処理アプリケーションでそれぞれの反復計算クライアントが 1 台の時を基準とした時の処理能力と反復計算クライアント台数の関係」（一辺の行列サイズ 30000、要素数 20 個、5 回実行した平均値）

6.5 1 台でのスレッドと分散処理の比較

1 台でスレッドによる分散処理アプリケーションと、DPDK を用いた緩和法解析の分散処理アプリケーション、UDP を用いた緩和法解析の分散処理アプリケーションの 3 パターンを比較した。1 台 (実験機 1) でスレッドによる分散処理アプリケーションでは、収束判定を 1 スレッド、反復計算を 4 スレッドの計 5 スレッドで実験を行った。DPDK を用いた緩和法解析の分散処理アプリケーションと、UDP を用いた緩和法解析の分散処理アプリケーションでは、収束判定ホストを 1 台、反復計算クライアントを 4 台の計 5 台で実験を行った。これは、収束判定と反復計算を行う CPU コア数の合計が同じである。一辺の行列サイズを変えて行った実験の結果を図 6.5 のグラフに示す。図 6.5 のグラフでは、1 台でスレッドによる分散処理アプリケーションで、一辺の行列サイズ 40000 の時、スワップが発生し計算時間が大幅に長くなったため、描いていない。図 6.5 から、いずれの行列サイズ

でも DPDK による分散処理が速い結果となった。また、1 台でスレッドによる分散処理を行った場合よりも UDP による分散処理が速い結果となった。一辺の行列サイズ 30000 の時、DPDK を用いた緩和法解析の分散処理アプリケーションは、1 台でスレッドによる分散処理アプリケーションに比べて、計算時間が 24.6%短縮されている。収束判定と反復計算を行う CPU コア数の合計が同じ場合に、1 台のコンピュータ内で行うよりもオーバーヘッドが少ないことは、分散処理が緩和法解析に適しているということが考えられる。

6.6 DPDK による高速化

DPDK を用いた緩和法解析の分散処理アプリケーションと、UDP を用いた緩和法解析の分散処理アプリケーションで、反復計算クライアントでの反復計算の周回数の確認を行った。反復計算クライアントでの反復計算の周回数とは、反復計算クライアン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。

反復計算クライアントで、一回の共有データの送信を行うまでにする計算量、計算にかかる時間は同じである。DPDK を用いた緩和法解析の分散処理アプリケーション、UDP を用いた緩和法解析の分散処理アプリケーションで、一回の共有データの送信までに行う計算量、計算にかかる時間が同じとなると、通信による差が、計算時間の影響である。

DPDK を用いた緩和法解析の分散処理アプリケーション、UDP を用いた緩和法解析の分散処理アプリケーションで、反復計算クライアントでの反復計算の周回数を表 6.2 に示す。表 6.2 の DPDK と UDP それぞれの周回回数は、各収束判定クライアントの周回回数の平均である。

表 6.2:「DPDK を用いた緩和法解析の分散処理アプリケーションと、UDP を用いた緩和法解析の分散処理アプリケーションで反復計算クライアントでの反復計算の周回回数」（要素数 20 個、5 回実行した平均値）

一辺の行列サイズ	反復計算クライアントの台数	DPDK の周回回数 (回)	UDP の周回回数 (回)
30000	4	21.5	23.0
30000	7	21.4	23.1
50000	4	21.3	23.0
50000	7	21.4	23.2

表 6.2 から、DPDK は UDP に比べて少ない周回回数で済んでいる。これは、DPDK の高速通信により共有データの更新が速くなったことで、計算の加速が起きたと考えられる。このように計算の加速による効果から、DPDK は計算時間の短縮行われていると考えている。

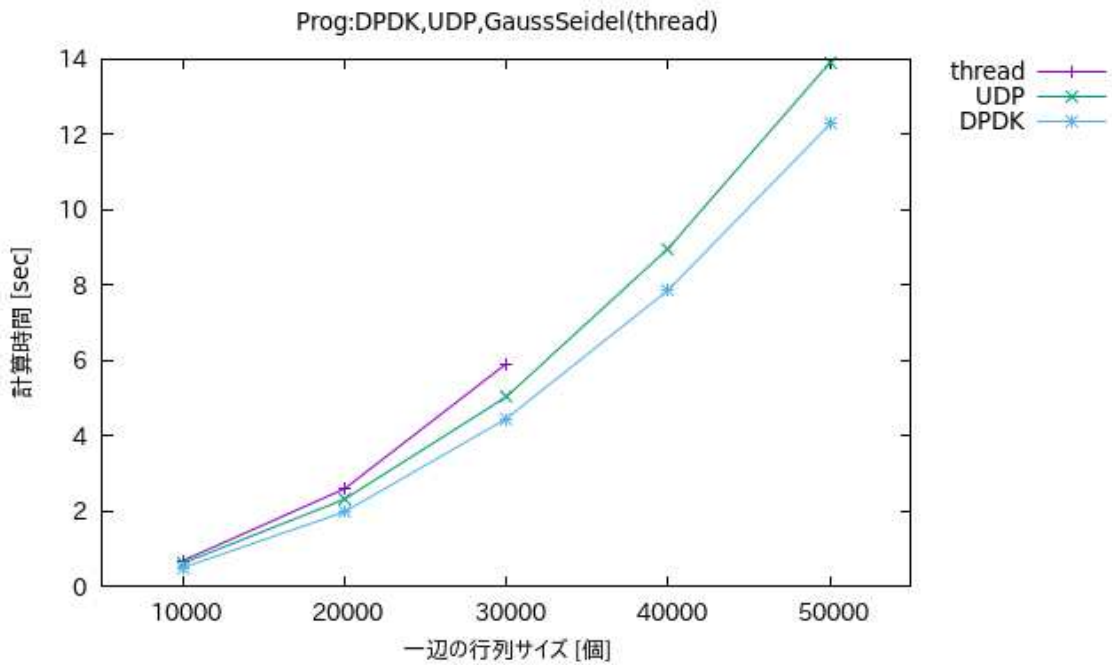

図 6.5: 「1 台でスレッドによる分散処理アプリケーションと、DPDK を用いた緩和法解析の分散処理アプリケーション、UDP を用いた緩和法解析の分散処理アプリケーションでの行列サイズと計算時間の関係」（収束判定 1 、反復計算 4、要素数 20 個、5 回実行した平均値）

第7章　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を用いた緩和法解析の分散処理アプリケーションの作成前に、緩和法解析の分散処理アプリケーションでDPDKを扱いやすくするオリジナルのライブラリの作成を行った。DPDKが提供するライブラリは、NICなどを高速に扱うための低レイヤ向け関数が細分化されている。さらに、プリミティブなEtherフレームを扱うことから、アプリケーションを作成するためのプログラミングが煩雑である。そのため、パケットを送信する、受信するという時に、システムコールが使えない。プログラミング時にヘッダーの作成や、Etherフレームからのデータの書き込み、読み出しまで面倒を見る必要がある。これらの理由により、緩和法解析の分散処理アプリケーションを作成しやすくするためのオリジナルのライブラリの作成を初めに行った。

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って実験を行った。緩和法解析の分散処理アプリケーションでは、行列データの配布等に信頼性のある通信が必要である。UDPを用いた分散処理アプリケーションでは、同じネットワークでTCPによる信頼性のある通信を使えていた。DPDKにはTCP/IPスタックがないため、ネットワークを2重化し、DPDKによる通信と、TCPによる通信を行うこととした。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ーに限って、L2レベルの通信を行うデータグラムを作成した。

UDPを用いた緩和法解析の分散処理アプリケーションと、DPDKを用いた緩和法解析の分散処理アプリケーションとで、同じ条件での実験を行った。この実験により、以下の

- 1つのデータグラムで送る要素数
- 反復計算クライアントの台数
- 行列サイズ

の観点から、比較を行った。

共有データの更新による計算の加速と、通信オーバーヘッドはトレードオフである。UDP を用いた分散処理アプリケーションでは、1 つのデータグラムで送る要素数を減らすと極端に遅くなってしまう。DPDK を用いた分散処理アプリケーションでは、1 つのデータグラムで送る要素数を減らしてもほぼ遅くならなかった。UDP が DPDK に比べて 1 つのデータグラムで送る要素数が少ない時に差が出るのは、通信オーバーヘッドの影響であるといえる。

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 を用いた場合よりも高速化することに成功した。通信オーバーヘッドを小さくするために、更新値をまとめて送らなけらばならない。更新値を多くまとめるすぎると、共有データの更新の遅れにより収束が遅くなり、計算時間が長くなってしまう。DPDK では UDP を用いた場合に比べて、一度に送る共有データの要素数をいくつにした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 で反復計算クライアントを 7 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ところ、DPDK は UDP に比べて、15.3%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た。また、DPDK が UDP よりも計算時間短縮できているのは、通信の高速化により反復計算の周回回数削減による効果もあると考えている。

1 台でスレッドによる分散処理アプリケーションと、DPDK を用いた緩和法解析の分散処理アプリケーション、UDP を用いた緩和法解析の分散処理アプリケーションの 3 パターンを比較した。行列サイズを変えた実験でいずれの場合も、速い方から、DPDK での分散処理、UDP での分散処理、1 台でのスレッドによる分散処理となった。一辺の行列サイズ 30000 の時、DPDK を用いた緩和法解析の分散処理アプリケーションは、1 台でスレッドによる分散処理アプリケーションに比べて、計算時間が 24.6%短縮されている。収束判定と反復計算を行う CPU コア数の合計が同じ場合に、1 台のコンピュータ内で行うよりもオーバーヘッドが少ないことは、分散処理が緩和法解析に適しているということが考えられる。分散処理をより高速に行うためには、DPDK のインタラプト等の通信オーバーヘッドを削減による超高速通信は有効であるといえる。

本研究では DPDK の超高速通信を用いた計算の分散処理アプリケーションを目的として、多元連立一次方程式の分散処理アプリケーションを題材に、計算の高速化を目標とした。今回の分散処理の結果から緩和法解析のように収束問題であり、非常に更新の多い計算問題においては、非常に DPDK の超高速通信が有効に活用できた。今後はその他の収束問題により広く活用できる枠組みを考えていきたい。

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クライアントで計算に使用した CPU コアは 1 コアである。反復計算クライアントでの計算を高速化させるために、反復計算クライアン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。

参考文献

[1] Data Plane Development Kit(DPDK), https://www.dpdk.org/

[2] Rajesh R, Kannan Babu Ramia, Dr. Muralidhar Kulkarni, Integration of LwIP stack over Intel DPDK for high throughput packet delivery to applications, 2014 Fifth International Symposium on Electronic System Design

[3] 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, 上智大学理工学¿研究科・電気電子工学専攻, 1992 年度

[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 年度

[5] Sophie PENG-CASAVECCHIA, Kohei NAKAMURA, and Kazumasa KISHIKI, Performance Evalutions of Network Based Optimisations for Change-Point Detection, 電子情報通信学会技術研究報告 (117 巻 153 号 P209-214), 電子情報通信学会 (2017.7.26) 報通信学会 (2017.7.26)

[6] 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, 情報処理学会第 45 回 (平成 4 年後期) 全国大会, 1992 年

[7] 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 年度

[8] 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 年度

[9] Munenori MAEDA, Yuta TAMURA, and Yuki MATSUO, High Performance I/O Processing Using Kernel-bypass and Light-weight User-level Threading by RMI on DPDK, 電子情報通信学会技術研究報告 (116 巻 117 号 P187-191)

[10] Fumihiko Sawazaki, Noritaka Horikome, and Naoki Takada, A study about SPP design with countermesure of data buffer overflow NetroSphere: Towards the Transformation of Carrier Networks, 電子情報通信学会技術研究報¿告 (117 巻 459 号 P23-25)、電子情報通信学会 (2018.3.1) する

[11] Kazumasa Kishiki, Korechika Tamura, and Hiroki Matsutani, FPGA and DPDK-Based Communication Acceleration Methods for Parameter Servers, 電子情報通信学会技術研究報告 (117 巻 459 号 P99-104)

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。