目次

Cover : Designed by TANI Takushi / T&T Design Lab. (June,2011)

Vol.10, No.1 2011

全国共同利用版

広報

特集「分散・並列処理のための最新ソフトウェア」

「分散メモリ型システムを対象とした高生産・高性能プログラミングモデルXcalableMP」中尾昌広、佐藤三久●「汎用自動チューニング機能インターフェース集OpenATLibおよび数値計算ライブラリXabclib の開発」片桐孝洋●「並列ファイルシステムの高速化の研究」堀　敦史●「Gfarm ファイルシステムによる広域ファイル共有-データインテンシブサイエンスを促進する基盤ソフトウェア-」建部修見●「バックトラックに基づく動的負荷分散フレームワークTascell」平石　拓、八杉昌宏

ISSN 1347-3581

京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

機構長に就任して

情報環境機構長　美濃　導彦

昨年の１０月１日より，情報環境機構長，CIO，CSIO に就任しました．それまでは，学術情報メディアセンター長，総長室副室長などをしていましたので，文科省，大学，全国の基盤センター等の動きがある程度は把握できていましたが，文系，理系を含む大学全体の情報環境となるとあまり真剣に考えておりませんでした．他分野の先生方にとって情報環境は道具であり，使えればいいだけの価値しかありません．情報系では，情報が社会の基盤だ，情報の研究は重要だと考えていますが，それは情報の中だけの話です．この立場になって再認識したことは，大学全体の情報環境はその利用者が多岐にわたっており技術だけでは済まないということ，及び情報系の学問は大学の中であまり存在感のない分野の一つであるということです．

情報技術は社会に広く浸透し，今では誰でもが利用しなければならない時代になって来ました．こうなると，情報は特別な分野ではなく数学や物理などの学問と同じく誰でもが必要な分野になります．数学や物理での専門家は一般人から見ると専門的すぎて何を研究しているのかよくわからないというのと同じレベルで情報の専門家が何をするのか分からなくなってきています．これまでは，「コンピュータが今後重要になる，だから情報を勉強すれば有利である」という論理が明快だったわけです．これが情報系の学科に対する高校生の人気が落ちてきている原因です．社会へのメッセージには常にわかりやすさが求められています．

情報系の学問は，コンピュータの利用技術から社会で扱われている情報の生成，流通，整理，蓄積というところに重点が移ってきており，今後とも社会において重要な分野です．社会の様々な活動が情報を扱うことによって行われていることを考えると，多くの人が必要な考え方，技術が情報であり，その専門分野での研究が社会を牽引していく状況になるでしょう．当面の間は，情報の専門家として様々な学問分野の研究者と共同研究を行うことによって，それぞれの分野がどう変わるか，イノベーションが起こせるかをポイントになります．同時に，その成果を情報技術の進展，情報技術そのものの研究に生かして，螺旋的に技術を発展させていかなければなりません．

こう考えると，計算機を提供して全国共同を進めてきたということは，様々な分野の研究者とのネットワークができていたという意味で先見の明があったのかもしれません．全国共同利用の基盤センター群が協力してネットワーク型の学際大規模情報基盤共同利用共同研究拠点を発足させました．計算科学をそれぞれの分野の研究者と情報系の研究者が共同研究を進める場として考え，計算を必要とするさまざまな科学分野に情報の考え方を持ち込み大きく進展させようとしています．このような試みが，社会にとってわかりやすい説明になっていると認識して，情報系の研究教育を進めていって頂くことを切に期待しています．

Vol. 10, No.1 号の発刊にあたって

京都大学学術情報メディアセンター
岩下　武史

まず、東日本大震災で被災された方々、ご家族の方々に心よりお見舞い申し上げます。国内の研究者にスーパーコンピュータサービスを提供しているセンターの一員として、このような震災の被害を最小限に食い止めるために、計算科学やシミュレーション技術がどういう貢献ができるのか、またセンターとしてどのような協力が可能なのか真摯に検討し、取り組んでいきたいと考えております。

今号では、「分散・並列処理のための最新ソフトウェア」と題する特集を組ませていただきました。現在、学術情報メディアセンターのスーパーコンピュータは2012年春のリプレースに向けて、調達の手続きが進行中です。また、同時期には、京コンピュータが神戸の地で稼働を開始します。京コンピュータでは10万以上の並列度を活用することが求められ、学術情報メディアセンターのスパコンについても現行のシステムと比べて高い並列度を有することになります。このように昨今の計算機環境は急速に変化し、特に大幅な並列度の拡大が見込まれているのに対して、一般に並列プログラミングはデバッグ等の面から困難さがあることが知られています。そこで、このような並列計算環境を簡便かつ効率的に活用するためのソフトウェア研究に関する特集を企画し、5編の寄稿をいただきました。中尾氏、佐藤氏（筑波大）からはXcalableMPと呼ばれる分散メモリ型並列システムを対象とした新しい並列プログラミングモデルを紹介いただきます。XcalableMPはPGASモデルに基づいており、OpenMPのような簡単な指示文を逐次プログラムに加えるだけで分散・並列プログラムを簡易に作成することが可能となるものです。次に、片桐氏（東大）からは、東京大学他で開発中のOpenATLib、Xabclibに関する記事を頂きました。これらのソフトウェアは、それぞれ近年注目される自動チューニング（AT）技術のAPI及びATを活用した数値計算ライブラリです。これらのソフトウェアを活用することで、近年の複雑な構成を有する計算環境に対して、プログラムやライブラリが自動的にチューニングされ、より高い実効性能を得ることができます。また、堀氏（理研）からは並列ファイルシステムに関する近年の技術動向に加えて、同氏が開発したCatwalkに関して紹介をいただきました。Catwalkは、細かい非連続なファイルへの同時並列的なアクセスが並列プログラムで一般的なIO処理であることに着目し、これらのIO処理を効率化することで並列プログラムからのファイルアクセスを改善するソフトウェアです。次に、建部氏（筑波大）からは、同氏が開発を行ってきた分散ファイルシステムGfarmの紹介を頂いています。Gfarmは複数の組織間でのファイル共有をより簡易かつ高性能に実現するソフトウェアであり、京コンピュータと各大学の情報基盤センターの連携を行うHPCIプロジェクトにも採用され、学術情報メディアセンターのユーザが京コンピュータを利用する際にも活用が必須となるソフトウェアです。最後に、平石氏、八杉氏（京大）により、Tascellと呼ばれる高生産並列言語・ランタイムソフトウェアの紹介を頂きました。Tascellはバックトラックベースの動的負荷分散機構を備え、並列計算をより効率的に実行することを可能とします。

本広報では、今後もスーパーコンピュータで活用が期待されるソフトウェア、ハードウェアに関する記事を掲載していく予定です。本センターの広報をご活用いただき、皆様の研究活動に役立つところがあれば幸いです。今後ともどうぞよろしくお願いいたします。

分散メモリ型システムを対象とした 高生産・高性能プログラミングモデル XcalableMP

中尾 昌広 *，佐藤 三久 * †

*筑波大学 計算科学研究センター
†筑波大学大学院 システム情報工学研究科

1 はじめに

分散メモリ型システムにおけるプログラミングモデルとしては MPI（Message Passing Interface）が広く用いられています．しかしながら，各プロセスにデータを分散して配置する，データ送受信のための通信関数を記述する等といった分散メモリ環境特有の処理を考慮する必要があるため，MPI のプログラミングコストの大きさが問題となっています．

そのような背景から，文部科学省が進めている e-Science プロジェクト「シームレス高生産・高性能プログラミング環境」[1] の一貫として，分散メモリ型システムを対象とした新しい並列プログラミングモデル XcalableMP（以下 XMP）[2, 3, 4] を開発しています．XMP は C と Fortran 言語の拡張であり，OpenMP[5] のようにコメントによる指示文を逐次プログラムに加えることで，並列プログラムを簡易に作成することが可能です．

XMP の仕様は，次世代並列プログラミング言語検討委員会によって作成されています．この委員会は，大学・研究機関・企業に所属している研究者から構成されており，様々な機関に属する研究者と議論を行いながら仕様を作成することで，広くそして長く使われるプログラミングモデルの開発を目指しています．

原稿執筆時点における XMP の実装は，C 言語版が筑波大学からオープンソースソフトウェアとして公開されており，`http://www.xcalablemp.org` からダウンロード可能です．XMP の仕様およびチュートリアル資料等もダウンロード可能です．また，メーリングリストによるサポートも行っています．

本稿では，2 章で関連研究について紹介し，3 章で XMP のプログラミングモデルの概要を，4 章で XMP のプログラム例について述べ，5 章で本稿のまとめを行います．

2 関連研究

MPI 以外の分散メモリ型システムを対象としたプログラミングモデルとしては Partitioned Global Address Space（PGAS）モデルが近年非常に注目されています（XMP も PGAS モデルです）．

PGAS モデルは，どのプロセスからでも参照可能なグローバルなメモリ空間（大域メモリ空間）を持ちます．つまり，各プロセスが持つメモリを 1 つの仮想的なメモリ空間として扱えるため，MPI と比較してプログラミングが容易になります．また PGAS モデルでは，どの大域メモリ空間の一部がどのプロセスと対応付けられているかが，抽象化されてユーザに提示されています．そのため，ユーザはデータのローカリティを意識したプログラミングが可能になります．PGAS モデルの特徴をまとめると，生産性と性能のバランスをとったモデルであるということができます．

PGAS モデルに基づいた様々な実装が提案されていますが，中でも比較的知名度が高いのは Unified Parallel C（UPC）[6] と Co-array Fortran（CAF）[7] でしょう．

UPC は C 言語の並列拡張であり，下記のように配列の宣言時に”shared”を追加することで，大域メモリ空間上にデータ領域を確保できます．下記配列は，すべてのプロセスから参照可能です．

```
shared int a[100];
```

さらに，for ループ文を並列化して実行するための upc_forall ループ文が提供されているため，MPI では手動で行っていたイテレーションの分割を自動的に行うことができます．

CAF は Fortran 言語の並列拡張であり，簡単な文で片側通信を記述できるという特徴があります．CAF で用いる配列および変数には codimension という次元が拡張されており，それを用いて通信を表現します．下記に例を示します．

```
y(1:3) = x(2:4)[3]
```

通常の配列の後ろにある角括弧が codimension です．この例では，image 3（image とは MPI のプロセスに相当）の配列 x の 2 から 4 までの要素を全 image の配列 y の 1 から 3 の要素に代入しています．

他の PGAS モデルの実装には X10[8]，Chapel[9]，Titanium[10] などがあります．しかしながら UPC や CAF も含め，まだ一般には利用されていないのが現状です．

3 XcalableMP の概要

3.1 特徴

XMP は分散メモリ環境でのプログラミングの生産性を向上させる言語として設計されています．

XMP は科学技術計算分野で広く用いられている C と Fortran 言語をベースとして，並列化のための言語拡張を行っています．そのため XMP には，C 言語版と Fortran 言語版の XMP がそれぞれ存在していますが，その基本概念は同一になるように工夫されています．また，言語拡張の大半は指示文であるため，どちらかの言語を習得済みであれば，XMP の習得も容易であります．

XMP の指示文は並列言語 High Performance Fortran（HPF）[11] の概念を多く受け継いでいます．HPF も指示文を用いてプログラムの並列化を行う言語ですが，データ通信等が HPF コンパイラによって自動的に行われてしまうため，性能チューニングが難しいという問題点がありました．そこで XMP では，指示文を用いてデータ通信を明示的に記述することで，性能のチューニングについてユーザ自身が工夫できるようにしています．

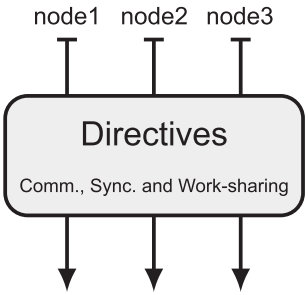


図 1: XMP の実行モデル

さらに，XMP プログラム内では MPI 関数および OpenMP の指示文も利用できるため，既存のソフトウェア資産を有効活用することができます．

3.2 実行モデル

XMP では，1 つのコア，もしくはメモリを共有する複数のコアからなる計算機の単位をノードと呼びます．XMP プログラムで宣言されたデータは，各ノードが持つメモリに配置され，各ノードで実行されるプログラムからアクセスされます．他のノードのメモリ空間を参照・更新するためには，XMP が提供する指示文を用います．

XMP の実行モデルは，MPI と同様に SPMD（Single Program Multiple Data）です．XMP の実行モデルを図 1 に示します．各ノードでは同じプログラムが並列に実行され，プログラム中の XMP 指示文により通信・同期・ループ文の分散処理が発生します．それ以外の箇所は各ノードにおいて重複実行されることになります．

XMP では，大域メモリ空間におけるデータ処理を行う操作をグローバルビューによるプログラミングといいます．また XMP では，ノード番号を指定することによって，他のノードに配置されたローカルデータもアクセスすることができます．この操作をローカルビューによるプログラミングといいます．ユーザはこれらのプログラミングを行いながら，並列アプリケーションの作成を行います．次節から，それぞれのプログラミングについて説明します．

3.3　グローバルビューモデル

グローバルビューは，大域メモリ空間に存在するデータに対し，典型的な通信・同期・ループ文の分散処理を行うための指示文を提供しており，それらの手順を簡潔に記述することができます．

具体的な手順としては，まずユーザは各ノードで共有して利用するデータを各ノードに分散配置する指示文を書きます．次に，各ノードに対し，各ノードが保持しているデータをそれぞれ実行するように指示します．その際，プログラマは必要な指示文を用いて，各ノードが計算を行うのに必要なデータはそのノードにあるように，事前にデータ通信を行う必要があります．

グローバルビューは，逐次プログラムのイメージから出発して，データをノードに分散させ，それに応じた計算の並列化を考えていくプログラミングスタイルに適しています．4 章でグローバルビューモデルのプログラム例を説明します．

3.4　ローカルビューモデル

ローカルビューは，各ノードが持つローカルデータに対して通信を行う機能です．通信を簡易に記述するために，CAF の記述法を採用しています．Fortran 言語版の XMP は CAF と等価です．C 言語版の XMP でも CAF と同等の記述ができるように言語拡張しています．2 章で示した CAF の例を C 言語版 XMP で記述すると下記のようになります．

```
#pragma xmp coarray
y[1:3] = x[2:4]:[3];
```

通常の配列の後のコロンの後にある角括弧が CAF の codimension に相当します．ローカルビューは，各ノードの振る舞いを詳細に記述するプログラミングスタイルに適しています．

4　XcalableMP プログラミング

本章では，C 言語版 XMP を用いたグローバルビューのプログラミングについて説明します．

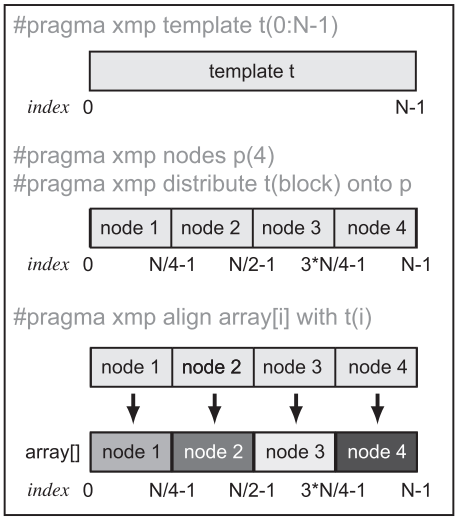


図 2: テンプレートの概念図

```
1  int array[N];
2  #pragma xmp template t(0:N-1)
3  #pragma xmp nodes p(4)
4  #pragma xmp distribute t(block) onto p
5  #pragma xmp align array[i] with t(i)
6
7  main(void){
8    int i, res = 0;
9
10 #pragma xmp loop on t(i)
11   for(i = 0; i < N; i++){
12       array[i] = func(i);
13       res += array[i];
14       }
15
16 #pragma xmp reduction (+:res)
17 }
```

図 3: XMP のプログラム例

4.1　テンプレートを用いたプログラム

グローバルビューでは，まず指示文を用いて各ノードに配列データを分散配置します．分散配置されたデータは，仮想的なインデックス配列であるテンプレートを用いて操作が行われます．図 2 にテンプレートの概念図を，プログラム例を図 3 に示します．#pragma xmp で始まっている文が XMP 指示文です．XMP 指示文は通常のコンパイラ（gcc など）では無視されます．

図 3 の 2 行目においてインデックスの下限値（0）と上限値（N-1）を設定したテンプレートを宣言しています．3 行目では計算に用いるノード集合（この例では 4 ノード）を定義しています．4 行目では

ノード集合に対するテンプレートの割り当て方法を設定し，5 行目では配列とテンプレートとの関連付けを行っています．XMP がサポートしているノード集合に対するテンプレートの割り当て方法は下記の通りです．

- block：配列を出来る限り均一なサイズにブロック状に分割し，各ノードに割り当てる
- cyclic：配列を 1 要素ずつラウンドロビンで各ノードに割り当てる
- block-cyclic：配列を n 要素ずつのブロックに分割し，ラウンドロビンで各ノードに割り当てる
- gblock：各ノードに割り当てる配列の要素数を 1 ノードずつ任意に指定する

11〜14 行目のループ文は，10 行目の指示文により各プロセスが独立に処理します．図 3 ではテンプレートは block 分割されているので，ノード 1 は i=0 から N/4 -1 を，ノード 2 では i=N/4 から N/2-1 を処理します．注意点として，図 3 におけるループ文終了時の変数 res の値は各ノードで異なります．そのため，16 行目で reduction 指示文を用いることにより，各ノードの変数 res の値の総計を求めています．reduction 指示文で用いることができる演算子は，定型的な演算子（図 3 で用いている+以外には* や MAX など）が提供されています．

このように，テンプレートを用いることで，逐次プログラムをほとんど変更せずに，並列プログラミングを行うことができます．

4.2 袖領域を用いた計算

差分法などを用いるアプリケーションでは，配列要素 a[i] の計算のために，a[i-1] や a[i+1] といった隣の要素を参照することがよくあります．あるノードが持つ配列要素の端が a[i] の場合，その隣の要素を参照するためには他のノードと通信する必要があります．その際，どのノードが隣の要素を持っているかを計算するのは煩雑ですし，参照の度に通信を行うのも非効率です．そこで，各ノードの持つ配列範囲を少しだけ拡張し，その拡張した範囲に隣の要素を保持しておくことがよく行われます．この拡張された範囲を袖領域または shadow と呼びます．

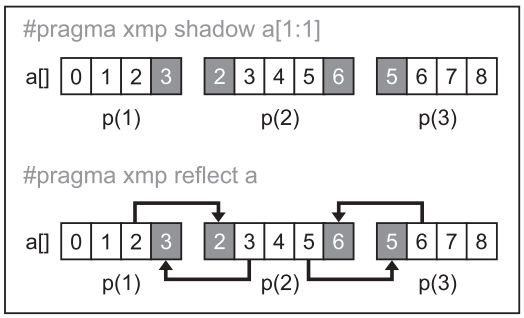


図 4: 袖領域の生成と同期の概念図

```
1   int a[9], b[9];
2   #pragma xmp template t(0:9-1)
3   #pragma xmp nodes p(3)
4   #pragma xmp distribute t(block) onto p
5   #pragma xmp align a[i] with t(i)
6   #pragma xmp align b[i] with t(i)
7   #pragma xmp shadow a[1:1]
8
9   #pragma xmp loop on t(i)
10  for(i = 0; i < 9; i++)
11    a[i] = init(i)    // a[]の初期化
12
13  #pragma xmp reflect a
14
15  #pragma xmp loop on t(i)
16  for(i = 1; i < 8; i++)
17    b[i] = a[i-1] + a[i] + a[i+1];
```

図 5: 袖領域の生成と同期のプログラム例

XMP では shadow 指示文を用いて袖領域の作成を，reflect 指示文を用いて袖領域の同期をとることができます．袖領域の生成と同期の概念図を図 4 に，プログラム例を図 5 に示します．

それぞれの例では，配列 a[] の上端と下端に 1 要素ずつ袖領域を作成しています（shadow 指示文中で定義している配列内のコロンによって上端と下端の要素数を指定しています）．図 4 の灰色の領域が，作成した袖領域です．分散された配列で端にあるものは，片側にしか袖領域は生成されません．図 5 のように，shadow と reflect 指示文を用いることによって，隣接ノードが持つ領域を参照する計算（15〜17 行目）を簡易に記述することができます．

4.3 分散された配列間の代入操作

XMP では，分散された配列間でデータの代入を行うための指示文 gmove が用意されています．gmove

指示文の例を下記に示します．配列 a と b は分散された配列です．

```
#pragma xmp gmove
a[0:3] = b[2:5];
```

この例では配列 b の 2 から 5 までの要素を配列 a の 0 から 3 の要素に代入しています．gmove 指示文では，Fortran 言語のようにコロンを用いて要素の範囲指定を行う記法を採用しています．また，ここで重要な点は，配列 a と b がどのように分散されているかを，ユーザは知る必要はない点です．配列 a と b の参照したい要素の実体はそれぞれ異なるノードにあるかもしれません．また配列 a は block 分割で，配列 b は cyclic 分割されているかもしれません．MPI を用いて同等の記述を行う場合，どの要素がどのノードに保持されているかを手動で計算する必要があるため，そのプログラムは非常に煩雑なものになります．XMP では gmove 指示文を用いることで直感的な配列間の代入操作を記述することができます．

4.4 その他の指示文

XMP では，これまで紹介した指示文以外にも様々な指示文が提供されています．その指示文と例を下記に示します．詳細な利用方法については，URL[4] を参考にして下さい．

- bcast 指示文
  データのブロードキャストを行います．下記の例ではノード番号 1 がローカル変数 *variable* を全ノードにブロードキャストしています．

  ```
  #pragma xmp bcast variable on p(1)
  ```

- barrier 指示文
  各ノードのバリア同期を行います

  ```
  #pragma xmp barrier
  ```

- task 指示文
  指定されたノードがこの指示文の直後の処理を実行します．下記の例では関数 func_a() はノード番号 1 のみ実行されます

  ```
  #pragma xmp task on p(1)
  func_a();
  ```

4.5 プロファイリングツールの利用

本節では，筑波大が実装した C 言語版の XMP コンパイラである Omni XcalableMP Compiler[4] 特有の機能を紹介します．Omni XcalableMP Compiler では，ユーザが性能分析を簡易に行えるように，指示文毎にそのプロファイル情報（データの通信量や処理に要する時間）を取得する機能があります．

プロファイル情報の取得には，既存の並列プロファイリングツールである Scalasca[12] と tlog[13] を用いています．Scalasca は並列アプリケーションの動作を計測・分析し，性能のボトルネックとなっている箇所を特定できるという機能を持ちます．tlog はガントチャートの形式でプロファイル情報を表示できる機能を持ちます．

各プロファイリングツールを使うには，XMP のコンパイル時に下記のようなオプションをつけます．xmpcc は XMP のコンパイルコマンドです．

```
$ xmpcc [-with-scalasca|-with-tlog] [-allprofile|
-profile] input-file ...
```

Scalasca のプロファイル情報を取得したい場合は-with-scalasca を，tlog のプロファイル情報を取得したい場合は-with-tlog を指定します．また，プログラムに用いているすべての指示文についてのプロファイル情報を取得したい場合は-allprofile を指定します．もし，特定の指示文のみのプロファイル情報を取得したい場合は，下記のようにプロファイリングしたい指示文の末尾に「profile」という文字列を挿入し，コンパイルオプションには-profile を指定します．

```
#pragma xmp loop on t(i) profile
```

Scalasca を用いたプロファイル結果を図 6 に示します．図 6 では，各ノードにおける各 XMP 指示文の処理時間を表示しています．左枠で時間やデータの転送量などの項目を選択すると，真中の枠に各指示文の各計測値が表示されます．真中の枠で計測値を選択すると，右枠に各ノードにおける計測値が表

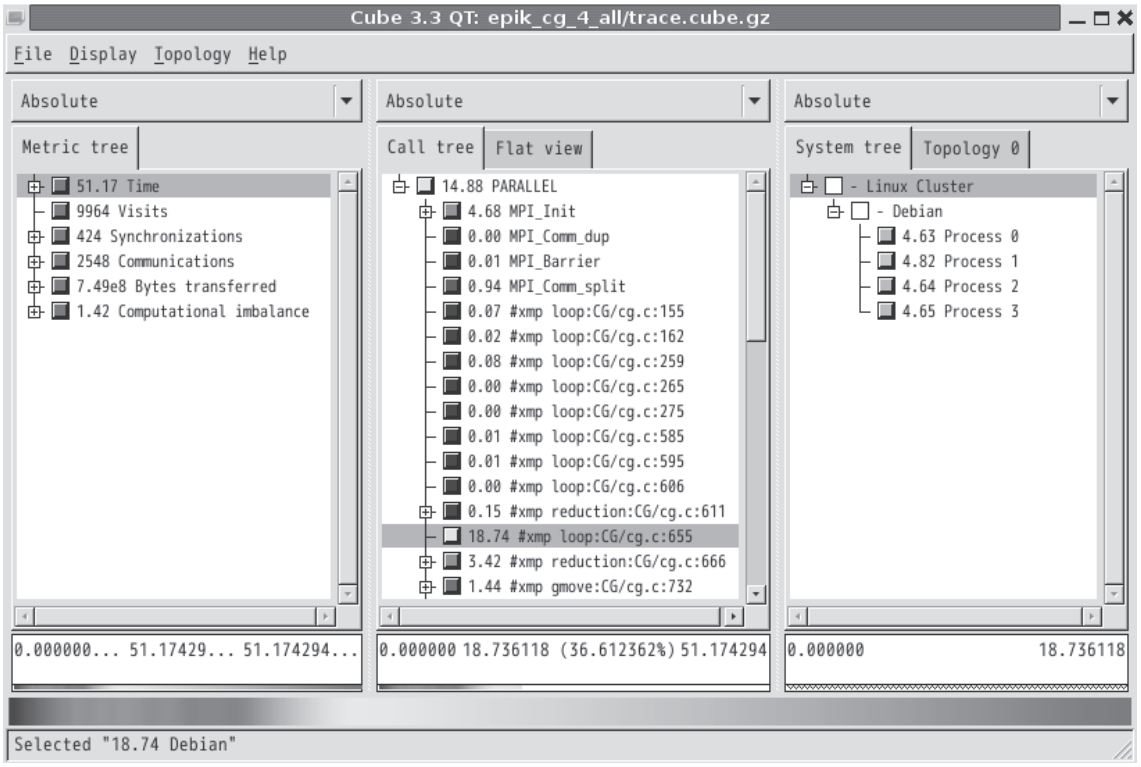


図 6: Scalasca を用いたプロファイリングの結果

示されます．図 6 では，左枠は処理時間，真中の枠は CG/cg.c というファイル中の 655 行目の loop 指示文を選択しています．

なお，Scalasca の機能により，他の並列プロファイリングツールである Vampir[14]，PARAVER[15]，Jumpshot[16] 上でも結果を表示することができます．

tlog の結果を図 7 に示します．図 7 では，横軸は時間を，縦軸は各ノードを示しています．tlog を用いることで，各ノードがどのタイミングでどの指示文を実行しているかを視覚的に理解することができます．

5　まとめと今後の展望

本稿では，分散メモリ型システム用の新しいプログラミングモデルである XMP について説明しました．XMP は C と Fortran 言語に対する指示文ベースの並列拡張であり，MPI よりも簡易に高性能な並列アプリケーションを開発することできます．

XMP の今後の開発予定として，スレッド分割と並列 I/O のための指示文拡張，既存の数値計算ライブラリに対するインタフェースの作成，フォールトトレランス機能，GPU への適用などを考えています．

また，国産の実用的な並列プログラミング言語として，国際的な普及を図ることが非常に重要です．その普及活動の第一歩として，2010 年度は並列プログラミングコンテスト [17] を行いました．今後はワークショップの開催などを予定しています．

今後，神戸の京速コンピュータやその後にやってくるエクサスケールコンピュータを有効利用するためには，数十万コアのオーダーの並列性に対応したプログラミングモデルが必要です．そのような環境においても，十分な性能と生産性を持つようなプログラミングモデルを，XMP をベースに作成していきたいと考えています．

XMP の仕様は，次世代並列プログラミング言語検討委員会によって作成されています．本活動に興味がある方は，下記のメールアドレス（筑波大 佐藤三久教授）にご連絡下さい．
E-mail：msato@cs.tsukuba.ac.jp

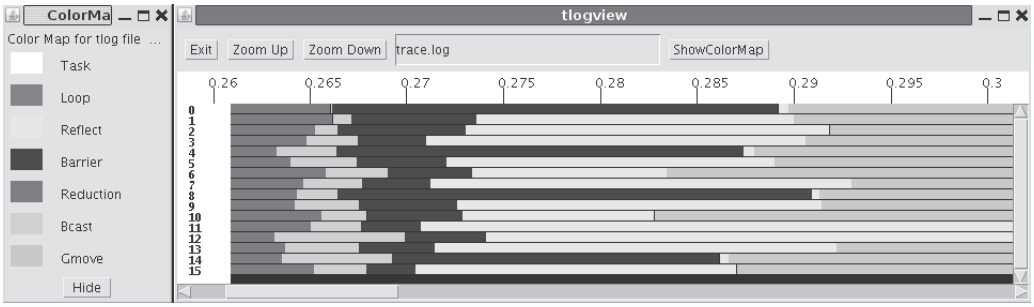


図 7: tlog を用いたプロファイリングの結果

参考文献

[1] 次世代 IT 基盤構築のための研究開発「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」に関する研究開発課題の決定について, http://www.mext.go.jp/b_menu/boshu/detail/08071609/002.htm

[2] 李珍泌, 朴泰祐, 佐藤三久. “分散メモリ向け並列言語 XcalableMP コンパイラの実装と性能評価“, 情報処理学会論文誌 コンピューティングシステム, 2010.

[3] Masahiro Nakao, Jinpil Lee, Taisuke Boku, Mitsuhisa Sato. “XcalableMP Implementation and Performance of NAS Parallel Benchmarks”, Fourth Conference on Partitioned Global Address Space Programming Model (PGAS10), 2010.

[4] XcalableMP. http://www.xcalablemp.org

[5] OpenMP.org. http://openmp.org/wp/

[6] Unified Parallel C at George Washington University. http://upc.gwu.edu

[7] Co-Array Fortran. http://www.co-array.org/

[8] X10 Home. http://x10.codehaus.org/

[9] Chapel Programming Language, http://chapel.cray.com/

[10] Titanium Project Home Page. http://titanium.cs.berkeley.edu/

[11] High Performance Fortran 言語仕様書 Version 2.0. http://www.hpfpc.org/jahpf/spec/hpf-v20-j10.pdf

[12] Scalasca. http://www.scalasca.org/

[13] CCS HPC Summer Seminar 2007, http://www.ccs.tsukuba.ac.jp/workshop/HPCseminar/2007/

[14] Vampir, http://vampir.eu

[15] PARAVER, http://www.bsc.es/plantillaA.php?cat_id=485

[16] Performance Visualization for Parallel Programs, http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm

[17] 第 3 回 クラスタシステム上での並列プログラミングコンテスト. https://www2.cc.u-tokyo.ac.jp/procon2010/

汎用自動チューニング機能インターフェース集 OpenATLib および数値計算ライブラリ Xabclib の開発

片桐　孝洋

東京大学情報基盤センタースーパーコンピューティング研究部門

1　はじめに

計算機の構成が、いままでになく複雑化している。深いメモリ階層と、非均質なメモリアクセスを特徴とするマルチコア計算機が広く浸透している。個人所有の PC も、いまやマルチコア計算機が搭載されている。

複雑化した近年の計算機環境においては、最高性能を達成するためにコンパイラ最適化のみに依存することができなくなっている。人手によるチューニングがいまだになされている。しかし、計算機構成の複雑化でチューニングすべきパラメータが複雑化していることに加え、性能挙動も複雑になっている。このことから、人手によるチューニングさえも困難になっている。

研究室が所有する PC クラスタから、国家が提供する世界最高レベルの性能をもつスーパーコンピュータまで、コードを改変せずに高い性能が維持されることが好ましい。このような＜性能の移植性＞を**性能可搬性**とよぶ。性能可搬性の実現は、低コストのソフトウェア開発に不可避な技術項目である。性能可搬性を達成するために、性能チューニングの自動化が昨今の計算機環境では急務となっている。

一方、数値計算ライブラリは数多くの性能パラメータが存在する。この性能パラメータの設定を誤ると、激しい性能劣化を引き起こすだけでなく、解への収束ができなくなるなど、数値計算の道具としての機能に決定的な影響を及ぼす。数値計算ライブラリへ入力される行列の特性は、数値計算ライブラリのレベルでは事前に分からない。したがって、汎用的なアルゴリズムと実装方式が、初期値として設定されている。また、適用する計算機構成において最高性能を達成するように、性能パラメータを手動で調整することが普通である。したがって、性能可搬性は自動的には達成できない。

そこで近年では、性能パラメータの自動チューニング（今後、単に自動チューニング（Auto-Tuning、AT）と記述する）の研究が盛んである。性能可搬性を達成するために、性能パラメータの自動チューニングを行う。数値計算ライブラリのレベルでは入力行列の特性が実行時にならないと分からない理由から、**実行時**情報を基にした性能パラメータの自動チューニングが、特に重要な課題になっている。

本報告は数値計算ライブラリの AT について解説するものである。数値計算ライブラリ、特に疎行列反復解法ライブラリで必要となる AT 機能の API（Application Programming Interface）をまとめたライブラリ OpenATLib について説明する。また、OpenATLib を用いた AT 機能付きの数値計算ライブラリ Xabclib の解説も行う。

本報告の構成は以下のとおりである。まず第 2 章で OpenATLib と Xabclib の説明をする。3 章は T2K オープンスパコンを利用した OpenATLib の性能評価である。4 章は、今後の開発課題について述べる。5 章はまとめである。

2　OpenATLib と Xabclib

2.1 OpenATLib とは

OpenATLibは、数値計算ライブラリにおけるAT機能をAPI化し、その参照実装を提供することを目的に開発された[1]。Krylov部分空間法を基にした疎行列反復解法ソルバに特化したAT機能について、**実行**

時に行うAT機能を中心に開発を行っている。

我々は平成21年度に開発したOpenATLibβ版を基に、平成22年度で機能を高度化したOpenATLib2011版を開発した。この2011版のOpenATLibの主要関数と機能説明を表１に示す。

表 1　OpenATLib2011 版における関数名と機能

関数名	機能の説明
OpenATI_INIT	OpenATLib と Xabclibのパラメタを設定する
OpenATI_DAFRT	Krylov部分空間のリスタート周期を増加するか判定
OpenATI_DSRMV	CRS形式において、倍精度演算　対称行列用の 疎行列－ベクトル積で、最もよい実装方式の選択
OpenATI_DURMV	CRS形式において、倍精度演算　非対称行列用の 疎行列－ベクトル積で、最もよい実装方式の選択
OpenATI_DSRMV_Setup	OpenATI_DSRMVのための初期データ設定処理
OpenATI_DURMV_Setup	OpenATI_DURMVのための初期データ設定処理
OpenATI_DAFGS	Gram-Schmidt直交化処理において、 4種類の実装から最もよい実装方式の選択
OpenATI_LINEARSOLVE	数値計算ポリシを適用する連立一次方程式ソルバ用 メタ・インタフェース
OpenATI_EIGENSOLVE	数値計算ポリシを適用する固有値ソルバ用 メタ・インタフェース

表1のOpenATLibの関数には命名規則がある。"OpenATI_"で関数名が始まる。最初の１文字は演算精度（S:単精度、D:倍精度）で、2文字以降について、AT機能の場合は"AT"、演算機能の場合は、2文字目は行列形状（S:対称、U:非対称）、3文字目以降は機能名となる。

平成21年度開発のβ版との差異は、2011版ではスレッドセーフ機能を追加した都合からOpenATI_INIT関数が追加された点である。

OpenATLibの最大の特徴は、解への収束性と実行速度に大きく影響する、算法上の性能パラメータの＜リスタート周期＞について、残差履歴を実行時にモニタリングすることでリスタート周期の増減を判定する関数"OpenATI_DAFRT"の提供にある。OpenATI_DAFRTの判定基準はMM比[2]を用いる。2011版についても、この機能に変更はない。

2.2 疎行列データ構造と疎行列-ベクトル積(SpMV)の実現

OpenATLib では、疎行列データの圧縮形式は行圧縮形式（CRS (Compressed Row Storage)、もしくは、CSR (Compressed Sparse Row)）を採用している。この理由は、CRS 形式は単純でわかりやすい形式であること、および、幅広い分野で使われている形式であること、があげられる。また行単位の自明な並列性があり、近年の超並列計算機に向く形式である。欠点は、ベクトル計算機や GPU など、最内ループの疎行列 - ベクトル積（Sparse Matrix-Vector Product, SpMV）演算自体の並列性を抽出する方式では、CRS 形式で行あたりの非零要素数が少ない場合において、SIMD (Single Instruction Multiple Data) 並列性が抽出できない点である。

OpenATLib は、OpenMP によりスレッド並列化がされている。2011 年 5 月現在、MPI 並列化機能は実装されていない。

OpenATLib の SpMV の実装として、以下の 5 種類のスレッド並列の実装を提供している。

- 疎行列 - ベクトル積（SpMV）関数：OpenATI_D{S|U}RMV の実装
  1. 行単位の並列性を利用する方式（対称・非対称行列の双方）
  2. 非零要素数を考慮し、並列実行時の計算負荷をバランス化させる方式（対称・非対称行列用の双方）
  3. 作業行列の零要素について、並列実行時の加算を省く方式（対称行列用のみ）
  4. 新規開発方式のBranchless Segmented Scan(BSS)方式（非対称行列用のみ）
  5. オリジナルのSegmented Scan（SS）方式[3]（非対称行列のみ）

上記の実装 1 は、OpenMP での並列化で通常行う実装方式である。OpenATLib の特徴は、実装 2 ～4 の並列実装方式の提供にある。特に実装 2、3 は、実行時に判明する疎行列の比零要素の分布形状を考慮して行う実装の最適化候補である。したがって、OpenATLib の実行時 AT の設計理念に合致する。

実装 4、5 は、非零要素の分布形状にかかわらず SIMD 並列性を抽出できる観点で面白い方式といえる。CRS 方式では、行単位の並列性を考慮したループを用いての実現になるため、一行当たりの非零要素数が少ない場合は、最内ループ長が短くなる。この理由から、SIMD 並列性の抽出ができない。また、各行の非零要素数の散らばりが大きい場合、並列処理時の負荷バランスが悪くなることで、並列処理効率が悪くなる。一方、実装 4、5 による SS 方式に基づく方式は、最内のルー

プ長を疎行列形状にかかわらず任意に一定長にできる。このことで、SIMD 並列性と計算負荷の均等化が達成できる。

従来の SS 方式による実装 5 は、最内ループに IF 文が存在し、スカラ計算機で激しい性能低下を起こす。そこで、この IF 文を取り除くため、新しいデータ構造を導入したのが、実装 4 の Branchless Segmented Scan (BSS)方式である。BSS のデータ構造を図 1 に示す。

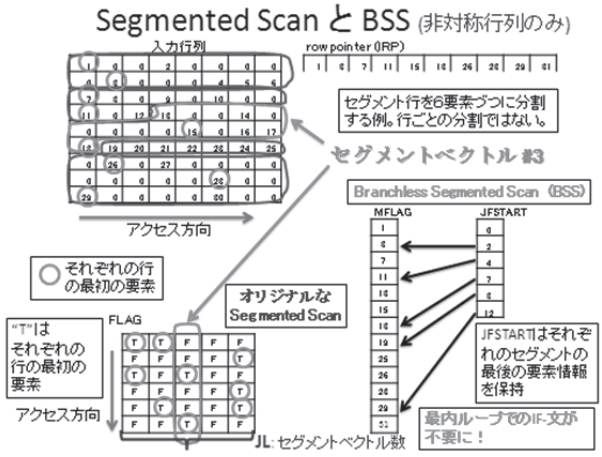


図 1　SpMV における Segmented Scan（SS）方式のデータ構造と BSS のデータ構造

図 1 では、従来の SS 方式では、分割された元の疎行列の要素ベクトル（セグメントベクトル）において、元の疎行列の行の終わりを判断するために持つ、論理型行列 FLAG が必要となる。この FLAG 行列を用い、行列の終わりかどうかを最内側ループで判断するのが、SS 方式の実装である。

一方 BSS では、この IF 文を削除し、単なるループにする。そのため、図 1 の MFLAG と JFSTART のデータ構造を導入した。MFLAG は、セグメントベクトルをアクセスするための非零要素配列のインデックス範囲を保持する配列である。JFSTART は、元の疎行列における最後の行の非零要素の場所を保持する行列である。FLAG 行列の"T"の場所を、MFLAG のインデックスとして JFSTART は保持している。

2.3 Xabclib とは

Xabclib (eXtended Automatically Blocking and Communication adjustment LIBrary) とは、OpenATLib を利用した AT 機能付き自動チューニングライブラリの参照実装として開発された。すなわち、OpenATLib を利用したアプリケーションの位置づけで開発された、数値計算ライブラリである[4]。

2011 年 5 月現在、OpenATLib2011 版を利用し、疎行列に対する標準固有値問題の解法であるリスタート付き Lanczos 法[5]（対称行列用）、陽的リスタート付き Arnoldi 法[6]（非対称行列用）、および、連立一次方程式の解法 GMRES(m)法[7]、伊藤による前処理を実装した BiCG-Stab 法[8]を数値ソルバとして実装している。

2.4 数値計算ポリシーとメタ・インタフェース

OpenATLib は、エンドユーザが所望する AT 方針を指定し、それに基づき AT を実行する機能を実現している。これを、**数値計算ポリシー**機能[9]と呼ぶ。ユーザが指定できる数値計算ポリシーは、OpenATLib2011 版では、実行時間、利用メモリ量、演算精度の 3 種である。

この 3 種のポリシーで行う AT 戦略には違いがある。最も特徴的なのは＜演算精度ポリシー＞である。このポリシーでは、ユーザ指定の精度（要求精度）について、反復解法内での収束基準と、実際の残差との間に乖離がある問題を考慮している。つまり、実際の残差の観点で要求精度を満たせない状況（これを、**偽収束**とよぶ）を回避する目的で実現された新しい実装方式である。

演算精度ポリシーの実装概略を説明する。まず、疎行列反復解法終了後に、実際の残差を計算する。その後、要求精度を満たしていない場合、疎行列反復解法の収束判定閾値 EPS を 10 分の 1 に小さくする。かつ、現在の解を初期推定値に設定する。この上で、再度、疎行列反復解法を呼び出す。これを精度改良ループとよぶ。実装の詳細は、文献[10][11]を参照されたい。

ここで注意する点は、OpenATLib で実現された演算精度ポリシーは、精度を改善する反復改良法の 1 種ではないという点である。また、多倍長演算を利用した高精度化でもない。すなわち、倍精度演算を利用した数値計算ライブラリ上の性能パラメータのチューニング手法の 1 種である。

AT ポリシーが作用する関数は、OpenATLib 関数の中でも、抽象度の高い機能レベルのものである。これはソルバレベルとなる。連立一次方程式のソルバ、固有値ソルバのレベルにおいて、OpenATLib は数値計算ポリシーが適用される API を提供している。それぞれ OpenATI_LINEARSOLVE と OpenATI_EIGENSOLVE である。これらの抽象度の高い、すなわち、内部で OpenATLib 関数がさらに使われて構成されている関数の API を、**メタ・インターフェース**とよぶ。

OpenATLib におけるメタ・インタフェースに対して、ユーザが望む数値計算ポリシーを記述するには、ポリシー記述用のファイル（ポリシーファイル）に、エンドユーザが数値計算ポリシーを記述することで行う。ポリシーファイルには、数値計算ポリシーのほか、要求精度や最大実行時間を記載することができる。ここでは、数値計算ポリシーの指定について説明する。図２がポリシー指定の概略である。

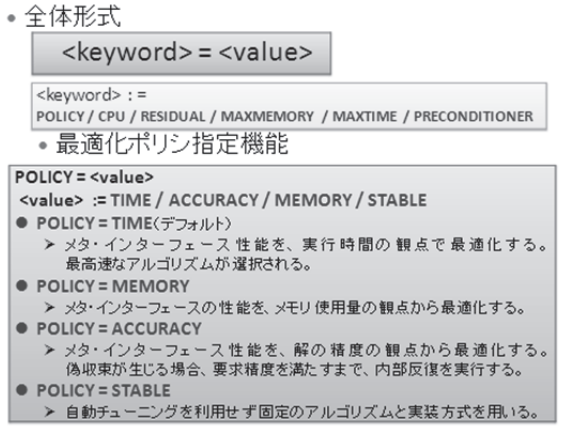


図 2　数値計算ポリシーの記述形式

図２から、ユーザは、演算速度（TIME）、メモリ量（ACCURACY）、演算精度（ACCURACY）、および、固定（STABLE）を指定する。ここで STABLE ポリシーとは、ユーザプログラムのアルゴリズムのデバックのため、自動チューニングを利用しないように OpenATLib 内の実装を固定するポリシーである。STABLE ポリシーは 2011 版から導入された。

2.5 リスタート周期の AT

Krylov 部分空間を利用した反復解法の中には、メモリ量を節約するため、ある反復回数 m（これを**リスタート周期**とよぶ）ごとに今まで保持していた部分空間情報を捨て、適切な初期値を再設定し反復をやり直す処理（これを、リスタートと呼ぶ）が実現されているものがある。このような反復解法を、リスタート付き反復解法とよぶ。

リスタート付き反復解法の問題点は、適切な m の取り方である。m を大きくとると、メモリ量と計算量が増加するが、解への収束性が良くなる。一方、m を小さくとると、メモリ量と計算量が削減されるが、解への収束性が悪くなる。場合によっては、指定の最大反復回数で解へ収束しない。

一般に最適な m は、入力行列の数値特性に依存する。入力行列の数値特性を調べるためには、多くの場合、行列の固有値が必要になる。しかし、行列の固有値計算の計算量は大きく、固有値を調べる間に、解へ収束してしまうことが多い。したがって、実行前に適切な m を推定することは、計算量の観点から困難である。

このような背景から、実行時に経験的に良い m を推定しながら、かつ同時に、反復解法を進めて解を探索していくアルゴリズムは興味深く、AT 研究の一環として多数、研究されている。数値アルゴリズムの特性を利用して m を推定する方法も、いくつかは知られている。しかし数値アルゴリズムを特定すると、AT 手法の汎用性がなくなる。

そこで我々は、数値アルゴリズムの特性を利用せず、実行時の数値モニタリングのみで適切な m の値を推定する方法を採用している。

我々は、上記のモニタリング手法を MM 比[2]という基準をもとに実装している。OpenATLib のルーチンは、OpenATI_DAFRT 関数である。

標記 $R_i\ (s, t)$は、過去の t 番目から s 番目までの値 $r_i(z),\ z=s-t+1, \cdots, s$ を利用するものとする。このとき、以下の式 (1) で、MM 比は定義される。

$$R_i(s,t) = \frac{\max_z \{r_i(z); z = s-t+1, \cdots, s\}}{\min_z \{r_i(z); z = s-t+1, \cdots, s\}} \quad \cdots(1)$$

すなわちMM比は、過去t個分のサンプリング点情報のうち、最大と最少の値の比である。MM比が小さいとき、過去t個分のサンプリング点の変動は小さいことを意味している。一方、MM比が大きいとき、変動が大きいと判断できる。

値$r_i(z)$については、実行時に定まる任意の情報が利用できる。いま、反復解法を考えているので、この値$r_i(z)$に残差ベクトルのノルムを取るのは妥当である。すなわち、残差ベクトルのノルムのMM比が小さいとき、解への収束が停滞していると判断できる。このときは、リスタート周期mを増加させればよい。一方、残差ベクトルのノルムのMM比が大きいとき、解への収束が加速していると判断できる。このときは、リスタート周期mを減少させればよい。解への収束の停滞を判断するMM比の値は、チューニングパラメタとなる。デフォルト値は、複数のテスト行列から適切な経験値を設定しておけばよい。なお、OpenATLib2011における、この初期値は100.0である。

2.6 再直交化アルゴリズムのAT

OpenATLibが有するその他のAT機能として、以下の再直交化機能がある。

- Gram-Schmidt直交化関数：OpenATI_DAFGS
  1. 古典Gram-Schmidt（CGS）
  2. Daniel-Gragg-Kaufman-Stewart型のGramm-Schmidt（DGKS）[12]
  3. 修正Gram-Schmidt（MGS）
  4. ブロック化古典Gram-Schmidt（BCGS）

多くの疎行列反復解法ライブラリは、Krylov空間情報の精度を保つため、直交化*を必要とする。多くのライブラリでは、アルゴリズムの数値計算安定性から実装3のMGS法が実装されている。しかしながら、並列性の観点や演算精度の観点から、実装3のMGS法が最適であるわけではない。並列性の観点では、実装1、4のCGS法系の実装がよい。また、演算精度の観点では、CGS法を2回行う実装2の方法が良いことが知られている。

そこでOpenATLibでは、ユーザが演算精度ポリシーを指定する場合は実装2を採用する。一方、ある閾値より大きな要求精度を利用する場合は、実装4のBCGS法を採用する。それ以外の場合は、実装3のMGS法が利用される。このように、直交化アルゴリズムに関する経験的AT方式を、OpenATLibでは採用している。

3 性能評価

3.1 計算機環境

東京大学情報基盤センターが所有するT2Kオープンスパコン（東大版）の1ノード（16コア）を利用した。最大実行スレッド数は、1ノードあたり16スレッドである。計算ノードは、4ソケットのQuad-Core AMD Opteron Processor 8356（2.3GHz）で構成されている。ノードあたりのメモリ量は32GBである。

OpenATLibはOpenMPを利用して並列化されている。コンパイラはIntel Fortran Compiler Professional Version 11.0である。コンパイラオプションは-O3 -m64 -openmp -mcmodel=mediumである。

3.2 テスト行列と評価条件

テスト行列として、フロリダ大学疎行列コレクションから、固有値ソルバに対して21種（対称行列）、連立一次方程式の解法から22種（非対称行列）を選んだ。

OpenATLib2011版の数値計算ポリシー機能を評価するため、OpenATLib2011版を利用して構築されたXabclib2011版の性能を比較する。最大実行時間制限を1000秒とした。エンドユーザにより与えられた要求精度は、1.0E-8とした。また、Arnoldi法により求める固有値・固有ベクトル数は10個とした。

3.3 リスタート周期のAT効果

OpenATLibの最も特徴的な機能である、リスタート周期調整機能OpenATI_DAFRT関数の効果を示す。表2、表3にGMRES法による実装

---

* より詳細には、1つの初期ベクトルと多数の直交化済みのベクトルを直交化する＜再直交化＞である。多くの直交化されていない初期ベクトルを同時に直交化する＜QR分解＞とは、並列性の観点で異なり、区別して議論しなくてはならない。

(OpenATI_GMRES 関数、非対称行列)、および、Lanczos 法による実装(OpenATI Lanczos 関数、対称行列)において、リスタートの自動チューニングをした場合と、しない場合の評価結果を載せる。なお、自動チューニングをしない場合のリスタート周期は *m=30* に固定されている。

表 2　OpenATI_GMRES におけるリスタート周期の自動チューニング効果（非対称行列）

行列名	固定リスタート周期			自動チューニング			ATによる高速化
	M	リスタート回数	time(sec)	最終M	リスタート回数	time(sec)	
vibrobox	30	24	1.13	35	20	1.04	1.09
Lin	30	1047	601.65	150	54	214.49	2.81
cfd1	30	55	12.42	80	24	11.16	1.11
cfd2	30	45	18.69	70	28	21.31	0.88
gyro	30	10	1.13	35	16	1.43	0.79
t3dl	30	1878	125.62	90	33	6.69	18.79
c-71	30	4	0.70	25	17	1.38	0.51
Si5H12	30	192	17.99	115	34	9.34	1.93
SiO	30	161	24.32	100	37	13.72	1.77
dawson5	30	1052	135.75	105	34	14.79	9.18
H2O	30	623	168.46	130	40	42.01	4.01
F2	30	27	15.04	50	21	15.42	0.97
oilpan	30	28	10.63	45	24	11.97	0.89
shipsec1	30	26	20.89	50	30	30.58	0.68
bmw7st_1	30	1	1.06	15	5	1.27	0.83
SiO2	30	699	805.68	145	45	207.74	3.88
shipsec5	30	53	59.31	75	27	60.41	0.98
Si41Ge41H72	30	411	679.91	150	40	274.68	2.48
bmw3_2	30	3	4.20	25	19	12.93	0.33
Ga41As41H72	30	10000	収束せず	150	44	204.05	収束

表 3　OpenATI_LANCZOS におけるリスタート周期の自動チューニング効果（対称行列）

行列名	固定リスタート周期			自動チューニング			ATによる高速化
	M	リスタート回数	time(sec)	最終M	リスタート回数	time(sec)	
chem_master1	30	22	2.55	52	14	2.25	1.13
torso2	30	1	0.68	7	2	0.31	2.19
torso1	30	1	2.54	2	1	0.72	3.53
torso3	30	12	33.57	32	14	34.11	0.98
memplus	30	5	0.25	22	10	0.20	1.25
ex19	30	1000	収束せず	100	60	26.23	収束
poisson3Da	30	3	0.48	17	7	0.54	0.89
airfoil_2d	30	7	0.73	22	14	0.83	0.88
poisson3Db	30	7	10.38	17	14	11.03	0.94
viscoplastic2	30	19	2.93	37	15	1.70	1.72
xenon1	30	30	20.16	62	19	16.18	1.25
xenon2	30	40	92.96	72	20	64.29	1.45
wang4	30	5	0.37	17	9	0.29	1.28
ecl32	30	1000	収束せず	92	22	11.61	収束
sme3Da	30	670	215.49	100	90	101.33	2.13
sme3Db	30	1000	収束せず	100	120	377.29	収束
sme3Dc	30	1000	収束せず	100	122	575.63	収束
epb1	30	11	0.38	32	14	0.35	1.09
epb2	30	3	0.22	12	9	0.21	1.05
epb3	30	11	3.22	42	14	3.02	1.07

表 2、表 3 から、いくつかの行列では AT を適用することで高速化が達成できる。しかし、すべての場合で高速化されるわけではない。AT を適用して速度低下する場合（速度向上が 1.0 以下の行列）は、リスタート周期 *m=30* に固定した実行でも、リスタート回数が少ない場合が多い。言い換えると、小さいリスタート周期ですぐに収束するような＜簡単な問題＞では、AT は不要である。

より重要なのは、リスタート周期 *m=30* では収束しないような＜難しい＞問題が、AT を適用すると収束することである。このことは、AT により、ユーザが自らリスタート周期のパラメタチューニングしなくても解を求めることができることを意味している。したがって、簡易なライブラリ利用の促進と、性能パラメタチューニングのためのコスト削減が達成できる。

3.4 演算速度ポリシーの評価

次に本節では、数値計算ポリシーの性能評価の一部を紹介する。Arnoldi 法による固有値ソルバ（非対称行列用）のメタ・インタフェースである OpenATI_ EIGENSOLVE (ARNODLI)において、演算速度、メモリ量、演算精度を指定した場合の実行時間を図 3 に載せる。

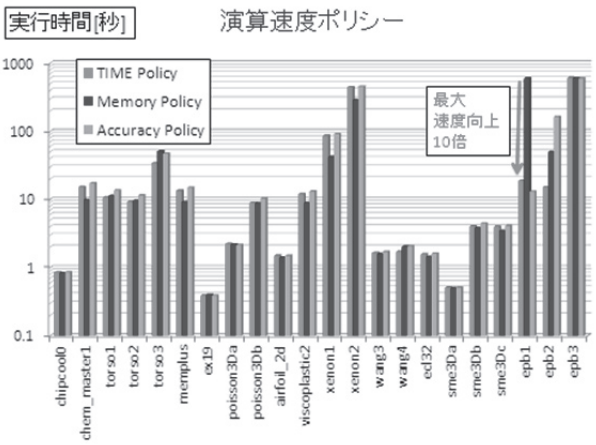


図 3　OpenATI_EIGENSOLVE(ARNORDI)の演算速度ポリシーの効果

本実験では、演算速度ポリシーがもっとも高速であることが期待される。ところが図 3 から、必ずしも演算速度ポリシーは高速とならない。3 種のポリシーの実行時間がほぼ同じになる場合が多い。この理由は、テスト行列において、小さいリスタート周期ですぐに収束する問題が多いことによる。すなわち、メモリを多く消費して高速収束させようと試みても、メモリ量を少なくして実行した場合と収束性はあまり変わらないことによる。

だが図 3 から、いくつかの行列では、演算精度ポリシーは、その他の 2 ポリシーに対して高速化

できる場合がある。特に、行列「epb1」では、メモリ量ポリシーの実行時間に対し、演算速度ポリシーは、約 10 倍程度も高速化ができる。

4　今後の開発課題

OpenATLib は、平成 23 年度の最終版開発期間において、さらなる機能拡張や規格化を予定している。我々は以下について、機能拡張やライブラリの充実をすべきと考えている。しかしながら、平成 23 年度において、すべての実現は不可能である。多くは、今後の研究課題として残る。

疎行列 - ベクトル積機能

- 非零要素を並び替えるオーダリング
- 疎行列データ形式の変換機能。たとえば、CRS 形式から ELL(Ell Pack)形式への実行時変換。
- 疎行列形状に特化した高速化手法。たとえば、7 点差分や 27 点差分行列の演算に特化する。
- 実行時の最適なループアンローリング段数選択
- 演算アクセレータと CPU との協調動作

数値計算ポリシー機能

- コア数の実行時最適化
- 消費電力の実行時最適化
- メタ・インタフェース内のソルバ切り替え
- 前処理方式の自動選択
- 部分的多倍長計算や精度保障
- チューニング情報のデータベース化のためのデータ構造の規格化

数値計算アルゴリズム

- ICCG など、よくつかわれる数値計算アルゴリズムと、先進的な算法、たとえば IDR（*s*）法の実装

応用プログラムへシームレスな接続

- 応用問題レベルの情報（メッシュの構造情報など）を利用した前処理方式の実装と API 化
- 分散並列環境用（MPI 版）の API を、応用プログラムの事例から規格化

5　おわりに

我々は、数値計算ライブラリにおける自動チューニング（AT）機能のプログラム上のインターフェースとしての規格化（API 化）のため、疎行列反復解法ソルバで必要な実行時 AT 機能を提供する OpenATLib を開発している。また、OpenATLib を利用した AT 機能付き数値計算ライブラリの参照実装を提供するため Xabclib を開発している。

OpenATLib および Xabclib は、現在も機能高度化のためインターフェースの変更が予定されている。だが平成 23 年度はプロジェクトの最終年度であるため、最終仕様は平成 23 年度中に定める。

OpenATLib および Xabclib のソースコートは、LGPL ライセンスで、PC クラスタコンソーシアム [13]で公開されている。

謝辞

本研究は、文部科学省 e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発「シームレス高生産・高性能プログラミング環境」の支援による。

日頃議論いただく、東京大学情報基盤センターの大島聡史助教、伊藤祥司特任准教授、中島研吾教授、愛媛大学の黒田久泰准教授(東京大学併任)、日立製作所中央研究所の櫻井隆雄氏、直野健博士、日立超 LSI システムズの猪貝光祥氏に感謝の意を表する。

参考文献

[1] 片桐孝洋, 櫻井隆雄, 黒田久泰, 直野健, 中島研吾, OpenATLib:汎用的な自動チューニングインターフェースの設計と実装, 情報処理学会研究報告:ハイパフォーマンスコンピューティング, 2009-HPC-121(3), 2009 年.

[2] 櫻井隆雄, 直野健, 恵木正史, 猪貝光祥, 木立啓之, リスタート付ランチョス法における実行時パラメータ自動チューニング方式の提案, 情報処理学会研究報告:ハイパフォーマンスコンピューティング, 2007(80), 173-178, 2007 年.

[3] Blelloch,G.E., Heroux, M.A.,and Zagha, M., Segmented operations for sparse matrix computation on vector multiprocessors, CMU-CS-93-173, Carnegie Mellon University, Pittsburgh, PA, 1993年.

[4] 櫻井隆雄, 直野健, 片桐孝洋, 中島研吾, 黒田久泰, OpenATLibを利用した疎行列ライブラリの開発と評価, 情報処理学会研究報告:ハイパフォーマンスコンピューティング, 2009-HPC-121(17), 2009年.

[5] Hernandez, V., Roman, J.E., and Tomas, A., Evaluation of Several Variants of Explicitly Restarted Lanczos Eigensolvers and Their Parallel Implementations, High Performance Computing for Computational Science -VECPAR2006, 403-416, 2007年.

[6]Lehoucq, R.B., Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration, Technical Report TR95-13, Department of Computational and Applied Mathematics, Rice University, 1995年.

[7] Saad, Y. ,Iterative methods for sparse linear systems, SIAM, 1996年.

[8] Itoh, S., Katagiri, T., Sakurai, T,, Igai, M.,Ohshima, S., Kuroda, H.,Naono, K.,Nakajima, K., An Improvement in Preconditioned BiCGStab Method, ２０１１年ハイパフォーマンスコンピューティングと計算科学論文集ＨＰＣＳ２０１１, ２０１１年1月１８日（火）－1月１９日（水）, ＨＰＣＳ２０１１論文集, 2011年.

[9] 直野健, 猪貝光祥, 木立啓之, 数値計算ポリシー入力型自動チューニング方式, 情報処理学会研究報告:ハイパフォーマンスコンピューティング, 2005(81), 31-36, 2005年.

[10]櫻井隆雄, 直野健, 片桐孝洋, 中島研吾, 黒田久泰, 猪貝光祥, 数値計算ポリシーインターフェース付行列計算ライブラリの開発と評価, ２０１１年ハイパフォーマンスコンピューティングと計算科学論文集ＨＰＣＳ２０１１、２０１１年1月１８日（火）－1月１９日（水)、ＨＰＣＳ２０１１論文集, 2011年.

[11]片桐孝洋, 櫻井隆雄, 黒田久泰, 直野健, 中島研吾, Xabclib：汎用的自動チューニングインターフェースOpenATLib を利用した反復解法ライブラリの開発,日本応用数理学会学会誌「応用数理」(岩波書店), 20 (4), 25-37, 2010年.

[12] Daniel, J.W., Gragg, W.B., Kaufman, L., and Stewart, G. W., Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization, Math. of Computation, 30, 772-795, 1976年.

[13]PCクラスタコンソーシアム
http://www.pccluster.org/

並列ファイルシステムの高速化の研究

堀 敦史 *

*理化学研究所計算科学研究機構システムソフトウェア研究チーム 研究員

1 はじめに

eScience プロジェクト

本研究は文部科学省の支援を受けた eScience プロジェクトの一環です。まず最初に eScience プロジェクトの概要を説明しようと思います[1]。

昨今の科学技術計算用コンピュータは、クラスタ構成のものが主流となり、その性能も年々向上し、2012年には日本の「京コンピュータ」が 10 PFLOPS を達成する見込みです。一方、ソフトウェアの方は、MPIで書いたプログラムをクラスタ上で走らせるという程度にしか、大きな進歩が見られません。eScienceプロジェクトでは、今後のペタスケールの時代におけるクラスタ用ソフトウェアの研究開発プロジェクトで、以下のサブプロジェクトから構成されています。

XcalableMP MPI に代わる新しい並列プログラミング言語

Xcrypt 複数のジョブを投入するためのスクリプト言語

Xabclib 自動チューニング数値計算ライブラリ

Xruntime 性能と可搬性を重視した実行環境

本稿で述べる内容は、Xruntime サブプロジェクトの一部なので、以下にもう少し詳細について説明したいと思います。

Xruntime

クラスタはコストパフォーマンスの高い科学技術計算用の並列コンピュータとして広く使われるようになってきました。研究室レベルで使われる数十ノード程度の比較的小型なクラスタから T2K 京大にあるような数百台規模のクラスタと幅広い性能のダイナミックレンジがあります。そこで動くプログラムの大半は MPI 通信ライブラリを用いて記述されていますので、一見、プログラムの可搬性が高く、研究室レベルのクラスタで開発したソフトウェアを大規模クラスタの上で動かすことは容易であるように思われます。これは、研究室が所有する小規模のクラスタで開発したプログラムを大きな規模の計算のために、例えば T2K クラスタ上で走らせたい、というような場合です。しかしながら実際には、システムソフトウェアのバージョンや、MPI ライブラリの細かい差異、並列ファイルシステムの違いなどが存在し、違いを全く意識せずに使う事はできないのが現状です。

同じハードウェア仕様をベースに構築された T2K の京大、東大、筑波大の 3 つのクラスタでさえも、MPI のバージョン、バッチスケジューラ、ファイルシステムがそれぞれ異なり、あるユーザがこれら 3 台のクラスタを使おうとすると、システム毎にプログラムを再コンパイルしたり、バッチスクリプトを修正したり、ファイル I/O の特性を考慮したりする必要があります。

本サブプロジェクトでは、このような問題意識から、以下の研究開発項目が挙げ、違うクラスタを使う場合にユーザの負担をできるだけ低減させようとするものです。

- 並列ファイル I/O
- 可搬な MPI プログラムの実行環境
- 汎用バッチスケジューラ

以上が前置きで、これからは本題である並列ファ

[1] http://www.open-supercomputer.org/escience/e-Science.html

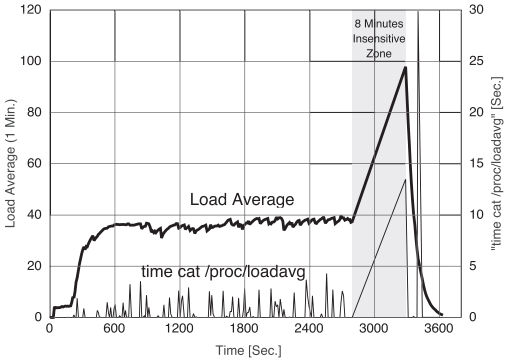


図 1: T2K 東京 4 ノードでの NFS へのファイル書込時の負荷の様子

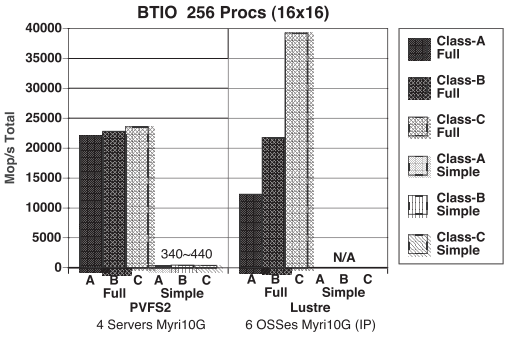


図 2: PVFS2 と Lustre の BTIO 性能

イル I/O についての話を進めます。

並列ファイル I/O

演算装置は毎年驚異的な性能向上を続けている一方で、ファイルシステムの要であるディスクの性能は、容量に関しては向上を続けているものの、ほとんど向上していません。最近ではより高速な SSD も普及が進みつつありますが、まだまだ高価なため、クラスタに導入されるケースは、これから増える見込みですが、現時点では希少です。いずれにせよ、演算性能と I/O 性能の性能ギャップは広がるばかりです。図 1 は T2K 東大の 4 ノード（計 64 プロセス）から 1 台の NFS サーバに対し、1 GByte のファイルを書込んだときの NFS サーバの LoadAverage（1 min.） の時間変化を示したものです（太線、左 Y 軸）。この図にあるように、たった 4 台からの書込みに対し、1 台の NFS サーバは過負荷状態に陥り、負荷をサンプリングするために発行したコマンドの応答時間（細線、右 Y 軸）もかなり悪く、終盤の特に負荷が高い期間においてはおよそ 8 分間も応答しない状態に陥っていることが分かります。

NFS はもともと分散ファイルシステムとして 20 年以上も前に開発された技術で、現在では広く使われています。しかしながら最近のクラスタの台頭により、並列かつほとんど同時に一斉にアクセスされると図 1 に示したように性能低下が顕著です。T2K のような大規模なクラスタでは、NFS のような分散ファイルシステムではなく、並列ファイルシステムが使われています。しかしながら、並列ファイルシステムを構築するには、多数のディスクやサーバ、さらにそれらをつなぐ高速な専用ネットワークが必要になり、それなりの金額がかかります。

並列ファイルシステムは、多数のディスクを同時に用い、それを並列にアクセスすることで性能を稼ぐという原理になっています。単純計算では、10 台のディスクで 10 倍の、100 台のディスクで 100 倍の性能を実現しようとするものです。一方、並列計算の方では、数百、数千といったプロセスから同時にファイルへのアクセス要求が出ます。並列ファイルシステムでは、ひとつのジョブが一度にひとつのファイル、これはファイルアクセス性能を向上させるために数十、数百のディスクに分散されている、に並列にアクセスします。あるプロセスはひとつのファイルの自分の担当の部分にのみアクセスし、別なプロセスはファイルの別な部分をアクセスします。多くの場合、並列ファイルシステムは並列ジョブのプロセス数より少ないディスクしか持たない場合が多いので、ひとつのディスクからみた場合、そのディスクが保持するファイルの一部分のデータに対し、複数のプロセスから違う場所をアクセスすることになります。ところが、ディスクが一番性能を発揮するのは、ひとつのファイルを最初から終わりに向かって連続にアクセスする場合なので、不連続なアクセスになりがちな並列ファイルアクセスではディスクの性能は大きく低下する可能性があります。

図 2 には、代表的な並列ファイルシステムである、PVFS2 と Lustre 上に、並列 I/O のベンチマークプログラムである BTIO を用いて評価をした結果を示します。この時、実際の I/O は MPI-IO を用いて記述されています。BTIO は問題の大きさにより

A、B、C と 3 つのクラスがあり、クラス A で 419.43 MB、クラス B で 1697.93 MB、クラス C で 6802.44 MB のファイルを新たに作ります。またファイルのアクセス方法についても simple と full の 2 種類があり、simple では非 collective な MPI-IO（主に `MPI_File_write_at`）、full の場合では collective な MPI-IO（主に `MPI_File_write_at_all`）が使われているという違いがあります。この図の縦軸は BTIO が自分で計測した、ファイル I/O を含んだ計算速度ですので、数値が大きい程、性能が良いことになります。この計測では T2K 東大の計算ノードを 16 台（256 プロセス）を用いました。PVFS2 では計算とは別な計算ノード 4 台をファイルサーバとして用い、Lustre では専用の 6 台のファイルサーバ（OSS）を用いています。また PVFS2 および Lustre において、計算ノードとファイルサーバは Myri10G（10Mbps）を用いています。

このグラフから、PVFS2 および Lustre とも full アクセス方式での性能が高いのですが、simple での性能が全く出ていません。特に Lustre では MPI-IO の内部エラーとなって計測不可能な状況でした。PVFS2 と Lustre において collective な方の性能が良いのは、MPI-IO の実装である ROMIO が、細かい I/O 要求をまとめて大きな要求に変換する最適化の処理を行っているからですが、この最適化は collective な MPI-IO にしか適用されません。BT-IO では collective と非 collective の両方 MPI-IO によるコードが書かれていますが、FORTRAN で collective な MPI-IO を呼び出すには、それなりに複雑なプログラムを書く必要があり、プログラマにとっては大きな負担になります。一方、非 collective な MPI-IO の記述では、逐次コードからの自然な延長として書かれています。これは私の主観的な判断ですが、プログラミングという観点からは collective な MPI-IO はあまり望ましいものではありません。

2 Catwalk

本稿で紹介する Catwalk は、そもそも並列プログラムからのファイルへのアクセスは細かい非連続なファイルへの同時並列的なアクセスが主な原因であるという仮定から、敢えて逐次のアクセスに変換することで、ディスクの性能を最大限に引き出そうとするものです。図 3 は Catwalk の内部のプロセス

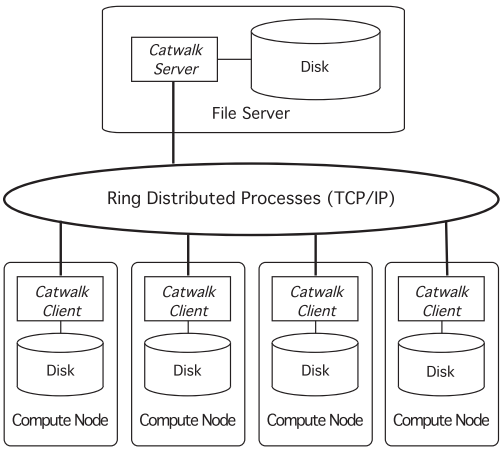


図 3: Catwalk-ROMIO の分散リングプロセス構造

構造を示しています。各計算ノードには Catwalk のクライアントプロセスが走り、ファイルサーバ上には Catwalk のサーバプロセスがあります。今のところ Catwalk は、並列ファイルシステムとは違い、ひとつのファイルサーバのみを使います。Catwalk クライアントプロセス間とサーバプロセス間は普通の TCP/IP で通信路が形成されています。

計算ノード上では、Catwalk クライアントプロセスが、ユーザのプログラムからの I/O 要求を受け付け、リングに沿ってその要求を最終的にサーバ上に送り、実際のファイルへのアクセスとなります。このリングに沿って転送される時に、I/O 要求を並べ替えたり、複数の細かい I/O 要求をより大きなひとつの I/O 要求に変換したりして、最終的にファイルサーバ上でのディスクの性能が最大限となるようにします。内部処理の詳細に関しましては、参考文献を参照してください。

Catwalk は、（MPI-IO でない）普通の I/O を受け付ける *Catwalk* と、MPI-IO を受け付ける *Catwalk-ROMIO* の 2 種類があります。両方ともオープンソースとして公開されており、インストールするだけで誰でもが使うことが可能です。実際、京大、東大、筑波大の各 T2K クラスタ上での実行が確認されています。

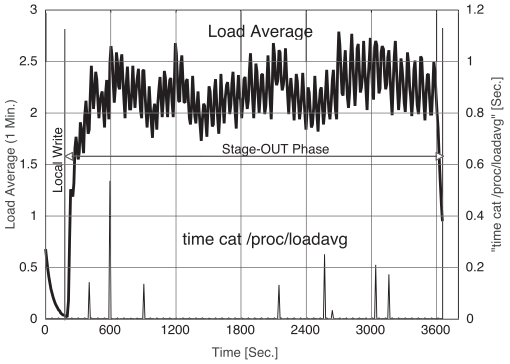


図 4: Catwalk を用いた場合のサーバの負荷の様子

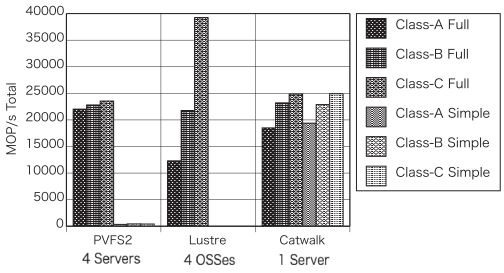


図 5: Catwalk-ROMIO の BTIO 性能

3　実験

図 4 は先に説明した図 1 と同じ条件で計測した Catwalk ファイルサーバの負荷状況を示します。縦軸のスケールが大きく異なっていて、Catwalk の場合のサーバの負荷が NFS に比べはるかに低いことが分かります。同時に、横軸の値を比較すると、NFS とほぼ同じ時間で終了していることが分かります。

図 5 は図 2 と同じ条件で計測した Catwalk を用いた場合の BTIO の結果を、図 2 上に重ねたものです。BTIO の full の場合では、Catwalk は PVFS2 とほぼ同等の性能を示し、問題サイズが小さい場合には Lustre を上回る性能を示しました。一方、Catwalk の simple の性能は full の場合とほぼ同じであり、PVFS2 や Lustre の性能を大幅に上回っています。Catwak はたった 1 台のディスクしか用いていないにも関わらず、複数のディスクを用いた PVFS2 や Lustre と同等あるいはそれ以上の性能を発揮していることが分かります。

4　遠隔ファイルアクセス

例えば、手元のラップトップにあるデータファイルを使って T2K クラスタで計算したいとします（図 6）。一般には、いったん手元のファイルを scp コマンド等でクラスタの共有ファイルシステム上にコピーし、それからクラスタで計算し、もしその結果が欲しければ、再度、出力結果を手元のラップトップにコピーしなければなりません。同じクラスタ上で何度も同じファイルを使って計算する場合はこれでもよいのですが、あちこちのクラスタを使って計算するような場合には、違うクラスタを使うたびにファイルをコピーしなければなりませんので煩雑です。

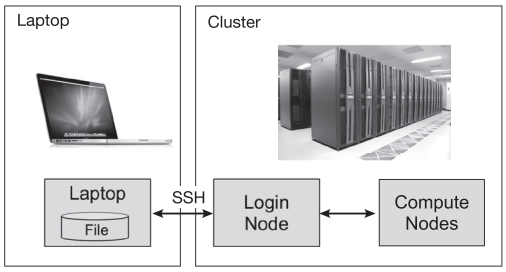


図 6: 遠隔ファイルアクセスの例

```
LoginNode$ catwalk mpirun catwalk a.out
```

図 7: クラスタ内で Catwalk を使う場合の実行例

```
LAPTOP$ catwalk -ssh
MyName@T2KLoginNode
LoginNode$ mpirun catwalk a.out
```

図 8: 遠隔ファイルアクセスするための実行例

Catwalk-ROMIO は遠隔にあるファイルを直接アクセスすることも可能です。通常、クラスタ内のファイルを Catwalk でアクセスする場合は、図 7 に示したように `mpirun` コマンドの前後に `catwalk` を指定します。`mpirun` の左側の `catwalk` は Catwalk サーバプロセス（図 3 参照）の起動で、右側の `catwalk` はクライアントプロセス（同じく図 3 参照）の起動を行っています。遠隔にあるファイルをアクセスする

場合、例えば図 8 にありますように、ssh コマンドでクラスタにログインするのではなく、catwalk コマンドでログインします。このようにログインしてしまえば、後は mpirun コマンドの後に catwalk を指定するだけで済みます。Catwalk サーバプロセスの起動は、ssh コマンドの代わりにラップトップ上で実行した “catwalk -ssh” が自動的におこないます。これだけのことで、ラップトップ上にあるファイルを陽にコピーすることなくクラスタからアクセスすることが可能になります。残念ながら現在の Catwalk は Linux しかサポートしていませんので、この場合のラップトップには Linux がインストールされている必要があります。

この遠隔ファイルアクセス機能は、残念ながら Catwalk-ROMIO だけで Catwalk では使えません。

5 おわりに

以上、Catwalk を御紹介してきました。Catwalk は並列プログラムからのファイルアクセスを効率的にするためのプログラムです。評価実験からは、Catwalk はひとつのディスクしか用いていないにもかかわらず、より多数のディスクを用いた並列ファイルシステム PVFS2 や Lustre と比肩できる性能を発揮することが判明しました。また、PVFS2 や Lustre が不得意とするようなデータアクセスパターンでは Catwalk が圧倒的に良い性能を示すことも示されました。残念ながらこれらの結果から、どんなファイルアクセスパターンでも Catwalk の方が良い性能を示すということが示された訳ではありません。Catwalk は普通のディスクでも NFS や PVFS2、Lustre といった分散／並列ファイルシステム上でも動きますので、お使いのクラスタがサポートしているファイルシステムで並列アクセス性能が低い場合に、Catwalk を試してみることが可能です。Catwalk を使う事で性能が良くなる可能性は少なくないと思われます。

Catwalk の欠点は単一のファイルサーバしか持てないことです。これはひとつのディスクの性能が Catwalk の性能の上限を意味しません。ファイルサーバに RAID など高性能はストレージ装置を導入すれば、Catwalk はその性能を引き出すことができます。多くの場合、Catwalk の性能の上限は、図 3 にあるリングを構成するためのネットワークの性能です。

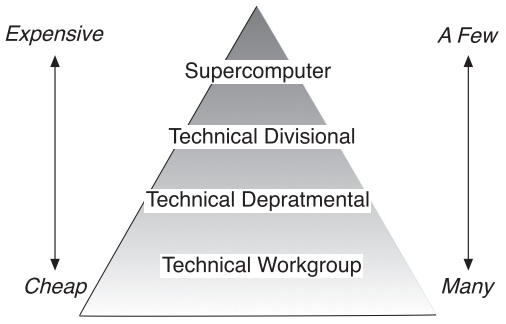


図 9: クラスタユーザの階層

Catwalk はファイルサーバ以外に、並列ファイルシステムのような高価なハードウェア投資が不要です。

図 9 は IDC[2] が出した 2008 年の科学技術計算用クラスタに関するレポートをベースに、ユーザ（所有者）の分類と数を簡単な図に表した物です。この図において、例えば T2K クラスタは、ピラミッドの頂点付近に位置づけされます。比較的規模の大きなクラスタにおいては専用の並列ファイルシステム装置を設けることが多いのですが、T2K 以上に大規模なクラスタでは、並列ファイルシステムの性能が十分にスケールせず、必要とされる性能を提供することができない場面も多々見られます。一方、この図の底辺付近に位置する小規模なクラスタでは、専用のファイルシステムよりもより多くの計算ノードに予算を割り振ることが多く、ファイルシステムは古典的な NFS が使われてるケースがほとんどでしたが、Catwalk を用いる事で NFS より高性能な並列 I/O が実現できることになりますし、図 9 のピラミッドの中位や下位に相当する多くの潜在的なユーザがいるものと考えられます。

本研究で開発されたソフトウェア Catwalk および Catwalk-ROMIO は PC クラスタコンソーシアム[3] で開発が進められている SCore クラスタシステムソフトウェアに組込まれ、オープンソースとして一般に配布されています。

参考文献

[1] 堀敦史，鴨志田良和，松葉浩也，安井隆，住元真司，石川裕: ファイルステージング再考：オンデマンド化

[2] http://www.idc.com/
[3] http://www.pccluster.org

と高速化に向けたプロトタイプ実装の評価，情報処理学会研究報告 Hokke'09，2009.

[2] 堀敦史 鴨志田良和 松葉浩也 安井隆 住元真司 石川裕: ファイルステージングシステム Catwalk の MPI-IO 実装，SWoPP2009，2009.

[3] Atsushi HORI, Yoshikazu KAMOSHIDA, Hiroya MATSUBA, Kazuki Ohta, Yutaka ISHIKAWA: On-Demand File Staging System for Linux Clusters, IEEE Cluster 2009, 2009.

[4] Atsushi HORI: Parallel File IO Using Ring Communication Topology, WPSE2010, Feb.2010.

[5] 堀敦史，鴨志田良和，松葉浩也，安井隆，住元真司，石川裕: リングトポロジーによる MPI-IO 書込の高速化技法，SACSIS 2010, pp.319-327, 2010.

[6] 堀敦史，鴨志田良和，松葉浩也，安井隆，住元真司，石川裕: MPI-IO 書込のパイプライン処理による高速化，情報処理学会 ACS 32 論文誌 (to appear).

Gfarm ファイルシステムによる広域ファイル共有

データインテンシブサイエンスを促進する基盤ソフトウェア

建部　修見

筑波大学計算科学研究センター

1　はじめに

Gfarm ファイルシステムは 2000 年より研究開発を続けているソフトウェアです。当時、CERN の LHC 実験計画がスタートし、LHC 加速器から生成される年間数十ペタバイトの実験データの解析、そのデータサイズを上回るシミュレーションデータの生成、という前代未聞ともいえる規模のデータ解析システムに関する議論が活発でした。これまでは、加速器を保持する計算センターでこれらのデータ解析は行われていましたが、LHC 実験規模のデータ解析となると CERN の計算センターだけではとてもまかないきれません。そのため、各国の計算センターで、協力してデータ解析をするということになったのです。そのようなとき、大規模データ処理を行うために、計算ノードのローカルディスクを利用し、性能をスケールアウトさせる、ファイル複製により複数の組織間で効率的にファイル共有を行う、という設計で Gfarm ファイルシステムの研究開発を始めました[1]。

2　Gfarm ファイルシステムの概要

Gfarm ファイルシステム[2]は、複数の組織間でのファイル共有、複数の組織による大規模データ解析のために設計されたファイルシステムです。各組織に設置されたストレージを単一の階層的名前空間（ディレクトリツリー）で束ね、単一のファイルシステムとしてアクセスすることが可能です。各組織からのアクセスは基本的には各組織のストレージに対し行われるように設計されており、広域環境においてもスケールアウトするアーキテクチャとなっています。各組織で自由にストレージの追加が可能で、ストレージの追加によりアクセス性能がスケールアウトします。遠隔の他組織に格納されているファイルも透明にアクセスできます。また、自拠点にファイル複製を作成してより高速にアクセスすることもできます。ファイル複製は、ファイルアクセス性能の向上だけではなく、ネットワーク、ストレージ障害時における耐故障性を向上させるために利用されます。

2.1　ソフトウェアコンポーネント

Gfarm ファイルシステムは、図 1 のようにファイルシステムメタデータを管理するメタデータサーバ（MDS）と、分散して設置されているストレージをアクセスするためのストレージサーバ（I/O サーバ）で構成されます。

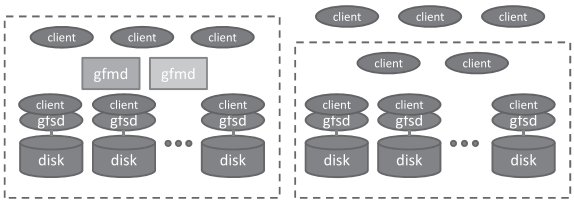


図 1 Gfarm ファイルシステムのソフトウェアコンポーネント

Gfarm ファイルシステムにおいて MDS は gfmd と呼ばれ、耐障害性を向上させるためマスタースレーブ構成をとることが可能です。I/O サーバは gfsd と呼ばれます。Gfarm ファイルシステムにおけるストレージは、ext3 などのローカルファイルシステムのディレクトリで、ディレクトリに対し I/O サーバを動作させることにより、そのディレクトリを Gfarm ファイルシステムの一部とすることが可能になります。

2.2 スケールアウトアーキテクチャ

MDS は、ディレクトリツリー、ファイル情報、ファイル複製格納位置などのファイルシステムメタデータを管理します。ファイルアクセスに際し、MDS はファイルのオープン時、クローズ時のみアクセスされ、実際の read、write、seek などのファイルアクセスは I/O サーバが直接アクセスされます。近くの I/O サーバ、またアクセス負荷の低い I/O サーバをそれぞれのクライアントがアクセスすることにより、アクセスを分散させることができます。これにより、MDS に対するアクセスがボトルネックとならない限り、全体としてクライアント数、I/O サーバ数に応じたスケールアウトするアクセス性能の実現が可能となります。

2.3 ソフトウェアリリース

Gfarm ファイルシステムのソフトウェア開発をオープンソースですすめるにあたり、Sourceforge.net を利用しています[3]。ここでは、最新リリースのダウンロード、不具合、要望などの登録、開発版、安定版ソースコードの閲覧、コミットログの確認などができます。

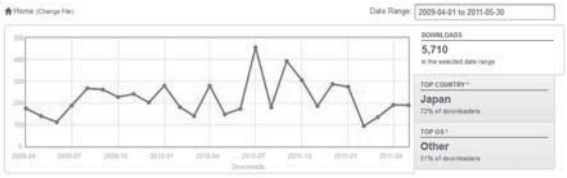


図 2 ダウンロード状況

図 2 は 2009 年 4 月 1 日から 2011 年 5 月 30 日までの約 2 年間のダウンロードの状況です。5,710 件のダウンロードがありました。

3 性能評価

本章では、スケールアウトするための条件となっている MDS の性能を示すとともに、クライアント数を増やしたときのファイルアクセス性能の評価を示します。

評価は科研費特定領域研究情報爆発 IT 基盤で構築された全国 14 拠点のクラスタからなる InTrigger プラットフォームを利用しました。評価にあたり Gfarm ファイルシステムの MDS は九州大に設置しました。I/O サーバは各拠点のクラスタの計算ノードで構成し、計 239 ノード、ストレージ容量は 146TB でした。クライアントは、I/O サーバにもなっている各拠点のクラスタの計算ノードです。各拠点から MDS までの RTT（往復遅延時間）は 0.04 ミリ秒〜47 ミリ秒でした。Gfarm ファイルシステムはバージョン 2.3.1 を利用しました。

3.1 メタデータサーバの性能

MDS の性能評価にあたり、クライアント数を増やしたときのディレクトリの並列作成性能の評価を行いました。図 3 に評価結果を示します。X 軸は同時にディレクトリを生成するクライアント数、Y 軸は全体として一秒間に作成されたディレクトリ数です。クライアントは図に示される拠点に従い増加していきました。ディレクトリ作成は MDS への問合せが必要となり、クライアントからの一往復分のネットワーク遅延がかかります。そのため、少ないクライアント数では MDS の性能を飽和させることはできません。

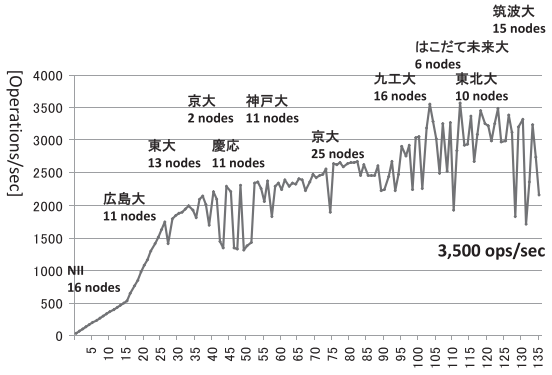


図 3 メタデータサーバ性能

占有していない広域ネットワークによる実測データであるため性能は安定していませんが、100 クライアントほどで毎秒 3,000 操作、最大で毎秒 3,568 操作の性能を示しました。これにより、MDS は毎秒 3,500 操作ほどの処理が可能であり、全体としての MDS アクセスがその性能を超えない限り MDS 性能はボトルネックとはならないことが分かりました。この性能までは、I/O 性能はスケールアウトすることが期待されます。

3.2 Nファイル並列書込・読込性能

次に、並列ファイルアクセス性能を示します。まず、図 4 にそれぞれのクライアントが 1GB の別ファイルを作成するときの性能を示します。MDS の性能評価と同様に、クライアントは図に示される拠点に従い増加させていきました。

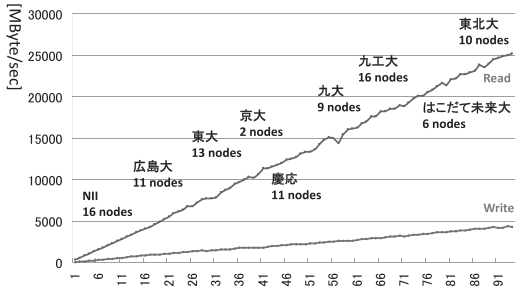


図 4 1GB の Nファイル書込・読込性能

各クライアントは I/O サーバでもあるため、各クライアントマシンのローカルディスクは Gfarm ファイルシステムの一部となっています。Gfarm ファイルシステムでは、ファイルの読込・書込に対し、ネットワーク的に近いストレージ、近いファイル複製を選択するようになっています。そのため、各クライアントが別ファイルを作成する場合、空き容量が十分あれば、各クライアントのローカルディスクが選択されます。並列書込については、それぞれのクライアントの書込はローカルディスクに対して行われ、書込 I/O に関するディスクアクセス競合はおこりません。MDS の性能評価で示されたように、MDS は毎秒 3,500 操作を行うことができます。そのため、94 クライアントからの並列アクセスでは MDS の性能はボトルネックとはなりません。結果として、I/O 性能はスケールアウトし、94 ノードで 4,370 MB/s の性能を達成しています。

並列読込は、並列書込で書き込んだファイルを読み込む性能です。書き込み時間は、書き込んだ後、ディスクに sync するまでの時間を計測していますが、読込は書き込んだ後すぐに行っているため、書き込んだデータがバッファキャッシュに載っており、性能が高くなっています。読込についても同様に MDS 性能はボトルネックとはならず、スケールアウトする性能を示し、94 ノードでは 25,170 MB/s の性能を達成しています。

3.3 1 ファイル並列読込性能

1 ファイルの並列読込性能を図 5 に示します。この評価では、1GB の 1 ファイルを各クライアントが並列に読み込みます。読み込むファイルは、あらかじめ各拠点に複製を作成しておきます。各拠点における複製数を 1、2、4、8 と変化させて性能を評価しました。これまでの評価と同様にクライアント数を図にかかれた拠点に従い増やしています。

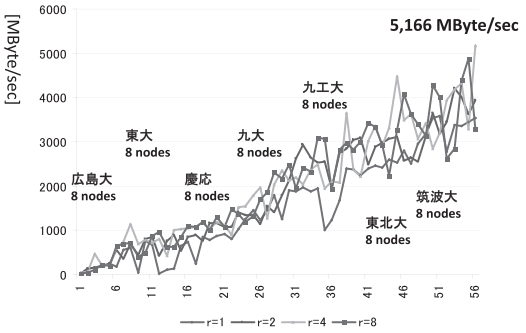


図 5 1GB の 1 ファイル読込性能

読込性能は安定していませんが、I/O 性能はスケールアウトし、7 拠点 56 クライアントで 5,166 MB/s の性能を達成しています。これにより、各拠点のクライアントは自拠点のファイル複製を参照していることが分かります。

4 基本的な利用方法

本章では Gfarm ファイルシステムの基本的な利用方法を説明します。

4.1 認証

Gfarm ファイルシステムをアクセスするために、まず認証の設定を行います。認証方法としては、共有鍵認証と GSI 認証をサポートしています。どの方式が利用可能かは管理者に問い合わせてください。

共有鍵認証の場合、gfkey コマンドで共有鍵を作成し、その鍵を MDS および全ての I/O サーバにコピーします。以下のように、31536000 秒（約

1 年）有効の共有鍵を作成し、MDS など remotehost にコピーします。

```
% gfkey -f -p 31536000
% scp –p .gfarm_shared_key remotehost:
```

GSI 認証の場合は、以下のように代理証明書を作成します。

```
% grid-proxy-init
```

デフォルトでは 12 時間有効のものが作成されます。

4.2 マウント

Gfarm ファイルシステムは gfarm2fs コマンドによりマウントすることができます。gfarm2fs コマンドは利用者が書き込み可能なディレクトリに Gfarm ファイルシステムをマウントするコマンドです。たとえば、以下のようにマウントポイント/tmp/username を作成し、マウントします。

```
% mkdir /tmp/username
% gfarm2fs /tmp/username
```

アンマウントは fusermount –u を利用します。

```
% fusermount -u /tmp/username
```

Gfarm ファイルシステムは他の拠点でもマウントできますので、これで拠点をまたいでのファイル共有が可能となります。

4.3 Gfarm コマンド

通常のファイルシステムに対するコマンドは、gfarm2fs でマウントすることにより利用することが可能ですが、ファイル複製管理やクオータ機能など、Gfarm ファイルシステム独自の機能については Gfarm コマンドを利用します。

gfwhere コマンドはファイルの格納位置を調べるコマンドです。

```
% gfwhere .bashrc
linux-1.example.com
```

上記の例では、.bashrc は linux-1.example.com に格納されていることが分かります。

gfrep コマンドはファイル複製を作成するコマンドです。ファイル複製を二つ作成するためには以下のようにします。gfwhere コマンドでどこに作成されたか確認できます。

```
% gfrep -N 2 .bashrc
% gfwhere .bashrc
linux-1.example.com linux-2.example.com
```

gfrep コマンドは多くのコマンドオプションを持ち、複製作成先を指定したり、また必要数以上作成された複製を消去したりすることもできます。

そのほかのコマンドについては、Web ページ（http://datafarm.apgrid.org/software/gfarm_v2/html/ja/ref/）あるいはマニュアルページを参照してください。

5 最新機能・状況紹介

Gfarm ファイルシステムの新しい機能、特徴的な機能について紹介します。

5.1 ファイル自動複製機能

ファイルの作成時、更新時に自動的にファイル複製を作成する機能です。バージョン 2.4.1 以降でサポートしました。Google ファイルシステムをはじめファイル複製をサポートする多くのファイルシステムでは固定数のファイル複製しか作成できませんが、Gfarm ではファイル複製数はディレクトリの拡張属性 gfarm.ncopy で指定することができます。このため、ディレクトリによりファイル複製数を変えることが可能です。たとえば、/（ルート）ディレクトリ以下の全てのファイルに対し、複製数を 3 としたい場合は、以下のように /ディレクトリの gfarm.ncopy 拡張属性で 3 を指定します。

```
% echo –n 3 | gfxattr –s / gfarm.ncopy
```

このとき、ファイル作成時、更新時のファイルクローズ後に、非同期的にファイル複製を作成します。非同期に作成しますので、クライアントが複製作成完了まで待たされることはありません。

5.2 XML 拡張属性

バージョン 2.3.0 以降では、通常の拡張属性だけではなく値に XML 文書を持つ XML 拡張属性をもてるようになりました。XML 拡張属性は XPath で特定ディレクトリ以下のエントリを検索できるところに特徴があります。たとえば、/ ディレクトリから foo というタグを含む XML 拡張属性を検索するためには以下のように行います。

```
% gffindxmlattr //foo /
```

/ディレクトリ以下の全てのエントリに対し、//foo という XPath 検索を行い、結果を表示します。

本機能により、アプリケーションのメタデータを、ファイルやディレクトリの XML 拡張属性に格納し、XPath 検索により該当ファイルを検索することが可能になります。

5.3 拡張 ACL 機能

バージョン 2.4.2 以降では、POSIX 1003.1e DRAFT 17 に基づく拡張 ACL をサポートしました。これにより、所有者、グループ、それ以外といった従来からの UNIX のファイルシステムのアクセス制御に加え、特定ユーザ、特定グループに対するアクセス制御が可能になります。

5.4 Gfarm ファイルシステムのフェデレーション機能

バージョン 2.4.2 以降では、Gfarm URL 形式（gfarm://metaserver:port/path/name）のシンボリックリンクをサポートしました。これにより、異なる Gfarm ファイルシステムのファイル、ディレクトリに対するリンクを作成することができます。利用者は、そのリンクを透明にたどることができますので、どの Gfarm ファイルシステムをアクセスしているのかを意識することなく、複数の Gfarm ファイルシステムをアクセスすることが可能です。

5.5 Debian パッケージ

Debian (Squeeze)のパッケージに Gfarm ファイルシステムが取り込まれました。

図 6 Debian パッケージ

これにより、Ubuntu を含む Debian 系のディストリビューションでは、apt-get install で Gfarm ファイルシステムをインストールすることが可能です。

6 大規模データ処理

Gfarm ファイルシステムは複数拠点間でのファイル共有だけではなく、スケールアウトするアーキテクチャにより、大規模データ処理用のファイルシステムとして利用することが可能です。ファイルアクセス性能をスケールアウトさせるためには、それぞれのクライアントが近くのストレージを、集中しないように分散してアクセスすることが大切です。大規模データ処理においては、特に入力ファイルの読込性能をスケールアウトさせるために、入力ファイルのファイル配置を考慮したジョブスケジューリングが重要となります。

以下、Gfarm ファイルシステムを利用した典型的な大規模データ処理について紹介していきます。

6.1 MPI-IO

アプリケーションから利用しやすいインターフェースに HDF5、Parallel netCDF などがありますが、それらは並列 I/O の標準インターフェースの MPI-IO を利用しています。

Gfarm ファイルシステムをマウントし、MPI-IO で Gfarm ファイルシステムを利用することも可能ですが、このとき MPI-IO では比較的典型的な N プロセスで 1 ファイルを並列に作成する場合にスケールアウトする性能が得られません。異なるファイル作成であれば、クライアントに近いストレージに分散して作成することによりスケールアウトする性能を達成することができますが、Gfarm ファイルシステムはファイルを分割しないため、1 ファイルへの並列書込は本当に 1 ファイルへの並列書込となってしまい、アクセスが集中してしまうためです。

そのため、そのようなケースもついても、意味を変えることなく、物理的には N ファイルに書き出しを行う MPI-IO の設計、実装を行っています[4]。性能評価によると、並列ファイルアクセス性能はスケールアウトし、4 ノード以上では PVFS

より良い性能を示しました。MPI-IO ライブラリのソフトウェアはまだ公開していませんが、今後公開を予定しています。

6.2 MapReduce

大規模データ処理では、MapReduce も利用されるようになりました。MapReduce のオープンソースの処理系として Hadoop があります。HadoopではHDFSと呼ばれるファイルシステムが利用されます。ただし、HDFS は POSIX 準拠ではなく、たとえばファイルの修正もできません。そこで、HDFS ではなく Gfarm ファイルシステムを Hadoop MapReduce で利用する研究開発を行っています[5]。

Hadoop MapReduce も、マウントすることにより Gfarm ファイルシステムをそのまま利用することが可能ですが、それでは、マップタスクの配置に対し、入力データの配置を考慮することができません。そのため、不必要に遠隔アクセス、アクセス衝突が発生してしまいます。これを防ぐために、Gfarm ファイルシステムに対する Hadoop プラグインを開発し、公開しています[3]。

MapReduce の代表的なアプリケーションである、Grep（UNIX の grep とは違い、指定文字列の出現回数をカウントする）と Terasort の性能を図 7 に示します。

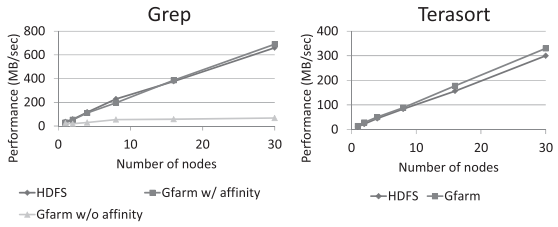


図 7 Grep と Terasort の性能

Grep では、青の HDFS と赤の Gfarm w/ affinity がほぼ同様のスケールアウトする性能を示しています。参考のため、Gfarm ファイルシステムを利用するものの、ファイル配置情報を利用しない場合の性能を、黄色の Gfarm w/o affinity で示します。ファイル配置が利用できない場合、マップタスクはデータの先頭から固定長ブロックごとを処理するように実行されますが、それらは物理的には単一ファイルに格納されているため、アクセスが集中してしまい、ノード数を増やしても性能向上していません。また、Terasort は、Gfarm ファイルシステムが HDFS を若干上回る性能を示しています。

プラグインの開発により、HDFS と同等あるいは HDFS をしのぐ性能を達成することができました。これにより、Hadoop MapReduce を使うために、HDFS を構築しデータを HDFS にコピーする必要はなくなり、全て Gfarm ファイルシステムにおいて、通常アクセス、MPI-IO による並列アクセス、MapReduce アプリケーションによる並列アクセスが可能となりました。

なお、Lustre、GPFS、PVFS などの並列ファイルシステムに対し Hadoop MapReduce アプリケーションを実行させるための研究もありますが、並列アクセス性能を向上させるため、ブロックサイズを通常より 1,000 倍も大きくするなど、通常とは大きく異なる設定が必要で、通常利用については逆に性能が落ちてしまいます。それに対し、Gfarm ファイルシステムは、もともと HDFS と設計が似ていることもあり、通常の設定のままで性能を出すことができます。

6.3 Pwrake によるワークフロー実行

大規模データ処理では、様々なプログラムを組み合わせてデータ処理を行うことが多く、それらをまとめてワークフローとして実行することも多く行われています。

ワークフローの記述はアイコンなどでグラフィカルに構成する Taverna、Kepler、スクリプト言語の Swift、MegaScript、依存関係を記述する Makefile などがあります。

Gfarm ファイルシステムに格納されているファイルに対し、効率的に大規模データ処理を行う場合、ファイルの格納位置を考慮したジョブのスケジューリングが重要となります。そのために、我々のグループでは Pwrake ワークフローエンジンの研究開発を進めています[6]。Pwrake は、UNIX におけるビルドツール make の ruby 版である rake を拡張し、同時に実行可能なジョブを並列分散実行します。ジョブの依存関係は rakefile により記述します。Rakefile は ruby で記述可能なため、極めて柔軟な記述ができます。複

雑な科学技術計算のためのワークフローでは、途中の実行結果によりワークフローが動的に変わるようなものも珍しくありませんが、そのような複雑なワークフローも記述可能です。

Pwrake では、並列分散実行の拡張だけではなく、Gfarm ファイルシステムにおけるファイルの格納位置を考慮したジョブスケジューリングを行います。

天文分野のデータ処理として、複数枚の天文画像を継ぎ合わせて一枚の天文画像にするモザイク処理があります。代表的なモザイク処理エンジンである Montage[8]によるケーススタディを紹介します。

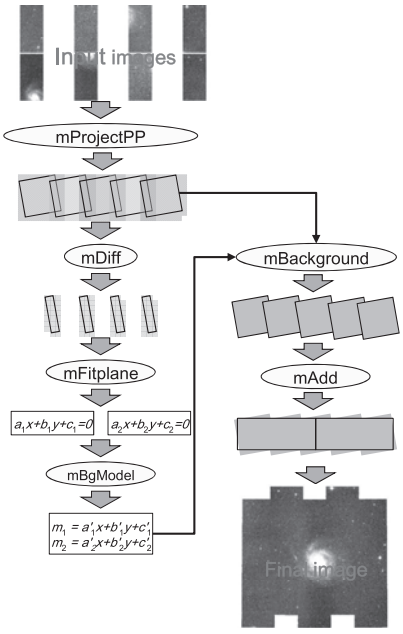


図 8 Montage ワークフロー

Montage のワークフローを図 8 に示します。複数の天文画像をまず同一平面上に射影し、オーバラップしているところで画像の明るさを揃えるなどの処理を行い、一枚のモザイク画像とします。天文画像は FITS 形式で保存されており、天体のどの場所をいつどのように撮影したものかなどのメタデータが画像ファイルのヘッダ部分に付加されています。ここで、オーバラップしている画像に対する処理をするために、全ての入力ファイルのヘッダ部分を解析して、どの画像がオーバラップしているかを調べる必要があります。つまり、実行開始時に全てのワークフローが決定しているわけではなく、ワークフロー実行中のジョブの出力に従って実行するワークフローが変わってきます。これまでのワークフローツールでは、このように動的にワークフローが決定されるものは扱えませんでしたが、Pwrake では rakefile により 100 行で完全に記述することが可能です。そのため、入力ファイルを指定するだけでワークフローの実行が可能となります。

Montage ワークフローに対し、クラスタのコア数を増やしながら実行時間をプロットしたものが図 9 です。

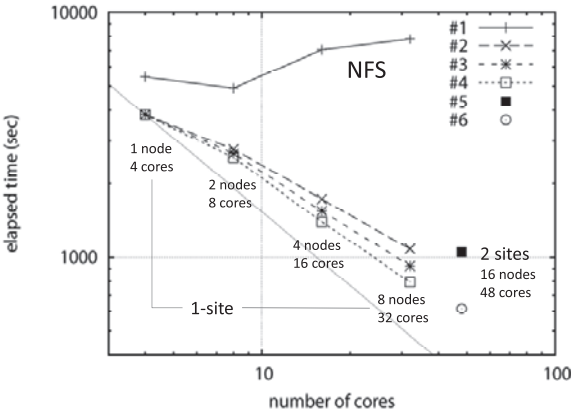


図 9 Montage ワークフローの実行時間

X 軸はワークフローを実行するコア数、Y 軸は実行時間です。データを NFS においた場合、コア数を増加しても実行時間が増えてしまっています。ディスクアクセス性能で実行時間が抑えられ、かつ並列アクセスによるアクセス衝突により全体性能が落ちています。Gfarm ファイルシステムでは、設定の違いにより三本の線が引かれていますが、どれもコア数を増やすと性能がスケールアウトしています。また、2 sites とかかれている筑波大と産総研の 2 拠点のクラスタによる広域分散データ処理では、初期データ配置を最適にすることにより■から○に性能向上し、1 拠点 32 コアから 2 拠点 48 コアに対し広域環境にもかかわらず性能がスケールアウトしています。

Pwrake および Montage ワークフローの rakefile は[7]で公開しています。

7 まとめ

Gfarm ファイルシステムは、2000 年から研究を進めているファイルシステムで、オープンソー

スで開発をすすめています。広域環境でも効率的にファイル共有が可能となるだけではなく、大規模データ処理に必要なスケールアウトするアクセス性能を達成できるようなアーキテクチャとなっています。MPI-IO、MapReduce、ワークフロー実行のそれぞれについて、効率的に処理するための研究開発も進めています。今後も引き続き研究開発を進め、あらゆる分野のデータインテンシブサイエンス（e-サイエンス）を促進することを目標としています。

参考文献

[1] 建部修見，森田洋平，松岡聡，関口智嗣，曽田哲之，“ペタバイトスケールデータインテンシブコンピューティングのためのGrid Datafarm アーキテクチャ”，情報処理学会論文誌 : ハイパフォーマンスコンピューティングシステム，Vol.43，No.SIG 6 (HPS 5)，pp.184-195，2002

[2] Osamu Tatebe, Kohei Hiraga, Noriyuki Soda, "Gfarm Grid File System", New Generation Computing, Ohmsha, Ltd. and Springer, Vol.28, No.3, pp.257-275, 2010

[3] http://sourceforge.net/projects/gfarm/

[4] 木村浩希，建部修見，“広域分散ファイルシステム Gfarm の MPI-IO の実装”, 情報処理学会研究報告 2010-HPC-124(15)，pp.1-6，2010

[5] 三上俊輔，太田一樹，建部修見，”POSIX準拠の広域分散ファイルシステム Gfarm 上での Hadoop MapReduce アプリケーション”，先進的計算基盤システムシンポジウム (SACSIS 2011) 論文集, pp.181-188, 2011

[6] Masahiro Tanaka, Osamu Tatebe, "Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing", Proceedings of ACM International Symposium on High Performance Distributed Computing (HPDC), pp.356-359, 2010

[7] https://github.com/masa16/pwrake

[8] http://montage.ipac.caltech.edu/

バックトラックに基づく動的負荷分散フレームワークTascell

平石 拓*，八杉 昌宏†

*学術情報メディアセンター
†情報学研究科

1 はじめに

Tascell [4] は，我々が開発を進めている work stealing ベースの高生産並列言語・ランタイムであり，以下の特徴を持つ．

- 木探索，グラフ探索など負荷が均等になるように問題を分割することが難しいアプリケーション（不規則なアプリケーション）に対しても効率良い並列化が可能である．
- 分散環境，さらには複数拠点のクラスタを跨って 1 つの計算を行うような広域分散環境にも対応している．また，ノードの性能が不均一であったり，一部のノードの負荷に急に高くなったような場合においても，良好な性能を発揮する．計算ノードの動的な追加にも対応している（動的脱退は今後サポートする予定）．
- プログラマは，逐次の C 言語プログラムを，並列化可能な箇所の指定やタスク生成の方法等を追加するという比較的小さなコストで Tascell のプログラムに書き換えることができる．

グラフ探索のような不規則なアプリケーションを効率良く実装できるほかの高生産並列言語で有名なものとしては，MIT で開発された Cilk [3]（最近では Cilk Plus [2] として Intel Parallel Composer の機能に取り込まれている）や IBM で開発されている X10 [1] が挙げられる．

例として，n 番目の Fibonacci 数を再帰的に求める C プログラムを 図 1 に，それを Cilk で並列化したプログラムを図 2 に示す．Cilk は，直後の関数呼び出しが非同期に実行されることを示す `spawn` と，同一関数内で実行した全ての `spawn` の終了を待ち合わせる `sync` という，基本的に 2 つのキーワードのみで並列性を記述できるようになっている（X10 もほぼ同様の言語機構を備えている）．`spawn` は直後の関数呼び出しを実行する新たなスレッドを生成するが，この新たなスレッドはすぐに実行される．一方，それまで実行していたスレッド（`spawn` 文の続きを実行するスレッド）はワーカのスレッドキュー（正確には deque）に蓄えられ，アイドルな（暇な）ワーカがそこからスレッドを盗む（スティールする）ことにより動的負荷分散を実現する．

なお，他のワーカからスレッドを盗む際には，最初に生成されたスレッドから優先的に盗む（First-in First-out, FIFO）ことで実行木の根元付近のなるべく大きな単位で仕事の分割が行われるようにし，ワークスティールの回数が少なくなるようにしている．ま

```
int fib (int n)
{
    if (n <= 2) return 1;
    {
        int s1, s2;
        s1 = fib(n - 1);
        s2 = fib(n - 2);
        return s1 + s2;
    }
}
```

図 1: C による Fibonacci 数計算プログラム

```
cilk int fib(int n)
{
     if (n <= 2)
          return (1);
     else {
          int s1, s2;
          s1 = spawn fib(n - 1);
          s2 = spawn fib(n - 2);
          sync;
          return (s1 + s2);
     }
}
```

図 2: Fibonacci の Cilk プログラム

た，スレッド生成にかかるコストのほとんどは Lazy Task Creation (LTC) [7] と呼ばれる実装技術により，実際にそのスレッドがスティールされるまで遅延される．そのため，図 2 のプログラムのように問題サイズの指数に比例するような多数のスレッド生成を行ったとしても，(たとえば pthread で同様のプログラムを書いたときと比較して）致命的な性能劣化は起こらない．これらの仕組みにより，Cilk では不規則なアプリケーションに対しても比較的小さなプログラミングコストで，効率良い並列化を行うことが可能になっている．

我々が提案する Tascell のワーカは，Cilk とは異なり，普段は一切のスレッド／タスク生成を行わず逐次プログラムのように動作し，他のアイドルなワーカからタスクを要求されると初めてタスク生成を行う．このとき，なるべく大きな単位でタスクを生成するために，Cilk の FIFO 戦略の代わりに，これまでの計算を**一時的バックトラック**により巻き戻し，最古のタスク生成可能状態を復元するということを行う．この手法は，普段のタスク生成のコストを完全に削減できる（バックトラックの準備のためのコストはかかるが，後の性能評価で示すように，そのコストは Cilk の LTC よりも小さい）ほか，アプリケーションレベルの作業空間の再利用性，参照局所性を改善できるという利点を持つ．また，バックトラック探索アルゴリズムなどにおいて，タスク生成時に発生する作業空間のコピーを必要時まで遅らせることでコピー回数を削減するという高速化も可能であり，Tascell 言語を用いればそのようなプログラムを比較的小さなプログラミングコストで書くことができる．

本稿では，Tascell について簡単な例を交じえて紹介し，京都大学スーパーコンピュータ Thin クラスタ，および広域分散メモリ環境として日本国内の多拠点のクラスタを相互に接続した計算環境である InTrigger [10] における性能評価の結果を示す．

2　均等な負荷分散が困難な例

提案手法の説明のために，一般に効率的な並列化が困難な 2 つの樹木再帰のアルゴリズムを取り上げる．それぞれのアルゴリズムの説明の後，並列化がなぜ困難かを説明し，それを解決するための我々の手法の詳細を述べる．

```
int a[12];   // 作業空間：未使用のピース
int b[70];   // 作業空間：盤面．(6+ 番兵 ) × 10 個のセル

// a[] 中の j0 番目から 12 番目までのピースの設置を試みる
// i 番目（i<j0）のセルは設置済み
// b[k] が盤面中の最初の空きセル
int search (int k, int j0)
{
 int s=0; // 見つかった解の数
 for (int p=j0; p<12; p++) { //全種類のピースを試す
  int ap=a[p];
  for (each possible direction d of the piece) {
   ... local variable definitions here ...
   if (ap 番目のピースが d の方向に置けるか？);
   else continue;
   ap 番目のピースを盤面 b に設置し，a も更新
   kk = the next empty cell;
   if (no empty cell?) s++;          // 解発見
   else s += search (kk, j0+1); // 次のピース
   ap 番目のピースを取り除き a も元に戻す（バックトラック）
  }
 }
 return s;
}
```

図 3: Pentomino パズル全解探索の C プログラム

最初の例は，冒頭でも取り上げた n 番目の Fibonacci 数を再帰的に求めるアルゴリズムである．C による逐次プログラムは図 1 のように書ける．各関数呼び出しにおいて，`fib(n-1)` と `fib(n-2)` は並列に実行することが可能である．

二つ目の例は，Pentomino パズルの全解探索アルゴリズムである．各 pentomino（ピース）は辺どうしが接続された 5 つの正方形からなるブロックであり，12 通りのピースが存在する．Pentomino パズルとは，6 × 10 セルの盤面に，この 12 種類のピースを隙間なく敷き詰める配置を求めるものである．これ自体は単なるパズルだが，多くの探索アルゴリズムがこれと似たような構造をしている．逐次の C プログラムは図 3 のように書ける．各関数呼び出しで，未使用のピースについての反復（最外ループ）と，ブロックを置く各方向についての反復（1 つ内側のループ）を行っているが，ここでは最外ループについての並列化を考える．Pentomino は 1 ステップ進むごとにピースを盤面に設置しバックトラックの際に取り除くというバックトラック探索を行っており，そのため，盤面の状態を管理するための**作業領域** が使われていることに注意する．

これらを並列化する素直な方法として，各ワーカがその時々の状態に基づいてタスク生成するかどうかを判断する方法が考えらえる．そのような方法に基づくタスク並列のプログラムを図 4 に示す．各ワーカは `fib(n)` の計算において，`fib(n-2)` の計算をタ

```
// タスクオブジェクトの構造定義
struct tfib {
  int n;  // 入力
  int r;  // 出力
};
// タスクのエントリポイント
void exec_fib_task (struct tfib *pthis)
{ pthis->r = fib (pthis->n); }

int fib (int n) {
 if (n <= 2) return 1;
 {
   int s1, s2;
   if (choose not to spawn?) {
     s1 = fib(n - 1);
     s2 = fib(n - 2);
   } else {
     Allocate a workspace of struct tfib as this.
     this.n = n - 2;  // 入力値をセット
     Send this as a newly spawned task.
     s1 = fib(n - 1);
     Wait and receive the result of this.
     s2 = this.r;      // 出力値を獲得
     Deallocate this workspace.
   }
   return s1 + s2;
 }
}
```

図 4: Fibonacci を率直にタスク並列化したプログラム

スクとして生成するか，生成せずに自分で計算するかを何らかの基準で判断している．「何らかの基準」としては，例えばタスクを要求しているワーカが存在するときのみタスクを生成する，などが考えられる．効率良い負荷分散のためには，全実行時間にわたって全てのワーカを busy にできるような最小数のタスク生成が行われることが理想であり，そのため各タスクはできるだけ大きな単位で生成されることが求められる．つまり，各ワーカは計算の初期段階で適切な数のタスクを生成しておき，その後はずっと（計算終盤の微調整を除き）生成を行わないという戦略を採るのが最善である．しかし，そのような戦略は実行全体についての正確な情報（予測）がなければ実行することは不可能である．

同様の戦略を Pentomino の最外ループに適用する場合，各ワーカは，全ての反復を自分で計算するか，ループの半分を新たなタスクとして生成するかを「何らかの基準」で選択する（生成することを選択した場合，残りのループを同様の方法で再帰的に生成することもできる）ことになるが，やはり同様の理由でうまくいかない．

3　Tascell の動的負荷分散手法

上記の問題の解決のため，Tascell では，**一時的後戻り**を行うことで，タスクの**遅延分割**を行う．

図 1 や図 3 の逐次計算は，左深さ優先の実行木の走査と見ることができる．2 章の素直な戦略（Fibonacci における図 4）で常に「生成しない」ことを選択した場合も同様である．

Tascell では，ワーカは**最初は**常に「生成しない」ことを選択するが，他のワーカからタスク要求を受けると，**過去の選択を変更したかのように**タスクを生成する．つまり，図 5・図 6 に示すように，ワーカは (1) まず過去の時点に後戻りし，(2) タスクを生成し（この時，実行経路をそのタスクの結果を受け取るように変更），(3) 自分が行っていた計算を再開する．

このように，最も古いタスク分割可能時点に遡ることにより，一般には最も大きいタスク（図 5 では木のルートの右側の fib(38)）を生成することができる．

逐次版の Pentomino（図 3）では，ワーカは自らの**作業領域**を持ち，そこにステップごとにピースを置いたり取り除い（**アンドゥ**）たりすることで探索を行っている．ワーカがタスクを生成する際には，新しいタスク用の作業領域を allocate した上で，そこに「現在の」状態をコピーする必要がある．提案手法においては，「現在の」内容とは過去の選択時点における内容のことである．そこで，ワーカは図 6 のように (1) の後戻り順に沿って適切な *undo* を行うことにより過去の作業領域の内容を復元し，(2) のタスク生成時にその復元した作業領域のコピーを作成する．その後，(3) の後戻りからの復帰時に適切な *redo* を行うことで作業領域の内容を undo 前に戻すことで，(4) における自らの計算の再開を可能にする．

Cilk が採用している LTC は，タスク生成のコストそのものが小さければワーカが常にタスクを「生成する」ことを選択しても性能上大きな問題にならない，という発想に基づくものだと考えることができる[1]が，本手法は LTC に対して，(1) スレッドを一切生成しないので，スレッドキューの管理コストが発生しない，(2) マルチスレッド言語では，各スレッド用に作業領域を確保する必要があるが，本手法で

[1]ただし，Cilk は `spawn` で生成したスレッドを生成したワーカ自身が先に実行し，その残りの計算（プログラム上では `spawn` の次の行以降の処理）がスティールの対象になるという点で，単に図 4 のタスク生成コストが小さくなったものとは挙動はやや異なる．

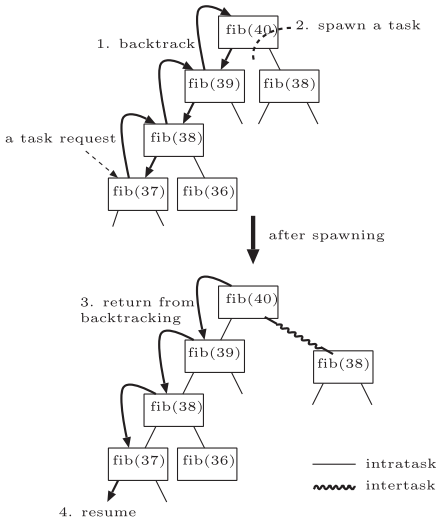


図 5: fib(40) におけるタスクの遅延生成．Tascell のワーカがタスク要求を（fib(37) で）検出すると，(1) まず最古のタスク生成可能時点まで後戻りし，(2) fib(38) のタスクを生成し，(3) 後戻りから復帰し，(4) 自らの計算を再開する．

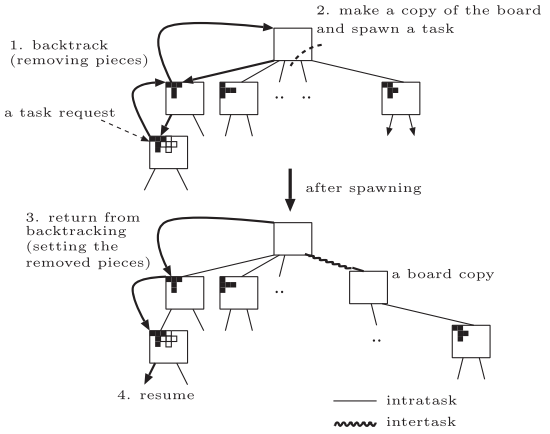


図 6: バックトラック探索 Pentomino におけるタスク遅延生成．図 5 との違いは，(1) 後戻り中のアンドゥ処理（ピース除去），(2) 半分の反復をタスクとして生成する際の，一時的に過去の状態に戻った盤面のコピー，および (3) 後戻りからの復帰中のリドゥ処理（ピースの再設置）．

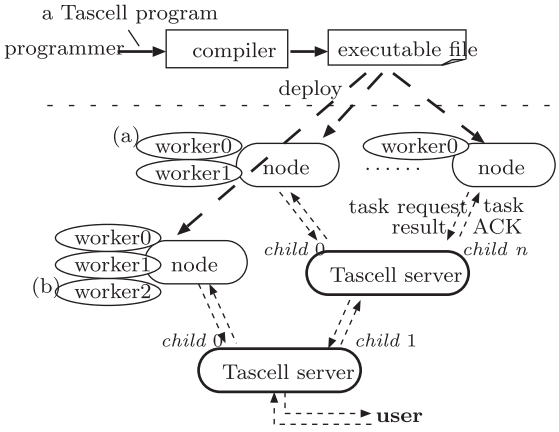


図 7: Tascell フレームワークにおけるプログラムのコンパイル・実行

は単一の作業領域を再利用し続けることが可能であり，参照局所性が向上する，[2] (3) バックトラック探索をマルチスレッド言語で実装する場合，親スレッドの作業領域のコピーを各スレッド生成時に用意する必要があるが，本手法では，一時的後戻りを用いることによりそのようなコピーを遅延できる，という優位点を持つ．

Tascell でも Cilk でも，実際に生成されるタスクの数（ワークスティールの回数）は並列化可能なポイント数に比べて極めて少ないことを仮定している．我々の手法は，LTC よりワークスティールのコストが高くなることを許容し，逐次計算のオーバーヘッドを極めて小さく抑えるものであり，上記の仮定のもとでは全体の性能は向上する．

4 Tascell フレームワーク

上記のような動的負荷分散を行うアプリケーションの実装をプログラマが行えるようにするため，*Tascell* **フレームワーク**を実装した．このフレームワークは Tascell サーバおよび Tascell 言語コンパイラで構成される．

4.1 フレームワークの概要

図 7 に，Tascell フレームワークにおけるプログラムのコンパイル，実行の様子を示す．コンパイルされた Tascell プログラムは 1 台以上の計算ノードで実行される（起動時に接続先の Tascell サーバを指定する）．各計算ノードでは 1 つ以上のワーカによる共有メモリ環境での並列計算が行われる．さらに，Tascell サーバを介して複数の計算ノードを接続することにより，分散メモリ環境での並列計算も実行できる．

Tascell サーバは，計算ノード間のメッセージの中継や，ユーザインタフェースとの入出力処理，各計算ノードの負荷情報の管理などを行う．ここで，Tascell サーバには計算ノードだけでなく別の Tascell サーバを接続することもできる．これにより，各クラスタの代表ノードにサーバを配置し，サーバおよび計算ノードからなるツリー状のネットワークを構成することで，WAN で接続された多拠点のクラスタに跨って 1 つの並列計算を行うことも可能となっている．

ワーカ間のタスクやその結果の授受は**タスクオブジェクト**を介して行われる．タスクオブジェクトは，タスクの入出力の値をセットするためのメンバを持つ構造体であり，その具体的な構造は Tascell プログラムで定義される．オブジェクトの受け渡しは，共有メモリ内のワーカ間であればポインタの受け渡しで行い，ノードを跨る場合はオブジェクトをシリアライズして Tascell サーバを介して送信する（そのため，ノード間でアーキテクチャが異なる場合でも正常に動作する）．

4.2 Tascell 言語

Tascell 言語は，タスク分割可能箇所の指定や，一時的 undo および redo の処理を記述できるようにした拡張 C 言語である．プログラマは図 9 のように，既存の逐次プログラムをベースにして，Tascell ワーカのプログラムを書くことができる．Tascell の追加コンストラクトについて以下で説明する．

[2]Cilk では `SYNCHED` という疑似変数を利用することで，子スレッド間については作業領域の再利用が可能だが，親子スレッド間での再利用はできない．

```
// タスク tfib の定義
task tfib {
  in:  int n;  // 入力
  out: int r;  // 出力
};
// tfib のエントリポイント
task_exec tfib
{ this.r = fib (this.n); }

worker int fib (int n) {
 if (n <= 2) return 1;
 {
   int s1, s2;
   do_two          // Tascell のコンストラクト
     s1 = fib(n - 1);
     s2 = fib(n - 2);
   handles tfib {
     // put 部（タスク送信前に実行）
     { this.n = n - 2; }
     // get 部（結果受信後に実行）
     { s2 = this.r; }
   }               // end of do_two
   return s1 + s2;
 }
}
```

図 8: Fibonacci の Tascell プログラム

4.2.1 タスク定義

タスク定義は下記のトップレベル宣言により行う．

`task` *task-name* `{[in:|out:]` *struct-declaration* ⋯ `};`

例として，図 8 中の “`task tfib {in:  int n; out: int r};`” は，`tfib` というタスクのオブジェクト構造を定義している．構文は C の構造体の定義とほぼ同様だが，`in:` または `out:` の属性を各フィールドに指定することができる．コンパイラはこれらの属性を参照して，外部ノードへタスクを送受信する手続きを生成する．

エントリポイントの定義　タスク *task-name* が実際に行う計算は，

`task_exec` *task-name* `{` *body* `}`

のトップレベル宣言により定義する．*body* 中では，図 8 中の “`this.r =fib(this.n);`” のように `this` を用いてタスクオブジェクトを参照できる（タスクの入力の値が含まれている），タスクの計算結果を適切なフィールドにセットすることをプログラムに要求する．また，*body* では以下で説明する**ワーカ関数**を呼び出すことができる．

```
task pentomino {
  out: int s;    // 出力
  in: int k, i0, i1, i2;
  in: int a[12]; // 作業空間：未使用のピース
  in: int b[70]; // 作業空間：盤面の状態
};
task_exec pentomino {
  this.s = search (this.k ,this.i0 ,this.i1 ,this.i2,
                   &this);
}

worker int search (int k, int j0, int j1, int j2,
                   task pentomino *tsk)
{
 int s=0; // 見つかった解の数
 // Tascell の並列 for 文
 for (int p : j1, j2)
  {
   int ap=tsk->a[p];
   for (each possible direction d of the piece) {
    ... local variable definitions here ...
    if(ap 番目のピースが d の方向に置けるか？);
    else continue;
    dynamic_wind // undo, redo 操作指示コンストラクト
     { //  dynamic_wind の do, redo 操作
       ap 番目のピースを盤面 tsk->b に設置し tsk->a も更新
     }
     { // dynamic_wind 本体
       kk = the next empty cell;
       if (no empty cell?) s++; // 解発見
       else                     // 次のピース
        s += search (kk, j0+1, j0+1, 12, tsk);
     }
     { // dynamic_wind の undo 操作
       ap 番目のピースを取り除き，tsk->a も元に戻す
        （バックトラック）
     } // end of dynamic_wind
   }
  }
 handles pentomino (int i1, int i2)
            // 範囲（i1-i2）の設定は暗黙になされる
  {
   // put 部（タスク送信前に実行）
   { // 反復の前半分に相当するタスクの入力を設定
     copy_piece_info (this.a, tsk->a);
     copy_board (this.b, tsk->b);
     this.k=k; this.i0=j0; this.i1=i1; this.i2=i2;
   }
   // get 部（結果受信後に実行）
   { s += this.s; }
  }  // end of parallel for
 return s;
}
```

図 9: Pentomino の Tascell プログラム

4.2.2 ワーカ関数

通常の関数定義に `worker` キーワードを付加すると，その関数は**ワーカ関数**となる．以下で説明するタスク分割コンストラクト（およびワーカ関数の呼び出し）の使用は，ワーカ関数および `task_exec` の本体でのみ可能である．(この制限は Cilk の `cilk` 手続きと同様.)

4.2.3 タスク分割指示文

Tascell で追加された文 (statement)

`do_two` $statement_1$ $statement_2$
`handles` *task-name* `{` $statement_{put}$ $statement_{get}$ `}`

により，$statement_2$（図 8 では “`fib(n-2)`”）の計算を $statement_1$ （“`s1=fib(n-1);`”）の計算中に生成してもよいことを指示する．より正確には以下の処理が行われる．(1) ワーカはまず暗黙の**タスク要求ハンドラ**を有効にし，$statement_1$ を実行する．ハンドラが起動すると，ワーカは新しいタスク *task-name* を生成し，$statement_{put}$ （図 8 の “`this.n=n-2;`”）によりタスクオブジェクトの入力フィールドに値をセットし，そのタスクオブジェクトをタスク要求元に送信した後，後戻りから復帰する．(2a) $statement_1$ の実行中にタスク要求ハンドラが起動しなければ $statement_2$ がそのまま実行される．(2b) ハンドラが起動した場合，$statement_2$ の実行は飛ばされ，生成したタスクの結果を待ち合わせる．その後，$statement_{get}$ （図 8 の “`s2=this.r;`”）を実行することで受け取った結果をマージする．なお，ワーカは別のタスク要求を出し，待ち合わせ中アイドル状態にならないようにする．

生成されるタスクの型は識別子 *task-name* により指定する．$statement_{put}$ や $statement_{get}$ 中では `this` キーワードによりタスクオブジェクトを参照できる．$statement_{put}$ ではこのオブジェクトの入力フィールドにタスクの入力をセットすることが要請され，$statement_{get}$ ではオブジェクトの出力フィールドを参照してタスクの結果を獲得し $statement_2$ と等価な処理を行うことが要請される．

さらに，反復計算を分割するための並列 `for` ループ文も用意している．構文は以下の通りである．

`for(int` *identifier* `:` $expression_{from}$ `,` $expression_{to}$`)`
$statement_{body}$
`handles`
task-name `(int` $identifier_{from}$ `, int` $identifier_{to}$`)`
`{` $statement_{put}$ $statement_{get}$`}`

図 9 の Pentomino プログラムでは並列 `for` ループ “`for (int p:  j1, j2) {...} handles pentomino`

`(int i1,int i2) {{...} {s+=this.s}}`.” により最外ループを並列化している．

この文は，$expression_{from}$ 以上 $expression_{to}$ 未満の整数に対して $statement_{body}$ を繰り返す．この繰り返し実行の間，暗黙のタスク要求ハンドラが有効になる．このハンドラが起動すると**未処理の反復の半分**が新しいタスクとして生成される．実際に新しいタスクに割り当てられる範囲は $statement_{put}$ 中で，$identifier_{from}$ と $identifier_{to}$ で参照できる．生成されたタスクの結果は $statement_{get}$ で処理する．

Tascell はまた，undo，redo 操作を定義するため，`dynamic_wind` コンストラクトを備えている[3]．構文は以下の通りである．

`dynamic_wind`
$statement_{before}$ $statement_{body}$ $statement_{after}$

この文は基本的には，$statement_{before}$（図 9 ではピースを置く “do” 処理），$statement_{body}$，$statement_{after}$（ピースを取り除くという “undo” 処理）の順で実行される．これに加え，$statement_{after}$ は $statement_{body}$ の実行中，この文よりも古いタスク要求ハンドラの起動前にも（“undo” 処理として）実行される．また $statement_{before}$ はそのようなハンドラの起動終了後にも（“redo” 処理として）実行される．

`do_two`，並列 `for`，`dynamic_wind` の各文は $statement_1$ や $statement_{body}$ 内で動的にネストさせることができる．この時，図 5 や図 6 のように複数のタスク要求ハンドラおよび undo，redo 節が同時に有効になる．各ワーカは `do_two` と並列 `for` の先頭においてポーリングによりタスク要求を検出する．検出すると一時的な後戻りにより最も古い要求ハンドラを起動することで，できるだけ大きいタスクを生成しようとする．後戻りの途中に undo，redo 節があった場合は，全ての undo 節が後戻りの経路順に実行され，ハンドラ起動後，全ての redo 節がその逆順に実行される．

[3]Scheme 言語 [6] をご存知の方はその同名のコンストラクトと同様のものだと考えていただきたい．

4.3 Tascell による並列プログラミング手順

図 8 の Tascell プログラムは，(1) 図 1 の C プログラムをベースとして (2)`fib` 関数定義に `worker` キーワードを追加し (3) 並列実行が可能な 2 つの文を見つけ (4) タスクのオブジェクト構造や名前を考えつつ `do_two` を適用し (5) タスクの定義をトップレベルに追加する，という手順で書くことができる．

図 9 の Tascell プログラムも同様に，図 3 の C プログラムから書き換えることができる．Fibonacci との主な違いは，(3) において「別々の作業領域を用意すれば」並列化可能であるようなループを見つけてそこに並列 `for` を適用すること，タスクオブジェクト内にその作業領域を用意すること，undo，redo 操作の箇所に `dynamic_wind` を適用する必要があることである．

5 Tascell コンパイラの実装

Tascell コンパイラは Tascell 言語から XC-cube 言語 [9, 11] への変換器として実装した．XC-cube は我々が提案している拡張 C 言語であり，拡張の一つとして，入れ子関数（関数の中で定義する関数）を定義することで “L-closure” あるいは “Closure” と呼ぶ軽量なクロージャを生成するための機能を備えている．この機能を用いることで，実行スタックの底で「眠っている」フレームにアクセスする，すなわち**自らの計算状態を書き換える**ことができ，Tascell ワーカのタスクの要求時生成や undo/redo 処理を実現できる（詳細な実現方法は文献 [4] を参照されたい）．XC-cube 自体の実装はアーキテクチャごとに GCC コンパイラを改造して行っているが，L-closure と Closure の機能に関しては，それぞれ LW-SC，CL-SC という名称で標準 C 言語への変換による実装も行っている [5]．やや性能を犠牲にするがこれを利用することで，C コンパイラを使える環境であれば Tascell の利用が可能である．

6 性能評価

京大のスーパーコンピュータ Thin クラスタ（最大 8 ノード）および InTrigger のうち，chiba，hongo，mirai，kobe，keio の最大 5 拠点 36 ノードを使用し

表 1: 評価環境

	京都大学オープンスーパーコンピュータ（Thin クラスタ）	InTrigger chiba，kobe，keio，kyushu クラスタ
プロセッサ	AMD Opteron 8350 Barcelona Quad-Core × 4	Xeon E5410 2.33GHz Quad-Core × 2
メモリ	32GB	32GB（chiba，hongo） 16GB（mirai，kobe，keio）
コンパイラ（Tascell）	X86-64 GCC 3.4.3 `-O2` ＋ LW-SC による変換ベースの L-closure 実装 [5]	XC-cube（X86-64 GCC 3.4.6 ベース）`-O2` Closure に基づく入れ子関数 [9, 11]
コンパイラ（Cilk）	Cilk 5.4.6 ＋ X86-64 GCC 3.4.3 `-O2`	—
Tascell サーバ	Allegro Common Lisp 8.1 `(speed 3) (safety 1) (space 1)`	Steel Bank Common Lisp 1.0.39 `(speed 3) (safety 3) (space 1)`

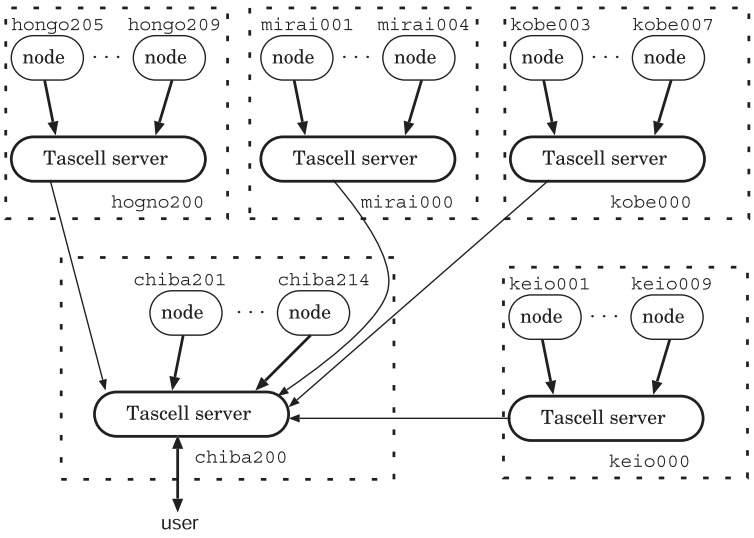


図 10: 性能測定時の InTrigger クラスタ間の接続．

た性能評価を示す．Thin クラスタの 1 ノード内の評価においては，Cilk との比較も行った．評価環境の詳細を表 1 に示す．また，InTrigger での評価におけるクラスタ間の接続図を図 10 に示す．InTrigger の評価においては，chiba（14 ノード），hongo（4 ノード），mirai（4 ノード），kobe（5 ノード），keio（9 ノード）の順で計算に参加するクラスタを追加していった．

ベンチマークプログラムとして，説明で用いた Fib(n)，Pentomino(n) のほかに n 女王問題の全解探索（Nqueens(n)），$n \times n$ の行列の LU 分解（LU(n)），2 つの n 要素の配列間の全要素ペア（(a_i, b_j) for all $0 \leq i, j < n$）について比較演算を実行するプログラム（Comp(n)），$(2n+1)^3$ 個の同一質量の質点からある点にかかる総引力を計算するプログラム（Grav(n)）を用いた．

Tascell の Nqueens は，Pentomino と同様，並列 `for` と `dynamic_wind` の組み合わせによる実装である．LU と Comp は，cache-oblivious な再帰アルゴリズムで実装されている．つまり，たとえば Comp においては，サイズ n と m $(n \geq m)$ の配列を比較する Comp タスクが，サイズ $n/2$ と m の配列を比較する 2 つのタスクに分割される．Grav は，Tascell では並列 `for` の三重ネスト（各ネストが座標軸に対応）で実装されている．なお，全てのアプリケーションにおいて，粒度を上げるための恣意的な閾値を設定しない細粒度並列の実装を行っている．

6.1 Thin クラスタでの評価

まず，逐次実行のオーバーヘッドを評価するため，Tascell を 1 ワーカで実行したときの実行時間を，C と Cilk のそれぞれのプログラム（Tascell とほぼ同

表 2: 1 ワーカ実行時の逐次 C プログラムとの実行時間の比較.

	Elapsed time in seconds (relative time to plain C)		
	C	Cilk	Tascell
Fib(40)	0.596	5.71	1.52
	(1.00)	(9.59)	(2.54)
Nqueens(15)	8.10	22.2	11.9
	(1.00)	(2.74)	(1.46)
Pentomino(12)	1.14	1.69	1.64
	(1.00)	(1.47)	(1.43)
LU(2000)	9.21	9.30	9.24
	(1.00)	(1.01)	(1.00)
Comp(30000)	3.32	6.02	3.99
	(1.00)	(2.08)	(1.20)
Grav(200)	2.85	5.80	4.29
	(1.00)	(2.03)	(1.50)

じアルゴリズムで実装[4]）の実行時間と比較した.

性能測定の結果を表 2 に示す．ほぼ全てのアプリケーションで，並列化のオーバーヘッドが Cilk より非常に低く抑えられていることがわかる．特に Fib では Cilk におけるタスクの生成頻度が非常に高いため，Naueens では Cilk における作業空間のコピーが頻繁に行われるため，差が顕著に現れる（Pentomino でも作業空間のコピー回数の差の影響はあるが，コピーやタスク生成の頻度が Nqueens より小さいため，性能差がでにくいと考えられる）．また，LU ではタスク分割可能な箇所の発生頻度が少ないため，Tascell でも Cilk でも並列化のオーバーヘッドがほとんど現れない.

1 ノード内（共有メモリ環境）で複数のワーカで並列計算を実行したときの性能測定の結果を図 11 に示す．逐次性能の差がそのまま並列計算の結果にも反映され，LU 以外の全てのベンチマークで Tascell が Cilk より高い性能を示している．例えば，Nqueens(16) の 16 並列の計算では，Tascell は Cilk に対して 1.84 倍（=0.684/0.372）の速度向上を達成している.

Tascell ，Cilk ともに LU において 8 並列以上で性能が急に低下しているが，これは，メモリバンド幅の飽和のほか，メモリが NUMA 構成である Thin ノードにおいて，全ての CPU から一つのメモリモジュールに配置された行列へのアクセスを多発させてしまっていることが原因と考えられる.

次に，複数の計算ノードを単一のサーバに接続して，並列計算を行ったときの性能評価の結果を図 12 に示す．LU 以外の各タスクの計算コストが通信コストに対して比較的大きいベンチマークでは良い速度向上が得られている．一方，LU では全く速度向上を得ることができない．これは，タスク分割のたびに，タスクの入力および結果として部分行列の送受信が発生するためである.（文献で [8] 等で考察されているように，一般に LU のように大きなサイズの共有データを要するアプリケーションで，ワークスティーリング方式の動的負荷分散で充分な速度向上を得るのは非常に困難である.）

[4]Nqueens と Pentomino では `SYNCHED` による子スレッド間の作業空間の再利用（3 章の脚注参照）を行っている.

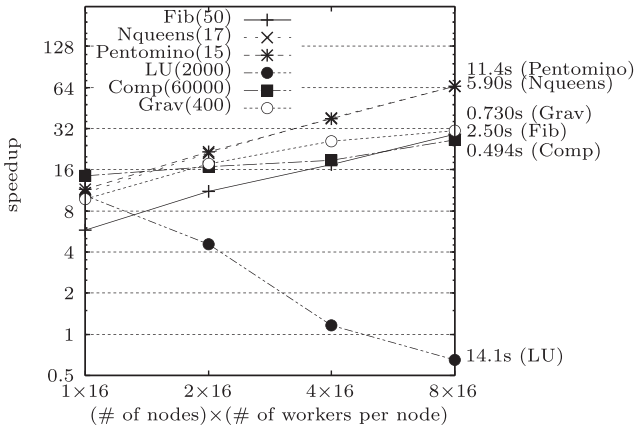


図 12: 複数の計算ノードを用いた並列化による逐次 C プログラムに対する速度向上（各ノード 16 ワーカ）．各グラフ右の秒数は 8 ノードでの実行時間.

6.2 InTrigger での評価

測定結果に基づく InTrigger における台数効果のグラフを図 13，実行時間および速度向上を数値でまとめたものを表 14 に示す．InTrigger においては，タスクオブジェクトのサイズに対して計算量が大きいベンチマークのみ評価を行った.

結果が示す通り，日本の各拠点に配置したクラスタ間で協調して並列計算を行ったような場合でもおおむね良好な速度向上が得られている．特に，Nqueens(18) においては 288 コアで，Tascell1 ワーカ比で 222 倍，C プログラム比でも 202 倍のスピードアップが得られている．Tascell ではタスクの分割回数が少なく抑えられているため，クラスタ間のネットワーク性能による影響は小さいと考えられる.

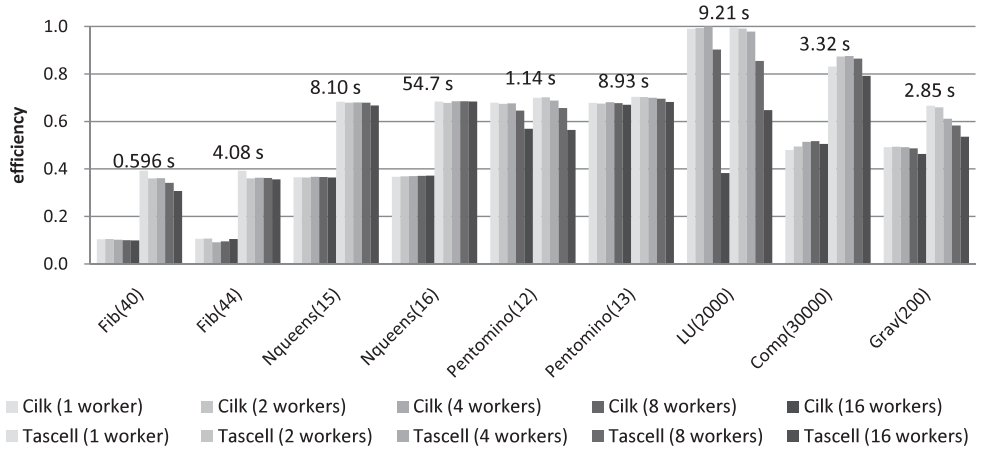


図 11: 1 ノード内の複数ワーカによる並列計算の性能測定結果．縦軸の efficiency は S/n_w（S: 逐次 C プログラムに対する速度向上，n_w: ワーカ数）で定義した値（つまり，efficiency = 1 で理想の速度向上）．各ベンチマークの上部の数値は C の実行時間．

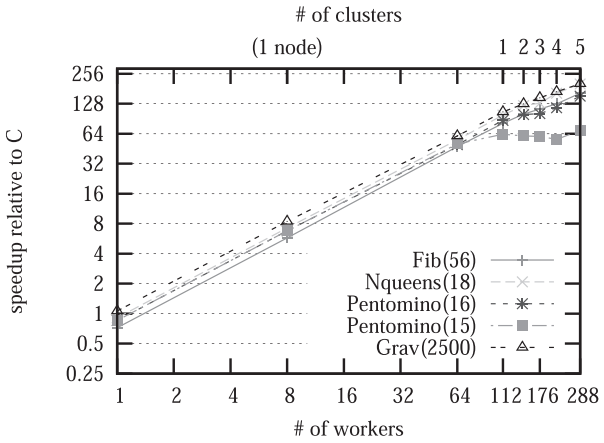


図 13: InTrigger クラスタにおける並列計算の性能測定結果．

図 14: InTrigger におけるプログラムの実行時間（s）および速度向上

	C	Tascell (Closure)				
# of workers	1	1	256	speedup		
	t_S	t_{TC1}	t_{TC288}	t_S/t_{TC1}	t_S/t_{TC288}	t_{TC1}/t_{TC288}
Fib(56)	2835	3763	17.3	0.721	164	218
Nqueens(18)	3574	3933	17.7	0.909	202	222
Pentomino(16)	7304	8564	47.7	0.853	153	180
Pentomino(15)	707	827	10.3	0.855	68.9	80.3
Grav(2500)	4526	4259	22.2	1.06	204	192

7　おわりに

本稿では，不規則なアプリケーションや不均質な広域分散環境においても比較的小さなプログラミングコストで良好な並列化性能が得られる，バックトラックベースの動的負荷分散による並列計算フレームワーク Tascell を紹介した．また，京都大学オープンスーパーコンピュータおよび，広域分散計算環境

InTriggerにおける性能評価結果によって，特にバックトラック系の探索アルゴリズムにおいて高い性能を得られることを示した．

今後は，実装の改善によって特に分散環境での並列化性能を向上させることや，ノードの動的脱退をサポートすることによる信頼性の向上を目指している．また，より実用的なアプリケーションを本フレームワークで並列化することも目指している．具体的なアプリケーションをお持ちで，本稿を読み興味を持っていただけた方はご一報いただければ幸いである．処理系は，`http://super.para.media.kyoto-u.ac.jp/sc/` で公開している．

参考文献

[1] Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., von Praun, C. and Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing, *SIGPLAN Not.*, Vol. 40, No. 10, pp. 519–538 (2005).

[2] Corporation, I.: A quick, easy and reliable way to improve threaded performance—Intel Cilk Plus. `http://software.intel.com/en-us/articles/intel-cilk-plus/`.

[3] Frigo, M., Leiserson, C. E. and Randall, K. H.: The Implementation of the Cilk-5 Multithreaded Language, *ACM SIGPLAN Notices (PLDI '98)*, Vol. 33, No. 5, pp. 212–223 (1998).

[4] Hiraishi, T., Yasugi, M., Umatani, S. and Yuasa, T.: Backtracking-based Load Balancing, *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2009)*, pp. 55–64 (2009).

[5] Hiraishi, T., Yasugi, M. and Yuasa, T.: A Transformation-Based Implementation of Lightweight Nested Functions, *IPSJ Digital Courier*, Vol. 2, pp. 262–279 (2006). (IPSJ Transactions on Programming, Vol. 47, No. SIG 6(PRO 29), pp. 50-67.).

[6] Kelsey, R., Clinger, W. and Rees, J.: Revised^5 Report on the Algorithmic Language Scheme, *ACM SIGPLAN Notices*, Vol. 33, No. 9, pp. 26–76 (1998).

[7] Mohr, E., Kranz, D. A. and Halstead, Jr., R. H.: Lazy Task Creation: A Technique for Increasing the Granularity of Parallel Programs, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 3, pp. 264–280 (1991).

[8] van Nieuwpoort, R. V., Kielmann, T. and Bal, H. E.: Efficient load balancing for wide-area divide-and-conquer applications, *PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, New York, NY, USA, ACM, pp. 34–43 (2001).

[9] Yasugi, M., Hiraishi, T. and Yuasa, T.: Lightweight Lexical Closures for Legitimate Execution Stack Access, *Proceedings of 15th International Conference on Compiler Construction (CC2006)*, Lecture Notes in Computer Science, No. 3923, Springer-Verlag, pp. 170–184 (2006).

[10] 斎藤秀雄, 鴨志田良和, 澤井省吾, 弘中健, 高橋慧, 関谷岳史, 頓楠, 柴田剛志, 横山大作, 田浦健次朗: InTrigger : 柔軟な構成変化を考慮した多拠点に渡る分散計算機環境, 情報処理学会研究報告-ハイパフォーマンスコンピューティング（HPC）, Vol. 2007, No. 80, pp. 237–242 (2007).

[11] 八杉昌宏, 平石拓, 篠原丈成, 湯淺太一: L-Closure：高性能・高信頼プログラミング言語の実装向け言語機構, 情報処理学会論文誌：プログラミング, Vol. 11, No. SIG 1 (PRO 13), pp. 118–132 (2008).

科学者の“対話力”トレーニングプログラムのための デジタルコンテンツの開発

加納圭[1](申請代表者), 水町衣里[1,2], 高梨克也[3], 元木環[3,4](センター代表者)

[1]京都大学物質－細胞統合システム拠点(iCeMS=アイセムス), [2]京都大学総合博物館, [3]京都大学学術情報メディアセンター, [4]京都大学情報環境機構

1 共同研究の背景と目的

近年、人々の日常生活や意思決定現場に、科学技術が深く関わるようになってきている。そのため、科学者自らが、研究成果や研究成果がもたらす社会的な影響について、専門外の人々と直接対話をすることなどが求められるようになってきた（Royal society, 2006）。

本年度から実施予定の「第４期科学技術基本計画(東日本大震災の影響で再検討中)」においても、「科学・技術コミュニケーションの活動の推進」がうたわれており、その具体策として本年度から「国民との科学・技術対話」が本格実施されることとなった(総合科学技術会議、2010)。これにより、１件あたり 3000 万円以上の公的資金を得た科学者は何かしらの科学コミュニケーション活動を行うことが義務づけられることとなり、また、3000 万円未満の公的資金を得た科学者が「国民との科学・技術対話」を行った際にはプラス評価が行われることとなった。そのため、科学者は今後ますます科学コミュニケーション活動を行う機会が増えていくことと予想される。

これからの科学者は、自身の研究をバランスよく伝えることができ、かつその社会的影響・意義について専門外の人々とコミュニケーションをとることができる能力、「科学コミュニケーション能力」を身につけることが必要とされるであろう。

しかし、科学者を対象とした科学コミュニケーションのトレーニングプログラムは模索段階にある。アメリカ科学振興協会、英国王立協会、オーストラリア国立大学などがすでに科学者のためのコミュニケーショントレーニングコースを提供しているのに対し、日本ではほとんどみられない。また、上記のアメリカ・イギリス・オーストラリアで提供されているプログラムは、マスメディアを通した情報発信や講演会などにおけるプレゼンテーションといった、科学者から一般市民に向けた一方向的な情報伝達に重点が置かれている。

これらの現状を踏まえ、筆者らは、より双方向的なコミュニケーションスキルをトレーニングするプログラムの開発を行うこととした。筆者らは科学に関する双方向コミュニケーションを「科学に対する複数の見方が同時に成立し得ることを学び合う過程」と捉えており、科学者から一般市民に向けた一方向的な情報伝達だけでなく、科学者が一般市民の考え方を学びとることも重要であると考えた。

双方向コミュニケーションの場として、「サイエンスカフェ」とよばれる「科学者と一般市民との対話の場」が近年日本において全国的に普及している[1)]。具体的には、「サイエンスカフェ」とは飲み物やお菓子を片手に気軽な雰囲気で、一般市民と科学者とがともに科学に関する話題について対話を行う場のことである(松田健太郎、2008、中村征樹、2008)。

そこで筆者らは「サイエンスカフェ」を実践トレーニングの場とした「科学者の”対話力”トレーニングプログラム（Dialogue Skills Training Program, 以下 DST プログラム)」を開発することとした。DST プログラムは「事前レクチャー」、「サイエンスカフェの実践」、「事後レビュー」の３つのパートからなる予定であり、本共同研究では「事前レクチャー」で用いるデジタルコンテンツの開発を目的とした。

具体的には、「サイエンスカフェ」のビデオ記録から、特徴的なコミュニケーション例を抽出・分類し、それらをデジタルコンテンツとして表現することを試みた。

2 サイエンスカフェの実施、記録

加納と水町が所属する京都大学物質－細胞統合システム拠点(iCeMS)・科学コミュニケーショングループが、”iCeMS カフェ”と題したサイエンスカフェを 2010 年 11 月, 12 月に計 3 回実施した。

iCeMS カフェとは、まず京都大学 iCeMS 所属の主任科学者が自身の研究室の研究テーマについて概説し、その後、研究室所属の若手科学者(大学院生含む)1 名が各テーブルに着き、3～5 人の一般市民と対話を行うものである(図 1)。

一般市民は京都大学及び京都大学 iCeMS の web ページ、メーリングリスト、ポスター掲示、新聞掲載等の広報手段を通じて公募で集められた。年齢も職業も多様な方々（画家、会社員、自営業、留学生、高校生、大学生、異分野の研究者などが各回 30 名程度参加した。

図 1　iCeMS カフェのテーブルの様子
(左奥の男性が若手科学者)

今回実施した 3 回の iCeMS カフェの実施日、テーマ、テーブル数は以下の通りである(表 1)。

No.	実施日	テーマ（タイトル)	テーブル数
1	2010 年 11 月 13 日	糖、いい仕事してますね	5
2	2010 年 12 月 11 日	未来の太陽電池	6
3	2010 年 12 月 18 日	からだの中の宅配便	6

表 1　iCeMS カフェの実施日、テーマ、テーブル数

iCeMS カフェ各回 2 テーブルをビデオカメラで撮影した。各テーブルにつき 2 台のビデオカメラを用い、テーブルの前後を挟み込むようにして撮影した。これにより、テーブルにつく人全員の表情やジェスチャを撮影することが可能となった(図 2)。音声は無線マイクをテーブル上に置いて録音した(図 2)。ビデオ撮影が行われたテーブルにおいては、若手研究者も含めて参加者全員から文書で研究に使用するためのビデオ撮影への同意を得た。

また、ビデオの音声記録を文字に書き起こしたトランスクリプトを作成した。

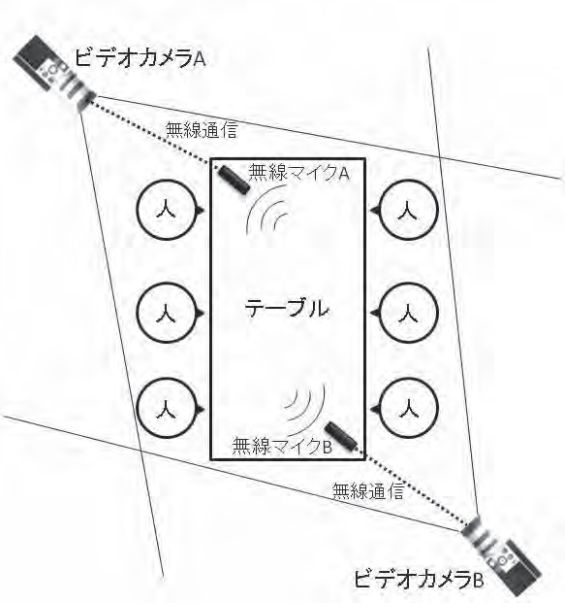


図 2　iCeMS カフェの各テーブルのビデオ記録方法

3 研究成果：ビデオ記録へのタギング

2011年11月13日に開催したiCeMSカフェを記録した2テーブルのビデオ記録から、典型的なコミュニケーションパターンを見出せないか、タグ付け（タギング：映像のある個所に、タグと呼ばれる短いコメントやキーワードを付け整理する手法）を試みた。タギングを行うことで、各テーブル2時間程度からなるビデオ記録を効率よく整理し共同研究者間で共有できるようになるだけでなく、サイエンスカフェでよく見られるコミュニケーションパターンを分類することも可能となった。

タギングされた情報を抽出・分類したところ、「サイエンスカフェで行われるべき理想の対話」「特徴的なコミュニケーション手段」の二つの視点で整理することができた。このような視点でビデオを観察することが、対話トレーニングにとって重要ではないかと仮説を立てた。

まず「サイエンスカフェで行われるべき理想の対話」という観点からタギングを行ったのは、サイエンスカフェは以下にあげる点において、講演会や講義とは異なると考えられるからである。

1. 科学を社会に説明したり、科学を広めたりするのではなく、社会の中に科学を位置づける場である。
2. 一方的な情報伝達が行われるのではなく、双方が学び合う場である。
3. 科学者が特殊な立場に立つのではなく、参加者と科学者が対等である。

これらの点を「サイエンスカフェで行われるべき理想の対話」と捉え、トップダウンの観点から、科学者が「1. きく」、「2. つたえる」、「3. 分かち合う」の3つのタグで映像を視聴することがトレーニングにつながると考えた。

次に、多人数インタラクションの分析手法（坊農・高梨，2009）を利用し，ビデオ記録中において特徴的なコミュニケーションが行われた箇所に注目し、以下の3つのタグを見いだした。

a. ジェスチャやアナロジーといった「**描写方法**」
b. 科学者の方が一般市民よりも知識レベルが高いことを暗に示すような「**知識分布の顕在化とメンバシップ対比の発動の機会**」
c. 物理的姿勢・視線・質問のタイミング・話題の切り替わり時の対応といった「**科学者(研究者)中心の参与枠組み**」

これらは「iCeMS カフェで行われた特徴的なコミュニケーション手段」というボトムアップの観点からのタグで、「サイエンスカフェで行われるべき理想の対話」のタグと、視点は異なるが関連があると考えた。

これらのことから、一回の iCeMS カフェのビデオ記録映像はトップダウンの観点から見いだされた3つのタグ、ボトムアップの観点から見いだされた3つのタグを組み合わせた、計3×3 = 9 種類のカテゴリーに相当する（図 3）映像シーンに分類できると考えた。

		特徴的なコミュニケーション手段		
		a. 描写方法	b. 知識分布の顕在化とメンバシップ対比の発動の機会	c. 研究者中心の参与枠組み
サイエンスカフェで行われるべき理想の対話	1. きく Listen	Scene 1-a	Scene 1-b	Scene 1-c
	2. つたえる Tell	Scene 2-a	Scene 2-b	Scene 2-c
	3. 分かち合う Equal Relationship	Scene 3-a	Scene 3-b	Scene 3-c

図3　「サイエンスカフェで行われるべき理想の対話」、「特徴的なコミュニケーション手段」という2つの観点の組合せによる映像シーンの分類

4 今後の展望

本研究で分類した映像シーンは、サイエンスカフェに代表される双方向コミュニケーションの場でよく見られるコミュニケーション事例である。

これらのシーンを科学者と見ながら、双方向コミュニケーションとはそもそもどういうものなのか、双方向コミュニケーションを成立させるには具体的にどのようなティップスがあるのか、といったことについて話合うことで、科学者の”対話力”トレーニングにつながると予想される。

今後は、それら映像シーンを補足説明と共にweb 上で視聴できるような（ひとつのタイムライン上の映像シーンを複数の視点で視聴できる）デジタルコンテンツを開発し、我々がこれまでに開発してきた DST プログラムに実装する予定である。

より効果的で効率的な「国民との科学・技術対話」を模索している科学者にとって意味のあるコミュニケーショントレーニングの場が提供できる体制を維持、整備していきたい。

5 謝辞

iCeMS カフェに参加し、本研究にご協力してくださった木曽真研究室の皆様、今堀博研究室の皆様、橋田充研究室の皆様に深く感謝いたします。また、本研究の趣旨にご同意くださったiCeMS カフェの一般参加者の皆様にも深く感謝申し上げます。最後に、平本毅さん、城綾実さんにはビデオ撮影で多大なる協力をいただきました。感謝いたします。

注釈

1) 100 を超えるサイエンスカフェ主催団体が存在する。それらの中には双方向コミュニケーションを目指さないものも含まれている。

参考文献

総合科学技術会議, 2010, 「「国民との科学・技術対話」の推進について（基本的取組方針）」

松田健太郎, 2008, 「日本のサイエンスカフェをみる：サイエンスアゴラ 2007 でのサイエンスカフェポスター展・ワークショップから」, 科学技術コミュニケーション, **3**, 3-15

中村征樹, 2008,「サイエンスカフェ—現状と課題」, 科学技術社会論研究, **5**, 31－43

The Royal Society, 2006, “Survey of factors affecting science communication by scientists and engineers”, The Royal Society, London.

坊農真弓・高梨克也（編著）, 2009, 「多人数インタラクションの分析手法」, オーム社

総合博物館に対する親しみと学際融合の場を醸成するための ビジュアルデザインポリシーの策定と実践

塩瀬隆之[1], 元木環[2,3](センター代表者), 山下俊介[1], 水町衣里[1], 永田奈緒美[2], 奥村昭夫[2], 大野照文[1](申請代表者)

[1]京都大学総合博物館, [2]京都大学学術情報メディアセンター, [3]京都大学情報環境機構

1　共同研究の背景

京都大学は多様な学問分野を擁する総合大学で、常に新奇な学問分野が生まれる土壌であり続けることを目指しているが、近年多い一過性のプロジェクトベースの連携のみでは、新奇な研究も教育も根付く間なく携わった人々の離散とともに終焉してしまう。学内の優れた研究者、能力の高い院生、学生が、一堂に会することができる研究教育交流の場を構築し、京都大学のユニバーシティ・アイデンティティの確立につながるような新たな仕組みが必要である。総合博物館はその機能を果たす一つの場となる存在となるべく、さまざまな企画展示、教育プログラムの実施において、特に「学際融合ネットワークの確立」「博物館への親近感を深める」「京都大学のアイデンティティを活かす」ことに注力した活動に取り組んでいる。

開館以来 10 年にわたり、１）異分野研究者が協働して企画した展示の開催等で学内での研究交流を活発化し、２）生涯学習等の多様な知的ニーズに応えて市民の支持を獲得する等の成果をあげてきたが、それは研究交流の活発化にせよ、市民の支持獲得にせよ、博物館への親近感や企画に込められたアイデンティティを実感してもらうことが重要かつ基本であるとの認識を持っているからである。他大学の博物館では、北海道大学のように大学のカリキュラムの一環として常設展示改良の試みがなされたり、東京大学での企業との連携や高級ブティックを会場とした博物標本の学外展示の試みなどの取り組みが見られるが、本博物館のように、学内の人的研究資源的リソースを元に、研究者から市民来館者までを対象に、「学際融合ネットワークの確立」と「博物館への親近感を深める」という両立しにくい特色を同時に打ち出している例は他にないものである。

2　共同研究の目的

この特色を示すためには、空間デザインから印刷物まで適応可能であり、博物館への親近感やアイデンティティを涵養するような、ポリシーを鑑みたデザイン的な統一性（VI）を持っていることが必要である。これまでも企画展、教育プログラムなどにおいて部分的にVI を導入することで、多様な分野の教員間学際融合ネットワークの構築や参加者の拡大に効果があったが、館全体に広げて、学内外の人たちに、学際融合という表現が難しい特色と、より大きな親しみやアイデンティティを感じてもらうデザインポリシーを策定することについては、入念な議論を重ねる機会もなく、体系的な試みがなされていない。

本研究では、館全体に広げてVI についての議論を重ねる機会を持ち、それをもとに学際融合と親近感を同時に伝えられるような一貫性のあるVI を策定、実践し、その手法についても考察する。

博物館／図書館などの公共空間において、その利用価値を高めるために、建築施行段階で用途に応じたトータルなVIが作成されることは、これまで建築デザインなどの分野において前例があるが、大学博物館の空間とコンテンツ（展示やプログラムといったソフトにあたる部分）における全体統一VIの策定の有効性についての研究とデザイン実施に例はない。「大規模大学の構成員間の知的協働」と「市民からの支持」は、法人化した京都大学が２１世紀に飛躍してゆくために不可欠な二大要素であり、大学構成員同士、大学と社会との接点ともいえる総合博物館においてVIを総合的に検討・策定すること、ひいてはこのプロジェクトの成功が、京都大学の体質強化につながる方法論をももたらすことにつながると考えられる。

3　実施概要と結果

京都大学の総合博物館として打ち出すべきビジョンやミッション、アイデンティティについての検討を行うため、次の３つを実施した。

1) VI の現状と課題を俯瞰するフィールド調査
2) VI を適用するメディアや場を検分した速報展示の実践
3) VI の方向性を編み出すワークショップ

3.1　VI の現状と課題を俯瞰するフィールド調査

博物館教員とミュージアムスタッフ（主に来館者の受付や案内、展示状況監視を担当）、ワークショップ担当の大学院生、メディアセンター教員とデザイナスタッフで館周辺から館内において、現状の空間やサイン、館の環境や構造についてフィールド調査と意見集約会を行った（表１）。（2010年9月13日、10月1日）

表１　意見集約された現状の課題と分類（抜粋）

案内	各種案内表示がわかりにくい／展示の場所が不明瞭／障害をもつ人への配慮不足
運営	内外の情報共有、役割分担がうまくいってない／今日何をやっているかが共有されにくい
展示	何も入ってない展示ケースが通路においてある／モノが多い／常設展に変化が欲しい／魚類が無い
施設	玄関に一部破損しているところがある／掲示物をセロテープで貼っている／自動改札が必要でなく怖い／休憩場所がない
イメージ	敷居が高い／堅苦しい／学内利用度低い／中高生にも来て欲しい
その他	解説パネル文字の問題（多い、小さい、高さ、フォントバラバラ）

3.2 VI を適用するメディアや場を検分した速報展示の実践

博物館での研究成果速報展示の機会を利用し、VI はメディアや場に関してどのように適用が可能か実践的に検分した。標本の生態をイメージさせる VI の作成、VI を適用した会場説明パネルの設置、VI 展示研究資料の写真のイメージと連動した映像での展示ガイドへの適用などを行った。

事例１：「レバノン―呪いの鉛板－」（平成 22 年 12 月 8 日―12 日、文学研究科　泉拓良教授　先史考古学）では、実物そのものが小さいために資料に刻まれた文字を判読することが難しい。そこでチラシ写真のイメージを映像に持ち込み、映像上でモノの見方までをもガイドする展示手法を試行した。

事例２：「クニマス-70 年ぶりの生存確認」（平成 23 年 1 月 14 日―23 日、総合博物館　中坊徹次教授　魚類系統分類学）では、個体採取の経緯や関係者相関を伝えるのが難しい。そこで、クニマスが湖底深くに生存していた生態をイメージさせる展示ポスターを作成すると同時に、過去の研究から大発見を報じるまでの経緯を同じ VI で展示した。

いずれもコンテンツ作成支援サービスも活用しながら、学内教員の最先端の研究成果を短時間で展示レベルまで引き上げる画期的な試みであり、学内構成員をはじめとする参加者間の一体感を高め、いずれも新聞、テレビで多数の報道で取り上げられ、デザイン性の高い情報発信方法が高評価を得た。

3.3　VI の方向性を編み出す教職員、ミュージアムスタッフとのワークショップ

全４回（2011 年 3 月 2 日、23 日、30 日、4 月 13 日）に渡って、博物館教職員、研究員、ミュージアムスタッフと研究分担者が集まりレクチャーおよびワークショップを実施した。

まず奥村、元木より、ロゴやサイン、展示ガイドブックなど、組織における情報発信をグラフィックデザインで表現する計画についての事例紹介とリ・デザインするための考え方について、レクチャーを行った。次に大野より「博物館の設立理念やミッション、10 年の沿革と活動歴からみる博物館」についてレクチャーを行った。レクチャーの中では「研究資源の保存・修復・活用」の重要性について、「学際融合拠点や社会連携の実験場」を目指していること、好奇心を刺激する場所でありたい、など設立から１０年間の思いも語られた。事務グループ長より近年の来館者数、企画

展など実施事例が紹介された（表２）。このことにより博物館には「資料の保存・研究」と「展示・教育」という２つの側面があることが確認された。

さらに、「博物館の現状・課題」「博物館のVIが応用される箇所」「博物館の VI で伝えたいことはなにか」、「１０年後に博物館がどうあって欲しいか」という問いかけを通じて参加者全員で言葉を生みだし、博物館 VI 計画の方向性を探るワークショップを実施した。ワークショップの中では、多数の意見が提案され、参加者がその意見を俯瞰し、それぞれで具体的な言葉に置き換え、VIにつながる形のアイデアを提案した（図１、２）。最終的に、奥村、永田が各参加者から出された言葉と形のアイデアを象徴化しシンボルマーク等のデザイン案の一例を提案した（図３）。

このことにより組織に置ける VI を策定するにおいて、より自分たちの意見が集約された結果としてとらえることができる、好みでデザインを判断しないという考え方が理解できるようになったという反応が得られた。

表２　京都大学博物館基礎データ

入館者数	年間３万５千人程度
開館日程	水曜～日曜（土日開館の大学博物館は珍しい）
企画展示	企画展（規模大）年１回、特別展（規模小）年２～３回、毎土曜の子ども博物館、公開講座を定期的に実施
配布物の例	博物館パンフ、展示ガイド、ニュースレター、プレスリリース、博物館構想時パンフ、年報、企画展ポスターなど

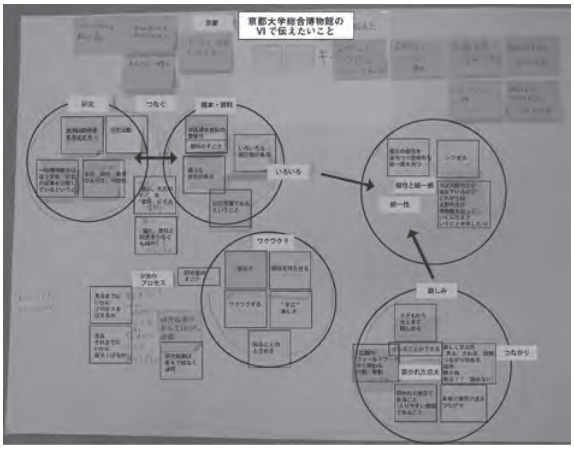


図１　博物館のVIを通じて伝えたいこと

（図１テキスト抜粋）

- 資料と標本＝研究、研究がないと資料価値がない。資料からワクワクさせるのは「研究」
- 研究の過程を見せるとワクワク
- いろんな研究、いろんな人がくるけど、「いろんな」の統一性が出したい。
- いろんな人に親しみとつながりを。

空豆＝『空に向かって実がなる』

向上心のあらわれを感じる（場所）。
空に向かって＝向上心。三つ実がなる＝文化史、自然史、技術史。根＝養分（知識）を吸収。花が咲いている、実のなっている横にも、また別の花が咲き、実がなっている様子が、さまざまな研究が、博物館で（研究成果の）花が咲いて、（研究成果の）実がなっている様子となぞらえることができると考えた。

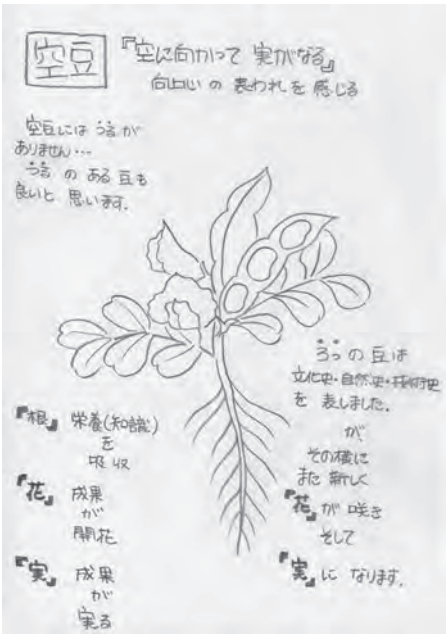


図２　スタッフが具体的な言葉と形のアイデア

図３　アイデアを象徴化したシンボルマーク例

また、この３つの調査や展示、ワークショップの実施を元に、開館時に作成されたシンボルマークの見直しをも含めた、導線の引き方、案内表示、来館者用パンフレットなど、網羅的に検討した結果、空間デザインから印刷物デザインまで、ビジュアルデザインによる改善の余地が大きく存在することが確認された。

4　今後の課題

本研究は２２年度において、博物館におけるミッションのうち「展示・教育」に関するVI 策定についての準備を完了した。ミュージアムスタッフが参加者の中心であったため、各自が来館者と接してきた経験をもとにその方向性を探ることで、「展示・教育」についての側面のVIの検討ができたと考えられる。ビジュアルになる前の段階をスタッフ間で共有しながら、VIについて言葉で考えることは、最終的なビジュアルについての是非を、好みでなく議論できることにつながり、あらわれたビジュアルに対して自分の意見が集約されていると感じることができるという意見が聞かれた。

本年度を終えて、これから必要になると考えられる課題は以下のとおりである。

1．VIの方向性に関する言葉にさらに根拠を持たせることが必要である。そのためには、博物館に対するイメージなどを調査するためのアンケートやヒアリングによる調査手法そのものの抜本的改革を図ることが必要である。これは、京都大学総合博物館の学際融合プロジェクトに集まる研究者・院生・学生、あるいは市民を対象に行っていく。

2．VIの効用・効果、博物館の空間デザインについて評価を実施するには、客観的な評価指標の確立が必要である。これについては、著者らは本共同研究の議論の一部から、研究チームを組み、科学研究費の申請を行ったところ、科学研究費基盤(B)「博物館におけるデザイン―評価サイクルのための展示評価ツールキットの開発」（代表者：塩瀬隆之）H23-25 に採択され、本共同研究の一部について研究プロジェクトとしてスタートさせることを達成した。

引き続き、もう一方のミッションである「資料の保存・研究」部分についても検討を進め、具体的なVI 策定と実践まで行い、評価実験を行う予定である。これは２２年度の速報展示で行ったような実際の展示での実験が必要になることから、引き続きコンテンツ作成支援などをうけつつ、VIの策定と実施、評価指標の確立を目指す。

参考文献

林大智、台湾伝統藝陣博物館における VI(ビジュアルアイデンティティ)の指針、千葉大学工学部工業意匠学科 平成 12 年度卒業論文、2000.

野宮謙吾、西田麻希子、越川茂樹、荒井剛、「岡山県立大学」のビジュアルアイデンティティ、第 56 回日本デザイン学会春期研究発表大会、日本デザイン学会、2009.

Thin SMP クラスタ運転状況 （2010/10 ～ 2011/3）

1）保守作業に伴うサービス休止およびシステムダウン障害発生状況

保守作業に伴うサービス休止

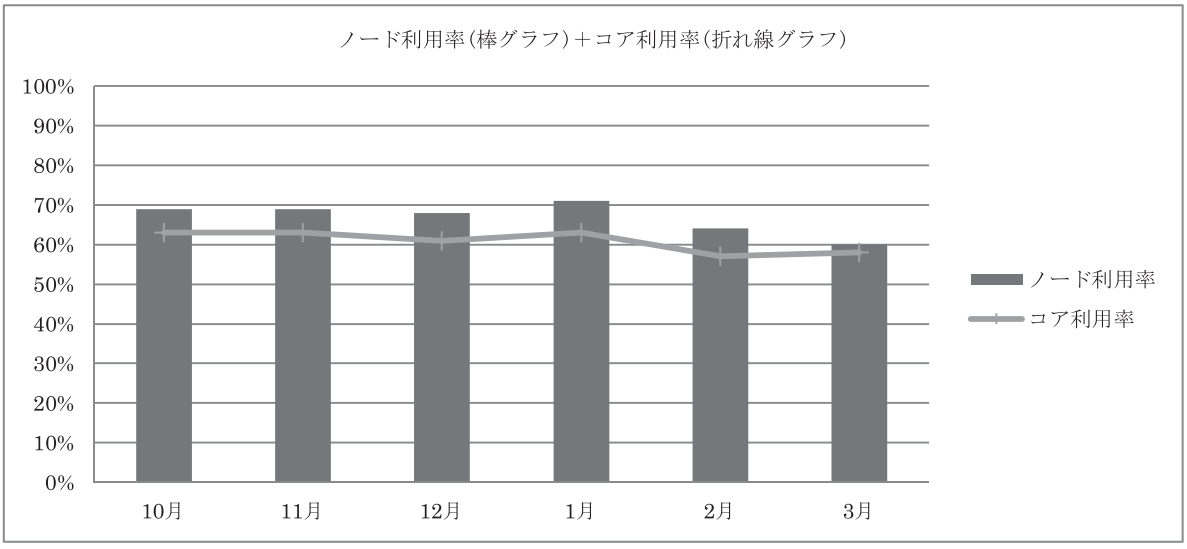
保守開始日時		サービス再開日時		保守時間[h]
2010/10/11	7:00	2010/10/12	19:00	36.0
2010/12/20	9:00	2010/12/20	17:00	8.0
2011/03/31	9:00	2011/03/31	24:00	15.0

システムダウン障害発生状況

障害発生日時		サービス再開日時		ダウン時間[h]
2010/10/7	12:00	2010/10/7	15:00	3.0

2）サービス状況

	サービス時間[h]	バッチ						TSS			
		処理件数	経過時間[h]	占有時間[h]	CPU時間[h]	平均稼動ノード数	ノード利用率	セッション数	セッション時間[h]	CPU時間[h]	平均稼動ノード数
10 月	705	19,950	199,592	2,596,010	2,505,937	345.5	69%	13,802	5,139	145,014	37.9
11 月	720	23,189	192,515	2,659,275	2,525,919	345.3	69%	14,014	5,604	165,413	38.0
12 月	736	26,772	257,101	2,652,197	2,475,578	345.3	68%	13,998	3,406	185,026	37.9
1 月	744	38,270	293,788	2,774,254	2,353,320	345.5	71%	16,181	5,172	178,772	38.0
2 月	672	24,805	241,133	2,249,407	2,118,075	345.5	64%	12,408	3,343	211,495	38.0
3 月	729	40,186	190,202	2,492,107	2,354,588	344.4	60%	13,163	5,127	153,704	37.9
計	4,306	173,172	1,374,331	15,423,251	14,333,416	345.3	67%	83,566	27,792	1,039,423	37.9



※ 占有時間 = 合計(経過時間×占有コア数)

※ 平均稼動ノード数 = 電源 ON 状態のノード数の月平均 (10 分間隔のサンプリングデータより算出)

※ ノード利用率 = 稼動ノードに対するジョブが実行されているノードの割合

※ TSS = ログインノード＋専用クラスタについてのデータ

Fat SMP クラスタ運転状況 （2010/10 ～ 2011/3）

1）保守作業に伴うサービス休止およびシステムダウン障害発生状況

保守作業に伴うサービス休止

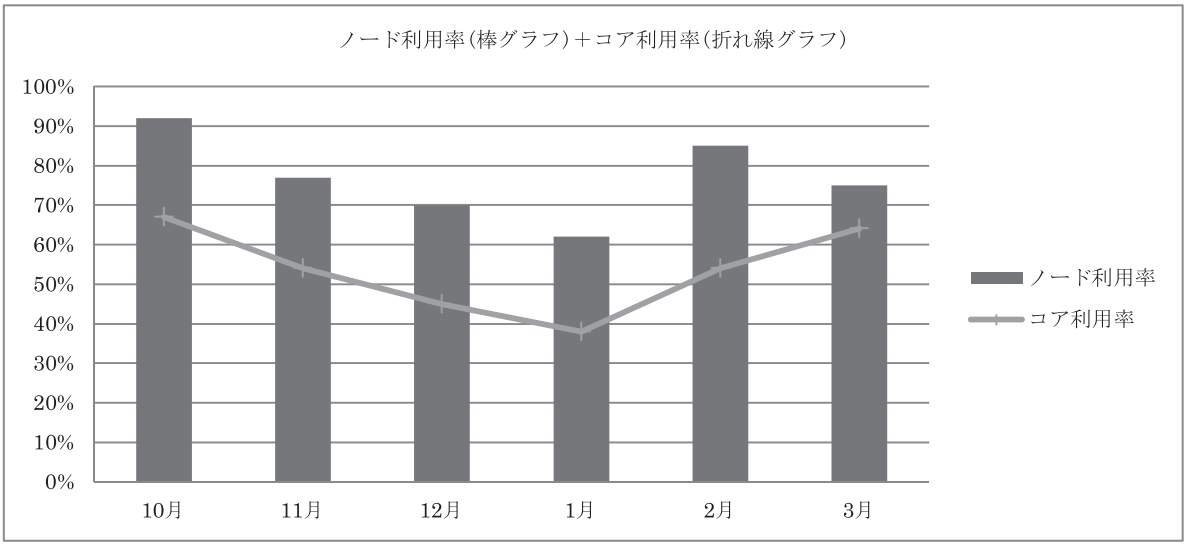
保守開始日時	サービス再開日時	保守時間[h]
2010/10/11 7:00	2010/10/12 19:00	36.0
2010/12/20 9:00	2010/12/20 17:00	8.0
2011/03/31 9:00	2011/03/31 24:00	15.0

システムダウン障害発生状況

障害発生日時	サービス再開日時	ダウン時間[h]
2010/10/7 12:00	2010/10/7 15:00	3.0

2）サービス状況

	サービス時間[h]	バッチ						TSS			
		処理件数	経過時間[h]	占有時間[h]	CPU時間[h]	平均稼動ノード数	ノード利用率	セッション数	セッション時間[h]	CPU時間[h]	平均稼動ノード数
10月	705	1,470	16,012	391,190	291,614	7.0	91%	2,900	564	5,751	1.0
11月	720	1,491	14,251	323,378	213,642	7.0	74%	2,192	347	888	1.0
12月	736	1,459	15,534	277,030	225,183	7.0	66%	2,159	483	17,626	1.0
1月	744	842	11,803	234,575	212,807	7.0	57%	1,753	189	4,135	1.0
2月	672	1,750	14,948	301,690	262,664	7.0	83%	2,107	387	1,124	1.0
3月	729	823	11,460	388,958	323,793	6.9	73%	1,725	224	718	1.0
計	4,306	7,835	84,008	1,916,820	1,529,703	7.0	74%	12,836	2,193	30,241	1.0



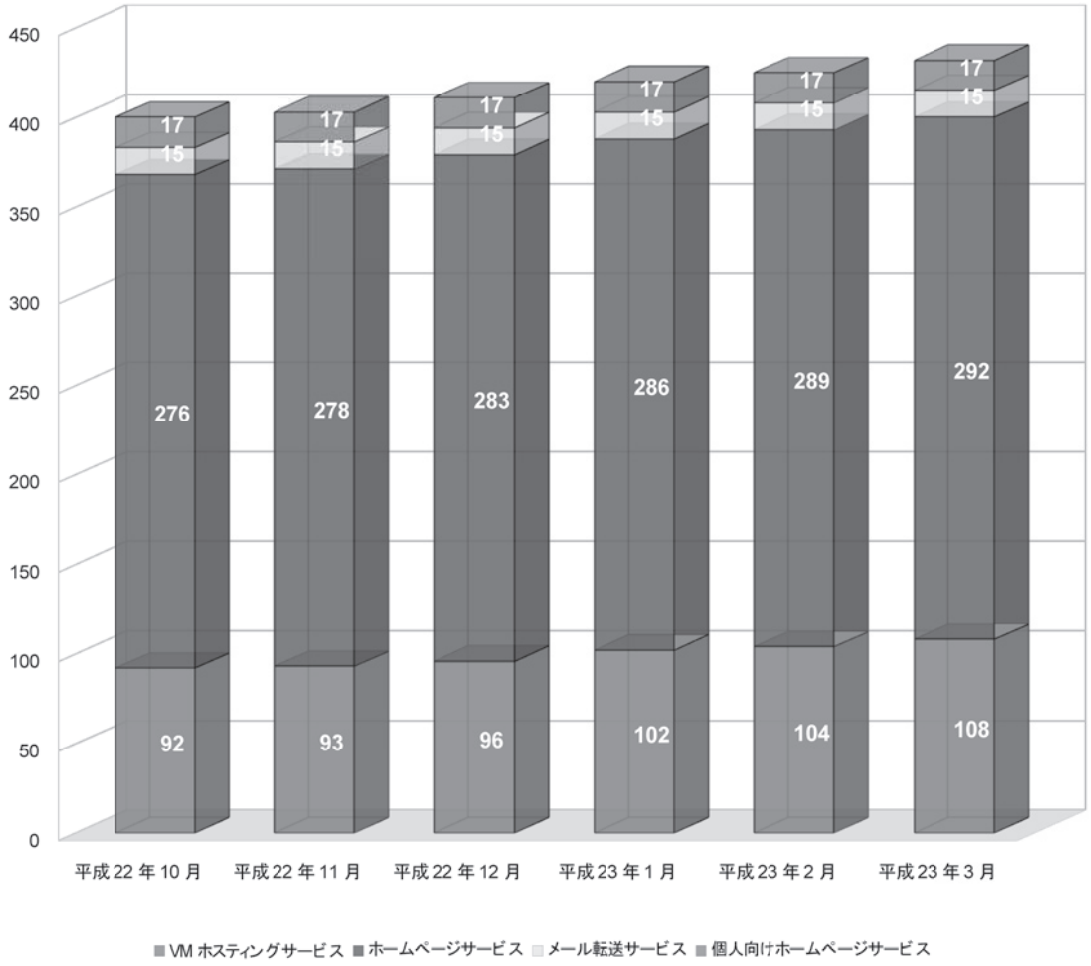
※ 占有時間 = 合計(経過時間×占有コア数)

※ 平均稼動ノード数 = 電源 ON 状態のノード数の月平均 (10 分間隔のサンプリングデータより算出)

※ ノード利用率 = 稼動ノードに対するジョブが実行されているノードの割合

汎用コンピュータシステムのサービス状況

1　ホスティング・ホームページサービス利用状況



（平成 22 年 10 月から平成 23 年 3 月）

大型計算機システム利用承認件数について

平成 23 年 3 月末現在大型計算機システムの利用件数は、2,574 件となっています。

2011年度 学術情報メディアセンター

コンテンツ作成共同研究募集のお知らせ

募集期間：2011年6月15日（水）～7月15日（金）

京都大学学術情報メディアセンターでは、新たな学術・教育コンテンツの開発を行うことを目的とした「コンテンツ作成共同研究」を以下の要領で実施致します。独自性・新規性があるコンテンツで、かつ本センターの設備や技能・発想を持ったスタッフとのコラボレーションが効果的な実験的要素の高い提案を募ります。


1


2

3

1：「ミクストメディアによる総合博物館における展示研究」での展示風景
2：「琉球方言デジタルアーカイブの構築と電子博物館との連携化」において作成された電子博物館ページ
3：「宮古島西原地区の文化と言語」研究において作成された創作方言絵本

募集要領

研究内容

具体的な内容としては、学術教育機関からの教育・研究活動の発信と活動そのものの発展に関わるコンテンツ開発を含む計画、学術教育資源の高度アーカイブ化とその利活用を促進するようなコンテンツ開発、インターフェイス設計などといった実験・研究的要素の高い計画が考えられます。採択された計画は、その実施に必要なコンテンツ作成室の作業工数負担金の一定の範囲内をセンターにて負担いたします。
コンテンツ作成支援サービスとは異なり、作成に専門の設備や技能が必要となるだけではなく、制作会社等に発注することが困難であるが、共同研究実施後その成果を広く社会に寄与しうるコンテンツ作成の提案を募集します。

応募資格

研究申請代表者は、京都大学学術情報メディアセンターの全国共同利用規程の資格を持つ研究者・教員であること（具体的には以下の通り）。また、本センターの教員が1名以上共同研究者として加わる研究グループであること。

- 大学、短期大学、高等専門学校又は大学共同利用機関の教員及びこれに準ずる者
- 大学院の学生及びこれに準ずる者
- 学術研究を目的とする国又は自治体が所轄する機関に所属し、専ら研究に従事する者
- 科学研究費補助金等の交付を受けて学術研究を行う者
- その他センター長が必要と認めた者

応募方法

所定の様式にしたがって計画書（3ページ）を作成の上、2011年7月15日までに郵送・学内便もしくは電子メールでご提出ください。記載された個人情報につきましては、本応募に関わることのみについて利用させて頂きます。ご不明な点は事前にご相談ください。

問い合わせ先・計画書提出先

応募に関するお問い合わせも、このアドレスまでお送りください。

郵送・学内便の場合　〒606-8501 京都市左京区吉田二本松町
京都大学学術情報メディアセンター（南館）
コンテンツ作成室
※封筒に「コンテンツ作成共同研究 申請書在中」と朱書きしてください。

電子メールの場合　京都大学学術情報メディアセンター
コンテンツ作成室
cpt@media.kyoto-u.ac.jp
※タイトルに「コンテンツ作成共同研究 申請書」とお書きください。

審査方法、審査

応募された提案は、京都大学学術情報メディアセンターコンテンツ作成共同研究企画委員会において、研究内容の新規性と独自性、計画の妥当性、また、本センターにおけるコンテンツ作成共同研究のための予算の範囲内で実施可能なものかを審査の上採否を決定します。採否は委員会での決定後、電子メールにて7月中に連絡いたします。

研究の実施、研究成果公開、知財登録

採択後は、研究グループとコンテンツ作成室で実施計画を作成し、それに基づき企画を実施することとなります。
研究終了時には、研究報告書をご提出頂き、京都大学学術情報メディアセンター「全国共同利用広報」にて公開することを条件とします。
共同研究の成果として得られたコンテンツは、京都大学産官学連携センター知的財産室ソフトウェア・コンテンツ分野に著作物として登録する、または本学の研究資源アーカイブに研究資源として登録する事を原則とします。
なお、研究成果の一部は、本センターの共同研究実施例として研究報告から抜粋し、センターWebサイトやパンフレット等へ情報の掲載を行うことがあります。また採択者が本共同研究による研究成果（コンテンツ）を学術論文誌や各種メディア等において公開する場合、本共同研究で開発作成を行ったことを明記してください。

学術情報メディアセンターコンテンツ作成共同研究企画委員会委員名簿

委員長　河原 達也（京都大学 学術情報メディアセンター 教授）
委　員　角所 考（関西学院大学 理工学部 教授）
菊池 誠（大阪大学 サイバーメディアセンター 教授）
黒江 康明（京都工芸繊維大学 大学院工芸科学研究科 教授）
柴山 守（京都大学 東南アジア研究所 教授）
杉万 俊夫（京都大学 大学院人間・環境学研究科 教授）
吉岡 洋（京都大学 大学院文学研究科 教授）
奥村 昭夫（京都大学 学術情報メディアセンター 客員教授）
土佐 尚子（京都大学 学術情報メディアセンター 特定教授）
美濃 導彦（京都大学 学術情報メディアセンター 教授）
椋木 雅之（京都大学 学術情報メディアセンター 准教授）
元木 環（京都大学 情報環境機構 助教）
赤坂 浩一（京都大学 情報部 情報基盤課 情報環境支援グループ）
小西 満（京都大学 情報部 情報基盤課 共同利用グループ）

京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

大型計算機システム利用負担金　（2009年10月1日より）

別表1　スーパーコンピュータシステム

			利用負担額	提供サービス					
コース	タイプ	セット		システム	バッチ	システム資源	経過時間（時間）	ディスク（GB）	利用者番号
エントリ	—	基本	12,600 円/年	Thin SMP	共有	最大1ノード相当（並列数16、メモリ32GB）	1	60	—
パーソナル	タイプ1	基本	100,000 円/年	Thin SMP	共有	最大2ノード相当（並列数32、メモリ64GB）	168	600	—
	タイプ2	基本	100,000 円/年	Fat SMP	共有	最大2ソケット相当（並列数8、メモリ64GB）	168	600	—
グループ	タイプ1	最小	250,000 円/年	Thin SMP	優先	2ノード（（16コア、メモリ32GB）×2）	336	2,000	6
		追加単位	250,000 円/年				—	2,000	6
	タイプ1B	最小	300,000 円/年	Thin SMP	準優先	4ノード（（16コア、メモリ32GB）×4）	336	2,400	12
		追加単位	150,000 円/年			2ノード（（16コア、メモリ32GB）×2）	—	1,200	6
	タイプ1C	最小	750,000 円/年	Thin SMP	占有	4ノード（（16コア、メモリ32GB）×4）	336	4,000	12
		追加単位	375,000 円/年			2ノード（（16コア、メモリ32GB）×2）	—	2,000	6
	タイプ2	最小	400,000 円/年	Fat SMP	優先	4ソケット(16コア、メモリ128GB)	336	4,000	12
		追加単位	200,000 円/年			2ソケット(8コア、メモリ64GB)	—	2,000	6
	タイプ2B	最小	240,000 円/年	Fat SMP	準優先	4ソケット(16コア、メモリ128GB)	336	2,400	12
		追加単位	120,000 円/年			2ソケット(8コア、メモリ64GB)	—	1,200	6
大規模ジョブ	タイプ1	最小	24,000 円/週(7日)	Thin SMP	優先	4ノード（（16コア、メモリ32GB）×4）	—	—	—
		追加単位	6,000 円/週(7日)			1ノード（16コア、メモリ32GB）	—	—	—
	タイプ2	最小	20,000 円/週(7日)	Fat SMP	優先	4ソケット(16コア、メモリ128GB)	—	—	—
		追加単位	5,000 円/週(7日)			1ソケット(4コア、メモリ32GB)	—	—	—
専用クラスタ	—	最小	750,000 円/年	Thin SMP	—	4ノード（（16コア、メモリ32GB）×4）	—	4,000	12
	—	追加単位	375,000 円/年			2ノード（（16コア、メモリ32GB）×2）	—	2,000	6
ライセンスサービス			20,000 円/年	可視化ソフト（AVS,ENVI/IDL)およびプリポストウェアの1ライセンスにつき					

備考

1. 利用負担額は、年度単位で算定している。また、総額表示である。
2. 大型計算機システムの全ての利用者は、上記表のサービスの他、次のサービスを受けることができる。
   1） 大判プリンタサービス
   2） その他、大型計算機システムが提供するサービス、機器の利用
3. 上記表の大規模ジョブコース、ライセンスサービスの申請には、大型計算機システムの利用者であることが必要である。
4. 「共有」: 当該カテゴリのユーザ間で一定の計算資源を共有するベストエフォートのスケジューリングを行う。
   「準優先」: 定常稼働状況において記載値（以上）の計算資源が確保されるように優先スケジューリングを行う。
   また、稼働状況によらず記載値の1/4の計算資源が確保されることを保証する。
   「優先」: 定常稼働状況において記載値（以上）の計算資源が確保されるように優先スケジューリングを行う。
   また、稼働状況によらず記載値の1/2の計算資源が確保されることを保証する。
   「占有」: 稼働状況によらず記載値（以上）の計算資源が確保されることを保証する。
5. ディスク容量はバックアップ領域（最大で総容量の1/2）を含む。
6. グループコース及び専用クラスタコースのシステム資源は、下記の負担額を支払うことにより増量することができる。
   なお増量は各月１日に実施し、増量した資源は当該年度末までの期間にわたって利用されるものとする。

コース	タイプ	追加負担金額（増量単位あたり）	システム資源増量単位	ディスク増量（GB）
グループ	タイプ1	25,000 円/月	2ノード（（16コア、メモリ32GB）×2）	2,000
	タイプ1B	15,000 円/月	2ノード（（16コア、メモリ32GB）×2）	1,200
	タイプ1C	37,500 円/月	2ノード（（16コア、メモリ32GB）×2）	2,000
	タイプ2	20,000 円/月	2ソケット(8コア、メモリ64GB)	2,000
	タイプ2B	12,000 円/月	2ソケット(8コア、メモリ64GB)	1,200
専用クラスタ	—	37,500 円/月	2ノード（（16コア、メモリ32GB）×2）	2,000

7. グループコース及び専用クラスタコースを通年でなく利用する場合には、下記の負担額を支払うものとする。
ただし、利用期間は当該年度内に限るものとする。

利用期間			3ヶ月	6ヶ月	9ヶ月
グループコース	タイプ1	最小	100,000 円	150,000 円	225,000 円
		追加単位	100,000 円	150,000 円	225,000 円
	タイプ1B	最小	120,000 円	180,000 円	270,000 円
		追加単位	60,000 円	90,000 円	135,000 円
	タイプ1C	最小	300,000 円	450,000 円	675,000 円
		追加単位	150,000 円	225,000 円	337,500 円
	タイプ2	最小	160,000 円	240,000 円	360,000 円
		追加単位	80,000 円	120,000 円	180,000 円
	タイプ2B	最小	96,000 円	144,000 円	216,000 円
		追加単位	48,000 円	72,000 円	108,000 円
専用クラスタ コース	—	最小	300,000 円	450,000 円	675,000 円
	—	追加単位	150,000 円	225,000 円	337,500 円

8. グループコース及び専用クラスタコースの利用者番号は利用者あたり年額5,000円を負担することで追加できる。
9. 機関・部局定額制度
他機関又は学内における部局(『国立大学法人京都大学の組織に関する規程』第3章第2節から第11節で定める組織をいう。)の組織が、その組織単位でグループコースサービス(年間)の利用を申請する場合、料金表(年間)に掲載額の1.5倍を利用負担金とする。なお、利用負担金額が150万円未満の場合は100人、150万円を超える場合は、150万円毎に100人までの利用者を認める。

別表2(汎用コンピュータシステム)

区分	利用負担額	単位
VMホスティングサービス	126,000円／年	1仮想マシンにつき
ホームページサービス	31,500円／年	1ドメイン名につき
個人向けホームページサービス	12,600円／年	1アカウントにつき
メール転送サービス	12,600円／年	1ドメイン名につき

備考
1．利用負担額は、総額表示である。
2．上記表の汎用コンピュータシステムのサービスを利用するためには、大型計算機システムの利用者であることが必要である。
3．ホームページサービス及びVMホスティングサービスにおいて、下記の負担額を支払うことによりオプションサービスを利用することができる。

オプションサービス種別	利用負担額	単位
データベース(Oracle)	63,000円／年	1アカウントにつき
ストリーミング(Helix Server)	31,500円／年	1アカウントにつき

4．VMホスティングサービスのシステム資源は、下記の負担額を支払うことにより増量することができる。

種別	利用負担額	単位
ディスク	10,500円／年	100GBにつき
システム資源	100,800円／年	1台につき

システム資源1台とは、CPU:2コア,メモリ:2GB である。

5．VMwareを用いたVMホスティングサービスは、下記の負担額を支払うことにより利用・増量することができる。ただし、システム資源が非常に限られているためサービスを提供できる場合が限定される。

種別	利用負担額	単位
標準機能サポート	25,200円／年	1仮想マシンにつき
ディスク	10,500円／年	100GBにつき
システム資源	201,600円／年	1台につき

システム資源1台とは、CPU:1コア,メモリ:2GB である。

6．利用負担額は、当該年度（4月から翌年3月まで）の利用に対して年額として算定するが、年度途中から利用を開始する場合には月数に応じて減額する。

別表3　スーパーコンピュータシステム(民間機関利用)

システム	システム資源	経過時間(時間)	ディスク(GB)	利用者番号	利用負担額
Thin SMP	4ノード((16コア、メモリ32GB)×4)	336	2,400	12	1,200,000 円/年
	6ノード((16コア、メモリ32GB)×6)	336	3,600	18	1,800,000 円/年
	8ノード((16コア、メモリ32GB)×8)	336	4,800	24	2,400,000 円/年

備考
1. 利用負担額は、年度単位で算定している。また、総額表示である。
2. ディスク容量はバックアップ領域(最大で総容量の1/2)を含む。
3. 通年でなく利用する場合には、下記の負担額を支払うものとする。
ただし、利用期間は当該年度内に限るものとする。

システム資源	利用期間		
	3ヶ月	6ヶ月	9ヶ月
4ノード	300,000 円	600,000 円	900,000 円
6ノード	450,000 円	900,000 円	1,350,000 円
8ノード	600,000 円	1,200,000 円	1,800,000 円

全国共同利用版広報・Vol.9(2010)総目次

[巻頭言]

[特集「センターのコンピューティング研究」]

[スーパーコンピュータ共同研究制度（若手研究者奨励枠）研究報告]

[プログラム高度化支援事業研究報告]

[スーパーコンピュータ共同研究制度（大規模計算支援枠）研究報告]

[2009 年度京都大学学術情報メディアセンターコンテンツ作成共同研究　研究報告]

[サービスの記録・報告]

[資料]

[編集後記]

― サービス利用のための資料一覧 ―

1. スーパーコンピュータシステム・ホスト一覧

- Thin SMP クラスタ：thin.kudpc.kyoto-u.ac.jp
  - Exceed onDemand 接続：thinX11.kudpc.kyoto-u.ac.jp
- Fat SMP クラスタ：fat.kudpc.kyoto-u.ac.jp

※ ホストへの接続は SSH(Secure Shell)のみ、telnet, ftp は不可

2. 問い合わせ先 & リンク集

- 情報環境機構のホームページ
  http://www.iimc.kyoto-u.ac.jp/

- 学術情報メディアセンターのホームページ
  http://www.media.kyoto-u.ac.jp/

- スーパーコンピュータシステムに関する問い合わせ先
  - 利用申請などに関する問合わせ先
    【共同利用支援グループ・共同利用担当（北館窓口）】
    E-mail：zenkoku-kyo@media.kyoto-u.ac.jp / Tel：075-753-7424 / Fax：075-753-7449
    URL: http://www.iimc.kyoto-u.ac.jp/ja/services/comp/
  - システムの利用など技術的な問い合わせ
    【研究支援グループ】
    E-mail：consult@kudpc.kyoto-u.ac.jp / Tel：075-753-7426
    URL: http://web.kudpc.kyoto-u.ac.jp/

- ホームページ・ホスティングサービスに関する問合わせ先
  【情報環境支援グループ】
  E-mail：whs-qa@media.kyoto-u.ac.jp / Tel：075-753-7494
  URL: http://www.iimc.kyoto-u.ac.jp/ja/services/whs/

- コンテンツ作成支援サービスに関する問合わせ先
  【コンテンツ作成室】
  E-mail：cpt@media.kyoto-u.ac.jp / Tel：075-753-9012
  URL: http://www.iimc.kyoto-u.ac.jp/ja/services/content/

編 集 後 記

先日、片付けのついでに昔使っていた天体望遠鏡を引っ張り出しました。保管状態が悪かったのか使い物にならなくなっていましたが、小さい頃なけなしの小遣いを貯めて買ったものだけにそっと元の場所に戻しておきました。そういえばあの頃は、お月さん・金星・火星・木星・土星と毎晩飽きもせず何時間も望遠鏡をのぞいていました。ズームアップすると、まるで自分が宇宙船に乗ってどんどんどんどんその近くまで飛んで行っているように思いながら。初めて見た時のあのお月さんのきれいだったこと。今考えると、たわいもないことをと思われますが、まだまだ夢や感動があったように思えます。ひるがえって今は、トホホ…、われながら情けない。

世捨て人

子供が保育園に通い始め、あっという間に 1 年が過ぎました。子供の病気は諸先輩に聞いていた通り多いのですが、それ以上に自分たちの病気が多くなったことに驚いています。普段、大人だけの環境では接しないようなウィルスにでも感染するのか、ひどい風邪で寝込むこともしばしばあります。また、病気にかかってから治るまでの日数は完全に 子供 ＜ 大人です。自分の体調が悪くても子供の相手をしたり、ある程度は家事をこなしたりするからと思っていますが、「若さの違いじゃない？」という意見もあり・・・。

保育園児の母

京都大学学術情報メディアセンター全国共同利用版広報　Vol. 10, No. 1

2011 年　6 月 30 日発行

編集者　京都大学学術情報メディアセンター
　　広報教育委員会・全国共同利用版広報編集部会
発行者　〒606-8501　京都市左京区吉田本町
　　京都大学学術情報メディアセンター
　　Academic Center for Computing and Media Studies
　　Kyoto University
　　Tel. 075-753-7400
　　http://www.media.kyoto-u.ac.jp/
印刷所　〒616-8102　京都市右京区太秦森ヶ東町21-10
　　株式会社 エヌジーピー
　　http://www.nextgp.jp