Vol.2010-HPC-127 No.6
2010/10/13

マルチスレッドによるTwo-Phase I/Oの高速化の検討

六車英峰[†1]　辻田祐一[†2]

科学技術計算の分野において近年取扱うデータ量は増加しており，高性能なファイル入出力が求められている．並列計算環境における標準的な通信インタフェースであるMPIにおいて，並列入出力インタフェースであるMPI-IOがあり，良く知られたMPI-IOの実装にROMIOがある．ROMIOにおける集団型入出力の最適化手法としてTwo-Phase I/Oが採用されている．Two-Phase I/Oは集団型入出力を行うプロセス間でデータの並べ替え操作を行うことで，ファイルに対する不連続なアクセスパターンを連続なアクセスパターンにし，ファイルアクセスを高速化する最適化手法である．しかしながら，Two-Phase I/Oはデータ並べ替え操作においてプロセス間で通信を行うため，並べ替え操作のコストが高く，これがファイルアクセスのボトルネックとなってしまう．本稿では，マルチスレッド処理を用いることで，Two-Phase I/Oのデータ並べ替え処理とI/O処理をオーバーラップさせ，集団型不連続入出力の性能を向上する方法を提案する．また，今回読み込みの場合のみに対して行ったTwo-Phase I/Oのマルチスレッド化の実装について，高い性能を実現できることを示す．

Multithreaded Two-Phase I/O Towards High Performance Collective I/O

Hidetaka Muguruma[†1] and Yuichi Tsujita[†2]

High performance I/O system has a big role due to the rapid growth in the scale of data generated by recent large scale parallel computing. MPI is the defact standard in parallel computations. MPI-IO is a parallel I/O interface specified in the MPI standard, and ROMIO is one of the well-known MPI-IO implementations. It utilizes Two-Phase I/O as an optimization in collective I/O. The Two-Phase I/O consists of multiple cycles of contiguous file accesses and data exchanges among user processes to improve its performance. However, inter-process communications for the two-phase I/O may be bottleneck with a big data size or a large number of processes. We propose a multithreaded two-phase I/O to overlap a part of the data communications with file accesses. In this paper, we explain an architecture of modified read operations and report its good performance relative to the original ROMIO implementation.

1. はじめに

近年の科学技術計算分野において計算量の増加と共に取り扱うデータの量は増加の一途を辿っている．しかしながら，CPUやメモリの性能向上に対して，I/Oの性能向上が追いついていないという現状があり，高性能なファイル入出力が求められている．ROMIO[1)]はMPI-2[2)]の規格に準拠したMPI-IO実装の一つである．この実装の中で並列入出力を高速に行うために様々な最適化が実装されている．Two-Phase I/O[1)]はROMIOに採用されている最適化の一つであり，不連続なアクセスパターンをプロセス間でデータを並べ替えることによって連続なアクセスパターンにすることで高性能な入出力を実現している．しかしながら，Two-Phase I/Oはデータ並べ替え操作においてプロセス間で通信を行うため，並べ替え操作のコストが高く，これがファイルアクセスのボトルネックとなってしまう．

そこで我々は，このROMIOに実装されているTwo-Phase I/Oをマルチスレッド化することで，ファイルアクセスとデータ並べ替えをオーバーラップさせてデータ並べ替え操作の処理の一部を隠蔽し，派生データ型による集団型並列入出力の高性能化を実現する方法を提案する．以下，2においてTwo-Phase I/Oの説明を行い，3においてTwo-Phase I/Oのマルチスレッド化とその実装について説明を行う．そして，4で我々の実装の評価結果を報告し，5で関連研究について述べ，最後に6で本稿のまとめと今後の課題について述べる．

2. Two-Phase I/O

Two-Phase I/OはROMIOに実装されている派生データ型による集団型入出力の最適化の一つである．Two-Phase I/Oは各計算プロセスが持つ不連続なファイルアクセスの要求を，一つの連続したファイルアクセスの要求にまとめることで，各プロセスが断片的なファイルアクセスを何度も行うのを避け，高い入出力性能を実現するものである．図**1**にファイルを読み込む場合のROMIOにおけるTwo-Phase I/Oの動作を示す．Two-Phase I/Oはデータ並べ替えとI/Oの二つの操作からなり，データの並べ替え操作を行うためにコレクティブバッファ(以下，CB)と呼ばれる一時的なデータの保持を行うバッファを持つ．CB

†1 近畿大学大学院システム工学研究科
Graduate School of Systems Engineering, Kinki University
†2 近畿大学工学部
Faculty of Engineering, Kinki University

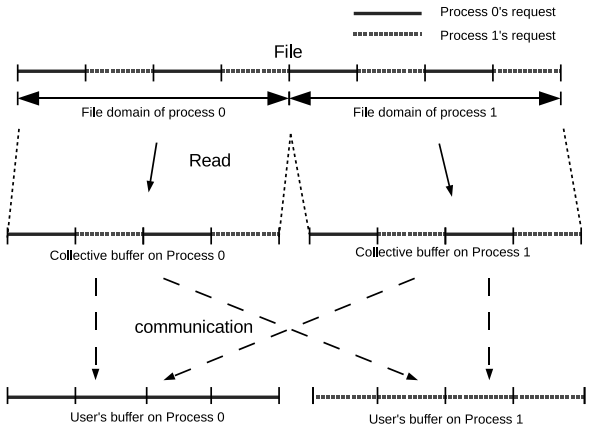

図 1　Two-Phase I/O による集団型読み込みの例

は実行時に MPI_Info_set により与えられるヒントによってサイズを指定することが出来る．I/O を行うプロセスは I/O アグリゲータと呼ばれ，デフォルトでは集団型入出力関数を呼び出した全てのプロセスが I/O アグリゲータとなるが，CB サイズと同様に MPI_Info_set により関数を呼び出したプロセス数以下に設定可能である．各 I/O アグリゲータにはアクセスするファイル領域が割り当てられる．アクセスするデータのサイズが CB サイズより大きい場合には，CB サイズ単位にアクセスを分けて，I/O と並べ替えの一連の処理を複数回実行する．

ファイルの書き込みの場合はファイル読み込みとは逆で，データ並べ替え操作の後に I/O 操作を行う．並べ替えた後のデータが連続にならず隙間が生じる場合には，そのまま書き込むと隙間に不定の値が上書きされてしまう．これを回避するために，この場合には事前にアクセスするデータ領域から隙間の部分も含めたデータを CB へ読み込む操作が行われる．これにより，隙間の部分のデータの消失を防いでいる．

3. Two-Phase I/O のマルチスレッド化の提案と実装

MPICH2-1.2.1p1 において CB のサイズはデフォルトで 16MB である．プロセスが読み出すファイルのサイズがこれより大きい場合，複数に分けられて実行されることになるが，読み込みの場合，ある回 (本稿では，これを以下，サイクルと呼ぶ) のデータ並べ替え操作

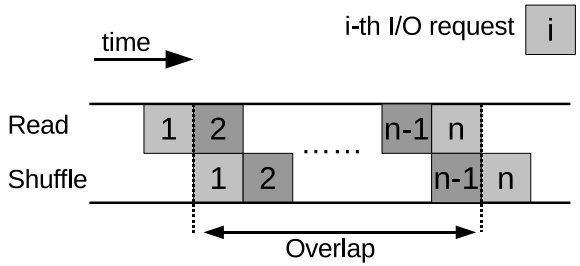

図 2　読み込みにおけるマルチスレッド化 Two-Pahse I/O のパイプライン処理の様子

とそのすぐ後に実行されるサイクルの I/O 操作は対象となるデータ範囲が異なるため互いに独立な操作である．そこで，これらの操作を同時に実行させるために，二つの操作をそれぞれスレッドを用いてパイプライン処理で実行する．

図 2 はファイル読み込みにおける Two-Phase I/O をスレッド化した場合の各処理のオーバーラップの様子である．あるサイクルのデータ読み込みとその前のサイクルのデータ並べ替え操作はオーバーラップし，全体の処理時間が短縮される．

書き込みの場合も同様であり，事前の読み込み操作とデータの並べ替え操作，書き込み操作がオーバーラップし，全体の処理時間が短縮されると考えられる．今回我々は，ファイル読み込みの Two-Phase I/O のみに対してパイプライン処理の実装を行った．

3.1 ファイル読み込みのスレッド化の実装

図 3 はスレッド化したファイル読み込みの Two-Phase I/O の動作を示した図である．本実装はメインスレッドと I/O スレッドによって構成され，I/O 操作と並べ替え操作用にそれぞれ I/O 要求を保持しておくためのキューを用意した．メインスレッドは I/O 要求の作成と並べ替え操作を行い，I/O スレッドはデータ読み込みを行う．並べ替え操作を別スレッドで行わないのは，I/O 要求の作成の処理ではファイルアクセスもプロセス間通信も行わないため，I/O や並べ替え処理と比較して処理に要する時間が極めて小さかったことと，スレッド数を減らすことで，スレッド間の同期のコストを減らすためである．

また，それぞれの I/O 要求には異なる CB が割り当てられる．そのため，I/O と並べ替え操作のサイクル数が多い場合には，メモリ資源を多量に消費してしまう．これを回避するため同時に存在する I/O 要求の数の上限値を定めている (今回はデフォルトを 4 とした)．

メインスレッドは，まず作成した I/O 要求を I/O スレッドのキューへ上限値まで詰める．その後は，メインスレッドは並べ替え用のキューから I/O スレッドが処理を終えた I/O 要

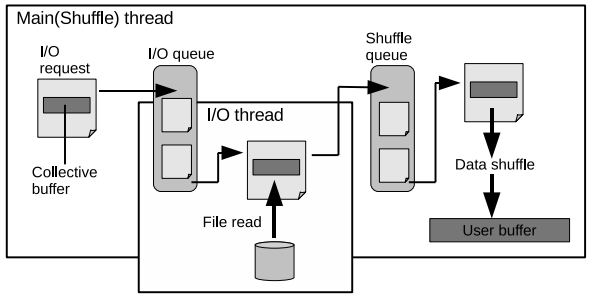

図 3　スレッド化した Two-Pahse I/O の実装

求を待ち，要求が来たら並べ替え処理を行う．一つ並べ替えの処理を終えると，新たに一つの I/O 要求を作成して I/O スレッドのキューへ詰める．I/O スレッドは I/O スレッドのキューに要求があればそれを処理して，処理し終えた要求を並べ替えのキューへ渡す．

並べ替えの処理には他のプロセスとの同期型の通信が含まれる．一方，I/O スレッドの処理には他のプロセスとの通信は行われない．従って，Two-Phase I/O における同期による待ち時間の一部も I/O の処理とオーバーラップすることとなる．また，I/O スレッドではプロセス間の通信は行われないため，並べ替え処理で行われる通信と衝突することによって双方の処理性能が低下する恐れはない．ただし，NFS や PVFS2 などのネットワークを利用するファイルシステムを利用し，それが MPI 通信と同じネットワークを利用している場合には，I/O による通信と並べ替え処理におけるプロセス間通信が衝突することが予想される．この場合，双方の処理性能が低下し，オーバーラップの効果が期待できないことが考えられる．

4. 評　　価

4.1 評 価 環 境

我々が実装したスレッド化した Two-Phase I/O のファイル読み込みと，オリジナルの Two-Phase I/O のファイル読み込みの性能比較を HPIO ベンチマーク Ver.1.55[3] を用いて行った．HPIO ベンチマークは無償のベンチマークプログラムであり，ユーザが指定したパラメタによって様々な不連続なファイルアクセスの性能を評価することが出来る．入出力対象の並列ファイルシステムには PVFS2 Ver.2.8.2[4] を用いた．今回 PVFS2 はメタデータノードに 1 ノード，I/O ノードに 4 ノードの計 5 ノードを用いて構築した．計算プロセス実行と PVFS2 構築に用いた PC クラスタの各ノード及び使用したスイッチの仕様を表 **1** に示す．

この二つのクラスタには二つの 1Gbps のネットワーク接続があり，それぞれを MPI 通信用と PVFS2 用に利用している．PVFS2 の I/O ノードでは，データの格納先として RAID-0 で構成された ext4 ファイルシステムを利用した．

4.2 オーバーラップ効果に対する性能評価

本提案手法は，Two-Phase I/O における I/O と並べ替え操作の一連の処理が複数回実行された場合のみに，I/O と並べ替え操作がオーバラップし性能向上が見込める．そこで，アクセスするデータサイズを 256 MB に定め，CB サイズを 1，4，16，64，128，256 MB と変更し，サイクル数が 256，64，16，4，2，1 回となるようにして，それぞれのバンド幅を測定した．HPIO ベンチマーク実行には計算ノード 4 つを用い，各ノードに１プロセスづつ起動させ，合計 4 プロセスが Two-Phase I/O を利用する派生データ型を用いた集団型入出力を行った．ファイルへのアクセスパターンはデータ 4kB が 4kB×(プロセスの数) おきに全部で 65536 個並び，一つのプロセスは 256MB のファイルアクセスを行う．従って，一つのファイルサイズは 1GB となっている．

測定した結果を図 **4** に示す．オリジナルの Two-Phase I/O は CB サイズが 256 MB(サイクル数が 1) の時にバンド幅が最大となり，サイクル数の増加に伴って性能が落ちている．これは，複数サイクルに分けて行うと，データ並べ替えのための通信やファイルアクセスに要するオーバーヘッドが増えるためである．

一方，スレッド化した Two-Phase I/O の場合では，CB サイズが 16 MB の時にバンド幅が最大になっており，これより CB サイズが大きい場合と小さい場合のどちらの場合で

表 1　評価環境

File System node	CPU	Intel Pentium D 930,3.2 GHz
	Memory	1.5 GB,533 MHz
	HDD	RAID0(160 GB×2,SATA)
	RAID card	3ware 9550SXU-4LP
	NIC	Broadcom5721(1 Gbps)
	OS	Fedora 12,Kernel 2.6.34
Computation node	CPU	Intel Xeon E5530,2.4 GHz
	Memory	8 GB,1333 MHz
	NIC	Broadcom BCM5716(1 Gbps)
	OS	Fedora 12,Kernel 2.6.32
Switch	HP procurve 2910al-24G switch J9145A(1 Gbps)	
	3Com SuperStack3 switch 3824(1 Gbps)	

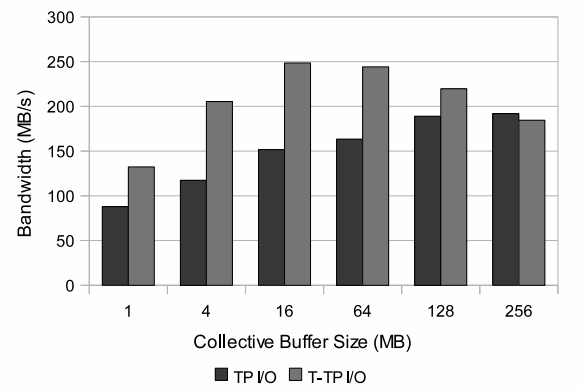

図 4　CB サイズに対する読み込み性能 (TP I/O:オリジナル, T-TP I/O:スレッド版)

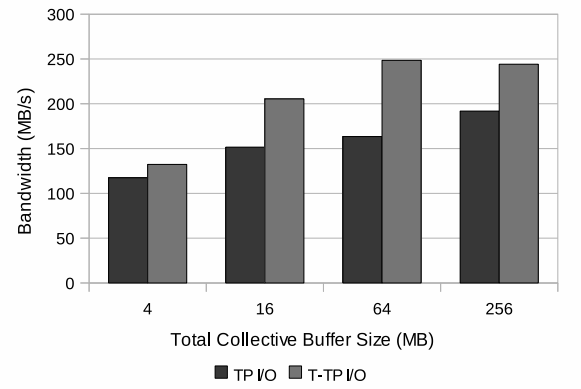

図 5　合計の CB サイズに対するの読み込み性能 (TP I/O:オリジナル, T-TP I/O:スレッド版)

も性能が低下している．CB サイズが 16 MB より大きい場合では，サイクル数が少ないことで，処理のオーバーラップする部分が少ないために性能が低下していると考えられる．一方，CB サイズが 16MB より小さい場合では，サイクル数が多くなったことで，スレッド間の同期や，データ並べ替えのための通信やファイルアクセスのオーバーヘッドの増加による性能低下が，オーバーラップによる性能向上を打ち消したためであると考えられる．

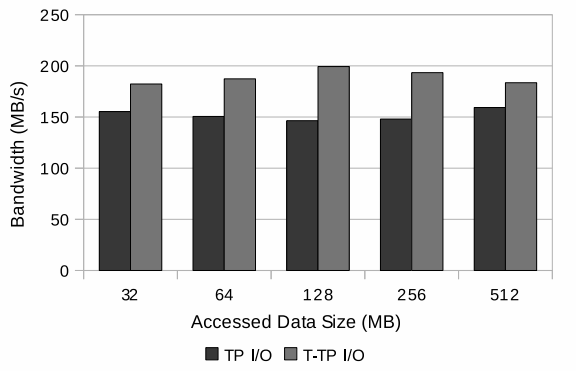

図 6　アクセスデータサイズに対する読み込み性能 (TP I/O:オリジナル, T-TP I/O:スレッド版)

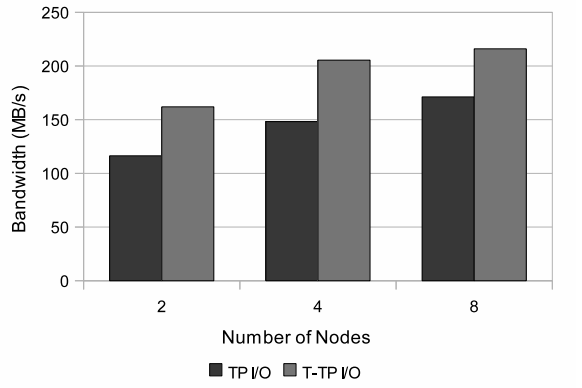

図 7　ノード数に対する読み込み性能 (TP I/O:オリジナル, T-TP I/O:スレッド版)

オリジナルの Two-Phase I/O とスレッド化した Two-Phase I/O を比較すると，CB サイズが 256 MB の時のみ，スレッド化した場合の方がオリジナルの Two-Phase I/O のバンド幅よりも小さくなっている．これは，256MB の場合ではサイクル数が 1 回であるため，スレッド化した場合でもデータ並べ替えとファイル読み出しの操作は全くオーバーラップせず，加えてスレッドの生成やスレッド間の同期のコストが含まれたためであると考えられる．一方，128 MB の場合のサイクル数は 2 回である．この時のバンド幅はオリジナルの

Two-Phae I/O の 128MB 及び 256MB の場合のバンド幅に対してそれぞれ約 12%並びに 14%向上しており，このことから少なくとも 2 回に分けて実行すればオーバーラップによる性能向上が得られることがわかる．その他の場合では，いづれもスレッド化した場合の方が性能が上回っており，CB サイズが 1，4，16 及び 64MB の各場合の性能は，オリジナルの Two-Phase I/O に対して，それぞれ約 51%，75%，63%並びに 49%向上している．このことから，スレッド化した場合の方の I/O と並べ替えのオーバーラップの効果が確認できる．また，オリジナル及びスレッド化した Two-Phase I/O で得られた最高性能について比較すると，スレッド化した Two-Phase I/O の最大のバンド幅 (CB サイズ 16MB の時) は，オリジナルの Two-Phase I/O の最大のバンド幅 (CB サイズ 256MB の時) を約 30%上回っている．

本実装では，スレッド間で最大で同時に 4 つの I/O 要求が存在するため，CB として割り当てられているメモリ量の合計は，最大で CB サイズの 4 倍である．図 **5** は，図 4 の結果から，CB に割り当てられている合計メモリサイズが同じもの同士を，オリジナル及びスレッド化した Two-Phase I/O 間で比較したものである．CB サイズの合計が同じ場合でもスレッド化した場合の方が優れていることがわかる．

これらのことから，スレッド化した実装では，CB として同じかそれ以下のメモリの割り当てで，オリジナルよりも優れた性能を得られ，より少ないメモリ利用量でより優れた性能を出すことが可能であることがわかる．

4.3 データサイズとノード数に対する性能の評価

4 つのノードで HPIO を実行し，各プロセスがアクセスするデータサイズを 32 MB から 512 MB まで 2 倍づつ変化させた時のファイル読み込み性能を図 **6** に示す．この時の CB サイズはデフォルトの 16MB であり，I/O と並べ替えのサイクル数はそれぞれ 2，4，8，16 及び 32 回である．すべての場合においてスレッド化した場合の方が高い性能を出しており，スレッド化した場合の性能はオリジナルの性能に対して，26%から 62%向上している．しかしながら，性能向上度はアクセスデータサイズが 128MB の時の 62%が最大であり，これより小さくなる場合でも大きくなる場合でも性能向上度は低下している．この原因はサイクル数だと考えられる．先の実験の結果では，性能向上度は 16 回 (CB=16 MB) で最大となっており，今回の場合でもやはり 16 回の場合に最大となっている．

図 **7** は，ファイルのサイズを 1GB に固定して，ベンチマークを実行するノード数を 2，4，8 と変化させた場合のファイル読み込み性能である．この時のサイクル数はそれぞれ 32，16 及び 8 回である．全ての場合においてスレッド化した場合のほうが性能が勝っている．スレッド化した場合の性能はオリジナルに対して，それぞれ 53%，65%及び 44%向上しているが，やはりサイクル数が 16 回の時が最大でそれより回数が多い場合でも少ない場合でも性能向上度が低下している．ファイルサイズを一定にした場合では，プロセス数を増やすと各プロセスがアクセスするデータのサイズが減少する．アクセスするデータサイズが CB サイズより小さくなった場合には，サイクル数が 1 回となるため，この場合の性能はオリジナルより劣ると予想される．

以上のことから，アクセスするデータサイズに合わせて CB サイズを変更し，サイクル数を調整しなければ，ノード数やデータ量によっては性能向上は得られないことが考えられる．しかしながら，現段階では，性能を最大化する CB サイズ及びサイクル数の決定方法が分かっていない．また，図 4 の結果からは，サイクル数が 4 回 (CB サイズが 64MB) から 64 回 (CB サイズが 4M) の間に性能のピークがあると考えられるが，その根拠は明確でなく，どのような場合においても，ピークとなるサイクル数が同じであるのかもまだ分かっていない．これらは今後，更なる詳細な評価試験と検討を行うことで明らかにしたいと考えている．

5. 関連研究

入出力をスレッドによってバックグラウンドで行わせる実装はこれまでにも数多くなされている[5)–6)]．しかしながら，それらは計算と I/O をオーバーラップさせるものであり，集団型入出力内部のデータ並べ替えのための通信と I/O をオーバーラップさせる我々の方法とは異なる．View-based I/O[7)] は，不連続なファイルアクセスを行う集団型並列入出力の最適化の手法である．View-base I/O は，データ並べ替えを行うことで連続なファイルアクセスを行うという点では Two-Phase I/O と同じであるが，Two-Phase I/O がアクセス時にファイル領域の割り当てを動的に行い，データのオフセットとサイズを I/O アグリゲータへ送信するのに対して，View-based I/O ではファイル領域の割り当てを静的に行い，また，アクセス時のオフセットと長さの通信を不要にしたことで，通信量や計算量を減らし，I/O 性能を向上させている．また，実装内部にキャッシュバッファを用意することでも性能を向上している．8) では View-Based I/O に GPFS のライブラリを用いて Write-back と Prefetching 機能を加えた実装が提案されている．この実装ではスレッドが用いられており，I/O がデータ並べ替え処理とは別スレッドで実行されている．I/O とデータ通信のオーバーラップという点では，本研究はこれに近い．しかしながら，本研究はファイルシステムに依存しない点でこれとは異なる．我々の提案手法は GPFS を含む ROMIO がサポートしてい

る全てのファイルシステム上で動作可能である．

6. おわりに

本研究は，Two-Phase I/O をマルチスレッド化することで，I/O 処理とデータ並べ替え処理をオーバーラップさせて，集団型並列入出力の性能を向上させる手法について述べた．今回我々はファイル読み込みの場合に対してのみマルチスレッド化の実装を行い，その性能評価を行った．その結果，Two-Phase I/O のファイルアクセスと並べ替えのサイクルが複数回実行される場合において，スレッド化していないオリジナルの Two-Phase I/O よりも優れた性能を実現し，またオリジナルよりも少ないメモリ使用量でより高い入出力性能を実現出来る可能性を示した．一方で，I/O と並べ替えのサイクル数によって性能向上度が変わり，オリジナルよりも性能が低下することが明らかになった．今後はスレッド化した Two-Phase I/O の性能を最大化するサイクル数の決定方法について検討するつもりである．また，ファイル書き込みについても実装を行い，性能評価を行うつもりである．また，9) では，Two-Phase I/O のデータ読み込み操作が遅延したプロセスがあると，他のプロセスがそれを待つことで性能低下につながる問題が指摘されている．この待ち時間をスレッド化によって隠蔽出来ていることを，ドイツ気候計算センター (DKRZ) で開発されている PIOviz の後継ツールである PIOsimHD[10] によるトレース機能との連携により検討したいと考えている．

謝辞　本研究を進めるにあたり，DKRZ およびハンブルク大学の Thomas Ludwig 教授,DKRZ の Julian Kunkel 氏，ハンブルク大学の Michael Kuhn 氏からは有用なコメントを頂きました．ここに感謝を申し上げます．なお，本研究の一部は科研費 (若手研究 (B) 課題番号 21700063) の支援を受けています．

参考文献

1) Rajeev Thakur, William Gropp and Ewing Lusk: Optimizing Noncontiguous Accesses in MPI-IO. Parallel Computing (28)1:83-105, January 2002
2) Message Passing Interface Standard: `http://www.mpi-forum.org/`
3) Avery Ching, Alok Choudhary, Wei-keng Liao, Lee Ward, Neil Pundit: Evaluating I/O Characteristics and Methods for Storing Structured Scientific Data. In *Proceedings of the International Parallel and Distributed Processing Symposium*, April 2006.
4) PVFS:http://www.pvfs.org/.
5) Phillip M. Dickens and Rajeev Thakur: Improving Collective I/O Performance Using Threads. In *Proceedings of the 13th Iternational Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing*, pp.38-45, 1999.
6) Xiaosong Ma, Marianne Winslett, Jonghyun Lee and Shengke Yu: Improving MPI-IO Output Performance with Active Buffering Plus Threads. In *Proceeding of the 17th International Symposium on Parallel and Distributed Processing*, pp.68.2, 2003.
7) Javier Garcia Blas, Florin Isaila, David E.Singh and Jesus Carretero: View-Based Collective I/O for MPI-IO. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, Washington, DC: IEEE Computer Society, pp.409-416, 2008.
8) Javier Garcia Blas, Florin Isaila, Jesus Carretero, David Singh and Felix Garcia-Carballeira: Implementation and evaluation of file write-back and prefetching for MPI-IO over GPFS, *The International Journal of High Performance Computing Applications*, pp.78-92, Spring 2010．
9) Julian M. Kunkel,Yuichi Tsujita, Olga Mordvinova, and Thomas Ludwig: Tracing Internal Communication in MPI and MPI-I/O. In *Proceedings of PDCAT 2009.IEEE Computer Society Press*, 2009.
10) PIOsimHD: http://wr.dkrz.de/Projects/PIOsimHD/wiki.