システムソフトウェアと
オペレーティング・システム 83－8
（2000．2．25）

MPI-IO を支援する SR8000 の並列入出力機構

薗田浩二*、宇都宮直樹*、岩月秀樹**、熊崎裕之**
* 日立製作所システム開発研究所
** 日立製作所ソフトウェア開発本部

並列科学技術計算プログラムの I/O 性能を高速化する標準 I/O インタフェースとして MPI-IO(Message Passing Interface I/O)が提唱されている。我々は MPI-IO を高性能実装するために並列ファイル入出力機構(PFF)を開発した。PFF が提供する不連続ファイル領域アクセス機能や collective I/O 機能は、MPI-IO インタフェースの与えるユーザプロセス I/O 情報と OS 内部の並列ファイル構成情報を利用し効率のよい I/O を行う。PFF を用いて実装した MPI-IO の性能は、8 プロセスの WRITE 実行で、UNIX 型インタフェースの 13.5 倍を達成した。

Parallel File I/O Feature for efficient implementation of MPI-IO on SR8000

Koji Sonoda*, Naoki Utsunomiya*, Hideki Iwatsuki**, Hiroyuki Kumazaki**
* Systems Development Laboratory, Hitachi, Ltd.
** Software Development Division, Hitachi, Ltd.

MPI-IO(Message Passing Interface IO) has been proposed as a standard I/O interface for parallel scientific program. For high performance implementation of MPI-IO, we've developed Parallel File I/O Feature(PFF). The noncontiguous file region access feature and the collective I/O feature of PFF use the I/O information of user process and the file structure information in OS to perform efficient I/O. The performance of MPI-IO, which implemented with PFF, achieves 13.5 times as fast as UNIX type interface under 8 processes execution environment.

1 はじめに

並列計算機の演算性能は、プロセッサアーキテクチャおよび動作周波数の向上に支えられ年々確実に向上している。既に理論演算性能だけでなく、実効演算性能も TFLOPS 級の並列計算機が稼働しており、PFLOPS 級の性能を持つ並列計算機の登場も近い将来現実となりつつある。

一方、演算性能に比べディスク性能の向上率は低いままであり、扱う問題が大規模化するに従い、ファイル I/O 性能がボトルネックとなってきている。この問題を解決するため、ファイルデータを複数のディスクに分散格納し、各ディスクを並列動作させて I/O 性能向上を図る、並列ファイルシステムが開発されてきた。並列ファイルシステムを用いることで、単一プロセスからの I/O 性能や、複数プロセスがそれぞれ異なるファイルにアクセスする場合の I/O 性能は大きく改善できる。

並列科学技術計算アプリケーションで多く用いられる SPMD(Single Program Multiple Data streams)型の並列プログラムでは、配列データを領域分割し、分割された各領域を複数のノード上に生成されたプロセスが担当して計算を行う事が多い。このような SPMD 型の並列プログラムにおける各プロセスのファイルアクセスパターンは、ファイルを一定パターンで不連続アクセスする特徴がある[2]。

一方 UNIX の I/O インタフェースは、連続したファイル領域へのアクセスインタフェースしか提供していない。従って、ファイル上で不連続なデータをアクセスする場合、連続領域数分の I/O 要求と、ファイルポインタ移動要求を発行する必要があり、システムコールオーバヘッドが増加する。

また、複数プロセスが同一ファイルをほぼ同時にアクセスするため、プロセス数に比例した I/O 要求が発生する。OS は、各 I/O 要求の関係を把握できないため、I/O 要求毎にディスクアクセスを行う。このため、ランダムなディスクアクセスを引き起こし、I/O 性能の低下を招く。

この問題を解決するインタフェースとして MPI-IO(Message Passing Interface IO)が提唱されている。MPI-IO は、MPI フォーラムが並列アプリケーションプログラム向けに策定した並列ファイル I/O インタフェースであり、MPI2[5]の一部として'97 年 7 月に仕様が決定された。MPI-IO は、前述した UNIX インタフェースの問題点を解決するため、ファイルビュー設定インタフェースや collective I/O インタフェースという特徴的なインタフェースを備えている。

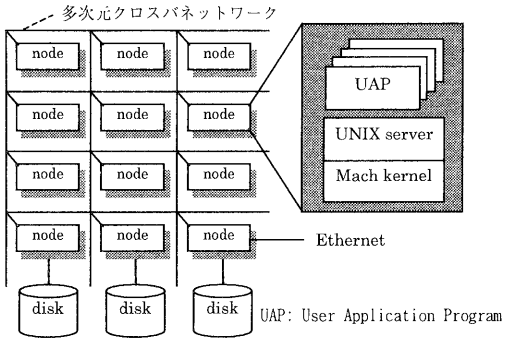

図 1 SR8000 の構成

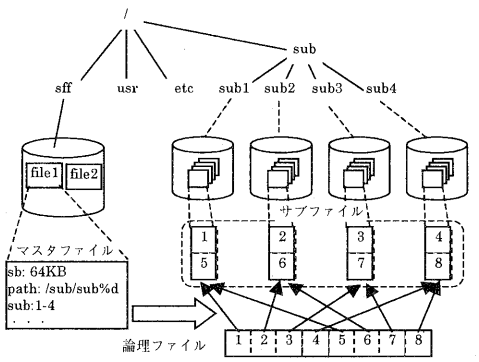

図 2 SFF の構成

なインタフェースを備えている。

ファイルビュー設定インタフェースは、自プロセスのアクセスするファイル領域のみが見える様に、ファイルに対し論理的なビューを設定する。このインタフェースにより、ファイル上の不連続領域を、ユーザプロセスが連続領域としてアクセスできるため、1回の I/O 要求で不連続ファイル領域への I/O 要求を発行することができる。

collective I/O インタフェースは、関連する全てのプロセスが同一ファイルへの I/O 要求を発行するインタフェースである。このインタフェースにより、システムは関連する I/O 要求の発行プロセスをあらかじめ把握できるため、個々のプロセスが発行する I/O 要求を待ちわせ、マージする最適化が可能となる。多くの場合、個々のプロセスからの I/O 要求は不連続領域アクセスでも、マージした I/O 要求は連続領域アクセスである。

MPI-IO のインタフェースにより、システムはユーザプログラムから I/O に関するより多くの情報を得ることができる。我々は、MPI-IO インタフェースの情報を有効に活用し、MPI-IO を高性能実装するための OS 支援機構として並列ファイル入出力機構(Parallel File I/O Feature: PFF)を開発した。PFF は SR8000 が提供するストライピングファイル機構(Striping File Feature: SFF)に対するインタフェースを提供する。PFF は不連続ファイル領域アクセスや collective I/O を初めとする機能により、MPI-IO の高性能実装を可能とする。

本論文では、PFF が提供する機能の概要について述べ、性能評価を行う。

2 並列ファイル入出力機構

ここでは、並列ファイル入出力機構の概要について述べる。はじめに 2.1 で SR8000 の概要とその並列ファイルシステムについて説明する。その後、2.2 で並列ファイル入出力機構の説明を行う。

2.1 SR8000 の並列ファイルシステム

図 1に、SR8000 の構成を示す。SR8000 は分散メモリ型の並列計算機であり、最大 512 ノードを多次元クロスバネットワークで結合して構成する。各ノードは複数の RISC プロセッサが、最大 16GB のメモリを共有するマルチプロセッサである。任意のノードにディスクや Ethernet 等の I/O デバイスを接続可能である。以下ではデバイスを接続しているノードを I/O ノードと呼ぶ。

各ノードには、Mach カーネルと UNIX サーバが常駐し、その制御下でユーザプログラムが動作する。また、各ノードの UNIX サーバが協調しシングルシステムイメージをユーザに提供する。

図 2に SFF の構成を示す。SFF は、論理ファイルを一定サイズ（ストライピングブロックサイズ)のブロックに分割し、各ブロックを複数ディスク上に作成するサブファイルにストライピングして格納する。ストライピングブロックサイズやサブファイル数などのメタデータを格納するファイルをマスタファイルと呼ぶ。ユーザプログラムはマスタファイルの名前を用いて SFF の論理ファイルをアクセスできる。

SFF のファイルは、UNIX ファイル同様、ユーザプログラムに対して連続したバイトストリームとして提供されるが、データが物理的に複数のディスクに分散しているため、それらの並列動作により高いスループット性能を実現する。

2.2 PFF の概要

MPI-IO 機能を高性能実装するためには、MPI プロセス間の関係やファイルの物理的な配置情報を適宜参照する必要がある。ところが、MPI プロセス間の関係やユーザが定義したデータ構造は MPI ライブラリが管理するが、ファイルの物理的な配置情報は OS が管理する。従って、MPI ライブラリと OS のどちらか一方のみで実装すると、十分な最適化が困難となる。

PFF は、UNIX サーバが提供する OS 機能であり、SFF の上位に位置し、SFF に対する並列入出力インタフェースを提供する。

PFF は、リスト I/O やストライド I/O による不連続領域アクセス機能、collective I/O 機能、非同期 I/O 機能、ファイルオフセット明示 I/O 機能により、MPI ライブラリが持つ I/O 情報を効率的に OS 内部に取り込み、最適化を行う。

以下で、これら PFF が提供する機能について説明する。

2.2.1 不連続領域アクセス機能

MPI-IO のファイルビュー設定インタフェースは、プロセスに対し論理的に連続なファイルのビューを提供するが、その実装は不連続領域アクセスを必要とする。

UNIX の I/O インタフェースは、連続ファイル領域に対する I/O 要求しかサポートしていない。従って、不連続ファイル領域に対する I/O 要求機能を MPI ライブラリで単純に実現すると、ファイルポインタ移動のための lseek()システムコールと、read()/write()システムコールを繰り返し発行することになり、システムコールオーバヘッドが増大する。

ライブラリレベルでシステムコールオーバヘッドを削減する方式として、Data Sieving[3]が提案されている。これは、個々の連続領域に対応する I/O 要求を多数発行する代わりに、それら複数の連続領域を含む少数の大きな連続領域 I/O 要求を発行し、ライブラリ内部の一時バッファ内にデータを格納した後、必要なデータのみをユーザバッファにコピーする方式である。この方式により、システムコールオーバヘッドを低減することができるが、ユーザが要求する領域間のギャップが大きいと、I/O データサイズに占める有効データの割合が小さくなり効率が悪い。

PFF では不連続領域アクセスを行うリスト I/O 機能とストライド I/O 機能を提供し、システムコールオーバヘッドの低減と高効率データ転送を実現する。

(1) リスト I/O 機能

リスト I/O 機能は、複数の I/O 要求をリスト形式で指定し一括して OS に要求する機能である。この機能により、不連続領域に対する I/O 要求を一回のシステムコールで発行でき、システムコールオーバヘッドの削減が可能となる。また、OS によるディスク I/O 要求のスケジューリングやマージが容易になるため、ディスク I/O 性能が向上する。

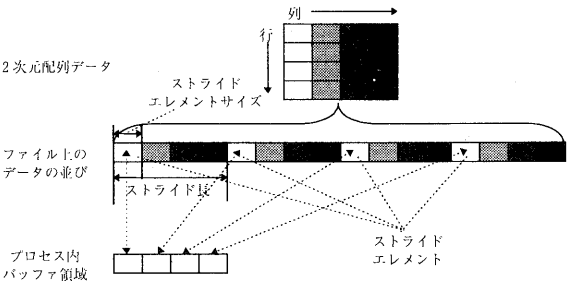

図 3 ストライドデータアクセス

(2) ストライド I/O 機能

複数のプロセスが、一つのファイルに格納されている配列データの、それぞれ異なる列（または行）をアクセスする場合、おのおののプロセスがアクセスするデータは図 3に示すようなストライドデータとなる。

PFF は、ストライドデータアクセスを指定するインタフェースを提供する。ストライドデータとは、一定サイズの要素データ(ストライドエレメント)が一定間隔で並んでいるデータ構造のことであり、そのデータの開始位置(ファイルオフセット)、要素データのサイズ(ストライドエレメントサイズ)、要素データ間隔(ストライド長)、I/O サイズの 4 つパラメタで指定することができる。

ストライド I/O 機能はリスト I/O 機能に比べて、MPI ライブラリの I/O 要求作成オーバヘッドと OS の解析オーバヘッドが小さく、データ間の関係をより容易に把握できる。

2.2.2 collective I/O 機能

collective I/O 機能の実現方式として 2 フェーズ I/O 方式[3]、server-directed I/O 方式[4]および disk-directed I/O 方式[1]が提唱されている。

2 フェーズ I/O 方式は、全計算ノードの I/O データを交換し連続 I/O 要求に変換する要求マージフェーズと、各ノードが並列に I/O を実行する I/O 実行フェーズの 2 フェーズで実現する。この方式は、並列ファイルシステムの物理的な配置情報を必要とせず MPI ライブラリのみで実現可能だが、I/O 要求のマージに伴い余分なノード間データ転送のオーバヘッドが発生する上、他計算ノード用のデータを格納するために余分なメモリが必要となる。

server-directed I/O 方式は、各 I/O ノード上のサーバプロセスが I/O データのマージとファイルシステムへのアクセスを行う方式である。計算ノード上のクライアントプロセスは、自プロセスのデータを格納する I/O ノード上のサーバプロセスと、ライブラリを通してデータ転送を行う。2 フ

ェーズ I/O 方式に比べ余分なノード間データ転送が発生しない。しかし、ユーザレベルで並列 I/O を実現するため、アプリケーションが作成した並列ファイルを通常の UNIX システムコールで一つのファイルとして扱うことができない。

disk-directed I/O 方式は、並列ファイルシステム自身が collective I/O を提供する方式で、server-directed I/O 方式同様、ディスクの存在するノードで I/O 要求のマージとディスクアクセスを行う。ファイルシステム自身が並列 I/O 機能を提供しているため、並列ファイルを通常の UNIX システムコールで扱うことができる。

以上から、disk-directed 方式が性能と互換性の面で有利と考える。PFF は、disk-directed 方式による collective I/O 機能を提供する。

collective I/O 機能を実現するためには、collective I/O を行うプロセスの集合を OS が知る必要がある。PFF は、プロセス ID の配列を用いてプロセスの集合を定義するインタフェースを提供する。MPI ライブラリが、このインタフェースを用いて collective I/O を行うプロセス集合をあらかじめ PFF に通知し、前述の不連続ファイル領域アクセス機能で I/O 要求を発行することで、PFF は複数プロセスの I/O 要求マージに必要な情報を取得できる。

さらに、PFF はファイルのストライプ情報を容易に参照できるため、プロセスが発行した collective I/O 要求を各 I/O ノード毎の要求に分割し、プロセス集合情報と共に I/O ノードに送信することで disc-directed 方式の collective I/O 機能を実現する。

2.2.3 ファイルオフセット明示 I/O 機能

MPI-IO のファイルポインタは、ユーザプロセスが設定したファイルビューに従って更新するため、ファイルビューの情報を必要とする。OS がファイルポインタを提供すると、ファイルビューを管理している MPI ライブラリが、I/O 要求の度にファイルポインタ更新要求のシステムコールを発行する必要がある。

このオーバヘッドを削減するため、PFF では I/O 要求時にファイルオフセットを指定するファイルオフセット明示インタフェースを提供し、ファイルポインタは MPI ライブラリで実現する方式とする。このインタフェースにより、I/O 要求発行時に任意のファイルオフセットを指定できるため、MPI ライブラリがファイルポインタ更新要求のシステムコールを発行する必要がなくなる。

2.2.4 非同期 I/O 機能

PFF は、同期 I/O インタフェースに加え、非同期 I/O インタフェースを提供する。同期 I/O インタフェースでは、ファイル I/O が完了するまで呼び出し側プロセスはブロックし続けるが、非同期 I/O インタフェースでは、I/O 要求を登録するだけですぐにリターンする。このため、I/O 処理と演算のオーバーラップ実行や、異なるファイルに対する I/O 要求の並列実行が可能になる。

2.2.5 ストライピング定義機能

SFF はファイルのデータを一定サイズ（ストライピングブロックサイズ）で分割し、各ブロックを複数ノード上のディスクにストライピングして格納する。ストライピングブロックサイズやサブファイル数等のパラメタはシステム毎にデフォルト値が設定されている。ところが、一般に最適なパラメタ値は、アプリケーションプログラムのアクセスパターンで決定される。ファイル I/O 性能を上げるためには、異なるプロセスがそれぞれ異なるディスクをアクセスするようにストライピングパラメタ値を設定する必要がある。

PFF は、ファイル毎にストライピングパラメタ値を変更できるインタフェースを提供する。設定できるパラメタは、ストライピングブロックサイズ、サブファイル数およびストライピングを行うサブファイル順である。同一ファイル上の異なるファイル領域に異なるストライピングパラメタ値を設定することも可能である。

このストライピング定義機能は、MPI-IO のヒント情報提供インタフェースを通じてアプリケーション毎の最適化を可能とする。

3 性能評価

ここでは、PFF を用いて実装した MPI-IO インタフェースを使用し、性能評価を行う。

3.1 測定プログラム

複数プロセスが、1 つのファイルに格納された配列データを領域分割してアクセスする MPI-IO プログラムを用いて、I/O スループット性能を測定した。

このプログラムは、図 4に示すとおり配列データを列方向に分割し、各分割領域を各プロセスがアクセスする。図はプロセス数が 4 の場合の例である。このように分割すると、各プロセスがアクセスする配列データはファイル内でストライド状に配置される。

このデータに対し、プログラムは以下の 3 種類の MPI-IO インタフェース(IF)で I/O を行う。

(1) UNIX タイプ IF (UT)

ファイルビューを設定せず、UNIX インタフェースと同等の MPI-IO インタフェースを用いて I/O を行う。ファイルビューを設定しないため、

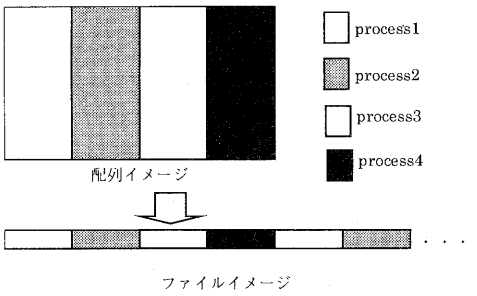

図 4 テストプログラムのファイルデータ構成

各プロセスはファイルに格納されている全データが見える。そのため、各プロセスはファイルポインタの移動と I/O 要求を配列の行数分繰り返し発行する。

(2) ファイルビュー設定 IF (FV)

各プロセスがファイルビューを設定し、それぞれが独立に I/O 要求を発行する。ファイルビューの設定により、ファイルが連続イメージで見えるため、各プロセスは I/O 要求を 1 回だけ発行する。

(3) ファイルビュー+collective I/O IF(FV+COL)

各プロセスがファイルビューを設定しcollective I/O 要求を発行する。ファイルビューの設定によりファイルが連続イメージで見えるため、プロセスは collective I/O 要求を 1 回だけ発行する。

本測定では、配列データの総データサイズを1GB、各プロセス担当領域の行方向サイズを16KB 固定とし、プロセス数に応じて列方向サイズが変化するように設定した。また、プロセス数は2，4，8の3通りとし、各 IF の WRITE、READ 性能を測定した。

3.2 測定結果および考察

測定には、8 ノード構成の SR8000 を用いた。各ノードが搭載する主記憶サイズは 8GB である。また、RAID ディスク DF350 を Ultra/WIDE-SCSI で各ノードにそれぞれ接続し、8 ディスクを用いて SFF を構成した。各 DF350 は 64MB のディスクキャッシュを持ち、各 I/O ノードは 250MB のバッファキャッシュを持つ。SFF のストライピングブロックサイズは 64KB である。

測定結果を図 5、図 6に示す。結果は、UT を 2 プロセスが実行した場合の I/O スループットを 1 とした性能比で示している。

(1) ストライド I/O 機能の効果

UT と FV を比較すると、WRITE 性能、READ 性能とも FV の方が大幅に性能が良い。8 プロセス時の性能比は、WRITE 性能で 5 倍、READ 性能で 9 倍であり、ファイルビュー設定機能が性能向上に大きく貢献していることが解る。

ストライド I/O 機能もしくはリスト I/O 機能がなければ、FV でも、MPI ライブラリが不連続領域数分のシステムコールを発行する必要があるため、UT と同等の性能となると考えられる。

また FV では、READ/WRITE 共、プロセス数の増加率以上に性能が向上している。特に、READ 性能においてこの現象は顕著である。

これは、I/O ノードにおけるバッファキャッシュとプリフェッチの効果である。各 I/O ノードとも 256MB のバッファキャッシュを備えており、最初に I/O ノードに到着した READ 要求を実行した結果、その READ 要求がアクセスする領域を含むディスクブロックがバッファキャッシュ上に読み出される。SR8000 の UFS が使用するディスクブロックサイズは 64KB であり、これは本テストプログラムで使用したストライドエレメントサイズの 4 倍である。従って、4 プロセスの I/O 要求までは、各プロセスの I/O 要求はファイルの全ディスクブロックをアクセスする事になり、最初の READ 要求実行時に他のプロセス分のデータもバッファキャッシュに読み出される。また 8 プロセスの場合は、各プロセスの I/O 要求はディスクブロックを 1 つおきにアクセスするが、UFS のプリフェッチ機能により隣のディスクブロックもバッファキャッシュに読み出される。

従って、最初のプロセス以外のプロセスからの I/O 要求は、バッファキャッシュ上のデータをアクセスするだけで処理が終了する。そのため、プロセス数が増加するに従い、バッファキャッシュをアクセスするプロセス数が増加し、プロセス数の増加率以上に性能が向上する。

(2) collective I/O 機能の効果

FV+COL の WRITE 性能は、FV に比べて 8 プロセス時で 2.6 倍の性能を示している。この理由は、FV における各プロセスの I/O 要求はストライドデータであり、ディスク I/O 処理をファイルのロックを確保してから行う必要があることによる。ロックにより、各プロセスからの I/O 要求は逐次的に処理される。

collective I/O では、ディスク I/O 前に各プロセスからの I/O 要求をマージし、連続領域に対する一つの大規模 I/O 要求に変換する。従ってファイルに対するロックが不要であり、マージ後の I/O 要求も連続領域アクセスのため性能がよい。

一方 READ 性能は、FV の性能とほぼ同じか若干低下している。この原因は 2 つ考えられる。

1 つは、前述したバッファキャッシュの効果である。バッファキャッシュにデータが存在する場合、I/O ノードの処理は転送バッファとバッファ

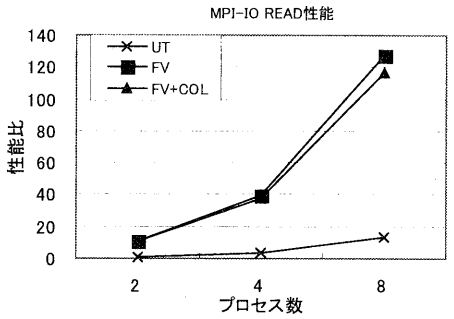

図 5 MPI-IO READ 性能

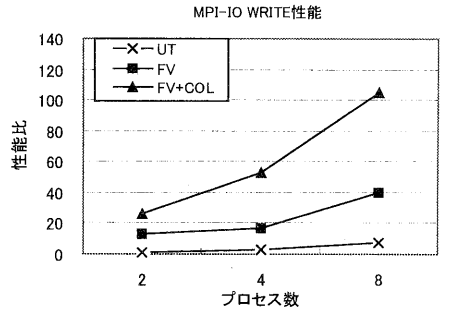

図 6 MPI-IO WRITE 性能

キャッシュ間のメモリコピーであり、ディスクアクセスを伴わない。また、WRITE と異なりファイルのロックが不要であり、複数プロセスからの I/O 要求は並列に処理される。従って、collective I/O 使用時と比べてオーバヘッド要因がなく、FV+COL との性能差が発生しない。

もうひとつは、collective I/O 時の同期オーバヘッドである。collective I/O では、複数プロセスの I/O 要求待ち合わせを行うため、プロセス間の I/O 要求発行時刻がずれると、先に I/O 要求を発行したプロセスには無駄な待ち時間が発生する。I/O ノードでの I/O 処理がディスクアクセスを伴う場合、ディスク I/O 待ち時間が大きいためこの待ち時間が問題とならないが、前述の通りメモリコピーだけで I/O 処理が完了する場合にはオーバヘッドとなる。これが、プロセス数増加時に FV+COL が FV に比べて READ 性能が低い原因と考える。

以上の結果から、collective I/O は WRITE 処理に対して大きな効果があるが、UFS のプリフェッチ機能が有効なケースでは、READ 処理に対する効果は小さい事が分かった。

4 結論

MPI-IO の高性能実装を支援する PFF の機能と実装方式について述べ、性能評価を行った。

PFF は、複数の I/O 要求を一括して発行できるリスト I/O 機能と、ストライドデータを指定するストライド I/O 機能を提供する。これらのインタフェースを用いて実装した MPI-IO のファイルビュー設定機能により、UNIX タイプのインタフェースを使用した場合に比べ、8 プロセスの WRITE 性能が 5.2 倍、READ 性能が 9.2 倍の I/O スループット性能を実現した。

また、PFF は OS レベルで collective I/O 機能を提供する。OS が collective I/O 機能を提供することにより、ライブラリで実現する場合に比べて無駄なノード間通信が発生せず、メモリ使用効率のよい実装を可能とした。この機能は、ファイルに対するロックが必要な WRITE 処理の場合に効果が高く、8 プロセス実行時で collective I/O 機能を使用しない場合に比べて 2.6 倍、UNIX タイプのインタフェースに比べて 13.5 倍の性能を示した。

今後は、より大規模な構成での測定と、さまざまなデータアクセスパターンでの測定を行い、PFF の有効性を検証していく必要がある。

参考文献

[1] David Kotz,"Disk-directed I/O for MIMD Multiprocessors", Dartmouth college, OSDI, 1994.11

[2] David Kotz, Song Bac Tho, and Sriram Radhakrishanan. "Dynamic file-access characteristics of a production parallel scientific workload", Proc. of Supercomputing '94, pp. 640-649, November 1994

[3] Rajeev Thakur, William Gropp, Ewing Lusk, "Data Sieving and Collective I/O in ROMIO", Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation, February 1999, pp. 182-189

[4] Y. Chen, Y. Cho, S. Kuo, K. E. Seamons, M. Subramaniam, and M. Winslett. "Server-directed input and output in panda: A commodity-parts approach to high-performance I/O", Submitted for journal publication, 1997.

[5] "MPI-2: Extensions to the Message-Passing Interface", Message Passing Interface Forum, http://www.mpi-forum.org/

・ UNIX は，X/Open Company Limited が独占的にライセンスしている米国ならびに他の国における登録商標です。