# OpenXM — an Open System to Integrate Mathematical Softwares

Masahide Maekawa
Kobe University
maekawa@math.kobe-u.ac.jp

Masayuki Noro
Fujitsu Labs
noryo@flab.fujitsu.co.jp

Katsuyoshi Ohara
Kanazawa University
ohara@air.s.kanazawa-u.ac.jp

Yukio Okutani
Kobe University
okutani@math.kobe-u.ac.jp

Nobuki Takayama
Kobe University
takayama@math.kobe-u.ac.jp

Yasushi Tamura
Kobe University
tamura@math.kobe-u.ac.jp

## ABSTRACT

OpenXM is a free, or Open Source, infrastructure for mathematical softwares. It provides methods and protocols for interactive distributed computation and for integrating mathematical softwares. OpenXM package is a set of softwares that supports OpenXM protocols. It is currently a collection of softwares `Risa/Asir` [8], `Kan/sm1` [16], `PHC` pack [17], `GNUPLOT`, `Mathematica` interface, and `OpenMath`/XML [1] translator. These are wrapped with the OpenXM stack machine to connect each other. *Availability*: The OpenXM package is obtainable at [12] from January 24, 2000.

## KEYWORDS

Asir, IAMC, Interactive Distributed Computation, Integration of Mathematical Softwares, MP, OpenMath, OpenXM.

## 1 DESIGN OUTLINE

As Schefström clarified in [14], integration of tools and softwares has three dimensions: data, control, and user interface.

Data integration concerns with the exchange of data between different softwares or same softwares. OpenMath [1] and MP (Multi Protocol) [3] are, for example, general purpose mathematical data protocols. They provide standard ways to express mathematical objects. For example,

```
<OMOBJ>  <OMI> 123 </OMI> </OMOBJ>
```

means the (OpenMath) integer 123 in OpenMath/XML expression.



Control integration concerns with the establishment and management of inter-software communications. Control involves, for example, a way to ask computations to other processes and a method to interrupt computations on servers from a client. RPC, HTTP, MPI, PVM are regarded as a general purpose control protocols or infrastructures. MCP (Mathematical Communication Protocol) by Wang [18] is such a protocol for mathematics.

Although data and control are orthogonal to each other, real world requires both. NetSolve [7], OpenMath+MCP, MP+MCP [18], and MathLink [19] provide both data and control integration. Each integration method has their own features determined by their own design goals. OpenXM (Open message eXchange protocol for Mathematics) is a project aiming to integrate data, control and user interfaces with design goals motivated by the followings.

1. Noro has been involved in the development of a computer algebra system Risa/Asir [8]. An interface for interactive distributed computations was introduced to Risa/Asir in 1995. The model of computation was RPC (remote procedure call). A robust interruption protocol was provided by two communication channels like the File Transfer Protocol (ftp). As an application of this protocol, a parallel speed-up was achieved for a Gröbner basis computation to determine all odd order replicable functions (Noro and McKay [9]). However, the protocol was local in Asir and we thought that we should design an open protocol.

2. Takayama has developed a special purpose system Kan/sm1 [16], which is a Gröbner engine for the ring of differential operators $D$. In order to implement algorithms in $D$-modules due to Oaku (see, e.g., [13]), factorizations and primary ideal decompositions are necessary. Kan/sm1 does not have an implementation for these and called Risa/Asir as a UNIX external program. This approach was not satisfactory. Especially, we could not write a clean interface code between these two systems. We thought that it is necessary to provide a data and control protocol for Risa/Asir to work as a server of factorization and primary ideal decomposition.

3. We have been profited from increasing number of mathematical softwares. These are usually "expert" systems in one area of mathematics such as ideals, groups, numbers, polytopes, and so on. They have their own in-

terfaces and data formats, which are fine for intensive users of these systems. However, a unified system will be more convenient for users who want to explore a new area of mathematics with these softwares or users who need these systems only occasionally.

4. We believe that an open integrated system is a future of mathematical softwares. However, it might be just a dream without realizability. We want to build a prototype of such an open system by using existing standards, technologies and several mathematical softwares. We want to see how far we can go with this approach.

Motivated with these, we started the OpenXM project with the following fundamental architecture.

1. Communication is an exchange of messages. The messages are classified into three types: DATA, COMMAND, and SPECIAL. They are called OX (OpenXM) messages. Among the three types, *OX data messages* wrap mathematical data. We use standards of mathematical data formats such as OpenMath and MP as well as our own data format *CMO* (*Common Mathematical Object format*).

2. Servers, which provide services to other processes, are stack machines. The stack machine is called the *OX stack machine*. Existing mathematical softwares are wrapped with this stack machine. Minimal requirements for a target software wrapped with the OX stack machine are as follows:

   (a) The target must have a serialized interface such as a character based interface.

   (b) An output of the target must be understandable for computer programs; it should follow a grammar that can be parsed with other softwares.

3. Any server may have a hybrid interface; it may accept and execute not only stack machine commands, but also its original command sequences. For example, if we send the following string to the `ox_asir` server (OpenXM server based on Risa/Asir)
   `" fctr(x^100-y^100); "`
   and call the stack machine command
   `SM_executeStringByLocalParser`
   then the server executes the asir command
   `fctr(x^100-y^100);` (factorize $x^{100} - y^{100}$ over $\mathbf{Q}$)
   and pushes the result onto the stack.

OpenXM package is implemented on above fundamental architecture. For example, the following is a command sequence to ask $1+1$ from the Asir client to the `ox_sm1` server:

```
P = sm1_start();
ox_push_cmo(P,1); ox_push_cmo(P,1);
ox_execute_string(P,"add"); ox_pop_cmo(P);
```

Here, `ox_sm1` is an OpenXM server based on Kan/sm1.

The OpenXM package is implemented on the OpenXM for TCP/IP, which uses the client-server model. The OpenXM on MPI [6] is currently running on Risa/Asir as we will see in Section 8.2. In this paper, we discuss only on systems for TCP/IP to concentrate on the core part of our design.

## 2 OX MESSAGES

An OX message for TCP/IP is a byte stream consisting of a header and a body.

| Header | Body |
|---|---|

The header consists of two signed 32 bit integers. The first one is an OX tag and the second one is a serial number of the OX message. Several byte orders including the network byte order are allowed and the byte order is determined as a part of the establishment of a connection. See Section 5 for details.

The OX messages are classified into three types: DATA, COMMAND, and SPECIAL. We have currently the following general tags for the OX messages.

```
#define OX_COMMAND                 513  // COMMAND
#define OX_DATA                    514  // DATA
#define OX_SYNC_BALL               515  // SPECIAL
#define OX_DATA_WITH_LENGTH        521  // DATA
#define OX_DATA_OPENMATH_XML       523  // DATA
#define OX_DATA_OPENMATH_BINARY    524  // DATA
#define OX_DATA_MP                 525  // DATA
```

A new OX tag may be added. The new tag should be classified into DATA or COMMAND. For example, `OX_DATA_ASIR_LOCAL_BINARY` was added recently to send internal serialized objects of Asir via the OpenXM protocol. This is a tag classified to DATA. See the web page of OpenXM to add a new tag [12].

## 3 DATA FORMAT

OpenXM admits multiple mathematical encodings such as OpenMath, MP, CMO (Common Mathematical Object format). OpenXM itself does not exhibit a bias towards a particular encodings as a main mathematical data carrier and an OpenXM compliant system do not need to implement all possible data formats. However they should at least implement seven primitive data types of the CMO, which are necessary to carry several control informations such as a *mathcap*. Mathcap is a list of supported CMO's, OpenXM stack machine codes, and necessary extra informations. If a program sends an OX messages unknown to its peer, an unrecoverable error may occur. By exchanging mathcaps, a program knows its peer's capability and such an error can be avoided. Mathcap is also defined as a CMO. See [10] for the details.

Encoding types of OX data are distinguished with tags of OX messages. For example, an OX message with the tag `OX_DATA` is followed by a CMO packet. An OX message with the tag `OX_DATA_OPENMATH_XML` is followed by an OpenMath XML string.

Let us explain the data format of CMO. Any CMO packet consists of a header and a body. The size of the header is 4 bytes that tags the data type of the body. Data type tags are signed 32 bit integers which is called *int32* in this paper. Following tags are registered in the OpenXM.

```
#define CMO_ERROR2                        0x7f000002
#define CMO_NULL                          1
#define CMO_INT32                         2
```

```
#define CMO_DATUM                            3
#define CMO_STRING                           4
#define CMO_MATHCAP                          5
#define CMO_LIST                             17

#define CMO_MONOMIAL32                       19
#define CMO_ZZ                               20
#define CMO_QQ                               21
#define CMO_ZERO                             22
#define CMO_DMS_GENERIC                      24
#define CMO_DMS_OF_N_VARIABLES               25
#define CMO_RING_BY_NAME                     26
#define CMO_RECURSIVE_POLYNOMIAL             27
#define CMO_LIST_R                           28
#define CMO_INT32COEFF                       30
#define CMO_DISTRIBUTED_POLYNOMIAL           31
#define CMO_POLYNOMIAL_IN_ONE_VARIABLE       33
#define CMO_RATIONAL                         34
#define CMO_64BIT_MACHINE_DOUBLE             40
#define CMO_ARRAY_OF_64BIT_MACHINE_DOUBLE    41
#define CMO_BIGFLOAT                         50
#define CMO_IEEE_DOUBLE_FLOAT                51
#define CMO_INDETERMINATE                    60
#define CMO_TREE                             61
#define CMO_LAMBDA                           62
```

The first seven primitive types should be implemented on all OpenXM compliant systems. The formats are as follows.

| | | |
|---|---|---|
| *int32* `CMO_ERROR2` | *CMObject* ob | |
| *int32* `CMO_NULL` | | |
| *int32* `CMO_INT32` | *int32* n | |
| *int32* `CMO_DATUM` | *int32* n | *byte* data[0] |
| ... | *byte* data[n-1] | |
| *int32* `CMO_STRING` | *int32* n | *byte* data[0] |
| ... | *byte* data[n-1] | |
| *int32* `CMO_MATHCAP` | *CMObject* ob | |
| *int32* `CMO_LIST` | *int32* n | *CMObject* ob[0] |
| ... | *CMObject* ob[n-1] | |

As to the formats of other CMO's, see [10].

When one wants to implement CMO on a server, the person proceeds as follows.

1. Look for the list of CMO's at the web cite [12]. If there is a CMO that fits to one's requirement, then use this CMO.

2. If there is no suitable CMO, design a new CMO and register the new CMO to [12] with a description and examples.

# 4 OPENXM STACK MACHINES

In OpenXM specification, all servers are stack machines. When a server ox_xyz gets an OX data message, it translates the data into a local object of ox_xyz and pushes the object onto the stack. The translation scheme is called the *PhraseBook* of ox_xyz (cf. OpenMath [1]). For example, the Asir command `ox_push_cmo(P,1)` (push integer 1 onto the server $P$) sends an OX data message `(OX_DATA,(CMO_ZZ,1))` to the server $P$. Here, `OX_DATA` stands for `OX_DATA` header and `(CMO_ZZ,1)` is a body standing for 1 expressed by the CMO. The server translates `(CMO_ZZ, 1)` to its internal object of the integer 1 and pushes the object onto the stack.

If the server gets an *OX command* message, then the server executes the command. Any OX command message starts with the int32 tag OX_COMMAND. The body is a stack machine operation code expressed by int32. The codes are listed below [10].

```
#define SM_popSerializedLocalObject                 258
#define SM_popCMO                                   262
#define SM_popString                                263
#define SM_mathcap                                  264
#define SM_pops                                     265
#define SM_setName                                  266
#define SM_evalName                                 267
#define SM_executeStringByLocalParser               268
#define SM_executeFunction                          269
#define SM_beginBlock                               270
#define SM_endBlock                                 271
#define SM_shutdown                                 272
#define SM_setMathCap                               273
#define SM_executeStringByLocalParserInBatchMode    274
#define SM_getsp                                    275
#define SM_dupErrors                                276
#define SM_control_kill                            1024
#define SM_control_to_debug_mode                   1025
#define SM_control_exit_debug_mode                 1026
#define SM_control_reset_connection                1030
```

OpenXM does not have a standard for mathematical operation sets while it is a work in progress in the GAP group [5]. Each OpenXM server has its own set of mathematical operations, which are performed as follows. First, arguments for a mathematical operation and the number of the arguments are pushed. Second, the mathematical operator name, such as `fctr` (the factorization command of Asir), is pushed as a string. Finally, the stack machine command `SM_executeFunction` (269) evaluates the operator and pushes the result onto the stack after poping the operator name, the number of arguments and arguments. For example, the following code factorizes $x^{100}-1$ by calling `ox_asir` from Asir.

```
P = ox_launch();
ox_push_cmo(P,x^100-1); ox_push_cmo(P,ntoint32(1));
ox_push_cmo(P,"fctr");  ox_push_cmd(P,269);
Ans = ox_pop_cmo(P);
```

When an error has occurred on an OpenXM server, an error object is pushed onto the stack instead of a result of the computation. The error object consists of the serial number of the OX message which caused the error, and an error message.

```
[340] P = ox_launch()$
[341] ox_rpc(P,"fctr",1.2*x)$
[342] ox_pop_cmo(P);
error([8,fctr : invalid argument])
```

OpenXM server won't send error messages to the client except when it receives a `SM_pop*` command. OX stack machines work in the asynchronous mode which is similar to X servers. For servers of graphic and sound applications, it may be an advantageous feature. It is also possible to emulate RPC and a web server for MCP [18] on our asynchronous OX stack machines.

# 5 SESSION MANAGEMENT

In this section we explain our control integration in OpenXM. We assume that various clients and servers establish connections dynamically and communicate to each other. Therefore it is necessary to give a dynamical and unified method to start servers and to establish connections. In addition to that, interruption of execution and debugging facilities are necessary for interactive distributed computation.

## 5.1 Invocation of servers

An application called *launcher* is provided to start servers and to establish connections as follows.

1. A launcher is invoked from a client. When the launcher is invoked, the client informs the launcher of a port number for TCP/IP connection and the name of a server.
2. The launcher and the client establish a connection with the specified port number. One time password may be used to prevent launcher spoofing.
3. The launcher creates a process and executes the server after setting the data channel appropriately.

After finishing the above task as a launcher, the launcher process acts as a control server and controls the server process created by itself. As to the details of the control server see Section 5.2.

As the data channel is used to exchange binary data, the byte order conversion is necessary when a client and a server have different byte orders. It is determined by exchanging the preferable byte order of each peer. If the preference does not coincide with each other, then the network byte order is used. This implies that all servers and clients should be able to handle the network byte order. Nevertheless it is necessary to negotiate the byte order to skip the byte order conversion because its cost is often dominant over fast networks.

## 5.2 Control server

In OpenXM we adopted the following simple and robust method to control servers.

An OpenXM server has logically two I/O channels: one for exchanging data for computations and the other for controlling computations. The control channel is used to send commands to control execution on the server. The launcher introduced in Section 5.1 is used as a control process. We call such a process a *control server*. In contrast, we call a server for computation an *engine*. As the control server and the engine runs on the same machine, it is easy to send a signal from the control server. A control server is also an OpenXM stack machine and it accepts `SM_control_*` commands to send signals to a server or to terminate a server.

## 5.3 Resetting an engine

A client can send a signal to an engine by using the control channel at any time. However, I/O operations are usually buffered, which may cause troubles. To reset an engine safely the following are required.

1. Any OX message must be a synchronized object in the sense of Java.

   As an OX message is sent as a combination of several `CMO` data, a global exit without sending all may generate broken data.

2. After restarting an engine, a request from a client must correctly corresponds to the response from the engine.

   An incorrect correspondence occurs if some data remain on the stream after restarting an engine.

`SM_control_reset_connection` is a stack machine command to initiate a safe resetting of an engine. The control server sends `SIGUSR1` to the engine if it receives `SM_control_reset_connection` from the client. Under the OpenXM reset protocol, an engine and a client act as follows.

*Client side*

1. After sending `SM_control_reset_connection` to the control server, the client enters the resetting state. It discards all `OX` messages from the engine until it receives `OX_SYNC_BALL`.
2. After receiving `OX_SYNC_BALL` the client sends `OX_SYNC_BALL` to the engine and returns to the usual state.

*Engine side*

1. After receiving `SIGUSR1` from the control server, the engine enters the resetting state. The engine sends `OX_SYNC_BALL` to the client. The operation does not block because the client is now in the resetting state.
2. The engine discards all OX messages from the engine until it receives `OX_SYNC_BALL`. After receiving `OX_SYNC_BALL` the engine returns to the usual state.

Figure 1 illustrates the flow of data. `OX_SYNC_BALL` is used to mark the end of data remaining in the I/O streams. After reading it, it is assured that each stream is empty and that the subsequent request from a client correctly corresponds to the response from the engine. We note that we don't have to associate `OX_SYNC_BALL` with any special action to be executed by the engine because it is assured that the engine is in the resetting state when it has received `OX_SYNC_BALL`.

## 5.4 Debugging facilities

Debugging is sometimes very hard for distributed computations. We provide two methods to help debugging on X window system: 1. the diagnostic messages from the engine are displayed in a `xterm` window; 2. the engine can pop up a window to input debug commands. For example `ox_asir`, which is the OpenXM server of Risa/Asir, can pop up a

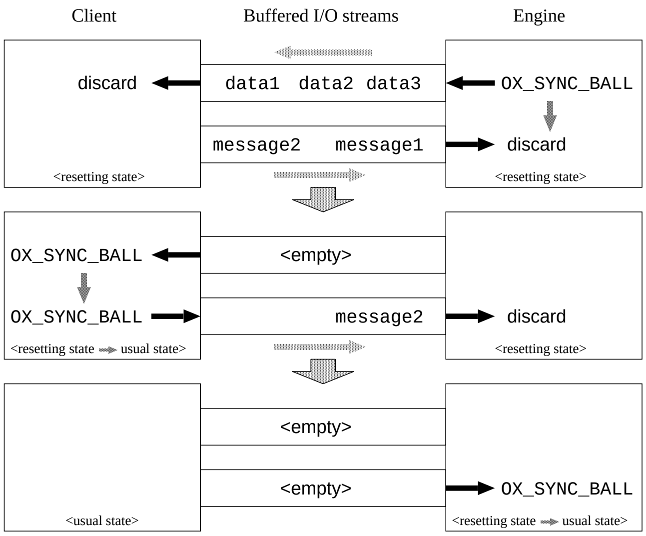


Figure 1: OpenXM reset procedure

window to input debug commands and the debugging similar to that on usual terminals is possible. One can also send `SIGINT` by using `SM_control_to_debug_mode` and it provides a similar functionality to entering the debugging mode from a keyboard interruption.

# 6 OPENXM CLIENTS

## 6.1 Risa/Asir

Risa/Asir provides a launcher `ox_launch` to invoke an OpenXM server and to set up the communication between the server and itself. Fundamental operations on OpenXM servers are exchange of OX data and sending of stack machine commands. As a client, Asir provides the following functions to execute these primitive operations: `ox_push_cmo()` for pushing data, `ox_push_cmd()` for sending a stack machine command and `ox_get()` for receiving data from a stream.

Some operations including the reset operation are implemented by combining these primitives. Among them, frequently used ones are provided as built-in functions. We show some of them.

- `ox_pop_cmo()`

  It requests a server to send data on the stack to the stream, then it receives the data from the stream.

- `ox_cmo_rpc()`

  After pushing the name of a function, arguments and the number of the arguments to the stack of a server, it requests the server to execute the function. It does not wait the termination of the function call.

- `ox_reset()`

  After sending `SM_control_reset_connection` to a control server, it completes the operations stated in Section 5.2.

Furthermore `ox_select()` is provided to detect if streams are ready for reading. It is implemented by the `select()` system call and is used to avoid blocking on read operations.

## 6.2 Mathematica

We provide an OpenXM client `math2ox` written as an external module for Mathematica. The module `math2ox` communicates with Mathematica by MathLink and with any OpenXM server by the OpenXM protocol. By using the module `math2ox`, we can call OpenXM servers from Mathematica; here is a demonstration of a computation of the de Rham cohomology groups of $\mathbf{C}^2 \setminus V(x^3 - y^2)$ from Mathematica.

```
In[1]:= Install["math2ox"]
In[2]:= OxStart["../lib/sm1/bin/ox_sm1_forAsir"]
In[3]:= OxExecute[" [(x^3-y^2) (x,y)] deRham "]
In[4]:= OxPopString[]
Out[4]=  [ 1 , 1 , 0 ]  (* The dimension of
                           cohomology groups *)
```

The `math2ox` adds the following functions to Mathematica.

```
OxStart[s_String]
OxStartInsecure[s_String]
OxExecuteString[s_String]
OxParse[s_String]
OxGet[]
OxPopCMO[]
OxPopString[]
OxClose[]
OxReset[]
```

Although the list of functions speaks itself, we add some explanations. The function `OxPopCMO[]` executes the same operation as `ox_pop_cmo()` in Risa/Asir; it pops data from the server stack. The `OxGet[]` receives an OX data message and returns its translation to an local object. The function `OxParse[]` helps debugging to connect Mathematica and ox servers. By using the function, one can send OX messages, written by the OX expression, to a server. OX expressions are Lisp-like expressions for OX messages and are defined in [10].

# 7 OPENXM CVS SERVER

OpenXM package is currently developed and maintained on a CVS server. The CVS is an open system for a version control. Any mathematical programmer is welcome to join with the committer team of OpenXM. The CVS web is used for reading the source tree and the change logs. The cvsup server is used to synchronize the local source tree with the main source tree. These software tools are used to develop free software packages like the FreeBSD operating system. These tools are very useful in development of the OpenXM mathematical software package by a team.

# 8 APPLICATIONS

## 8.1 Heterogeneous Servers

By using OpenXM, we can treat OpenXM servers essentially like a subroutine. Since OpenXM provides a universal stack machine which does not depend each servers, it is relatively easy to install new servers. We can build a new computer math system by assembling different OpenXM servers. It is similar to building a toy house by LEGO blocks.

We will see two examples of custom-made systems built by OpenXM servers.

### *8.1.1 Computation of annihilating ideals by kan/sm1 and ox_asir*

Let $D = \mathbf{Q}\langle x_1, \ldots, x_n, \partial_1, \ldots, \partial_n\rangle$ be the ring of differential operators. For a given polynomial $f \in \mathbf{Q}[x_1, \ldots, x_n]$, the annihilating ideal of $f^{-1}$ is defined as

$$\mathrm{Ann}\, f^{-1} = \{\ell \in D \mid \ell \bullet f^{-1} = 0\}.$$

Here, $\bullet$ denotes the action of $D$ to functions. The annihilating ideal can be regarded as the maximal differential equations for the function $f^{-1}$. An algorithm to determine generators of the annihilating ideal was given by Oaku (see, e.g., [13, 5.3]). His algorithm reduces the problem to computations of Gröbner bases in $D$ and to find the minimal integral root of a polynomial. This algorithm (the function `annfs`) is implemented by kan/sm1 [16], for Gröbner basis computation in $D$, and `ox_asir`, to factorize polynomials to find the integral roots. These two OpenXM compliant systems are integrated by the OpenXM protocol.

For example, the following is a sm1 session to find the annihilating ideal for $f = x^3 - y^2z^2$.

```
sm1>[(x^3-y^2 z^2) (x,y,z)] annfs ::
Starting ox_asir server.
Byte order for control process is network byte order.
Byte order for engine process is network byte order.
[[-y*Dy+z*Dz, 2*x*Dx+3*y*Dy+6, -2*y*z^2*Dx-3*x^2*Dy,
-2*y^2*z*Dx-3*x^2*Dz, -2*z^3*Dx*Dz-3*x^2*Dy^2-2*z^2*Dx],
 [-1,-139968*s^7-1119744*s^6-3802464*s^5-7107264*s^4
     -7898796*s^3-5220720*s^2-1900500*s-294000]]
```

The last polynomial is factored as $-12(s+1)(3s+5)(3s+4)(6s+5)(6s+7)$ and the minimal integral root is $-1$ as shown in the output.

Similarly, an algorithm to stratify singularity [11] is implemented by kan/sm1 [16], for Gröbner basis computation in $D$, and `ox_asir`, for primary ideal decompositions.

### *8.1.2 A Course on Solving Algebraic Equations*

Risa/Asir [8] is a general computer algebra system which can be used for Gröbner basis computations for zero dimensional ideal with $\mathbf{Q}$ coefficients. However, it is not good at graphical presentations and numerical methods. We integrated Risa/Asir, ox_phc (based on PHC pack by Verschelde [17] for the polyhedral homotopy method) and ox_gnuplot (GNUPLOT) servers to teach a course on solving algebraic equations. This course was presented with the text book [2], which discusses on the Gröbner basis method and the polyhedral homotopy method to solve systems of algebraic equations. We taught the course with a unified environment controlled by the Asir user language, which is similar to C. The following is an Asir session to solve algebraic equations by calling the PHC pack (Figure 2 is the output of `[292]`):

```
[287] phc(katsura(7));
The detailed output is in the file tmp.output.*
The answer is in the variable Phc.
0
[290] B=map(first,Phc)$
[291] gnuplot_plotDots([],0)$
[292] gnuplot_plotDots(B,0)$
```

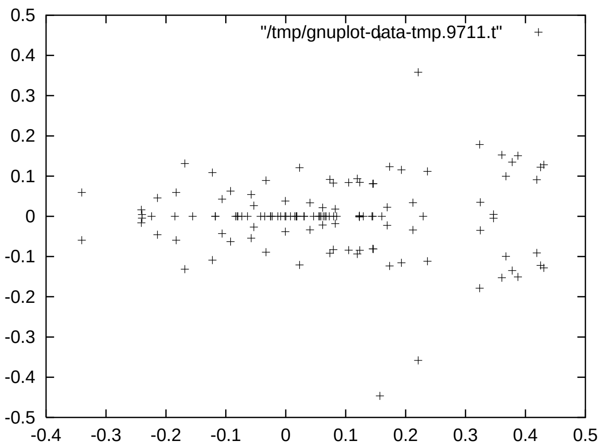


Figure 2: The first components of the solutions to the system of algebraic equations Katsura 7.

## 8.2 Distributed computation with homogeneous servers

One of the aims of OpenXM is a parallel speedup by a distributed computation with homogeneous servers. As the current specification of OpenXM does not include communication between servers, one cannot expect the maximal parallel speedup. However it is possible to execute several types of distributed computation as follows.

### *8.2.1 Product of univariate polynomials*

Shoup [15] showed that the product of univariate polynomials with large degrees and large coefficients can be computed efficiently by FFT over small finite fields and Chinese remainder theorem. It can be easily parallelized:

Input : $f_1, f_2 \in \mathbf{Z}[x]$ such that $deg(f_1), deg(f_2) < 2^M$
Output : $f = f_1 f_2$
$P \leftarrow \{m_1, \cdots, m_N\}$ where $m_i$ is an odd prime,
  $2^{M+1} | m_i - 1$ and $m = \prod m_i$ is sufficiently large.
Separate $P$ into disjoint subsets $P_1, \cdots, P_L$.
for $j = 1$ to $L$ $M_j \leftarrow \prod_{m_i \in P_j} m_i$
Compute $F_j$ such that $F_j \equiv f_1 f_2 \bmod M_j$
  and $F_j \equiv 0 \bmod m/M_j$ in parallel.
  (The product is computed by FFT.)

return $\phi_m(\sum F_j)$
(For $a \in \mathbf{Z}$, $\phi_m(a) \in (-m/2, m/2)$ and $\phi_m(a) \equiv a \bmod m$)

Figure 3 shows the speedup factor under the above distributed computation on Risa/Asir. For each $n$, two polynomials of degree $n$ with 3000bit coefficients are generated and the product is computed. The machine is FUJITSU AP3000, a cluster of Sun workstations connected with a high speed network and MPI over the network is used to implement OpenXM.

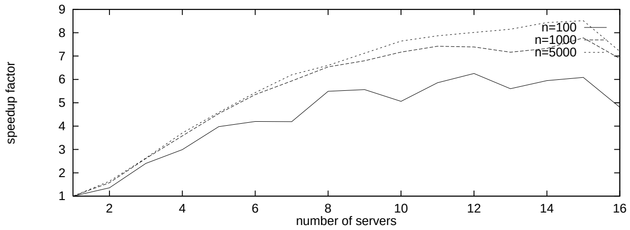


Figure 3: Speedup factor

If the number of servers is $L$ and the inputs are fixed, then the cost to compute $F_j$ in parallel is $O(1/L)$, whereas the cost to send and receive polynomials is $O(L)$ if `ox_push_cmo()` and `ox_pop_cmo()` are repeatedly applied on the client. Therefore the speedup is limited and the upper bound of the speedup factor depends on the ratio of the computational cost and the communication cost for each unit operation. Figure 3 shows that the speedup is satisfactory if the degree is large and $L$ is not large, say, up to 10 under the above environment. If OpenXM provides operations for the broadcast and the reduction such as `MPI_Bcast` and `MPI_Reduce` respectively, the cost of sending $f_1$, $f_2$ and gathering $F_j$ may be reduced to $O(\log_2 L)$ and we can expect better results in such a case.

### *8.2.2 Competitive distributed computation by various strategies*

SINGULAR [4] implements *MP* interface for distributed computation and a competitive Gröbner basis computation is illustrated as an example of distributed computation. Such a distributed computation is also possible on OpenXM. The following Risa/Asir function computes a Gröbner basis by starting the computations simultaneously from the homogenized input and the input itself. The client watches the streams by `ox_select()` and the result which is returned first is taken. Then the remaining server is reset.

```
/* G:set of polys; V:list of variables */
/* O:type of order; P0,P1: id's of servers */
def dgr(G,V,O,P0,P1)
{
  P = [P0,P1]; /* server list */
  map(ox_reset,P); /* reset servers */
  /* P0 executes non-homogenized computation */
  ox_cmo_rpc(P0,"dp_gr_main",G,V,0,1,O);
  /* P1 executes homogenized computation */
  ox_cmo_rpc(P1,"dp_gr_main",G,V,1,1,O);
  map(ox_push_cmd,P,262); /* 262 = OX_popCMO */
  F = ox_select(P); /* wait for data */
  /* F[0] is a server's id which is ready */
  R = ox_get(F[0]);
  if ( F[0] == P0 ) {
    Win = "nonhomo"; Lose = P1;
  } else {
    Win = "homo"; Lose = P0;
  }
  ox_reset(Lose); /* reset the loser */
  return [Win,R];
}
```

## REFERENCES

[1] The OpenMath Esprit Consortium (Caprotti, O. and Cohen, A.M. Editors), The OpenMath Standard. D1.3.2a (Public) `http://www.nag.co.uk/projects/OpenMath`, February, 1999.

[2] Cox, D., Little, J., O'Shea, *Using Algebraic Geometry*, Springer, 1998.

[3] Gray, S., Kajler, N. and Wang, P. S., Design and Implementation of MP, a Protocol for Efficient Exchange of Mathematical Expressions, *Journal of Symbolic Computation*, 1996.

[4] Greuel, G.-M. et al., SINGULAR : a computer algebra system for polynomial computations, `http://www.mathematik.uni-kl.de/~zca/Singular/`.

[5] Linton, S. and Solomon, A., OpenMath, IAMC and `GAP`, preprint, 1999.

[6] Message Passing Interface, `http://www.mpi-forum.org`

[7] NetSolve, `http://www.cs.utk.edu/netsolve`

[8] Noro, M. et al., A Computer Algebra System `Risa/Asir`, 1993, 1995, 2000 `ftp://archives.cs.ehime-u.ac.jp/pub/asir2000/`

[9] Noro, M. and McKay, J., Computation of replicable functions on Risa/Asir. Proceedings of the Second International Symposium on Symbolic Computation PASCO'97, ACM Press, 130-138 (1997).

[10] Noro, M and Takayama, N., Design and Implementation of OpenXM, 1996, 1997, 1998, 1999, 2000.

[11] Oaku, T., Algorithms for $b$-functions, restrictions, and algebraic local cohomology groups of $D$-modules. Advances in Applied Mathematics, **61**, 61–105, 1997.

[12] `http://www.math.kobe-u.ac.jp/OpenXM`

[13] Saito, M., Sturmfels, B. and Takayama, N., *Gröbner Deformations of Hypergeometric Differential Equations*. Algorithms and Computation in Mathematics **6**. Springer, 1999.

[14] Schefström, D., Building a highly integrated development environment using preexisting parts. In IFIP 11th World Computer Congress, San Francisco, USA.

[15] Shoup, V., A new polynomial factorization algorithm and its implementation, *Journal of Symbolic Computation*, 20, 364-397, 1996.

[16] Takayama, N., *Kan: A system for computation in algebraic analysis,* 1991 version 1, 1994 version 2, the latest version is 2.991106. `ftp.math.kobe-u.ac.jp/pub/kan`

[17] Verschelde, J., PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. ACM Transaction on Mathematical Softwares, 25(2) 251-276, 1999.

[18] Wang, P., Design and Protocol for Internet Accessible Mathematical Computation. Technical Report ICM-199901-001, ICM/Kent State University, 1999.

[19] Wolfram, S., *The Mathematica Book, Fourth Edition.* 1999, Cambridge University Press.