様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　　元 年　　６ 月 ２５ 日現在

機関番号：１４４０１
研究種目：挑戦的萌芽研究
研究期間：2016～2018
課題番号：１６Ｋ１２４１９

研究課題名（和文）ネットワーク並列性記述のためのディレクティブ処理系OpenMNの実現

研究課題名（英文）OpenMN directive fo network parallelism description

研究代表者
下條　真司（Shimojo, Shinji）

大阪大学・サイバーメディアセンター・教授

研究者番号：００１８７４７８

交付決定額（研究期間全体）：（直接経費）　　2,600,000円

研究成果の概要（和文）：本研究では、OpenFlow相互結合網で構成したクラスタシステムを対象とし，Message Passing Interface (MPI)のプログラミングモデルを想定したプロセス間通信に対して，分散並列システムを構成するネットワーク資源に対する並列処理を指定したプログラミングを可能とするネットワークディレクティブ(network directive)処理系OpenMN(Open Multi-Networking)の提案・実装を行なった。特に、集団通信でのコミュニケータ分割時のネットワーク並列処理に着目し、OpenFlow制御を行うことで、効率的なコミュニケータ分割をおこなう手法を提案した。

研究成果の学術的意義や社会的意義
高性能計算機システムを構成する相互結合網は、低遅延・広帯域なネットワーク性能が求められる。一般的に、今日の相互結合網はハードウェア的な制御を行われる。そのため、高性能計算機システムの規模が大きくなると、そのコストは大きくなる。本研究では、相互結合網のソフトウェアでのトラフィック制御を行うことで、効率的な高性能計算の実現を目指した挑戦的研究である。本研究での成果は、いまだ実用にむけて改善の余地があるが、挑戦的なアプローチそのものに学術的意義がある。

研究成果の概要（英文）：This research proposed and implemented Open MN, network directive, which allows developers to specify parallelism of network resources composing a distributed parallel system, targeting inter-process communication based on MPI in OpenFlow-based Cluster system. In particular, we focus on MPI_Comm_Split operation for MPI collective communications and applied OpenFlow control to the OpenFlow-based interconnect for efficient traffic control.

研究分野：情報ネットワーク

キーワード：ネットワークアーキテクチャ　SDN　並列計算

様　式　C－１９、F－１９－１、Z－１９、CK－１９（共通）

１．研究開始当初の背景

様々な構成のクラスタシステムが提案される中，計算ノード間の相互結合網（インターコネクト）を，他のデバイス（ストレージ等）通信網と共有するクラスタシステムも増加している．共有型のインターコネクトは安価に構成することができるが，プロセス間通信（MPI 通信）だけでなく，ストレージ I/O 通信もインターコネクト上に発生し，パケット混雑な状態，パケット輻輳が発生しやすい状態となる．こうしたクラスタシステムのインターコネクトの問題に対し，Software Defined Networking (SDN) を活用した，輻輳を回避しようとする研究開発が報告されつつある．SDN を活用して構築された相互結合網を有するクラスタシステムでは，相互結合網内の輻輳を回避し，アプリケーションの実行性能の低下を抑制することが可能であると示されている．

一方，高性能計算機において，並列処理技術を用いることで処理性能の向上を図るのが一般的になりつつある．CPU や GPGPU のメニーコアに代表されるように，コア数を増やすことによる CPU の性能向上技術が多く利用され，プロセッサのもつプロセッサコアの並列性が急速に向上している．事実，世界の高性能計算機の処理性能上位 500 位までをランキング化した TOP500 の計算機は，約 87.4%がクラスタシステムであることから，メニーコアの計算ノードをクラスタシステムとして構成する場合がほとんどであるといえる．またクラスタシステムの相互結合網は，システムごとの設計思想により多様化しており，例えば京コンピュータでの相互結合網トポロジは６次元トーラスメッシュ，Oakridge National Laboratory の summit は Non-blocking ファットツリー，Titan は，３次元トーラスと，それぞれのシステムで想定しているアプリケーション通信特性を考慮した設計が取られている．

２．研究の目的

本申請研究では、複数の計算機をネットワークで接続して構成する分散並列システムの性能向上を目指し、 (1) ネットワークディレクティブ(network directive) API、(2)ディレクティブプリプロセッサ、(3)ネットワーク制御エンジンを開発し、これらを中核技術としてネットワークディレクティブ処理系 OpenMN(Open Multi-Networking)を実現することを目的とした。特に、OpenFlow 相互結合網で構成したクラスタシステムでの，Message Passing Interface (MPI)のプログラミングモデルを想定し，クラスタシステムで用いられるプログラミングモデル，Message Passing Interface (MPI)を用いたプロセス間通信による並列処理での，通信輻輳を抑制し，かつ開発者がシステムごとの相互結合網トポロジを意識せずに開発可能とする開発言語の開発を目的とした。

３．研究の方法

クラスタシステムでのプログラミングモデルは，通常，計算ノード間は Message Passing Interface (MPI)を用いたプロセス間通信による並列処理，計算ノード内では，OpenMP など共有メモリを用いたデータ交換による並列処理が行われる．MPI によるプロセス間通信ライブラリは，多対多の通信も行うことができ，そうした集団通信は，コミュニケータ (Communicator) を指定して記述され，指定されたコミュニケータに含まれるプロセス間でマルチキャストやデータの集約・演算などを行う．コミュニケータとは，MPI の通信が行われるプロセスの空間的範囲である．コミュニケータをどのように分割するか決定する際，実行するシステムの構成を意識し，通信遅延を抑える工夫を含めた実装をする必要がある．つまり，クラスタシステムの計算機システムアーキテクチャ，インターコネクトのトポロジや構成はシステム毎に異なり，開発者は環境に応じて，コミュニケータ分割方法を検討する必要がでてくる．

本研究では，コミュニケータ分割が行われる MPI アプリケーションでの，集団通信の遅延時間増加を抑制し，MPI アプリケーションの実行時間を短縮することを目指し，開発者がインターコネクトのトポロジなど環境依存な問題や通信遅延などの問題を考慮することがないプログラミングモデル，ネットワークディレクティブを提案した。

コミュニケータ分割を明示的に，かつ，システム構成，トポロジに影響されない，いわばシステム依存を隠蔽化できるディレクティブ命令セットとして実装した．それをネットワーク指向型ディレクティブと定義し，OpenMN というソフトウェアとして開発した．本申請研究では，定義した命令セットをネットワークディレクティブとし，開発したソフトウェアスィートを OpenMN とする．ネットワークディレクティブは，プリコンパイラで解釈され，プリミティブな言語に置き換えられる．置き換えられたソースコードは，OpenFlow 相互結合網を制御するコントローラに接続し，クラスタのトポロジ，構成情報を得たのち，コミュニケータ分割を行う．

ネットワークディレクティブ

ネットワークディレクティブとプリコンパイラが，実行ファイルを生成する流れ（init 指示文，split 指示文）を図１に示す．

#pragma omn init

init 命令は，MPI アプリケーションにおいて，MPI_Init() が記述された直後の行に挿入される．Init 命令は，プリコンパイラに解釈されると，omn_init 関数に置き換えられる．omn_init 関数には以下の処理が含まれている．

- 各プロセスでランクと IP アドレスを取得
- MPI アプリケーションの識別 ID を生成
- 取得したランク，IP アドレス，生成した識別 ID を PMM へ送信

PMM は，OpenFlow コントローラとして動作しているため，同時に動作している複数の MPI アプリケーションからの通信が発生する．そのため，計算ノード予約・確保，トポロジによる最短経路探索などを行う必要があり，ランクおよび IP アドレスを PMM に教える必要があるが，同一の MPI アプリケーションであると識別する必要があるため，init 命令の中で識別 ID を生成し，PMM に送信する．

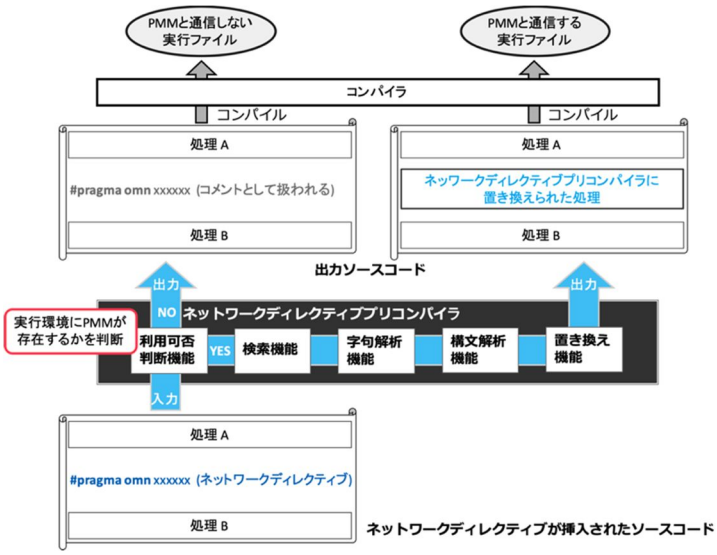

図 1　ネットワークプリコンパイラの処理の流れ

#pragma omn split

split 命令は，MPI_Comm_split() が記述された直前の行に挿入される．
開発者は，必要なコミュニケータの数を comms の位置に記述する．
プリコンパイラは，split 命令を解釈すると，直後に記述された MPI_Comm_split() の引数に基づき，実装した MPI_Omn_split 関数にソースコードを置き換える．MPI_Omn_split 関数には以下の処理が含まれている．

- ランクを PMM に送信
- 必要なコミュニケータ数を PMM へ送信
- PMM で計算されたコミュニケータ識別子を受信し，コミュニケータを分割

必要なコミュニケータ数の送信では，一つのプロセスからのみ送信される．コミュニケータの分割の際には，受信したコミュニケータ識別子を引数とし MPI_Comm_split() を呼び出す．これにより，MPI アプリケーションの各プロセスは，新たなコミュニケータに所属して実行することとなる．置き換え処理を擬似コードとしてソースコード 1 に示す．また split 指示文のサンプルソースコードをソースコード 2 に，split 指示文のプリコンパイラ処理後をソースコード 3 に示す．

```
comms : split 指示文に記述されたコミュニケータの分割数
SEACH : split 指示文が挿入された行数を取得する関数
LEXICAL : 入力された文字列をトークンのリストを生成する関数
DEL : 指定された行を削除する関数
ADD : 指定された行に入力された文字列を挿入する関数
ERROR : エラー出力し，置き換えを行わずに終了
function replace directive
        i ← SEARCH
        List ← LEXICAL(i + 1)
        if List[1] == "MPI Comm split" then
                for j in 4, 6 do
                        if List[j]≠ ", " then
                                ERROR
                        end if
                end for
                StrA ← List[3]
                StrB ← List[9]
                StrC ← "MPI Omn split(" +StrA+ ", " + comms+","+StrB+");"
                DEL(i)
                DEL(i + 1)
                ADD(i, StrC)
        else
                ERROR
        end if
end function
```

ソースコード１　split 指示文の置き換え処理

```
#pragma omn split number
MPI_Comm_split(comm, color, key, *newcomm);

MPI_Bcast(buf, count, type, 0, newcomm);
/*
 数値計算
 */
MPI_Gather(buf, count, type, buf, count, type, 0, newcomm);
```

ソースコード２　split 指示文記述例

```
MPI_Omn_split(comm, number, *newcomm) {
    get_optional_color(comm_number) {
        return opt_color;    // PMM で決定した識別子
    }
    MPI_Comm_split(comm, opt_color, key, *newcommon);
}

MPI_Bcast(buf, count, type, 0, newcomm);
/*  数値計算 */
MPI_Gather(buf, count, type, buf, count, type, 0, newcomm);
```

ソースコード３　split 指示文 プリコンパイラによる置き換え後のコード

４．研究成果

本申請研究では，OpenFlow 相互結合網で構成したクラスタシステムでの MPI プログラム通信間の通信輻輳を避ける OpenMN ディレクティブ言語の開発を行った．評価では，MPI 集団通信で行われるプロセスの MPI_Allreduce 関数と MPI_Scatter 関数で通信の遅延時間の測定を行い，提案手法による遅延抑制できることを確認した．

評価実験では，コミュニケータの分割を行う MPI プログラムにネットワークディレクティブを挿入した MPI アプリケーションを作成し，クラスタシステム上でジョブとして実行する．クラスタシステム上では，複数のユーザからのジョブが実行されていることを想定し，要求する計算ノードの数（並列数）が異なる複数のジョブと併せて，ネットワークディレクティブを挿入した MPI アプリケーションをクラスタシステムに対して割り当てる．実験で使用するジョブの種類を表１に示す．ジョブ A が，コミュニケータの分割を行う MPI アプリケーションであり，今回の評価計測を行うジョブである．一方で，ジョブ B，ジョブ C，ジョブ D は，コミュニケータの分割を行わず，MPI_Alltoall()を(並列数 10000) 回繰り返すプログラムである．4 種類のジョブは，ランダムな順序でジョブキューに投入される．また評価環境を図２に示す．

表１　評価で利用するジョブ．

ジョブ名	並列数	コミュニケータ分割数
ジョブ A	8	2
ジョブ B	6	なし
ジョブ C	3	なし
ジョブ D	2	なし

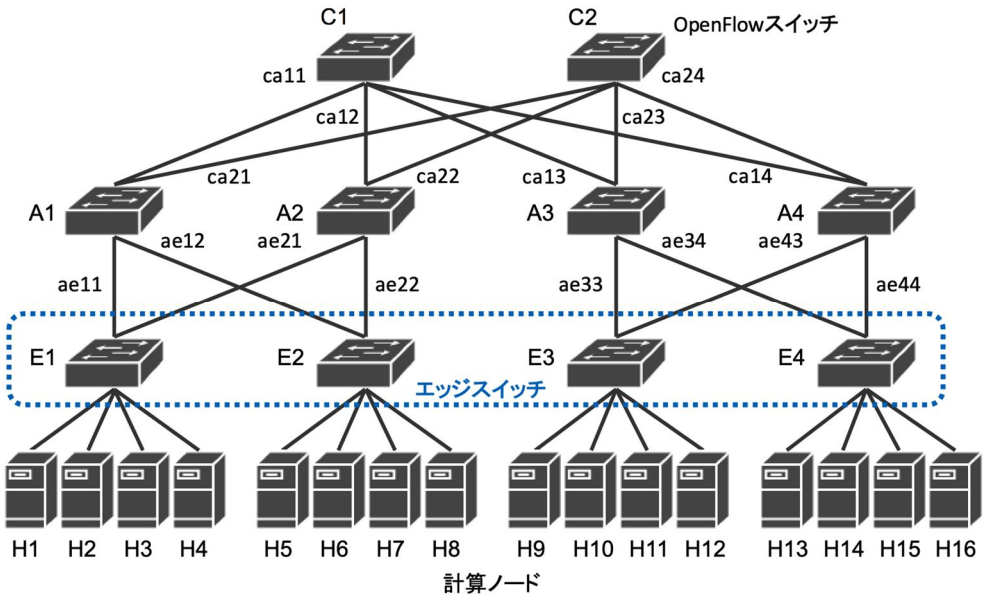

図２　クラスタシステム

MPI アプリケーションの実行時，分割後のコミュニケータ内の集団通信の遅延時間を，提案機構を利用する場合，しない場合とで比較する．MPI アプリケーションには，MPI ベンチマークプログラム OSU Micro Benchmarks を使用した．OSU Micro Benchmarks の内，使用したプログラムは，MPI_Allreduce() の遅延時間を計測する osu_allreduce と，MPI_Scatter()の遅延時間を計測する osu_scatter である．これらのプログラムにコミュニケータの分割処理を加えた MPI アプリケーションをジョブ A としてそれぞれクラスタシステムで実行し，遅延時間を計測する．

表１に示した通り，osu_allreduce と osu_scatter の実行時の，並列数は 8 であり，コミュニケータの分割数は 2 である．osu_allreduce を使用した場合と，osu_scatter を使用した場合の遅延時間を比較した結果,osu_allreduce と osu_scatter ともに，提案機構を利用した場合に遅延時間が抑制されていることを確認した. さらに, OSU Micro Benchmarks の osu_alltoall など，他の集団通信で計測した場合も同様に遅延時間増加が抑制されていた.

５．主な発表論文等

〔雑誌論文〕（計 3 件）

(1) Keichi Takahashi, Susumu Date, Dashdavaa Khureltulga, Yoshiyuki Kido, Hiroaki Yamanaka, Eiji Kawai, Shinji Shimojo, “UnisonFlow: A Software-Defined Coordination Mechanism for Message-Passing Communication and Computation”, IEEE Access, vol. 6, no. 1, pp. 23372-23382, Apr. 2018. [DOI:10.1109/ACCESS.2018.2829532] (査読有)

(2) Kohei Ichikawa, Pongsakorn U-chupala, Che Huang, Chawanat Nakasan, Te-Lung Liu, Jo-Yu Chang, Li-Chi Ku, Whey-Fone Tsai, Jason Haga, Hiroaki Yamanaka, Eiji Kawai, Yoshiyuki Kido, Susumu Date, Shinji Shimojo, Philip Papadopoulos, Mauricio Tsugawa, Matthew Collins, Kyuho Jeong, Renato Figueiredo and Jose Fortes, "PRAGMA-ENT: An International SDN Testbed for a Cyberinfrastructure in the Pacific Rim", Concurrency and Computation: Practice and Experience, Mar. 2017 [DOI: 10.1002/cpe.4138]. (査読有)

(3) Susumu Date, Hirotake Abe, Dashdavaa Khureltulga, Keichi Takahashi, Yoshiyuki Kido, Yasuhiro Watashiba, Pongsakorn U-chupala, Kohei Ichikawa, Hiroaki Yamanaka, Eiji Kawai, Shinji Shimojo, "SDN-accelerated HPC Infrastructure for Scientific Research", International Journal of Information Technology, Volume 22, Number 01, 2016. (査読有)

〔学会発表〕（計　6　件）

(1) 木戸善之, 片岡祐介, 伊達進, 下條真司, "OpenMN：ネットワーク指向型ディレクティブを用いたMPI コミュニケータ分割機構", 日本ソフトウェア科学会 第 16 回ディペンダブルシステムワークショップ, 金沢, 2018 年 12 月

(2) Yohei Takigawa, Keichi Takahashi, Susumu Date, Yoshiyuki Kido, Shinji Shimojo, "A Traffic Simulator with Intra-node Parallelism for Designing High-performance Interconnects", The 2018 International Conference on High Performance Computing & Simulation (HPCS 2018), July 2018. [DOI:10.1109/HPCS.2018.00077]

(3) Arata Endo, Ryoichi Jingai, Susumu Date, Yoshiyuki Kido, Shinji Shimojo, "Evaluation of SDN-based Conflict Avoidance between Data Staging and Inter-Process Communication", The 2017 International Conference on High Performance Computing & Simulation (HPCS 2017), pp. 267-273, Genoa, Italy, July 2017. [DOI: 10.1109/HPCS.2017.48]

(4) Masaharu Shimizu, Yasuhiro Watashiba, Susumu Date, Shinji Shimojo, "Adaptive Network Resource Reallocation for Hot-spot Avoidance on SDN-based Cluster System", NetCloud 2016 workshop, 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom2016), Dec. 2016. (10.1109/CloudCom.2016.0105)

(5) Takuya Yamada, Keichi Takahashi, Masaya Muraki, Susumu Date, Shinji Shimojo, "Network Access Control Towards Fully-controlled Cloud Infrastructure", PhD. Consortium, 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom2016), Dec. 2016. (10.1109/CloudCom.2016.0076)

(6) Susumu Date, Takashi Yoshikawa, Yasuhiro Watashiba, Yoshiyuki Kido, Shinji Shimojo, Masahiko Takahashi, Masaki Kan, Masaki Muraki, "A Proposal of On-demand Staging leveraging Job Management System and Software Defined Networking", Workshop on Sustained Simulation Performance (WSSP), Stuttgart, Germany, Dec. 2016.

６．研究組織

(1)研究分担者

研究分担者氏名： 伊達　進

ローマ字氏名： DATE, Susumu

所属研究機関名： 大阪大学

部局名：サイバーメディアセンター

職名：准教授

研究者番号（8 桁）：20346175

(2)連携研究者

連携研究者氏名：木戸善之

ローマ字氏名：KIDO, Yoshiyuki

所属研究機関名： 大阪大学

部局名：サイバーメディアセンター

職名：講師

研究者番号（8 桁）：70506310