数理解析研究所講究録 1395 巻 2004 年 138-143

OpenXM の新サーバ, 新プロトコル

OpenXM 開発グループ
(岩根, 小原, 野呂, 高山)
OpenXM committers
openxm.org

1 OpenXM の目的と歴史

OpenXM は • 数学での並列計算, • 数学ソフトウエアの統合化 または Conglomerate 化 (A.Solomon) • 数学的知識のマネージメント (Mathematical Knowledge Management) • 実際に数学の研究や数学の応用に使えるパッケージの開発, などを目標に数年前からパッケージの開発, プロトコルの開発をしている. この小文では講演に用いたスライドを元に OpenXM の新しいサーバおよび新しい通信プロトコルを紹介する.

OpenXM の歴史

• OpenXM 1.1.1 (January 24, 2000): 最初の実験版.

• OpenXM 1.1.2 (March 20, 2000): とりあえず使える版.

• OpenXM 1.1.3 (September 26, 2000): 1.1 系の最終版. OpenXM RFC 100 形式のプロセス木. 1077 個の数学関数を提供. 提供しているサーバは `ox_asir`, `ox_sm1`, `ox_phc`, `ox_gnuplot`, `ox_m2`, `ox_tigers`, `ox_math`(ematica), `OMproxy`.

• OpenXM 1.2.1 (March 2, 2002): Cygwin (Windows) への対応開始. マニュアル自動生成 (gentexi) など.

• OpenXM 1.2.2 (May 13, 2003): RFC 103 (部分的), autoconf

(•) Digital formula book, OpenMath, tfb, CD (hypergeo*, weylalgebra*, intpath*), 数値計算.

2 新しいサーバ ntl

http://www.shoup.net/ntl, NTL is a high-performance, portable C++ library providing data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields.

ox_toolkit + NTL ライブラリ ⇒ ox_ntl

```
[1145]PID=ox_launch_nox(0,
       "/home/taka/OX4/OpenXM/bin/ox_ntl");
[1028] F=ntl.ex_data(4)$
[1029] F = F * subst(F, x, x + 1)$
[1030] ntl.factor(PID, F);
/* NTL の knapsac factorizer */
[[1,1],
 [x^16+16*x^15- ... , 1]
 [x^16-136*x^14+ ..., 1]]
```

実装により明らかになったこと: ox_toolkit の有効性, ox_toolkit の問題点の改善, LLL をよぶために CMO array も必要.

この機能に関する ChangeLog

OpenXM/src/ox_ntl の下の全てのファイル (2003 年 12 月まで). OpenXM/src/ox_toolkit の下の次のファイル. • cmo.c 1.17, ox.c 1.13–1.26, ox_toolkit.h 1.24–1.27.

開発: 岩根 (小原)

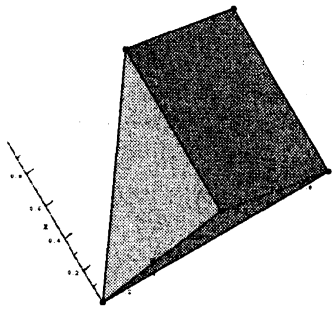
3 新しいサーバ polymake と oxshell

Polymake : 多面体用ソフトウエア. `http://www.math.tu-berlin.de/polymake`, by Ewgenji Gawrilow, Michael Joswig.
Facets, Convex hull, Triangulation, 構成, polytope data, Visualization by JavaView and Povray.

Example: 点 (1,0,0), (1,1,0), (1,0,1), (1,1,1) 上の cone の facet を求めよ.

`polymake` ではつぎような入力ファイル `square.poly` をまず作成する.

```
POINTS
1 0 0
1 1 0
1 0 1
1 1 1
```



入力: polymake square.poly FACETS
結果:

```
FACETS
0 0 1
0 1 0
1 0 -1
1 -1 0
```

3.1 Polymake の OX 100, 101 サーバ化で何が問題であったか?

新しい OX RFC 100 準拠のサーバを作る場合にはターゲットとする数学ソフトウエアシステムのソースコードに OX RFC 100 対応部分を加える作業をしないといけない. この作業はソースコードの理解とかなりの手間を要する. C++ で書かれた polymake のソースは先端的な C++ の機能を利用しておりコンパイルが容易でない.

Polymake はあらかじめファイルにコマンドやデータを書き出しておくことにより unix の shell から利用可能なように作られている: このようなアプリケーションを バッチ処理対応アプリケーションと呼ぶ. サーバとの通信の頻度は問題によるがある程度大きい計算の場合は通信時間は無視できる. Polymake は対話的利用は想定しておらずサーバの計算を中断して, 再度開始するなどの必要がない. したがって **system** 関数で polymake を呼び出して, 結果が書き出されたファイルを解析すればよい. しかしこの **system** 関数は使いにくい. 問題点はファイルシステムや OS の違いを吸収できない, プロセス管理のやりにくさなどである. 我々は **system** の強化版として oxshell を導入する.

3.2 Oxshell とは

Oxshell は上記のようなバッチ処理対応アプリケーションをソースコードの改変なく OX RFC 100 準拠にするためのいわゆるラッパー型の OX サーバを書くための補助機能を提供する sm1 への組み込み関数である.

Oxshell を用いて OX サーバを実現するのが適切な数学ソフトウエアは以下のような特徴をもつものであろう.

• バッチ処理対応アプリケーションである. • ソースコードの変更ができないか困難. • サーバとの通信が頻繁におきない. • サーバの計算を中断して, 再度開始するなどの必要がない. • Windows でも unix でも動かしたい.

3.3 OX RFC 100 の復習

クライアント-サーバモデル. サーバは stackmachine. (1) スタックマシンへのメッセージを送ることにより計算が進行する. (2) 計算の中断などのメッセージもある. (3) サーバはエンジンとコントロールプロセスよりなる.

OX RFC 100 には ファイルの概念がない. したがって,

polymake ファイル名 動作

みたいなプログラムは OX スタックマシンの上のデータをファイルに書き出してから実行して, それからまたスタックマシンの上のデータに変換しないといけない. (単純仕事, しかし, ZPG の極致; unix と windows 違い, /bin/sh の存在, プログラムが大変読みにくく保守もしにくい)

oxshell ではこの仕事は次の 1 行で書く.

[(polymake) (stringInOut://スタックマシン変数名.poly) 動作] oxshell

スタックマシンの変数をファイルをみなしている.

このような考え方は, なにも新しいものではない.

例: Java の io.StringReader
(io.InputStreamReader の代わり).

例: スクリプト言語である Python

```
x=os.popen("abc","r").read();
```

新しい部分: この考えを数学ソフトウエアの統合化のために使うための実装は存在していなかった. こ

れが oxshell コマンドの導入によって解決される点の一つである. 考え方が整理されプログラムの保守性が大変よくなり, また新しいバッチ型アプリケーションの OX サーバ化が楽になった.

Example: 文字列 S に格納された単語 dog, cat, lion を unix のツール sort で並べ変えるには次のように書く. 入力は oxshell を実行しているスタックマシンの変数 S に格納されている.

```
[1163]:= S="dog\ncat\nlion\n";
[1164]:= oxshell.set_value("S",S);
[1165]:= T=oxshell.oxshell(
          ["sort","stringIn://S"]);
[cat
dog
lion
,]
```

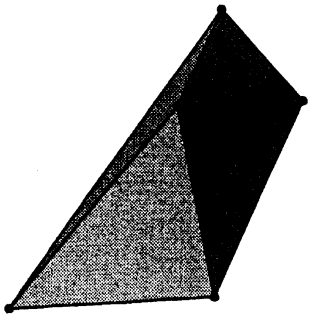
3.4 oxshell.facets() の実装, 数学関数の実装例

• データ変換には tfb/2 (田村)format を利用, OpenMath アーキテクチャを利用. 成果: その有効性を実証できた.)

```
[$polymake.data(
    polymake.POINTS([[1,0,0],[1,1,0],[1,0,1],[1,1,1]]),
  polymake.FACETS([[0,0,1],[0,1,0],[1,0,-1],[1,-1,0]]),
  polymake.AFFINE_HULL())$
 CMO tree expression of the data above
 Outputs to stdout and stderr ]
```

taka_ahg.b(A,Idx)(OpenXM/src/asir-contrib/packages/src/taka_ahg.rr) oxshell による polymake 呼び出し機能を用いた数学関数. この関数は 行列 A に付随した 方向 Idx のある b 関数を計算する. これは A の定義する cone のファセットをあらわす一次式の積で表現されることが, 知られている (Saito, Mutsumi; Sturmfels, Bernd; Takayama, Nobuki; Hypergeometric polynomials and integer programming. Compositio Math. 115 (1999), no. 2, 185–204 および Saito, Mutsumi; Parameter shift in normal generalized hypergeometric systems. Tohoku Math. J. (2) 44 (1992), no. 4, 523–534 をみよ.)

```
[1163]:= load("oxshell.rr");
[1164]:= load("taka_ahg.rr");
[1165]:=  A=[[1,0,0],[1,1,0],[1,0,1],[1,1,1],[1,2,0]];
[1166]:= fctr(taka_ahg.b(A,0,[s1,s2,s3]));
[[1,1],[s1-s3,1],[2*s1-s2-s3,1],[2*s1-s2-s3-1,1]]
```



```
def b(A,Idx,V) {
  F = oxshell.facets(A);
  /* Computing all the facets by polymake server.*/
  F = F[1];        /* F is the set of the facets */
  P = A[Idx];
  Nfacets = length(F);
  F = toPrimitive2(P,F);
  /* Translate into primitive supporting function */
  Bf = 1;
  for (I=0; I<Nfacets; I++) {
    H = matrix_inner_product(P,F[I]);
    if (H != 0) {
      B = matrix_inner_product(P,V);
      for (J=0; J<H; J++) {
        Bf = Bf*(B-J);  /* See the papers above. */
      }
    }
  }
  return(Bf);
}
```

この機能に関する ChangeLog

OpenXM/src/kan96xx の下の次の各ファイル. • Doc/oxshell.oxw 1.1–1.2. • trans の下のファイルすべて (2003 年 12 月). • Kan/kanExport0.c 1.19–1.20, option.c 1.13, primitive.c 1.10–1.11, shell.c 1.1–1.9, stackmachine.c 1.13–1.14, kclass.h 1.4–1.5. • kclass/tree.c 1.1–1.7, tree.hh 1.1–1.2.
OpenXM/src/k097 の下の次の各ファイル. • help.k 1.11–1.12, ki.c 1.16, ox_k0.c 1.5, slib.k 1.11, slib.sm1 1.10.
OpenXM/src/asir-contrib/packages/src の下の oxshell.rr 1.1–1.3.
OpenXM/src/util の下の ox_pathfinder.c 1.8–1.17.

開発: 高山

4 OX-RFC 103 (100, 101 補遺)

この RFC は OpenXM RFC 100 (および 101) の実装により明らかになった種々の問題点をもとに RFC 100 プロトコルへの幾つかの追加を提案する.

- ファイルへの読み書き
- 新しい CMO
- エンジン認証手続き
- 中断および変数の伝播

1. 自分の子供プロセスをすべてリストする (asir の場合:`ox_get_serverinfo()`)
2. 子供プロセスに順番に OpenXM-RFC 100 の中断メッセージを送る.

ox103_set_shared_variable(CMO string `Name`, CMO object `value`) エンジンスタックマシンの変数 `Name` に値 `value` が設定, さらに子どものプロセスすべてのエンジン関数 `ox103_set_shared_variable` をよぶ.

この機能に関する ChangeLog

OpenXM/src の下の次のファイル.

asir-contrib/packages/src/oxrfc103.rr 1.1–1.4, kan96xx/Doc/oxrfc103.sm1 1.1–1.3, util/ox_pathfinder.c 1.17.

開発: 高山

5 OX-RFC 102 – 本格的なサーバ間通信を用いた分散計算

OX-RFC-100, 101 : master-server 間通信を用いた分散計算

OX-RFC-102 : server-server 間通信

目標 : 本格的分散並列計算を可能にすること

応用例

- broadcast を効率化する N 個の server への broadcast が $O(\log_2 N)$ でできる
- LU 分解の分散並列計算 ScaLAPACK 風に, 行列を分散保持して並列計算

仕様

MPI-2 の, 動的プロセス生成, プロセスグループ間 broadcast の仕様を参考にする. `http://www-unix.mcs.anl.gov/mpi`

新しい点

OX RFC 100 を前提 ⇒ データ型の管理の必要がない, 100 の中断プロトコルの拡張, など MPI より容易に利用可能.

5.1 server の起動, server 間通信路の開設

server は OX RFC-100, 101 により起動する. この通信路に以下の SM コマンドを送る.

- `SM_set_rank_102` *nserver rank*

  server に, グループ内の server の総数 *nserver* と, その中での識別子 *rank* ($0 \leq rank \leq nserver$) を通知する.

- `SM_tcp_accept_102` *port peer*

  ポート番号 *port* の TCP ポートで, bind, listen, accept を実行して connect を待つ. 通信が成立したら, byte order negotiation を行い, 相手先テーブルに登録する.

- `SM_tcp_connect_102` *peerhost port peer*

  ホスト *peerhost* のポート番号 *port* の TCP ポートに connect する. 通信が成立したら, byte order negotiation を行い, 相手先テーブルに登録する.

5.2 server 間通信, broadcast, reduction

server 間通信は, 相互の信頼に基づき行う — 送り手が送信したら, 受け手はちゃんと受信動作に入ること

データは OX タグ付きで — `OX_SYNC_BALL` による通信路リセットに必要

以下の SM コマンドは collective 操作である. すなわち, 同一引数でグループ内の全ての server で実行されなければいけない.

- `SM_bcast_102` *root*

  識別子 *root* の server のスタック上のデータを pop し, グループ内に broadcast する. 各 server

のスタックに broadcast されたデータが push される.

- SM_reduce_102 *root opname*

  各 server のスタック上のデータが pop され, *opname* で指定される二項演算 (結合則が必要) を順に行い, 結果を *root* で指定される server のスタックに push する. 他の server には 0 が push される.

5.3 broadcast の手続き

SM_bcast_102 の実行

root = 0 で, 識別子が $b2^k$ (b は奇数) の server の動作

data ← 識別子が $(b-1)2^k$ の server からのデータ
for $i = k-1$ down to 0
  識別子が $b2^k + 2^i$ の server に *data* を送信
end for

2 で割り切れる回数が多い識別子を持つ server が先にデータ送信

⇒ デッドロックにならない

独立なペアどうしの通信が同時に行えるなら、高々 $\lceil \log_2 N \rceil$ ステップ (N は server の総数) で broadcast 完了.

5.4 reduction の手続き

SM_reduce_102 の実行

server 数 N, *root* = 0 で, 識別子が b の server の動作

手持ちのデータを *data* とする

for $i = 0$ to $\lfloor \log_2 N \rfloor$
  if (b に 2^i の bit がある) then
    識別子 $b - 2^i$ の server に *data* を送信して終了
  else if ($b + 2^i < N$) then
    $data_0$ ← 識別子 $b + 2^i$ の server からのデータ
    data ← *data* と $data_0$ の二項演算結果
  end if
end for

この場合も, 独立なペアどうしの通信が同時に行えるなら、高々 $\lceil \log_2 N \rceil$ ステップ (N は server の総数) で reduction 完了. 結果は root に残る.

5.5 broadcast 時のデータの流れ

$N = 16$, *root* = 0 の場合

step 1	step 2	step 3	step 4
0 → 8	0 → 4	0 → 2	0 → 1
	8 → 12	8 → 10	8 → 9
		4 → 6	4 → 5
		12 → 14	12 → 13
			2 → 3
			10 → 11
			6 → 7
			14 → 15

reduction の場合, データの流れは逆になる (step 4 → step 1, 矢印が逆)

5.6 エラー処理

master-server 間通信路は, OX RFC-100 で規定されている.

server-server 間通信路を空にするための, 識別子 i の server での操作

for $j = 0$ to $i-1$ do
  do
    data ← 識別子 j の server からの OX データ
  while *data* ≠ OX_SYNC_BALL
end for
for $j = i+1$ to *nserver* − 1 do
  OX_SYNC_BALL を 識別子 j の server に送信
end for

master-server リセット後 : 各 server はコマンド待ち状態

⇒ 次の SM コマンドを各 server に送信すればよい

- SM_reset_102 (引数なし, collective)

5.7 Asir (master) 上での API

- ox_set_rank_102(*Server*,*Nserver*,*Rank*)

  Server に SM_set_rank_102 を送る.

- ox_tcp_accept_102(*Server*,*Port*,*Rank*)

  Server に SM_tcp_accept_102 を送る.

- ox_tcp_connect_102(*Server*,*Host*,*Port*,*Rank*)

  Server に SM_tcp_connect_102 を送る.

- ox_reset_102(*Server*) (collective)

  Server に SM_reset_102 を送る

5.8 Asir (server) 上での API

- ox_send_102(*Rank*,*Data*)

  識別子 *Rank* の server に *Data* を OX データとして送信する. 識別子 *Rank* の server は対応する受信を開始しなければならない.

- ox_recv_102(*Rank*)

  識別子 *Rank* の server から OX データを受信する. 識別子 *Rank* の server は対応する送信を開始しなければならない.

- ox_bcast_102(*Root*[,*Data*]) (collective)

  識別子 *Root* の server を root として, グループ内で broadcast する. *Data* が指定された場合, スタックにプッシュされる. 識別子が *Root* に等しい server で, スタックからデータがポップされ, そのデータが, 各呼び出しの戻り値となる.

- ox_reduce_102(*Root*,*Operation*[,*Data*]) (collective)

  グループ内の各 server のスタックからポップしたデータに対し *Operation* で指定される二項演算を行い, 結果を *Root* で指定される server での関数呼び出しの戻り値として返す. *Data* が指定された場合, スタックにプッシュしてから上記の操作を実行する. *Root* 以外の server での戻り値は 0 である.

5.9 実行例 : 一変数多項式の積 (master 側)

```
def d_mul(F1,F2)
{
    Procs = getopt(proc);
    /* process 指定がない場合には, 自分で計算 */
    if ( type(Procs) == -1 ) return umul(F1,F2);
    if ( !var(F1) || !var(F2) ) return F1*F2;
    NP = length(Procs);
    /* 引数を server 0 に送る */
    ox_push_cmo(0,[F1,F2]);
    /* 各 server に, server 0 を root として仕事を始めるよう依頼 */
    for ( I = 0; I < NP; I++ )
        ox_cmo_rpc(I,"d_mul_main",0);
    /* server 0 から結果を受け取る */
    R = ox_pop_cmo(0);
    return R;
}
```

5.10 実行例 : 一変数多項式の積 (server 側; collective)

```
def d_mul_main(Root)   /* Shoup's algorithm */
{
    /* server の総数, 自分の id を知る */
    Id = ox_get_rank_102(); NP = Id[0]; Rank = Id[1];
    /* Root にある引数を broadcast で共有 */
    Arg = ox_bcast_102(Root); F1 = Arg[0]; F2 = Arg[1];
    L = setup_modarrays(F1,F2,NP);
    Marray = L[0]; MIarray = L[1]; M = L[2];
    R = umul_chrem(F1,F2,MIarray[Rank],Marray[Rank],M);
    Arg = 0; F1 = 0; F2 = 0;
    /* 各 server の結果を足し合わせて, Root に置く */
    R = ox_reduce_102(Root,"+",R);
    /* Root のみ, 結果を normalize する */
    if ( Rank == Root )
        R = uadj_coef(R%M,M,ishift(M,1));
    return R;
}
```

この機能に関する ChangeLog

OpenXM_contrib2/asir2000/ の下の次の各ファイル.

- lib/dmul102, 1.1–1.2.
- include/ox.h, 1.18–1.19.
- parse/glob.c, 1.41.
- io/tcpf.c 1.43–1.50, ox.c 1.21–1.24, ox_asir.c 1.45–1.50.

開発: 野呂