2005－HPC－104（4）
2005／10／7

Parallel netCDF インタフェースによる計算機間入出力操作の試み

辻　田　祐　一†

netCDF (Network Common Data Form) は、計算アプリケーションにおける可搬性のあるデータ形式をサポートするインタフェースを定めており、様々な計算アプリケーションで利用されている。さらに近年、並列入出力機能をサポートした parallel netCDF が提案されている。parallel netCDF では、並列入出力部分の機能を MPI-I/O のインタフェースを用いる事で実現している。異なる MPI ライブラリを持つ計算機間でもこの機能を実現するために、異機種の計算機間で透過的な MPI-I/O 操作を実現する Stampi を用い、いくつかの関数について、試験的に機能の実装を行なった。この新しく実装した機能の性能をネットワークで繋がれた PC クラスタ間で計測を行ない、データ長が大きい時には、十分使用できる事を確認した。

Implementation of a Parallel NetCDF Interface in Remote I/O

Yuichi Tsujita†

NetCDF has been proposed to support common data I/O format and recently parallel netCDF has been developed with the help of an MPI-I/O library. The parallel netCDF provides parallel I/O operations with the netCDF data format. To realize this mechanism among computers which have different MPI libraries, some of the parallel netCDF interfaces have been introduced in a remote I/O mechanism of a Stampi library. Stampi is a flexible intermediate library to realize seamless MPI communications and MPI-I/O both inside a computer and among computers. This newly implemented mechanism has been evaluated on interconnected PC clusters, and sufficient performance has been achieved with huge amount of data.

1. はじめに

MPI[1),2)] は並列計算における標準的な通信インタフェースの一つであり、様々な並列計算機で MPI の仕様に基づいた通信ライブラリが利用可能である。しかし、それらは自計算機内でのみ利用可能であり、異なる MPI ライブラリ間での MPI 通信は利用できない。そこでこのような異なる MPI ライブラリ間の透過的な MPI 通信を提供するために Stampi[3)] が開発された。

近年、並列計算の技術は理論、実験と同様に重要な役割を担うものとなってきている。そのような並列計算でも特に大規模な並列計算では、計算の過程で、ある決まった間隔毎に中間データあるいは障害時の復旧用の情報をファイルに出力している。そのような大規模なデータを入出力する計算アプリケーションでは、効率的な入出力を実現するために、並列入出力機能が必要になってきた。並列入出力機能の一つとして MPI の仕様では MPI-I/O と呼ばれる並列入出力インタフェースが定められている[2)]。

並列計算で出力されるデータは、可視化などの目的で、別の計算機に転送される事がある。MPI-I/O の機能は様々な MPI ライブラリで実現されてきたが、自計算機内でのみ利用可能であり、異なる MPI ライブラリを持つ別の計算機への入出力はできない。このような入出力機能を実現すべく、Stampi-I/O[4)] が Stampi の拡張機能として実現された。利用者は MPI-I/O のインタフェースを通して、自由に異なる計算機への入出力操作が可能である。ターゲットの計算機上に起動された入出力操作を行なうプロセスが計算機ベンダが提供する MPI-I/O ライブラリを用い入出力を行なうが、これが利用できない場合、UNIX I/O を用いた入出力を行なう。

並列計算において、入出力データの形式や入出力インタフェースを標準化し、それらを用いた可搬性のある入出力操作を行なう取り組みがいくつか行なわれている。その一つとして、netCDF[5)] と呼ばれるイン

† 近畿大学 工学部 電子情報工学科
Department of Electronic Engineering and Computer Science, Faculty of Engineering, Kinki University

タフェースがある。netCDF は多次元データを自己記述的で可搬性の高いフォーマットとして作成やアクセスができることを目的に策定されたインタフェースで、C や Fortran などのインタフェースが提供されており、netCDF のインタフェースを通して透過的な入出力操作が可能である。さらに近年並列入出力へ拡張した Parallel netCDF[6]（以下、PnetCDF と記す。）が定められ、利用可能である。PnetCDF では、並列入出力の実装に MPI-I/O を利用しており、例えば、MPICH[7] における MPI-I/O ライブラリである ROMIO[8] との組合せで利用可能である。しかし、異なる MPI ライブラリ間での入出力は出来ない。

そこで、Stampi の計算機間 MPI-I/O 機能である Stampi-I/O を用い PnetCDF のインタフェースを通したリモート入出力操作の実装を試みた。以下、本実装の詳細と、基本機能の性能評価の結果について述べた後、今後の課題について述べる。

2. PnetCDF インタフェースを用いたリモート入出力

PnetCDF インタフェースを用いて計算機間で入出力を行なうために Stampi の計算機間 MPI-I/O 機能を用いて実装した。以下、この機能のアーキテクチャと動作メカニズムについて説明する。

2.1 PnetCDF インタフェースへの対応

PnetCDF には、様々な入出力インタフェースが用意されており、それらのインタフェースを実装した関数の中で、数種類の MPI 関数が利用されている。いくつかの PnetCDF インタフェースに関してそれらの内部で呼び出される MPI 関数との対応を**表 1** に示す。

オリジナルの PnetCDF では、MPI 関数の部分を計算機ベンダが提供する MPI ライブラリや MPICH などを利用している。このような場合、自計算機内部で利用することは出来るが、異なる MPI ライブラリを持つ別の計算機への入出力は利用出来ない。そこで、Stampi が提供する MPI インタフェースを用いてこの機能が実現出来るかどうか、実装を試みた。表 1 に示すように、PnetCDF で用いている MPI 関数は Stampi でサポートしており、PnetCDF のライブラリを作成する際に Stampi のライブラリをリンクするように変更し、実装を行なった。

表 1 PnetCDF インタフェースから呼び出される MPI 関数の例。

`ncmpi_create()`	`MPI_File_open()`、`MPI_File_delete()` など
`ncmpi_open()`	`MPI_File_open()`、`MPI_File_delete()` など
`ncmpi_put_var_int()`	`MPI_Comm_rank()`、`MPI_File_set_view()`、`MPI_Type_hvector()`、`MPI_Type_commit()`、`MPI_Type_free()`、`MPI_File_write()` など
`ncmpi_close()`	`MPI_Allreduce()`、`MPI_File_close()` など

2.2 入出力機能のアーキテクチャ

入出力機能のアーキテクチャを**図 1** に示す。計算機

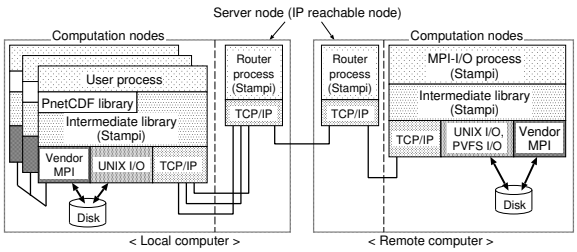

図 1 PnetCDF インタフェースをサポートする計算機間入出力機能のアーキテクチャ。

内部のユーザ・プロセス間はベンダ提供の MPI ライブラリを用いた高速な通信が行なわれる。PnetCDF インタフェースを用い、自計算機上で入出力を行なう場合、Stampi の MPI-I/O インタフェースを通して、計算機ベンダ提供の MPI-I/O ライブラリが利用可能な場合、これを用いた高速な入出力が、利用できない場合は、UNIX I/O を用いた入出力操作が行なわれる。

一方、異なる計算機との通信では、通信相手の計算機にユーザ・プロセスを起動し、TCP ソケットにより接続された通信経路を用いて MPI 通信を行なう。また、各計算ノードが直接外部と通信が出来ない場合は、管理ノードのような外部と直接通信可能なノードに通信中継プロセス（router process）を起動し、これを経由して通信を行なう。なお、プロセス起動においては、リモートシェルコマンド（rsh 又は ssh）を用いている。リモート計算機への入出力でも同様のメカニズムを用いる。ユーザ・プロセス内部でファイルを開くための PnetCDF インタフェースが呼び出されると、まず Stampi の MPI-I/O インタフェース（`MPI_File_open()`）が呼び出される。次に、このインタフェースが、指定したリモート計算機上に入出力を行なう MPI-I/O プロセスを起動し、ユーザ・プロセスからの入出力要求に従った処理を行なう。ベンダ提供の MPI-I/O ライブラリが無ければ Stampi が独自に用意している UNIX I/O 関数による入出力ライブラリを利用する。

2.3 動作メカニズム

PnetCDF インタフェースを用い入出力を行なうプログラムの起動メカニズムについて**図 2** を用いて説明する。 この図に示すように、ユーザはまず始めに

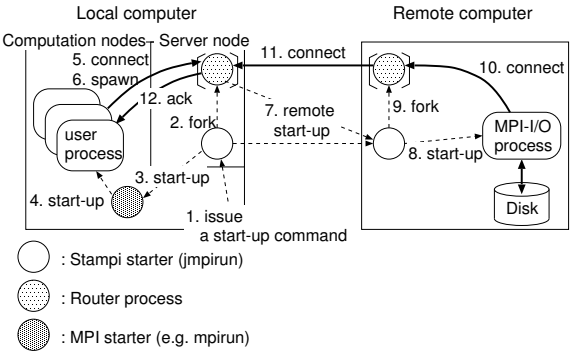

図 2 PnetCDF インタフェースを用いた計算機間入出力の動作メカニズム。

Stampi の起動コマンドにより、Stampi の持つ起動プロセス（starter）を起動する。すると starter は、MPI ライブラリの起動プログラム（例えば mpirun）を起動し、この起動プログラムによりユーザ・プロセスが起動される。ユーザ・プロセス内部で、`ncmpi_create()`（ファイル新規作成時）又は `ncmpi_open()` が呼び出されると、これらの関数内部にある `MPI_File_open()` が呼び出される。この関数呼び出しにより、対応する Stampi の MPI インタフェースが呼び出され、`info` オブジェクトに指定した計算機上に入出力操作を行なう MPI-I/O プロセスを起動する。このプロセス起動時において、ユーザ・プロセスから直接外部と通信出来ない時は、外部と通信可能なノードに通信中継プロセスを起動し、これを介してプロセス起動を行なう。この処理の後に、ユーザ・プロセスと MPI-I/O プロセスの間で通信経路を形成し、これを通して入出力操作が行なわれる。

入出力操作の後、`ncmpi_close()` が呼び出されると、MPI-I/O プロセスはファイルを閉じ、自分自身が消滅することにより、入出力操作が完了する。

3. 性 能 評 価

本実装について、ネットワークで繋がれた PC クラスタ間で入出力性能を評価した。それぞれの PC クラスタの仕様を**表 2** に示す。 それぞれの PC クラスタには 1 台の管理ノードと 4 台の計算ノードがあり、ノード間はギガビットのイーサネットスイッチを介し、1 Gbps の帯域で繋いだ。PC クラスタ II には、SCore[9] をインストールした。さらに PC クラスタ II には、PVFS[10]（version 1.6.3）を導入し、4 台の計算ノードからそれぞれ 73 GByte のディスク領域を束ねて管理ノード上に 292 GByte の並列ファイルシステムを構築した。計算機間の入出力試験では、この PVFS への入出力性能を計測した。また PC クラスタ間はそれぞれのイーサネットスイッチを 1 Gbps の帯域で繋いだ。

計算機間入出力の評価では、PC クラスタ I から PC クラスタ II の PVFS ファイルシステムへの入出力性能を計測した。この試験では、PnetCDF のインタフェースから Stampi の MPI-I/O インタフェースを呼び出した場合と、直接 Stampi の MPI-I/O インタフェースを用いた場合の評価を行なった。

まず、PnetCDF インタフェースを用いた試験では、3 次元データを用い、基本データ型を integer とし、16×16×16、64×64×64、256×256×256 の 3 種類（それぞれ、約 16 KByte、1 MByte、64 MByte に相当）について測定を行なった。測定に用いたプログラムの一例として、書き込み試験用プログラムの一部を**図 3** に示す。 リモート計算機への入出力に必要な情報は `MPI_Info_set()` により `info` オブジェクトに設定される。書き込み操作の場合、これを用い `ncmpi_create()` により netCDF データを新規作成する。さらにデータに関連する情報を `ncmpi_put_att_text()`、`ncmpi_def_dim()`、`ncmpi_def_var()` などにより登録した後に、`ncmpi_enddef()` により、設定モードを終了する。次に `ncmpi_begin_indep()` により独立型（非集団）の入出力モードを開始し、`ncmpi_put_var_int()` により integer（`NC_INT` を使用）のデータ型で入出力（この場合はデータの書き込み）を行なう。そして `ncmpi_end_indep_data()` により、入出力モードの終了操作を行なう。当該ファイルは、`ncmpi_close()` により閉じられ、この時に、リモート計算機上に起動された MPI-I/O プロセスは消滅する。

一方、読み出し操作の場合、`ncmpi_create()` の代わりに `ncmpi_open()` により、既存のファイルをオープンする。さらに当該ファイルに記録されているデータの情報を、`ncmpi_inq_varid()`、`ncmpi_inq_dimid()` や `ncmpi_inq_dimlen()` などで取り出し、次に入出力モードに切替えて `ncmpi_get_var_int()` によりデータを読み出した。

測定では、ファイルをオープンしてから閉じるまで（`ncmpi_create()` 又は `ncmpi_open()` から `ncmpi_close()` まで）の時間と入出力のみ（例えば `ncmpi_put_var_int()`）に要した時間を計測した。測

表 2 使用した二つの PC クラスタ（クラスタ I 並びにクラスタ II）の仕様。

	PC クラスタ I	PC クラスタ II
	管理ノード（Dell PowerEdge800）1 台 計算ノード（Dell PowerEdge800）4 台	管理ノード（Dell PowerEdge1600SC）1 台 計算ノード（Dell PowerEdge1600SC）4 台
CPU	Intel Pentium-4 3.6 GHz × 1	Intel Xeon 2.4 GHz × 2
チップセット	Intel E7221	ServerWorks GC-SL
メモリ	1 GByte DDR2 533 SDRAM	2 GByte DDR 266 SDRAM
ディスク装置	80 GByte (Serial ATA) × 1	73 GByte (Ultra320 SCSI) × 1（管理ノード） 73 GByte (Ultra320 SCSI) × 2（計算ノード）
ネットワーク インターフェース	Broadcom BCM5721 （PCI-Express オンボード）	Intel PRO/1000-XT（PCI-X ボード）
イーサネットスイッチ	3Com SuperStack3 Switch 3812	3Com SuperStack3 Swicth 4900
OS	Fedora Core 3 kernel 2.6.11-1.14_FC3smp	RedHat Linux 7.3 kernel 2.4.20-29.7smp（管理ノード） kernel 2.4.21-2SCOREsmp（計算ノード）
ネットワークドライバ	Broadcom BCM5700 Linux driver version 7.3.5	Intel e1000 version 5.5.4
MPI ライブラリ	MPICH version 1.2.6	MPICH-SCore（MPICH version 1.2.5 ベース）

```
...
char fname[] = ‘‘test.nc’’;
...
MPI_Info_create(&info);
MPI_Info_set(info, ‘‘host’’, ‘‘foo’’);
MPI_Info_set(info, ‘‘user’’, ‘‘aaa’’);
...

for(i = 0; i < 100; i++){
  MPI_Barrier(MPI_COMM_WORLD);

  /* create netCDF file */
  ncmpi_create(comm, fname, NC_CLOBBER, info,
    &ncid);
  ncmpi_put_att_text(ncid, NC_GLOBAL, ‘‘title’’,
    strlen(title), title);
  ncmpi_def_dim(ncid, ‘‘lon’’, transd[i].xsize,
    &lon_dim);
  ncmpi_def_dim(ncid, ‘‘lat’’, transd[i].ysize,
    &lat_dim);
  ncmpi_def_dim(ncid, ‘‘time’’, transd[i].zsize,
    &time_dim);

  /* assign each dimension */
  rh_dimids[0] = time_dim;
  rh_dimids[1] = lat_dim;
  rh_dimids[2] = lon_dim;

  ncmpi_def_var(ncid, ‘‘rh’’, NC_INT, 3,
    rh_dimids, &rh_id);
  ncmpi_put_att_text(ncid, rh_id, ‘‘description’’,
    strlen(description), description);
  ncmpi_enddef(ncid);  /* end of define mode */

  ncmpi_begin_indep_data(ncid); /* start of I/O */
  ncmpi_put_var_int(ncid, rh_id, rh_vals);
  ncmpi_end_indep_data(ncid);  /* end of I/O */

  ncmpi_close(ncid);  /* close netCDF file */
 }
 ...
}
```

図 3 試験に用いたプログラムコード（新規作成と書き込み）の一部。

定結果を**図 4** に示す。 この図で、”`total`” とあるのが、ファイルをオープンしてからファイルを閉じるまでに要した入出力全体に要した時間で、”`pure I/O`” とあるのが入出力操作のみに要した時間である。デー

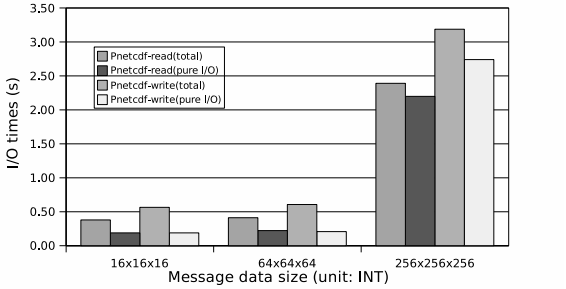

図 4 integer 型を用いた PnetCDF インタフェースによる PC クラスタ間の入出力に要する時間。

タ長が 16×16×16、64×64×64 の場合では、入出力操作のみに要する時間は入出力全体に要した時間に比べ約半分であった。又、読み出し、書き込みについて、データ長が変わっても、それぞれの時間では差はほとんど無かった。この程度のデータ長では、PnetCDF インタフェースを用いる事によるオーバーヘッドが大きいために、差がほとんど無かったと思われる。

一方、256 × 256 × 256 の場合、データ入出力に要する時間が増えたため、入出力全体に要する時間と入出力操作のみに要する時間の差は、読み出し、書き込み共に小さくなった。

次に、PnetCDF インタフェースを用いた場合（入出力操作だけに要する時間：図 4 の `pure I/O` に相当）と直接 MPI-I/O インタフェースを用いた場合の入出力処理に要する時間の比較を行なった。この試験において、MPI-I/O インタフェースを直接用いた場合については、PnetCDF のデータ長に合わせ、16 KByte、1 MByte、64 MByte のデータ長で入出力を行なった。ただしデータ型は派生データ型ではなく、

基本データ型（MPI_INT）を用いた。その結果を**図 5** に示す。 この図で、”MPI-I/O read(simple)”、”MPI-I/O write(simple)” とあるのが、それぞれ MPI-I/O 関数を直接用いた読み出し、書き込み操作である。データ長が小さいところでは、MPI-I/O インタフェースを直接用いる方がはるかに処理時間が短い。一方、256×256×256 の場合、PnetCDF で要した時間は、MPI-I/O インタフェースを直接用いた場合とさほど変わらなかった。この程度のデータ長であれば、PnetCDF インタフェースでリモート入出力を用いることが出来ることが分かった。

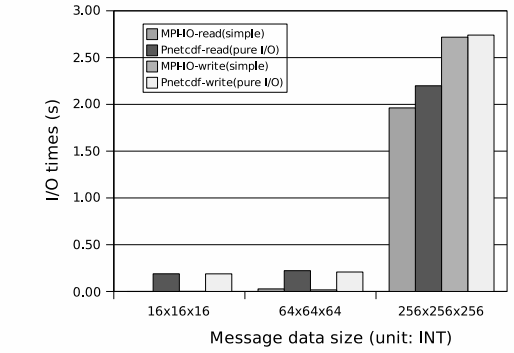

図 5 integer 型を用いた MPI-I/O 並びに PnetCDF インタフェースによる PC クラスタ間の入出力に要する時間。

さらに、PnetCDF インタフェースによる入出力に関し、データ型を double（NC_DOUBLE）にして評価を行った。この試験では、integer 型での試験に用いたデータ長に対応するようにデータ長を 16 × 16 × 8、64 × 64 × 32、256 × 256 × 128 のように変更した。その結果を**図 6** に示す。 この図と図 4 を比べて分かるように、データ長に関して処理に要する時間はほとんど変わらないことが分かった。

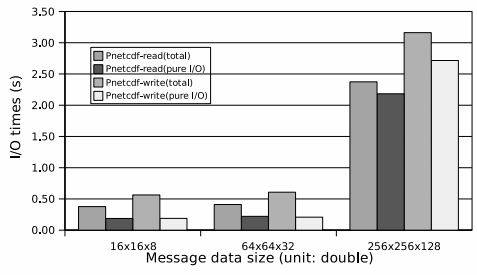

図 6 double 型を用いた PnetCDF インタフェースによる PC クラスタ間の入出力に要する時間。

4. 関 連 研 究

並列計算アプリケーションでは大規模なデータが発生し、それらを共通のフォーマットで取り扱うことで、データの可搬性が高まり、計算を実行する環境に依存する事なく自由にデータの入出力が可能になる。このような試みの代表的なものとして、netCDF と HDF5[11] がある。

netCDF は、多次元データを異なるアーキテクチャを持つ計算機間でも透過的に効率良く扱うための可搬性の高いデータフォーマットを定めており、netCDF のインタフェースを通して自由にアクセスすることができる。入出力処理は逐次処理であるが、並列入出力に対応させたものとして、Parallel netCDF が開発されている。Parallel netCDF インタフェースの下層レイヤには MPI-I/O インタフェースが利用されており、計算機ベンダが提供する MPI-I/O ライブラリあるいは MPICH の MPI-I/O 実装である ROMIO を用いる事で並列入出力において netCDF の形式のデータを扱うことが可能になっている。

一方、HDF5 は、階層構造を持つデータフォーマットを定めており、大規模データを効率良く扱うために開発され利用されている。HDF5 では、Dataset と Group という 2 つのオブジェクトがあり、Dataset では、データ要素の多次元配列を扱い、Group では HDF5 ファイルのオブジェクトの組織化のための仕組みを提供している。MPI-I/O インタフェースを利用した並列入出力インタフェースが用意されており、PVFS が利用可能な Linux が稼働する PC クラスタで、HDF5 から MPI-I/O、PVFS I/O の順に階層的にライブラリを呼び出す形で並列入出力機能を実現した事例も報告されている[12]。

5. まとめと今後の課題

Stampi が持つ計算機間入出力機能を用い、PnetCDF インタフェースによる計算機間入出力機能の実装の試みを行い、計算機間で入出力性能について評価を行なった。この試験の結果、PnetCDF のインタフェースから Stampi の MPI-I/O インタフェースを通して入出力操作を行う場合、PnetCDF のデータ形式で提供するデータに関連する情報を登録する操作などにより、PnetCDF インタフェースを導入する事によるオーバーヘッドがデータ長が短い領域では大きかった。一方、データ長を大きくすると、その影響は小さくなってゆき、256 × 256 × 256（約 64 MByte）の長さでは、直接 Stampi の MPI-I/O インタフェースを用い、

基本データ型による入出力を行う場合とほとんど変わらない時間で入出力処理が出来る事が分かった。このような大規模データであれば、複数の計算機で構成された環境で、対象のデータが存在する場所や、データが存在する計算機のアーキテクチャの違いなどを意識する事なく netCDF のデータ形式で自由に透過的な入出力操作が出来ることになり、有用であると考える。

今後の課題としては、未実装の入出力インタフェース（例えば集団型入出力関数）への対応、異なるサブネット間などでの性能評価、並びに計算アプリケーションへの適用を考えている。

謝辞 本研究を進めるにあたり、日本原子力研究所計算科学技術推進センター長の矢川 元基 氏には数々の援助をして頂きました。また日本原子力研究所計算科学技術推進センターの皆様には Stampi のライブラリを提供して下さり、数々のアドバイスを頂きました。ここに感謝致します。

なお、本研究の一部は文部科学省 科学研究費補助金（若手研究（B）課題番号 15700079）および財団法人 カシオ科学振興財団 平成 16 年度研究助成により行なわれた。

参 考 文 献

1) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard (1995).
2) Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface (1997).
3) Imamura, T., Tsujita, Y., Koide, H. and Takemiya, H.: An Architecture of Stampi: MPI Library on a Cluster of Parallel Computers, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 7th European PVM/MPI Users' Group Meeting, Balatonfüred, Hungary, September 2000, Proceedings* (Dongarra, J., Kacsuk, P. and Podhorszki, N.(eds.)), Lecture Notes in Computer Science, Vol. 1908, Springer, pp. 200–207 (2000).
4) Tsujita, Y., Imamura, T., Takemiya, H. and Yamagishi, N.: Stampi-I/O: A Flexible Parallel-I/O Library for Heterogeneous Computing Environment, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting, Linz, Austria, September 29 - October 2, 2002, Proceedings* (Kranzlmüller, D., Kacsuk, P., Dongarra, J. and Volkert, J.(eds.)), Lecture Notes in Computer Science, Vol. 2474, Springer, pp. 288–295 (2002).
5) Rew, R. K. and Davis, G. P.: The Unidata netCDF: Software for Scientific Data Access, *Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, American Meteorology Society, pp. 33–40 (1990).
6) Li, J., Liao, W.-K., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B. and Zingale, M.: Parallel netCDF: A High-Performance Scientific I/O Interface, *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, IEEE Computer Society, p. 39 (2003).
7) Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: A high-performance, portable implementation of the MPI Message-Passing Interface standard, *Parallel Computing*, Vol. 22, No. 6, pp. 789–828 (1996).
8) Thakur, R., Gropp, W. and Lusk, E.: On Implementing MPI-IO Portably and with High Performance, *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pp. 23–32 (1999).
9) PC クラスタコンソーシアム: `http://www.pccluster.org/`.
10) Carns, P. H., Ligon III, W. B., Ross, R. B. and Thakur, R.: PVFS: A Parallel File System for Linux Clusters, *Proceedings of the 4th Annual Linux Showcase and Conference*, USENIX Association, pp. 317–327 (2000).
11) NCSA: HDF5 home page, `http://hdf.ncsa.uiuc.edu/HDF5`.
12) Ross, R., Nurmi, D., Cheng, A. and Zingale, M.: A Case Study in Application I/O on Linux Clusters, *SC '01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing (CDROM)*, ACM Press, p. 11 (2001).