TSUBAME 共同利用 平成 27 年度 学術利用 成果報告書

利用課題名 PC クラスタにおける並列入出力の高速化に関する研究
英文：Study of High Performance Parallel I/O using PC Cluster System

辻田 祐一
Yuichi Tsujita

国立研究開発法人 理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science
URL

邦文抄録（300 字程度）
MPI における入出力インタフェースである MPI-IO の実装として ROMIO が広く利用されている．並列計算で多く用いられる集団型 MPI-IO に対し，ROMIO は Two-Phase I/O（以下，TP-IO）と呼ばれる最適化実装を用いて高速な入出力を実現しているが，ノードあたりに複数のプロセスが配置される場合には現行の TP-IO 実装ではファイル I/O を行うプロセスの配置やデータ通信の方法が最適化されていない．そこで我々はプロセス群のノード間配置を配慮したデータ通信に対する最適化を実装し，64 ノードで 768 プロセスによる並列 I/O で現行の ROMIO に対して約 67%の性能向上を実現した．

英文抄録（100 words 程度）
ROMIO is a well-known MPI-IO implementation, and it has been widely used in data-intensive parallel computing. ROMIO has an optimization named two-phase I/O for collective MPI-IO, which is frequently used in such applications. However two-phase I/O does not have ineffective operations when we have multiple processes per node regarding process layout and data exchanges inside two-phase I/O. In order to improve I/O performance, we propose optimization in data exchange phase of two-phase I/O, and we have achieved up to 67% performance improvements by the optimization relative to the original ROMIO.

Keywords: MPI-IO, ROMIO, Two-Phase I/O, **アグリゲータ**

背景と目的

並列ファイルシステムである Lustre を用いた PC クラスタにおける大規模並列入出力のさらなる高速化を目指すために、我々は MPI-IO 実装である ROMIO における集団型 MPI-IO の高速化に向けた最適化の実装と検証を進めている。特に集団型 MPI-IO においては Two-Phase I/O（以下、TP-IO）と呼ばれる最適化実装が用いられるが、近年の CPU のコア数増加やノード内に複数の CPU ソケットが配置されている現状に合った実装になっていない。そこで我々はこのような計算機環境においても高速化を実現するための実装の改変を行ってきている。小規模な PC クラスタ環境での検証を既に行ってきたが、TSUBAME2.5 を利用することによって大規模化に伴う性能低下に繋がる問題を洗い出すことと，それに対する最適化手法について検討および実装を行った。この中で我々は特に TP-IO 内部で行われるデータ並べ替えに伴う通信の最適化に着目した。現行の TP-IO が単純にランク順に通信相手を変えながらデータ交換を行っているのに対し、我々はノード間のランク配置に配慮した通信順でデータ通信を行うことで TP-IO の高速化が可能であることを確認した。64 ノードで 768 プロセスを起動した場合での HPIO ベンチマークによる評価では、現行の ROMIO に対して約 67%の性能向上を達成した。

概要

TP-IO による集団型 MPI-IO の処理の流れを図 1 に示す。この図では 4 プロセスが全て I/O 処理を行うアグリゲータとして機能している例を示しており、実際にファイルシステムに書き込みを行う前にデータの並べ替えを行った後に書き込みを行っている。このように TP-IO はデータレイアウトの並べ替えに必要な通信とファイル I/O の組合せで高速化を実現している。アグリゲータの配置は基本的にノードあたりに 1 個ずつだが、近年のマルチコア CPU やマルチソケットのノードが広く利用されるにつれて、ノードあたりに複数のプロセスを起動するケー

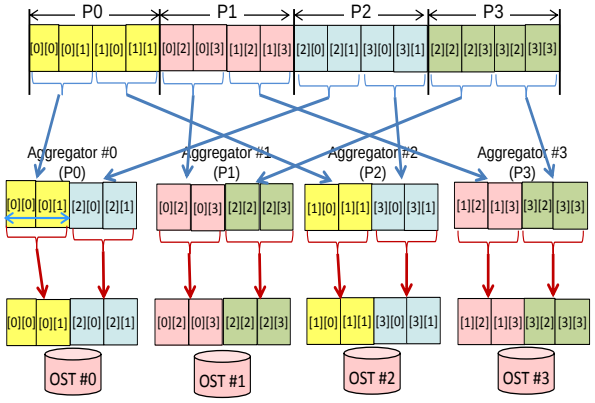


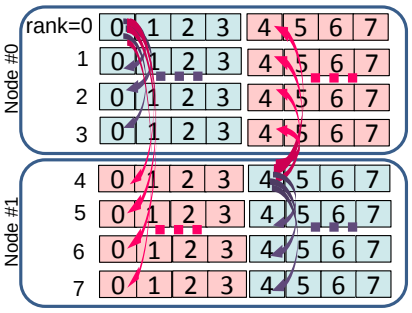
図 1：TP-IO による処理の流れの例

スも多くなり、アグリゲータをノードあたり複数配置させることも可能になってきた。しかしながら現行の TP-IO 実装ではノード毎に詰めて配置するために、特に Lustre のような並列ファイルシステムでストライピング処理を行うケースでは性能向上が望めない。我々はこのような問題に対してアグリゲータの配置をノード間でラウンドロビンにすることで高速化が実現できることを確認している。

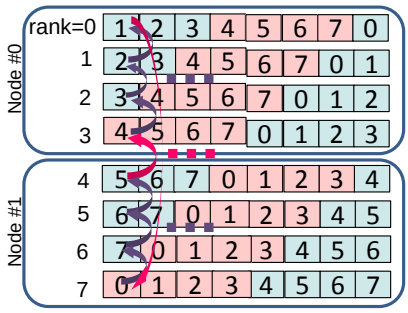
しかしながら、この最適化においてもデータ通信についての配慮は特に行っておらず、現行の TP-IO 実装と同様にランク順に非同期通信関数を発行する方式を用いていた。MPI プロセスのノード間のランク配置によっては、この方式だとランク間で通信時間の偏りが生じ、結果として性能低下を招く恐れがある。そこで我々はノード間のランク配置に配慮した通信関数発行順を適用することで通信コストの低下を実現し、結果として TP-IO を用いた集団型 MPI-IO の高速化に繋げることを目的として改変実装を提案している。特に我々はノード間通信を先行させて、最後にノード間通信を行う方式が最も有効と考え、ノード間配置を考慮しない方式を含めて実装モデルの検証と評価を行っている。その中で本課題では以下の 4 つの方式の比較検討を行った。

1. ランク順に発行する方式（normal）
2. 通信相手ランクを 1 つずつずらしていく方式（pairwise）
3. ノード間通信を先行させ、最後にノード間通信を実施する方式（ただし相手ノード内の通信相手の重なりは考慮しない.）（nd_shift）
4. 上記 3 の方式で、相手ノード内の通信相手の重なりが起きないようにずらして通信を行う方式（nd_rank_shift）

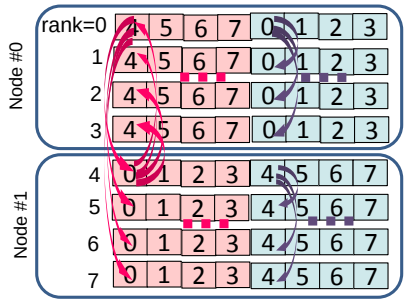
以下、1 の現行の方式を含め、新たに実装・検証を行った 2 から 4 の通信方式について図 2 に示す例を用いて簡単に説明する。



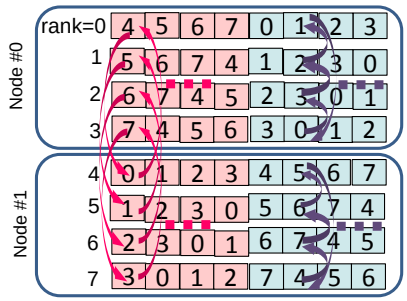
(a) ランク順の発行（現行 ROMIO）



(b) pairwise 方式



(c) nd_shift 方式



(d) nd_rank_shift 方式

図 2：通信発行順の違いによるデータ送受信の流れの例。（四角内の数字は通信相手のランク）

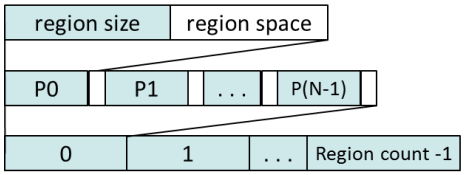
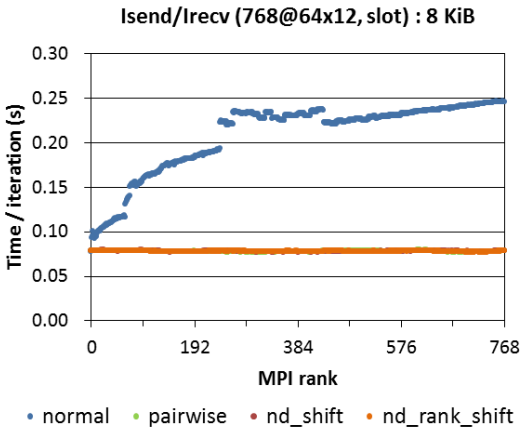


図 3：HPIO ベンチマークでの不連続アクセスパターンの生成例

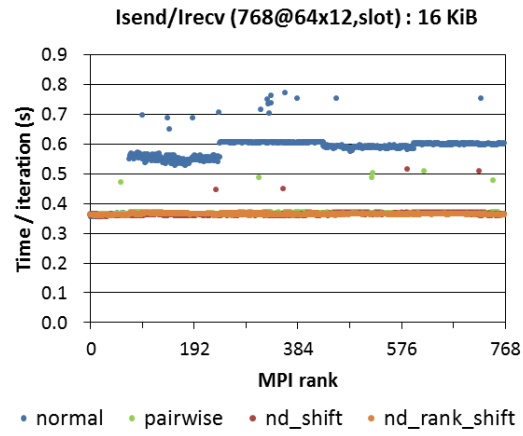
図 2 では 2 ノード間で 4 プロセスずつ起動し、ノード毎にランクを詰めて配置した場合での各々の通信発行順での MPI_Irecv での通信の流れ（時間は左から右へ進む方向）を示している。まず図 2(a)は現行 ROMIO で行われているランク順に発行するケースを示しており、この場合、通信開始時にランク 0 のプロセスに通信が集中し、順に後ろのランクへ通信の集中がシフトしてゆくために、全体的に通信処理の偏りが生じるため通信時間が長くなる問題がある。それに対して図 2(b)に示す stepwise 方式では隣接するプロセスから順に通信相手をずらしてゆくことで上記の通信の偏りを無くしている。しかしながらノード間のランク配置を配慮していないため、ノード間通信とノード内通信が混在してしまい、通信処理の負荷バランスが悪い。これに対して図 2(c)の nd_shift 方式はノード間通信を先行させ、最後にノード間通信を行うことで通信時間の負荷バランスを向上させている。しかしながらこの場合でも同じノード内のプロセス間では同じ通信相手と通信を行うため、通信処理の偏りが生じる可能性がある。これをさらに改善した方法が図 2(d)に示す nd_rank_shift 方式で、この方式では同じノードのプロセス間で通信相手をずらすことによって上述の通信の偏りを解消できる。

結果および考察

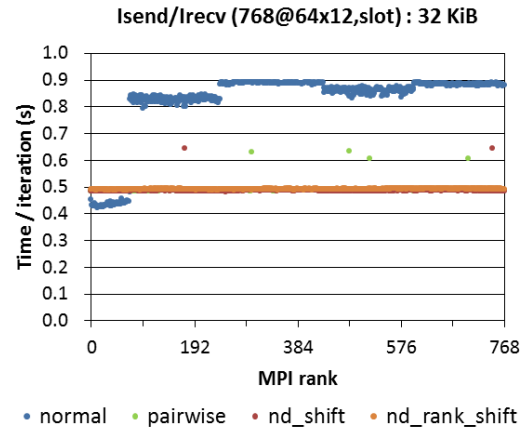
TSUBAME2.5 の Thin ノード 64 台を用い、各ノードに 12 プロセスずつ起動して合計 768 プロセスにより(1)TP-IO 内部の通信を模擬したプログラムによる通信順の影響の調査と(2)Lustre への集団型 MPI-IO による書き込み性能を HPIO ベンチマークにより評価を行った。TP-IO 内部の通信を模擬したプログラムはプロセス間で全体全による MPI_Isend/MPI_Irecv 対によるデータ通信をメッセージ長を変えて計測し、ノード間のランク配置を配慮した通信順の通信時間削減効果を検証した。次



(a) BK 配置、8KB/プロセス対/回



(b) BK 配置、16KB/プロセス対/回



(c) BK 配置、32KB/プロセス対/回

図 4： TP-IO を模擬したプロセス間での MPI_Isend/MPI_Irecv 対によるランクごとの通信時間

に HPIO ベンチマークによる評価では、ユーザが設定するパラメタは region size, region space および region count の 3 つがあり、図 3 に示すような不連続アクセスパ

ターンを生成させて集団型 MPI-IO による書き込み性能を評価した。派生データ型の生成に関しては region size=3,744B、region space=256B、region count=48,000 として評価を行った。使用した Lustre の OST 数はノード数と同じ 64 とした。また TP-IO でのアグリゲータの数を最大で全プロセス数となるまで変化させて I/O 性能を計測しており、各々の計測は 7 回繰り返した中での最大・最小値を除く計測値の平均値を記録し、同様の評価を日時を変えて計測した中から最大の平均値を最終的な性能として記録した。性能評価の日時を変えながら計測した理由は、使用する Lustre はログインノード等と同じネットワーク上に繋がっており、利用者の通信・I/O 等による外乱を受けやすいためである。

通信時間の評価ではランク順に発行するケース（normal）に加え、通信相手のランクを 1 つずつずらしてゆくケース（pairwise）やノード間通信を先行させてノード内通信を最後に行う 2 つのケース（nd_shift および nd_rank_shift）での評価を行った。特にノード毎に詰めてランク配置する場合に通信処理の偏りが顕著になるために、このランク配置での通信発行順の効果を検証した。図 4 にメッセージ長を変えた時の TP-IO 通信を模擬したプログラムでの通信時間（50 回の平均値）を示す。この結果からノード間通信を先行させる手法の優位性が確認でき、特に全体として nd_rank_shift 方式の優位性が確認できた。よって同様の手法が TP-IO 内部でも有効になる可能性が確認できた。

次に 64 ノードを用い 768 プロセスを起動して HPIO ベンチマークによる評価を行った。この評価で得られた I/O 性能を図 5 に示す。この図から nd_rank_shift 方式および nd_shift 方式の優位性が確認できる。現行の ROMIO（図中の orig）に対して最大で 67%の性能向上が確認できた。アグリゲータ配置の最適化のみを行ったケース（図中の normal）に対しても 16%の性能向上を達成している。また、nd_shift 方式に比べて nd_rank_shift 方式はアグリゲータの数を変化させる中で全体的に高い性能を示しており、ノード間・ノード内のランク配置を配慮した通信機構の有用性が確認できたが、今回得られているデータからは明確な優位性を確認することはできなかった。これについては使用するノード数を増やすなどによって比較検討をする必要があると考えている。

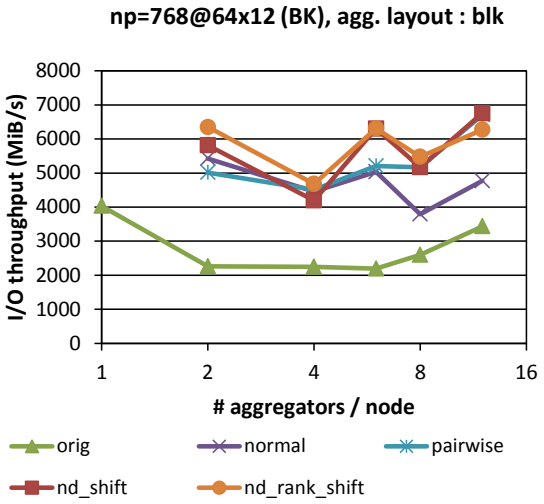


図 5： HPIO ベンチマークによる集団型 MPI-IO 書き込み処理の I/O 性能

まとめ、今後の課題

ROMIO による集団型 MPI-IO での TP-IO による高速化実装に対し、ノード間のランク配置を配慮したプロセス間データ通信の発行順の最適化を提案し、TSUBAME2.5 において実施した HPIO ベンチマークによる性能評価で提案手法の有用性を確認した。今回の評価においては、I/O 性能の最大値に関して現行 ROMIO に対し 67%の性能向上を確認した。

今後の課題としては、提案した 3 種類の通信順の最適化機構の優位性について確認することが必要と考えている。その意味で、使用する Lustre へのアクセスにおける他の利用者による外乱等を除いた上での評価を行う必要があると考えている。さらにデータ通信に対して集団型通信を行った場合との比較も今後検討したい。また、今回使用したノード数よりも多くのノード数・プロセス数での評価・検証も今後検討したい。