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

平成　２９　年　　５　月　２６　日現在

機関番号：　８２４０１
研究種目：　基盤研究(C)（一般）
研究期間：　2013～2016
課題番号：　２５３３０１４８

研究課題名（和文）大規模かつ高速な並列計算を支える高スケーラブルな並列入出力に関する研究

研究課題名（英文）Study of highly scalable parallel I/O systems for high performance computing with huge data management

研究代表者

辻田　祐一（Tsujita, Yuichi）

国立研究開発法人理化学研究所・計算科学研究機構・開発研究員

研究者番号：７０３６０４３５

交付決定額（研究期間全体）：（直接経費）　　3,800,000円

研究成果の概要（和文）：MPIにおける入出力インタフェースであるMPI-IOの代表的な実装であるROMIOでは、高速な並列入出力機能を有しているが、ノード内に複数のプロセスを起動する場合において十分な性能を発揮できない問題があった。そこで本研究では、ファイル入出力を行うプロセスの配置や、ファイル入出力操作に合わせて行うデータ通信の順番に関して、ノード間・ノード内のプロセス配置に配慮した最適化手法をROMIOに適用し、ユーザが設定したプロセス配置に関係無く、高い入出力性能が発揮できる実装を実現した。

研究成果の概要（英文）：ROMIO is well known as one of the representative MPI-IO implementations, and it provides high performance parallel I/O. However, it can not achieve enough I/O performance when we have multiple processes per node. In this research work, we have introduced topology-aware treatment in layout of processes that play file I/O and data aggregation task associated with the file I/O task in order to improve I/O performance of ROMIO.
Consequently, we have achieved high scalable parallel I/O in ROMIO using the proposed scheme with any kind of process layout.

研究分野：　並列計算機システム

キーワード：　ネットワークコンピューティング　並列入出力　PCクラスタ　MPI

１．研究開始当初の背景

近年のCPUやメモリなどの能力向上に伴い、より大規模なPCクラスタが構築・利用されるようになってきていた。そのような背景において、ファイルI/Oがボトルネックになるケースが多く発生してきており、並列ファイルシステムの機能向上や高度利用技術の確立が強く求められるようになってきていた。並列計算で標準的に用いられる通信インタフェースであるMPI（Message Passing Interface）では、ファイルI/Oのためのインタフェースである MPI-IO があり、代表的な実装としてROMIOと呼ばれるライブラリ実装が広く利用されている。特に、大規模計算で広く利用されているHDF5やPnetCDFと呼ばれるアプリケーション指向の入出力ライブラリでもMPI-IOの集団型入出力が多用されているが、このようなケースでは多次元配列データを効率良く処理する必要があり、ROMIO では図 1 に示すような Two-Phase I/O（以下、TP-IO）と呼ばれる最適化手法が用いられている。

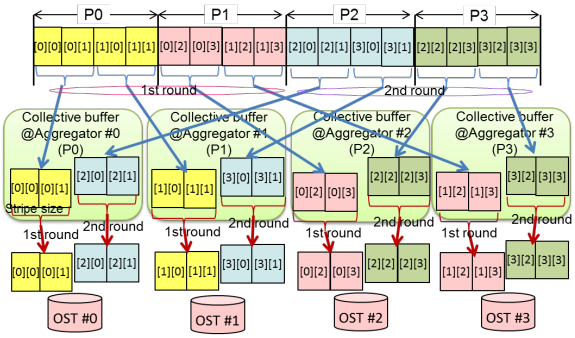


図 1：Two-Phase I/O の処理フロー

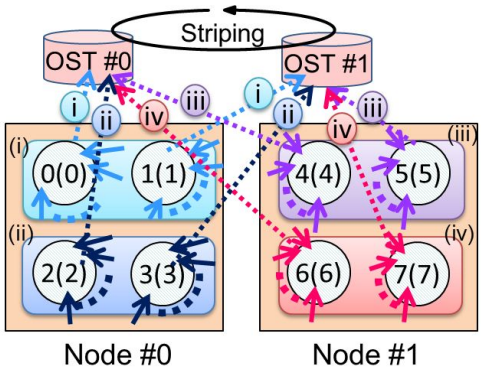
この手法では、各プロセスでファイルアクセス領域が連続になるように、アクセス領域を均等に分割し、データ通信による並べ替えとファイルI/Oの組合せで高速化を実現している。しかしながら、この実装では、ファイルアクセスを行うプロセス（アグリゲータ）の配置やアグリゲータにデータを集める際の通信順などについて、ノード内・ノード間の違いなどに配慮した実装になっておらず、アクセス先の並列ファイルシステムの特徴を活かしきれていない問題があった。

２．研究の目的

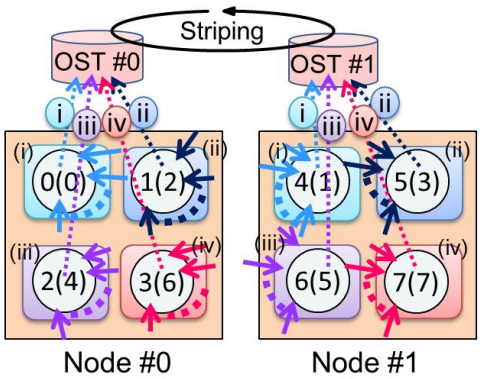
1 で述べた課題を解決するべく、ROMIO 内部の TP-IO に対して、特にプロセスのノード間配置や PC クラスタで広く用いられている Lustre へのファイル I/O のパターンを精査し、小規模から中規模の PC クラスタで有用な最適化実装を目指すこととし、この成果が更に大きな規模のケースでの最適化への足掛かりとなることを本研究の主目的とした。

３．研究の方法

近年の CPU コア数や PC クラスタのノード数の増加などに対して、アグリゲータのノード間配置やアグリゲータにデータを集める通信の順番など、ノード内・ノード間を意識した最適化実装の実現を目指すこととした。そこで本研究では、アグリゲータの配置や通信発行順に着目し、ノード構成を意識した最適化実装を目指した。並列ファイルシステムの一つである Lustre におけるストライピングアクセスに対して、デフォルトの場合と最適化したアグリゲータ配置の例を図 2 に示す。



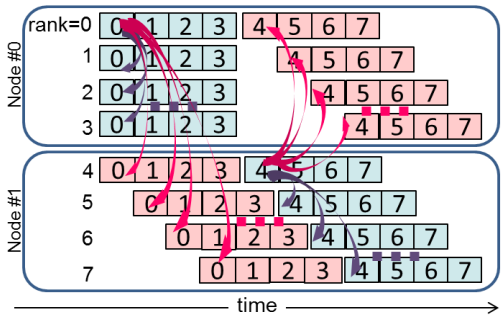
(a) デフォルトの配置



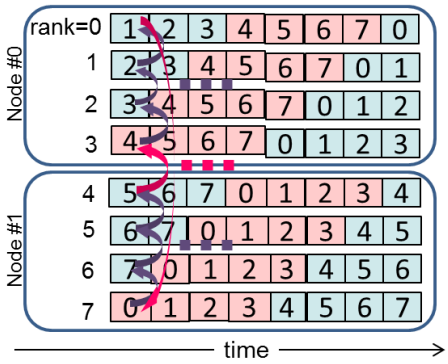
(b) 最適化した配置

図 2：ストライピングアクセスにおける 2 ノード間でのデフォルトと最適化したアグリゲータ配置の例（ノードあたり 4 プロセス）

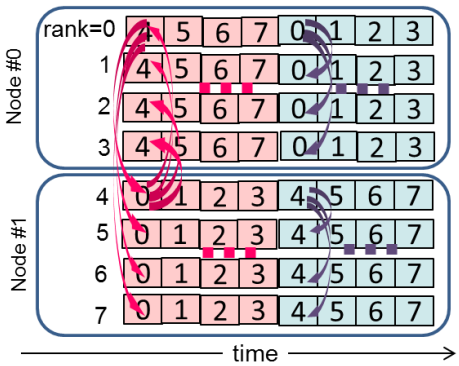
この図の四角で表されるのがノードであり、その中の白抜きの丸はプロセスを表している。さらに丸の中の数字は MPI ランクである。またその横の括弧内の数字はアグリゲータ配置順である。同じストライピングアクセスのラウンドに動作するプロセスを色分けしており、この図の例では 4 ラウンド（図中の(i)から(iv)）が行われている。この図のようにノードあたりに複数のプロセスがあり、それらをアグリゲータとして使う場合、図 2(a)に示すように、デフォルトではランク順でアグリゲータが配置される。



(a) ROMIO のデフォルトの通信パターン



(b) ring 型の通信パターン



(c) ノード間で重ならないパターン
(nd_shift)

(d) ランク配置を配慮したパターン
(nd_rank_shift)

図 3：2 ノードで 8 プロセスある場合でのアグリゲータへのデータ収集フローの例

この場合には、ファイルシステムへのストライピングアクセスにおいて、ノード#0 のアグリゲータ群が先にアクセス（(i)から(ii)）し、次にノード#1 のアグリゲータが動作（(iii)から(iv)）する、という流れになる。この場合、ターゲットとなるストレージサーバへアクセスする際に、ノードからの出口となる通信回線で複数のアグリゲータが同時に使用することになり、通信混雑による性能低下を招く恐れがあった。そこで、図 2(b)に示すように、ノード間でラウンドロビンとなる順にアグリゲータを配置させることで、この図での(i)から(iv)までの各ステップにおいて、各ノードからの通信回線の混雑を緩和し、全体のスループット向上に繋げることができる。

さらに、アグリゲータへのデータ通信の順番においても、ノード内・ノード間を意識した実装を行った。TP-IO での通信は非同期の送受信対の組合せで実現されており、全通信相手への通信関数発行後に完了通知関数によって通信データの送受信が完了する方式を採用している。 図 3(a)に既存の ROMIO のアグリゲータへの通信順の例を示す。この図では図 2 と同様のランク配置の場合での、各々のランクでのデータ通信順を左から右に向かって時間が経過する方向で示しており、各四角内の数字は通信相手のランクを表している。なお、同一ノード内の通信は青色、ノード間の通信は赤色で示している。この図に示すように、各ランクが一斉に相手ランクを 0 から順に通信を発行するパターンとなり、通信の局所化による混雑により、データの送受信完了が遅れるアグリゲータが発生する。また通信レイテンシや通信帯域の異なるノード間通信とノード内通信が同じステップで混在し、全体としてバランスが悪く、全体の通信時間の増加を招く恐れがある。これに対して、改善策として図 3 の(b)から(d)の 3 パターンを提案した。図 3(b)に示す ring 型のパターンは、単純に通信相手先のランクを 1 個ずつずらしながらランク間で通信相手が重ならないようにしているが、ノード内・ノード間の通信が各通信ステップで混在するために、通信時間の短縮が十分に行えない可能性はある。次に図 3(c)に示す方式では、通信遅延の大きいノード間通信を先行させ、かつ通信相手をノード間で重ならないようにしており、同じ方式でノード内通信も行っているが、同じノードにあるランク間で通信相手が重なるため、この部分で通信の局所化による混雑を招く恐れがある。最後に図 3(d)に示すパターンでは、図 3(c)に示す方式に対して、さらに相手ノード内のランク間で通信先をラウンドロビンでずらしながら通信を進めている。同じ要領でノード内通信も実施しており、これによって通信時間の短縮が狙える。その結果、TP-IO 自体の性能も向上するものと期待される。

４．研究成果

ファイル I/O を行うアグリゲータに関して、ROMIO のデフォルトの方式であるランク順で並べる方法と、アクセス対象となる Lustre のストライピングアクセスパターンに合わせノード間をラウンドロビンで配置させる方法を用い、HPIO ベンチマークで評価したところ、図 4 に示すように性能向上を確認した。

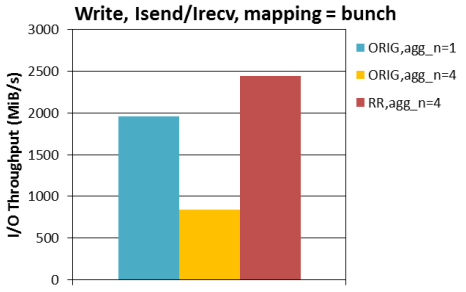


図 4：アグリゲータ配置の違いによる集団型書き込み性能

この評価では、MPI ライブラリとして MVAPICH2 ver.2.1rc1 を用い、内部の ROMIO 実装に対し、提案するアグリゲータ配置最適化の実装を行った。これを用いて InfiniBand（FDR）で繋がれた 4 ノードの PC クラスタに 16 プロセス（4 プロセス/ノード）を起動して、HPIO ベンチによる書き込みを行わせた。この際に、図に示すように ROMIO のデフォルトであるノードあたり 1 個のアグリゲータを起動する場合（図中の ORIG, agg_n=1）、デフォルトのレイアウトでノードあたり 4 個のアグリゲータを起動する場合（図中の ORIG, agg_n=4）、さらに提案手法を用いてノードあたり 4 個のアグリゲータを配置した場合（図中の RR, agg_n=4）の 3 パターンで評価した。その結果、全てのプロセスがアグリゲータとなると、デフォルトのレイアウトでは性能低下を招いているが、提案手法では、3 パターンの中でも、最も高い性能を出せることが確認できた。前者の場合、ノード毎にアグリゲータを配置させたことで、ファイルシステムへのアクセスで通信経路の帯域を共有することによる性能低下などが原因と考えられるが、後者の場合、このような混雑を回避でき、性能が向上したと考えられる。

次に、アグリゲータ群へのデータ収集での通信関数発行順に関する最適化について同様に HPIO ベンチマークで評価を行った。この評価では、東京工業大学の TSUBAME2.5 を用いたが、当該計算機で推奨されている MPI ライブラリが OpenMPI であるため、本実装のベースとして OpenMPI ver.1.8.2 を選択し、内部の ROMIO 実装に対して、上記のアグリゲータ配置最適化と合わせ、通信発行順の最適化実装を行ったもので評価を進めた。当該計算機の 64 ノードを用い、768 プロセス（12 プロセス/ノード）起動して HPIO ベンチマークにより性能評価を行った。その結果を図 5 に示す。

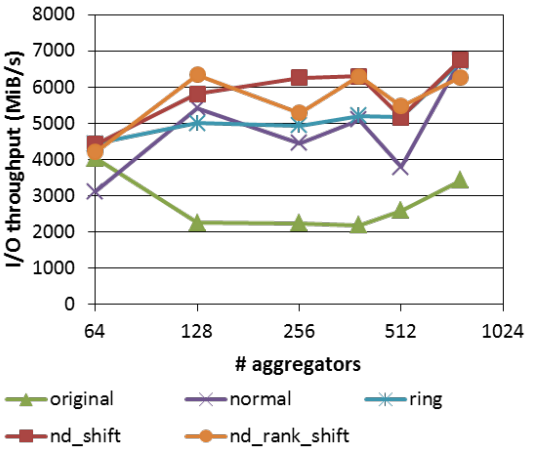


図 5：768 プロセス（12 プロセス/ノード）による HPIO ベンチマークによる通信パターンを変えた場合の入出力性能

ROMIO のデフォルトのランク順の通信パターン（図中の original）では、アグリゲータの数を増やすにつれて性能低下を招いているが、これは、3 で説明した通り、アグリゲータ配置が Lustre のストライピングアクセスに最適化されていないことによるものである。それ以外のパターンでは、ストライピングアクセスに合わせたアグリゲータ配置にしており、アグリゲータ数を増やすにつれて性能向上が確認できたが、通信発行順についてはランク順で行うパターン（図中の normal）に比べて、図 3(b)、(c)および(d)に示すパターン（それぞれ、図中の ring, nd_shift および nd_rank_shift に対応）において、さらに性能が向上する様子が確認できており、各々の最高性能で比較したところ、最大で 67%の向上を達成した。特に nd_shift と nd_rank_shift のパターンで高い性能が達成できた。同様の評価を 32 ノードを用いて 384 プロセス（12 プロセス/ノード）で行った場合でも nd_rank_shift のケースにおいて、最高性能での比較において最大で約 20%の向上を確認できたことから、ノード間のランク配置に配慮した通信発行順の有効性が確認できたと共に、ノード間のランク配置だけでなく、通信相手が重ならないように相手ランクをずらして通信を進めてゆく nd_rank_shift のパターンでの有効性が確認できたと考えている。

以上のことから、アグリゲータの配置並びにデータ通信の発行順についてノード内・ノード間のプロセス配置に配慮した最適化の有効性が確認できた。なお、ここで得られた知見の一部は、「京」コンピュータの MPI-IO ライブラリの性能改善版である EARTH on K と呼ぶ実装開発でも参考にしており、EARTH on K に関しては、バイナリのライブラリのみ

ではあるが、「京」コンピュータの利用者向けに公開している。

５．主な発表論文等
（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計　3 件）

① Y. Tsujita, A. Hori, T. Kameyama, Y. Ishikawa, “Topology-Aware Data Aggregation for High Performance Collective MPI-IO on a Multi-Core Cluster System,“ Proceedings of CANDAR’16, 査読有, pp. 37-46, IEEE, 2016
DOI:10.1109/CANDAR.2016.82

② Y. Tsujita, A. Hori, Y. Ishikawa, “Striping Layout Aware Data Aggregation for High Performance I/O on a Lustre File System,” Proceedings of ISC’15, Lecture Notes in Computer Science, 査読有, Vol. 9137, pp. 282-290, 2015
DOI:10.1007/978-3-319-20119-1_21

③ Y. Tsujita, A. Hori, Y. Ishikawa, “Affinity-Aware Optimization of Multithreaded Two-Phase I/O for High Throughput Collective I/O,” Proceedings of HPCS2014, 査読有, pp. 210-217, IEEE, 2014
DOI: 10.1109/HPCSim.2014.6903688

〔学会発表〕（計　2 件）

① 辻田祐一, 堀敦史, 亀山豊久, 石川裕, “プロセス配置を考慮した通信の最適化による集団型 MPI-IO の高速化, ” 第 154 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2016 年 4 月 25 日, 海洋研究開発機構　横浜研究所（神奈川県横浜市）

② 辻田祐一, 堀敦史, 石川裕, “Lustre のストライピングアクセスパターンに配慮した集団型 MPI-IO の性能向上に向けた試み, ” 第 149 回 情報処理学会ハイパフォーマンスコンピューティング研究会, 2015 年 6 月 26 日, 工学院大学（東京都新宿区）

〔図書〕（計　0 件）

〔産業財産権〕

○出願状況（計　0 件）

名称：
発明者：
権利者：
種類：
番号：
出願年月日：
国内外の別：

○取得状況（計　0 件）

名称：
発明者：
権利者：
種類：
番号：
取得年月日：
国内外の別：

〔その他〕
ホームページ等

６．研究組織
(1)研究代表者
辻田　祐一　（TSUJITA, Yuichi）
理化学研究所・計算科学研究機構・開発研究員
研究者番号：70360435

(2)研究分担者
（　　）

研究者番号：

(3)連携研究者
（　　）

研究者番号：

(4)研究協力者
（　　）