MetaCompiler: A Compiler System on Grid-like MetaComputing Environment

SATO Hiroyuki

Information Technology Center, the University of Tokyo
2-11-16, Yayoi, Bunkyo-ku, Tokyo, 113-8658, Japan.
e-mail address: schuko@cc.u-tokyo.ac.jp

Abstract:

GRID-like metacomputing environment provides a view in which a programmer can access distributed computational resources in a transparent way. In addition to the conventional technologies of distributed environment, a unification of local and distributed resources is required in order to fully utilize the resources of GRID.

In this paper, we propose a compiler system META-C on a GRID-like metacomputing system, and implement its prototype. Using this compiler, GRID environment is made transparent, and the compiler enables a programmer to compile source programs, without re-coding, on wide range of environments varying from local to metacomputing system. In our system, caching and partial computation are the key technologies in META-C together with conventional distributed technologies such as directory service.

Keywords:

Grid, Compiler, Numerical Computation, Agent System, Partial Computation, Caching

1. Introduction

Grid-like metacomputing environment provides a view in which a programmer can access distributed computational resources in a transparent way. This requires, in addition to conventional distributed technology, integrated access methods to local and remote resources. Recent development of the distributed technology has enabled us to access remote resource in a more abstract way. Abstraction of "resources" in distributed environment is a key technology in distributed technology. One of the most useful, but a long-history-having resource is "file." Today, resources are generally understood as "objects," and a number of objects can be treated as computational "resource" in a distributed environment.

In this situation, one of major problems is how to embed the space of remote resource names into a local environment, by which a programmer can access remote resources in the same way as local resources. There are two major solutions: one is to give unique name to each resource. Internet domain name and URI in Web systems are its typical examples. This solution is an extension of name space system, and a programmer can access names in this extended name space in almost the same way. The other solution is to map each remote resource name to the local name space. NFS is its typical example. This solution enables a programmer to access remote resource in the identical way to the local resource. However, the mapping must be carefully designed to avoid conflict in the name space.

The integration of local name space and remote name space is important because it is a key technology in providing transparency of site of resources. A programmer does not have to

consider the specific site or method in the access. Today, a number of directory services are provided as the database of resource names and access methods.

This paper studies numerical libraries as computational resources in distributed environment. Numerical libraries, together with compilers, are commonly used as a resource of high performance computing environments. They can be an interface with high performance computers. In other words, users can abstract a high performance computer as a high performance numerical library machine. This has much in common with RPC, in which remote resources are provided as procedure calls. Numerical libraries themselves have been, and are written and maintained with enormous efforts. BLAS and LAPACK are their typical examples. Furthermore, their performance is continuously improving. By using library calls, a programmer can fully utilize this performance improvement.

Distributed environment means the heterogeneous cluster of high performance computers connected via network with each other. The fact the number of high performance computers has been rapidly increasing, together with the increasing demands of high performance computing, makes reasonable the plan to construct Web-like network of high performance computers. GRID is the very project that aims at this goal.

Together with the interface of numerical libraries, in our case, the computational resource means the numerical libraries, not the computers themselves. There are at least two successful projects, Netsolve and Ninf. They provide numerical libraries as resources, and RPC-like interface.

Our goal is to provide a compiler by which a programmer can transparently use the remote computers via numerical libraries. This also makes environment transparent in the meaning that numerical programs written on a local environment can be transported to a distributed environment. We denote this compiler system by a META-compiler. META-compiler requires transparent handling of remote resource and embedding it into a local (=conventional) compiler. Handling a remote resource must be adaptive because organization of distributed environments is never fixed, but evolving. Netsolve and Ninf satisfy this monitoring of organization of remote resources. What remains to be solved is to hide the explicit remote procedure call. If this is solved, the program is portable in its true sense.

In this paper, we discuss the software architecture and implementation of META-compiler. Although our current implementation heavily depends on the RPC-like interface of Netsolve, the software architecture is independent of any specific agent system. We have improved functions of Netsolve in the version management and the directory service. The interface with Netsolve is taken on the linker level. As the performance-directed improvement, we added **#pragma** directives.

The organization of this paper is as follows: Section 2 discusses the related work. Section 3 and 4 studies the software architecture of META-compiler, particularly directory service and name resolution in a metacomputing environment, and caching and partial computation as their key technologies. Section 5 summaries the implementation of META-compiler. Section 6 gives a brief summary.

2. Related Work

Individual technologies for distributed environment have a long history. Before the word of metacomputer and GRID was coined, programming environment on heterogeneous computer network was studied. Jade (Rinard, et al, 1992) is its typical example of programming languages on heterogeneous environments. PVM (Geist, et al, 1994) is another system for heterogeneous environment, which is one of goals of PVM from its outset.

Netsolve (Casanova, et al, 1996, 1997) and Ninf (http://www.ninf.etl.go.jp) provide a GRID-like metacomputing environment from the numerical computation's view. WWVM (Dincer, et al, 1996) is another example of metacomputing environment using HTTP. Besides metacomputing, NEOS (http://www-c.mcs.anl.gov/home/otc) collects remote resources to solve large problems in linear programming.

Netsolve and Ninf have much in common, and have the interface by which Netsolve can call Ninf, and vice versa (Salts, et al, 1998). However, their strategies in identifying resources are slightly different from each other. Netsolve defines "problem" by extending functions, while Ninf uses both function name and location in identifying resources. A location is not necessary in identifying a problem in Netsolve, but is resolved by the agent of Netsolve.

Today, GRID (Foster and Kesselman, 1998) is a typical, and the largest project of metacomputing environment. GLOBUS (Foster and Kesselman, 1997, http://www.globus.org), a toolkit building distributed environments, and LEGION (http://www.cs.virginia.edu/legion), an object-based approach, are its core distributed technologies. NPACI (Karin, 1998) is an approach of GRID from the application's view.

MPICH-G (Foster and Karonis, 1998) is an approach that provides transparent metacomputing environment, and shares its objectives with our MetaCompiler. Using MPICH-G, a programmer can run as-is programs written in MPI on GLOBUS distributed environment.

3. Criteria of META-compiler

3.1 Criteria for Building Compiler Interface

In this section, we discuss the criteria for building META-compiler.

- There is no need for code rewriting. This is the ***transparency*** discussed in Section 1. This requires that the META-compiler processes the identical source programs to those targeted at conventional local environment.
- The remote resource can be accessed as a procedure call. This requirement is satisfied by many preceding systems including RPC, Netsolve and Ninf.

The META-compiler system makes the GRID-like metacomputing system transparent for a programmer. This means that the compiler system determines at which site a library call must be processed, whether local, or some remote site. A programmer does not need to consider any specific conditions of environments. This has two results: portability from the local environment to GRID, and furthermore that between metacomputing environments. The same program can be compiled and run both on small environments for debug and on large metacomputing environments for production-run. We implement a META-compiler on Netsolve, as a compiler interface on a GRID-like metacomputing environment.

3.2 Building Blocks

To satisfy the criteria stated above, we use the building blocks with the strategy listed below:
[C Compiler] We implement a META-compiler on conventional compilers. Specifically, a C compiler together with a linker is used. The extension is implemented as a preprocessor.
[Netsolve] As an interface with the metacomputing environment, we adopted Netsolve.

Using those blocks, we identify the problems to be solved in this project as:

1. identification of remote resources.
2. name management of remote resources together with the design of directory service.
3. caching.
4. parameter passing in RPC.
5. performance tuning.
6. parallelism consideration.

In this paper, we mainly discuss 1, 2, and 3, the technologies that provide the environment-transparency to META-compiler together with performance tuning.

4. Software Architecture of Meta-C

In this section, we discuss META-C, an implementation of a META-compiler.

4.1 Overview

We show the summary of META-C in Figure 1.

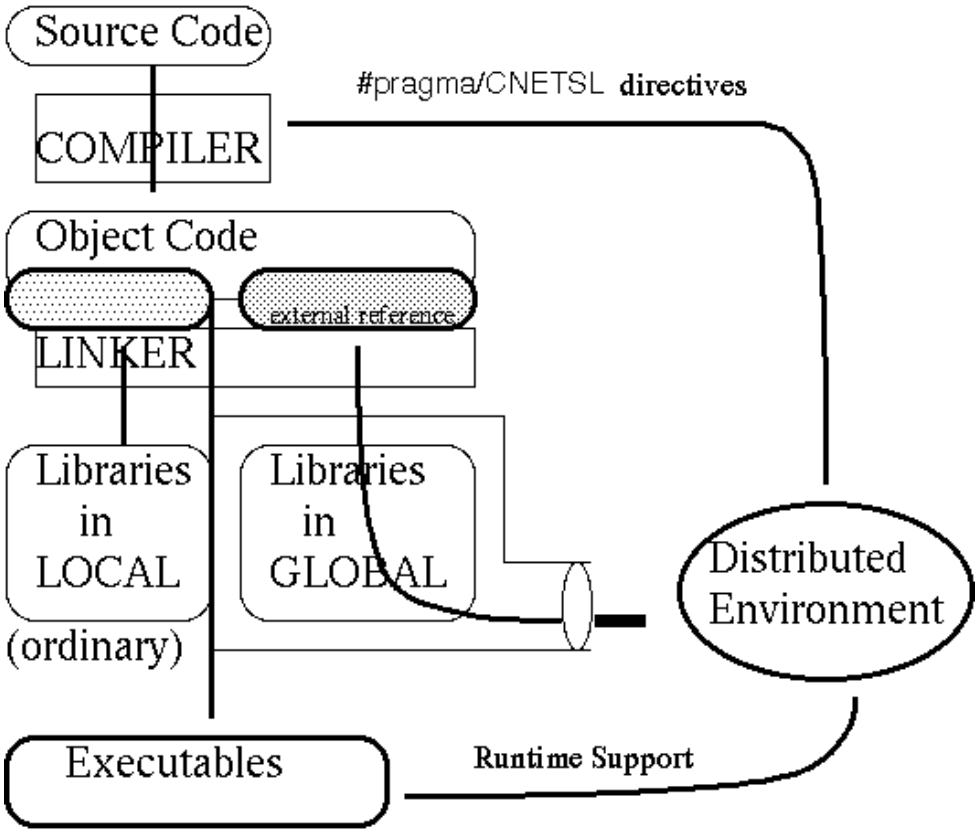

Figure 1. Software Architecture of MetaCompiler

By using META-C, we can process as-is programs on distributed environment, if they are written using numerical libraries.

META-C is connected with GRID via Netsolve, at the link phase. Function names are resolved in an extended name space reflecting the environment.

The interaction with the GRID is taken using the "agent" of Netsolve with a slight improvement. For performance tuning, we have implemented directives as hints for selection of servers of resources. Furthermore, we have made possible the selection of blocking/non-blocking computation for parallelism implemented on Netsolve.

By default, the META-C daemon together with the improved agent automatically does the selection by partial computation on the information on resources obtained from Netsolve, and caches the result on a local system.

4.2 Name Resolution and Directory Service

The resources are numerical libraries under control of Netsolve. Specifically, at the phase that the linker resolves the name of functions, numerical libraries at remote sites are also used in the name resolution. The information necessary for linking is provided by the agent, and listed below:

Name	Parameter Type	Result Type	Server List

Then, the improved agent of META-C generates the interface-stub as the form of archives. These archives are updated when the information from the agent of Netsolve is updated.

4.3 Enhancement of Directory Service

Directory service provided by Netsolve is based on abstract "problem." A problem is an abstraction of a library function name. In which the function is located is hidden from a programmer. The Netsolve agent controls the information on the location associated with function names. A client issues a query of a problem to the agent; the agent returns the list of locations with priorities; the client stub internally selects the server, and issues RPC to the server. What a client must do is just to explicitly select library calls that are to be processed by Netsolve, which is just the part where the code rewriting is necessary. When invoking the Netsolve client, the location of a server is transparent, but the invoking itself must be explicitly written.

For META-C, there is no (explicit) Netsolve call. A program-point where a library call is processed as a Netsolve call is automatically, or transparently determined by META-C. The Netsolve information that the linker and archivers use must be reflected in archives. This includes the organization of GRID and its update information. Moreover, the information of locations of functions is also necessary in building and updating the archives that the linker uses in transparent Netsolve calls. Furthermore, the information that the agent uses in the selection of servers is necessary so that META-C itself can tune performance.

To fulfil these requirements, we have made the enhancement of Netsolve agent:

- functions to obtain all list of problems under the control of the agent,
- functions to obtain all list of problems updated from the last query, and
- functions to obtain the information that the agent uses in server selection.

4.4 Caching

As Figure 1 shows, META-C uses archives to store the information (=function names which are remotely serviced) from the Netsolve agent. This is a kind of *caching* because the information obtained in the interaction with the agent is locally stored and used. Caching is proved to be effective when the access cost of network is high. Caching in Web systems is one of typical examples. In the rest of this subsection, we discuss caching.

The number of library functions existing in the GRID is generally too large to be handled by the linker of a local system. Saving all remote function names in local archives is not very effective. Therefore, we have designed the interface in twofold: library functions managed by the agent are used in local archives, and other function calls are issued directly to Netsolve.

The first interface in the form of archives stores the information from the agent. Specifically, META-C

1. obtains the information from the agent on the selection of servers, and
2. partially computes the information, and selects the server, and finally
3. stores the code that directly issue RPC to the selected server.

In the server selection, the protocol of Netsolve is divided into the essentially dynamic calls and the calculation using the information that the agent has. The latter part can be partially computed, by which some interactions with agents can be saved. The partial computation guarantees the same effect as the normal computation with better performance.

The caching process is the combination of partial computation and its store to the local system.

On the other hand, the second interface is invoked when the linker tells that the function is not in the interface archives, and directly issues the Netsolve client code. The code requests the service to an outer Netsolve agent that manages larger problems.

4.5 Performance Tuning

Usually, given a library call, META-C determines its optimal server with the information given by Netsolve. However, there are some cases that some hints are useful in determining servers for performance tuning. In Meta-C, we provide the hints listed below.

[heavy] The function call costs high.
[light] The function call costs low.
[block] The function is called in the blocking way, or
[nonblock] The function is called in the non-blocking way.
[server] Specify the server.

The option “heavy” is useful as a hint in server selection. Blocking/non-blocking call is closely related with parallel processing. These options can change the original logic of server selection. They are provided as **#pragma** directives.

The current version of META-C takes only library calls as the target of computation on remote server. However, we can extend this schema so that we can also take loops as the target of remote computation. This brings parallelization, or a typical performance tuning, into META-C, and the merge with OpenMP (http://www.openmp.org/specs) is possible at the future version.

5. Prototype Implementation

In this section, we discuss our prototype implementation of META-C. META-C consists of two parts: **nslcc**, the language processor, and **makar**, the daemon which interacts with the Netsolve agent.

5.1 nslcc

nslcc preprocesses the source file in the order that it
1. extracts and interprets **#pragma** directives, and
2. invokes the compiler and the linker at appropriate timings.

As major compiler options, we have added **-LOCAL** and **–GLOBAL** to specify that name resolution is done with priority given to the local or metacomputing environment, respectively.

5.2 Outline

The outline of processes of META-C is such that
1. first it extracts and interprets **#pragma** directives in the source files for performance tuning. The result is output to **netsl_**-prefixed files.
2. Next, it compiles the **netsl_**-prefixed files using **cc**, and finally
3. it makes linkage using **libmetafunc.a**.This archive saves the client stub of the numerical library functions. Its details are described in Section 5.3.

Figure 2 shows this outline of processing according to the software architecture discussed in the last section.

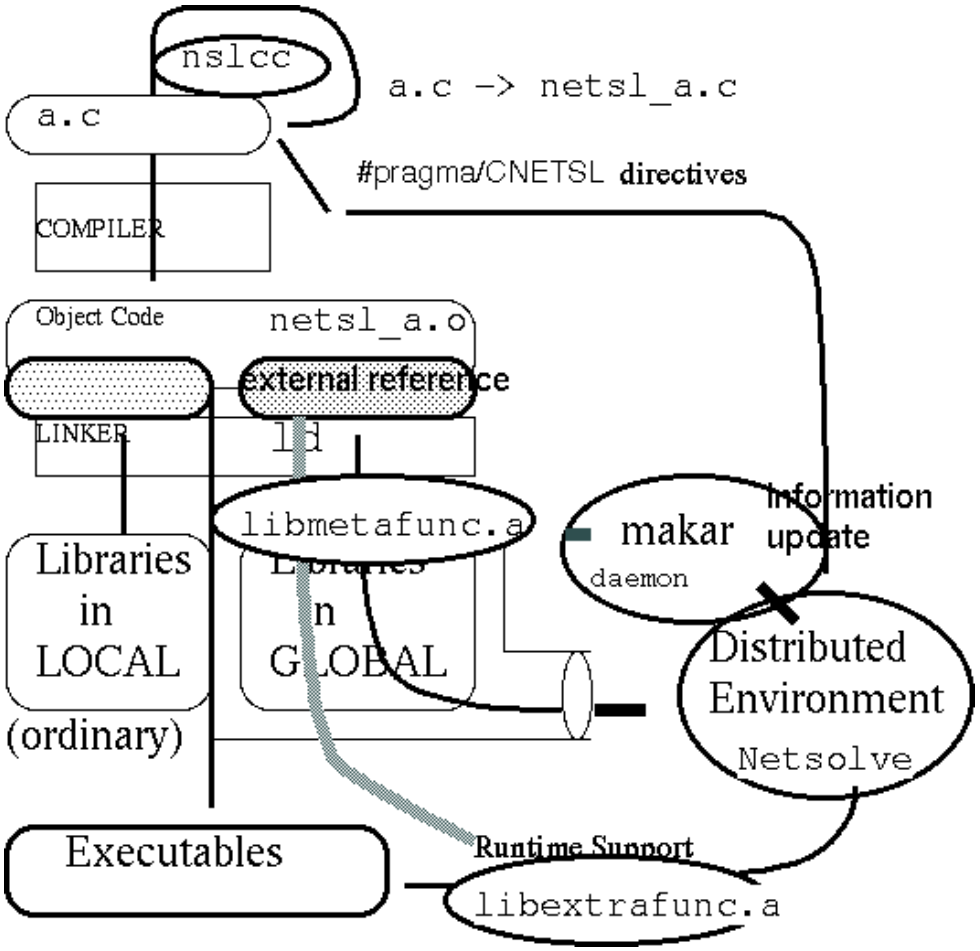

Figure 2. Outline of Prototype Implementation

5.3 Resource Name Resolution Environment

We provide three environments where the function names are resolved.
[local] Names are resolved locally. In other words, the linker searches names only in local archives.
[mc] The name space is extended to that including names provided by the Netsolve agents.
[netsolve] Names are resolved only in the name space of the Netsolve agent.

By switching environments, the same source program can be compiled in the environment varying from the local one to GRID.

The *transparency* is implemented as automatic RPC to GRID (i.e., mc or netsolve) environment. A programmer can use the same programs as those for local environment, and the name resolution is done in the space automatically extended to metacomputing environment.

We show in Figure 3 the algorithm of name resolution as the link strategy. Each stub in **libmetafunc.a** and **libextrafunc.a** is described in Section 5.5 and 5.6. In the implementation,

the compiler options **-LOCAL** and **-GLOBAL** correspond to local environment and mc environment respectively.

```
if  (switch == local) {
   cc -c a.c -o netsl_a.o;
   cc netsl_a.o -lib1 ...;
//  search local libraries first, and
//  do not use libmetafunc.a in link phase.
} else if (switch == mc) {
            cc -c a.c -o netsl_a.o; cc netsl_a.o -lmetafunc -lnetsolve -lnsl -lsocket -lib1 ...;
//  search libmetafunc.a(where client stubs are stored) first.
}
if (linker reports unresolved functions) {
            make libextrafunc.a;
//  Analyze the error message from the linker,
//  and for the unresolved names,
//  generate the stub requesting name resolution directly to Netsolve agent,
//  and save them to libextrafunc.a,
      cc netsl_a.o ... -lextrafunc;
//  and try again the name resolution using libextrafunc.a
}
```

Figure 3. Name Resolution Algorithm as the Link Strategy.

5.4 A Simple Example

We show a simple example in Figure 4.

```
 bcfe:9] cat a.c
 atest(double **a,double **b, double **c)
{
#pragma NETSOLVE heavy(dmatmul)
            dmatmul(a,b,c);
}
 bcfe:10] nslcc -GLOBAL -c a.c b.c
 bcfe:11] nm netsl_a.o
 netsl_a.o:

[Index]   Value     Size    Type  Bind  Other Shndx   Name

[3]   |       16|     60|FUNC |GLOB |0    |2      |atest
[2]   |        0|      0|NOTY |GLOB |0    |UNDEF  |dmatmul
[4]   |        0|      0|NOTY |GLOB |0    |UNDEF  |dmatmulheavyp
[1]   |        0|      0|FILE |LOCL |0    |ABS    |netsl_a.c
bcfe:12]
```

Figure 4. Example of Meta-C Process

In Figure 4, the generated code computes **dmatmul** in the metacomputing environment. **dmatmulheavyp** works as a performance option for META-C.

5.5 makar

makar is a daemon that interacts with Netsolve agents. It obtains problems and related information from the agent, makes client stubs, and makes their archives.

The outline of the processing is:

1. first, **makar** obtains all problem names and server names which the Netsolve agent is managing.
2. Next, it makes a client stub for each problem. The information from the agent includes servers of the problems, types of parameters and results. For example, in the case of **dmatmul**, the stub is generated as in Figure 5.

```
typedef struct { float r; float i; } scomplex;
typedef struct { double r; double i; } dcomplex;
int dmatmulheavyp=0;
int dmatmullightp=0;
int dmatmulnbp=0;
char dmatmul_d_servers_name = {“bcfe.cc.u-tokyo.ac.jp”,””};
unsigned int dmatmul_d_servers_addr = {2185622597, 0}
char dmatmul_f_servers_name = {“bcfe.cc.u-tokyo.ac.jp”,””};
unsigned int dmatmul_f_servers_addr = {2185622597, 0}
char dmatmul_l_servers_name = {“bcfe.cc.u-tokyo.ac.jp”,””};
unsigned int dmatmul_l_servers_addr = {2185622597, 0}

double **dmatmul(double  p0[][],double  p1[][])
{    if (ns_parallel != 0) {
        int ptr = nbrqtsptr;;
        nbrqts[nbrqtsptr++]=netslnb2(dmatmul_d_servers_name,
                          dmatmul_d_servers_addr,“dmatmul()”,p0,p1);
        if (nbrqtsptr >= 256) ns_barrier();
        return nbrqts[ptr];
    }
    if (dmatmulnbp) {
      if (dmatmulheavyp)
          return (double**) netslnb2(dmatmul_f_servers_name,
                   dmatmul_f_servers_addr,“dmatmul()”,p0,p1);}
      if (dmatmullightp)
          return (double **) netslnb2(dmatmul_l_servers_name,
                   dmatmul_l_servers_addr,“dmatmul()”, p0,p1);}
      return (double  **) netslnb2(dmatmul_d_servers_name,
                   dmatmul_d_servers_addr,“dmatmul()”,p0,p1);}
    else {
      if (dmatmulheavyp)
           return (double  **) netsl2(dmatmul_f_servers_name,
                   dmatmul_f_servers_addr,“dmatmul()”, p0,p1);
      if (dmatmullightp)
           return (double  **) netsl2(dmatmul_l_servers_name,
                   dmatmul_l_servers_addr,“dmatmul()”, p0,p1);
      return (double  **) netsl2(dmatmul_d_servers_name,
                   dmatmul_d_servers_addr,“dmatmul()”, p0,p1);
}
```

Figure 5. A C Stub Program for dmatmul

It is to be noted that the name of the server on which **dmatmul** is to be computed is already resolved. Therefore, the server is explicitly specified in **netsl2**, the RPC version of Netsolve with server name (Figure 5, 6). This means that this **dmatmul** directly issues RPC to the server.

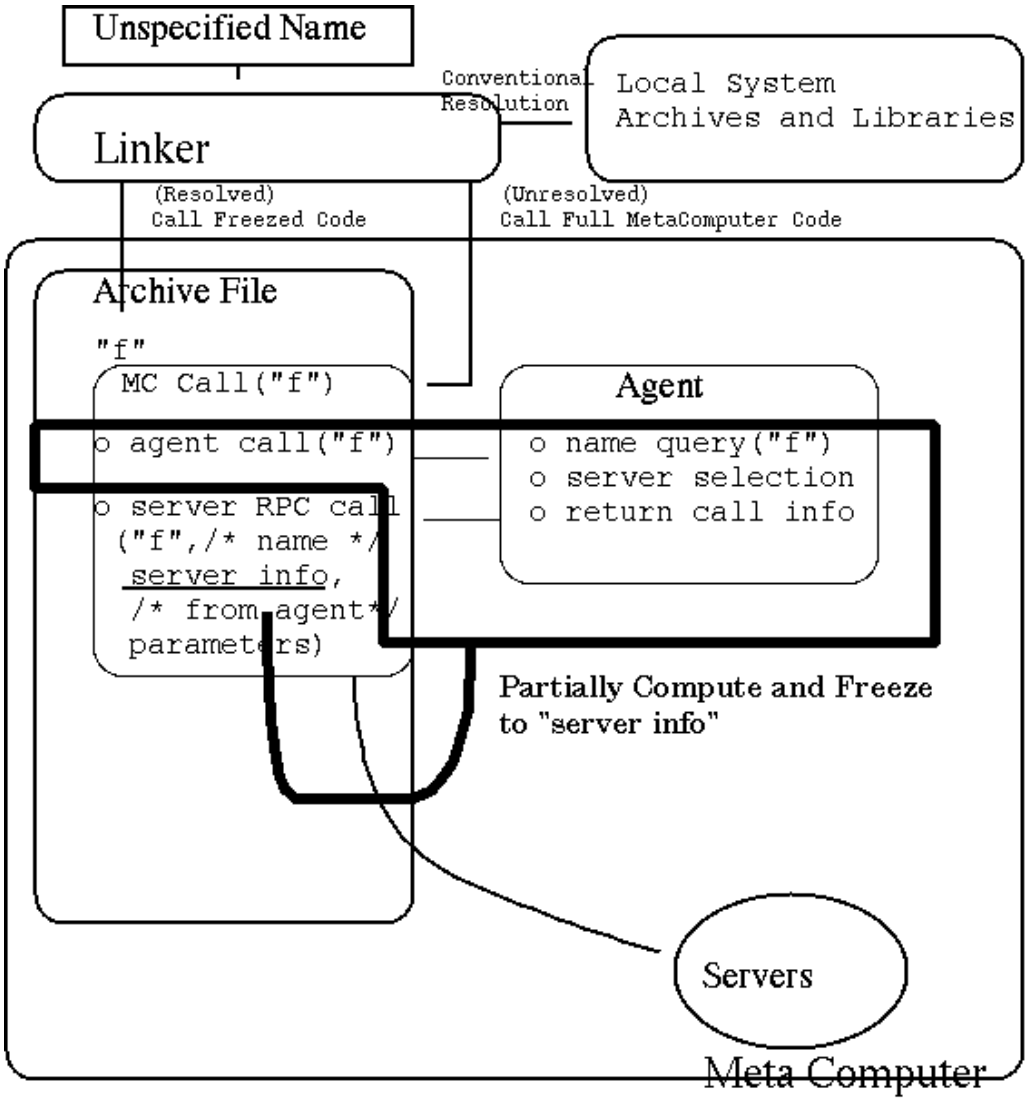

Figure 6. Server Specification by Partial Computation

3. Next, **makar** saves stubs into the archive **libmetafunc.a**. In other words, this archive represents the cache of the result of the partial computation of server-name resolution.
4. It periodically issues the request to the agent, and obtains the update information, then
5. merges the updated stuff into the archive.

In our implementation, we use a simple strategy in server selection. That is, if heavy attribute is specified, the fastest server is selected. If light attribute is specified, the server that has the least latency is selected. By default, we use the default logic of Netsolve.

The version management is indispensable to efficiently handle the update. Our implementation uses a simple timestamp, which needs further improvement.

5.6 Dynamic Interaction with Netsolve

Names that cannot be resolved by using libmetafunc.a are *dynamically* resolved in outer environment that the default Netsolve does not control. Specifically, META-C

1. lists the unresolved names from the error message from **ld**, and
2. for each unresolved name, it generates the corresponding stub that issues the Netsolve request. The target of the outer Netsolve agent is specified as **NETSOLVE_MASTERAGENT**, probably the agent that manages larger problem database.
3. Next, it saves the stubs in **libextrafunc.a**, and tries again.

We show the example stub in Figure 7. By this resolution, we can dynamically interact with the outer metacomputing environment.

```
main(p0,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,p16,p17,p18,
p19,p20)
{ char t[256],*ma;int x;
       strcpy(t, getenv("NETSOLVE_AGENT"));
       setenv("NETSOLVE_AGENT",ma=getenv("NETSOLVE_MASTERAGENT"));
       if (!strcmp(ma,"")) return 0;
       x= netsl("main",p0,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,
                      p14,p15, p16,p17,p18,p19,p20);
       setenv("NETSOLVE_AGENT",t); return x;}
```

Figure 7. A Stub that directly calls a Netsolve other than NETSOLVE_AGENT.

We show in Figure 8 the interactions of the linker with the metacomputing environment using **libmetafunc.a** and **libextrafunc.a**. Server specification by partial computation, its caching (**libmetafunc.a**), and dynamic name resolution using outer Netsolves are the key technologies.

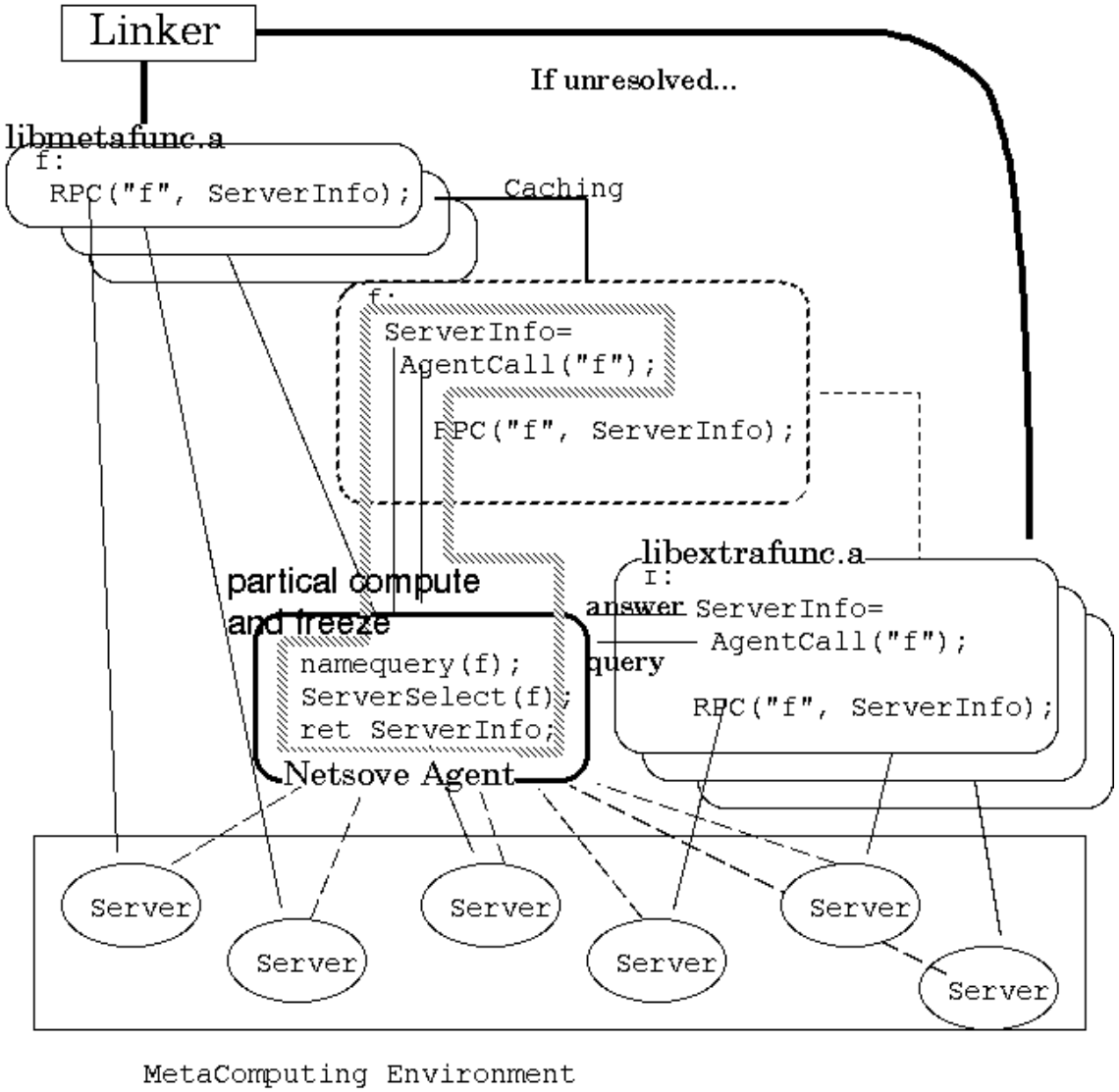

Figure 8. Partial Computation and Caching of its Results in Name Resolution

6. Concluding Remarks

In this paper, we have proposed a compiler system META-C on a GRID-like metacomputing system, and have implemented its prototype. Using this compiler, GRID environment is made transparent, and the compiler enables a programmer to compile source programs on wide range of environments varying from local to GRID.

Software architecture of META-C has been discussed. META-C is connected with Netsolve at the link phase. Function names are resolved in a name space extended to the metacomputing environment. Caching and partial computation are the key technologies in META-C together with conventional distributed technologies such as directory service. Implementation has also been discussed according to the software architecture.

Acknowledgement

This work was partially supported by the Ministry of Education, Science and Culture of Japan (Grant No. 13680397).

References

CASANOVA, H., and DONGARRA, J. (1996): Netsolve: A network server for solving computational science problems, Supercomputing 96.
CASANOVA, H., DONGARRA, J. and SEYMOUR K. (1997): Client User's Guide to NetSolve, draft.
DINCER, K. and FOX, G.C. (1996): Building a World-Wide Virtual Machine Based on Web and HPCC Technologies, Supercomputing 96.
FOSTER, I., and KARONIS, N. (1998): A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems, Supercomputing 98.
FOSTER, I., and KESSELMAN, C. (1997): Globus: A metacomputing infrastructure toolkit, J. Supercomput. Appl., 11(2), 115--128.
FOSTER, I., and KESSELMAN, C. Eds. (1998): The Grid. Blueprint for a new Computing Infrastructure.
GEIST, A., BEGUELIN, A., DONGARRA J., JIANG, W., MANCHECK, R., and SUNDERAM, V. (1994): PVM: Parallel Virtual Machine.
KARIN, S., and GRAHAM, S. Eds. (1998): The High Performance Computing Continuum, Comm. ACM, 41(11), 32--75.
RINARD, M.C., SCALES, D.J. and LAM, M.S. (1992): Jade: a high-level, machine-independent language for parallel programming, Supercomputing 92, 245--256.
SALTS, J., SUSSMAN, A., GRAHAM, S., DEMMEL, J., BADEN, S., and DONGARRA, J. (1998): Programming Tools and Environments, (Karin, et al, 1998), 64--73.