Re-scheduling invocations of services on RPC-Grid [1]

Thierry GAUTIER and Hamid-Reza HAMIDI

Projet APACHE Laboratoire ID-IMAG,
51 av Jean Kuntzmann, 38330 Montbonnot Saint Martin, France [2]

Abstract

RPC-based Grid infrastructures emphasize on the composition of services on a large number of computing resources. The key issue to reach high performance is to enable exploitation of parallelism on services invocations and communications. Moreover, this process should be transparent to reuse legacy codes. In this paper we present HOMA, an IDL compiler and a run-time support for automatic detection of the parallelism of invocations and their data dependencies on a set of CORBA objects. On homogeneous architecture, HOMA is accompanied by a predictable cost model. In the case of an application with a small critical path, among p processors the speed up of HOMA versus CORBA is asymptotically $O(p)$. The illustrations on a case study in computational chemistry validate our cost model.

Key words:

1 Introduction

Multi-scale and multi-physics simulations are emerging as solutions to achieve high fidelity in complex physical system simulations. Applicability of different physics, computational techniques, and programming models demands that these programs be developed as a largely independent collection of software components. For instance, [27] reports that simulation of the next generation of aircraft will require several hundreds of software components and will consume a lot of CPU time. Therefore, these simulations have to exploit the aggregate power of resources scattered from available clusters: such component-based programs have to exploit a computational grid [18]. Several

Email addresses: Thierry.Gautier@inrialpes.fr, Hamid-reza.Hamidi@imag.fr (Thierry GAUTIER and Hamid-Reza HAMIDI).

[1] This research is supported by the French government (ACI-Grid / GRID2 project
[2] The project was supported by C.N.R.S.-I.N.P.G.- I.N.R.I.A.-U.J.F.

environments have been proposed to build applications for a computational grid by providing services to encapsulate (server) components. A Network Enabled Server system (NES) [29] is a Metacomputing environment where a client requests the services of the servers using Remote Procedure Call (RPC). There are many research projects that adopt this basic model of computation Netsolve [11], Ninf [38], DIET [10]. High performance scheduling (for single or multiple requests), automatic extraction of parallelism between several requests and optimization of data movement are some of the main challenges of such systems.

This paper presents the HOMA environment [20–22] to schedule invocations of method onto a large scale distributed memory architecture. HOMA was motivated by multi-physics numerical simulations building by code coupling technics. Iterative and dynamic nature of coupling algorithms for numerical simulations orients HOMA toward dynamic composition of invocations to method with data dependencies. HOMA focus on the applications based on the standard CORBA from OMG, which have been successfully used in several projects for numerical simulation [4,10,27]. Moreover some high performance implementations are available [39,15] making CORBA well suited for building high performance distributed application on computational grid. For instance, some recent projects have proposed to facilitate access to core Grid services (CoG kits) [32] for CORBA based application. DIET [10] presents a NES system which relies on CORBA. CAPE-OPEN [4] is an European project that aimed to propose CORBA interface of components for high performance simulation environments in computer aided process engineering.

While our target environment is CORBA, the challenge addressed into this paper is to overcome some lacks of RPC-based environment for building easily high performance applications.

- The first main difficulty comes from the semantic of the invocation of method between a client and a server. An invocation is mostly a blocking instruction: the caller waits until the server returns values. CORBA limits non-blocking invocations to the methods which do not have output values. Therefore, in order to generate parallel flows of control, a programmer should mix blocking invocations with multi-thread computation, or he should mix non-blocking invocation with an other way to handle return values from server (callback, event driven model of execution, concept of future [25], ...). Whatever is a concrete choice, the natural way a client should invokes methods has to be forbidden. Using a specific compiler and runtime support, HOMA keeps the natural semantics of CORBA invocation to efficiently exploit parallel computation.
- The second limitation is due to the semantic of the communication of effective parameters during the RPC: all effective parameters of an *input* formal parameter of a method (**in** in CORBA) are communicated from the client to

the server; all *output* parameters (**out** in CORBA) are communicated back to the client. The *direction* of a parameter defined in the COBRA Interface Description Language (IDL) allows optimization of communication for one invocation between one client and one server. This local optimization does not imply a global optimization when a client invokes several methods on a set of servers to run a complex simulation: all parameters are moved between servers by passing through the client even if it does not need them; the client becomes the bottleneck for the scalability of using many components. The problem is exacerbated in the case of iterative simulation. Thus, the communication should be optimized by considering the whole invocations. In this paper we present our idea to keep standard CORBA semantics of invocation while generating only communications on the demand, *i.e.* if and only if a data is required on a component for an operation.

The outline of the paper is the following. Next section presents the HOMA project. We introduce the abstract interpretation of CORBA client invocations which allows to build at runtime the data flow graph between all invocations. This graph represents the future of the execution. Section 3.2 presents theoretical results about expected execution time of any CORBA client using HOMA versus a classical implementation: the gain could be linear with respect to the number of processors if the application exhibits enough parallelism. Efficient use of the data parallel CORBA object is described in section 3.3. Section 4 reports some experiments which fit our theoretical analysis. Then we conclude this paper.
This paper extends previous published results on the theoretical scheduling with respect of the memory and on the experiments [22,21,20].

2 A motivating example

This example comes from the project SIMBIO [6] in computational chemistry [3]. The goal of SIMBIO is to build a distributed application to compute complex simulation of molecular structure (a protein) surrounded by solvent.

2.1 SIMBIO application

Figure 1 shows the coupling algorithm of SIMBIO application as a multi step integration scheme. The molecular dynamics (*md* object) and the continuum method (*cm* object) are connected by a special task (*coupling* object) that implements the computation of the coupling terms of the physical models. All objects are parallel: the 'atoms' P and 'surface' S data as well as CORBA objects are in fact distributed vectors. These data and objects are aggregate

[3] SIMBIO was an project funding by INRIA, 1997-1999.

CORBA objects distributed among the processors. They are called *data parallel object* extension to CORBA sequential object [25,36,31]: the object state is distributed among the processors and can be accessed using an interface such as standard CORBA object, the implementation manages locality of data and automatically scatters and gathers data during invocation.

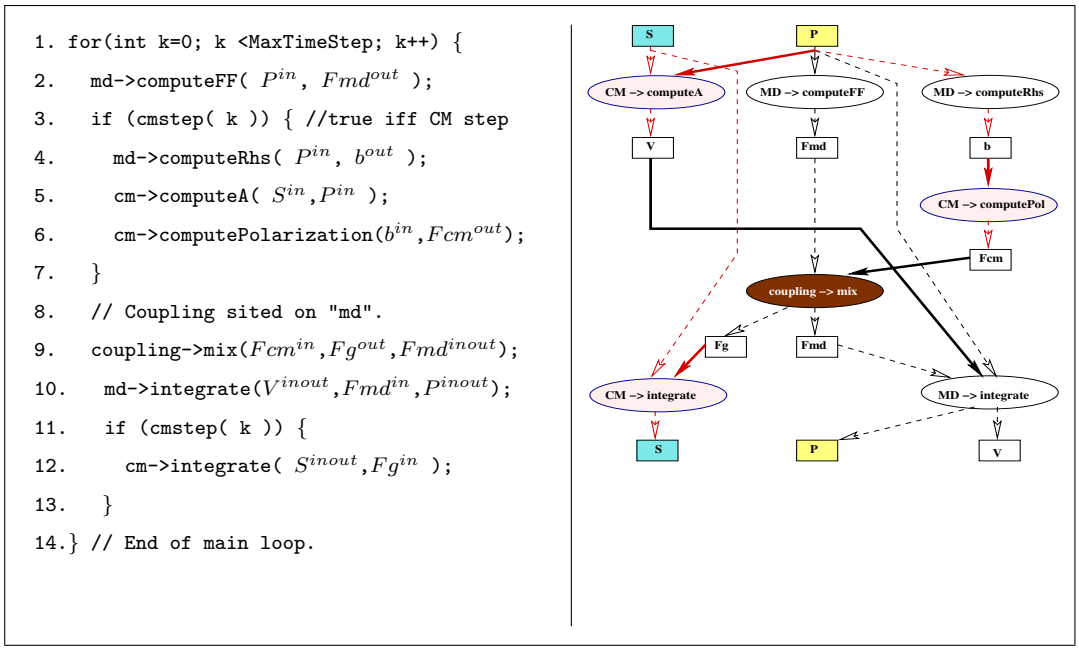


Fig. 1. Left: Client code of SIMBIO with superscripts to present the mode of parameter passing. Right: Data flow graph for one iteration if *cmstep* condition is true. Solid arcs represent inter-objects data communications.

The description of the data flow for one iteration depends on the current step and was given by a CORBA based client program sketched in the left part of figure 1. The right part represents the data flow graph between the invocations of the methods. Box node represents data, and ellipse node represents task. An edge from a task to a data means that the data is written by the task. An edge from a data to a task means that the data is read by the task. Each task in the data flow graph corresponds to invocation of method on CORBA object (at lines 4, 5, 6, 9, 10, 12).

On the top of standard CORBA implementation, the client program makes invocations to the servers. It is involved into each communications: the associated process receives and forwards all data requested by the servers. The client is the bottleneck for the communication. It is inherent from the semantic of method invocations in CORBA which requires explicit communication of effective parameters. HOMA implements less strict semantic of method invocations which allows to uncoupled the control flow and the data flow of the execution. The client always describes the invocations but does not participate into the communication of effective parameters between the servers.

3 Homa

HOMA is a research action to provide an environment which allows to reuse software components and assemble them in order to build high performance and scalable distributed application for numerical simulation. It is based on CORBA component technology [30]. The assumption implicitly made in HOMA is that application involves many components distributed onto several computers. First we present HOMA's features and then its architecture.

3.1 Features for high performances

This section describes three main features required to reach high performances. The first is the ability to predict the future of the execution by capturing all invocations in order to unfold the data flow graph. The second allows an efficient non preemptive execution in order to control the usage of resources at runtime. The later is a lazy scheme of communication of effective parameters. All of them are transparent to the final user and embedded into our IDL compiler overviews in section 3.4.

3.1.1 Abstract Interpretation

In order to achieve efficient execution on parallel architecture, the knowledge of the data dependencies related to the application appears as the key point for computing good schedule. HOMA handles at runtime the "abstract interpretation" to gain information about the parameters required for the invocations. At compile time and from an IDL definition, HOMA generates new client stubs and server skeletons which allow to capture the type and direction mode for each parameter involved in each invocation. At runtime, the control of the program is interpreted in the standard way while all invocations are intercepted to unfold the data flow graph of theirs executions. This graph is bipartite: the nodes form two sets; the "tasks" and the "data". A node of type "task" represents an invocation to a method on a server object. Input arcs are input parameters (`in` in IDL), output arcs are output (`out`) parameters. A node of type "data" represents a version of a variable. The right side of figure 1 represents the associated data flow graph of our motivating example in the case where `cmstep(k)` is true. Each node of the graph has attributes: the site of execution of the task (the site where the server object is instantiated) and the size of the data. Moreover, each node has a state (ready/ not ready) that represents the flow of data with respect of the dependencies. A data is ready if it is written by a task, and a task is ready if all of its input data are ready.

The generation of client stubs and server skeletons embeds all parameters of invocations into global references as provided by the runtime support ATHAPASCAN (see section 3.4). Such object acts as kind of *future* for results of methods that are not yet executed. The abstract interpretation detects read

operations at runtime and it blocks the control flow until the availability of the data. This concept comes from functional language community and it is used in PARDIS [25] to explicitly generate parallelism between invocations.

3.1.2 Efficient Non-preemptive Execution

After computation of a schedule of the data flow graph (section 3.2), the runtime evaluates all tasks accordingly the order induced by the data flow constraints. Remote synchronous method invocation delays execution of the caller until the end of the remote execution and the communication of parameters. This semantic limits the potential parallelism between invocations and requires preemptive scheduling at runtime. With HOMA, blocking invocation is decomposed in two non blocking invocations. This transformation is done into client stub and server skeleton generated from the IDL by our compiler. We call this transformation the "*invocation by continuation*" [22] as the adapted version of the "*wait by necessity*" [9] principle. In *invocation by continuation*, a task invoking a blocking method is transformed in two tasks which invoke non-blocking methods. The execution of the former task initiates the invocation on the server and creates a task that represents the continuation of the invocation. This task will became ready only when the server invokes a method to put back the parameters.

The most import fact is that this transformation allows a non preemptive execution [19] of the tasks with efficient execution: because all tasks are never waiting for results, all processors remain active while there exist ready tasks (in the graph). For instance, it is possible to execute the data flow graph using one and only one thread of control to invoke in parallel methods on several servers.

3.1.3 Automatic Data Movement

Due to the knowledge of the data flow of the (futur) execution of a sequence of invocations, HOMA is able to generate only the required communications to realize one invocation onto a server. As presented above, the data flow graph is automatically unfold thanks to IDL compilation and generation of new client stubs and server skeletons. The IDL compiler translates each parameter involved in invocations to a unique global reference. A reference is composed by a name and a version number. If a server which produces a parameter (*out* and *inout*) knows in advance all the servers that will consume it, the skeleton is able to send the data to the consumers just after its production. A receiver will store it until consumption. This mode of communication is called the *put mode*. Thanks to the knowledge of the data flow graph, data are broadcast to servers using a parallel algorithm. If a sever skeleton requires a data which was not yet received, it sends a request to the process that has

produced it; this is the *get mode* of communication. In both methods, the client that generates the sequence of invocations is never involved into the communications, except if it requires to read data. The advantage of *put mode* is to avoid extra communication. But, because it requires the storage of data on some servers, the put mode should be used carefully with big data in order to avoid memory consumption.

3.2 Theoretical Scheduling Results

HOMA relies on the macro data flow execution kernel of ATHAPASCAN [19,23,37], a runtime environment to manage parallel execution on heterogeneous distributed memory architecture.

3.2.1 Notations and background results

Let us recall the notations used in [19] defined on data flow graph: **Sequential time** T_1 denotes the sequential time that is defined from a serial execution of the program. **Parallel time** T_∞ is the arithmetic depth of graph taking into account the weights (computation costs) of task nodes. T_∞ is then a lower bound of the minimal time required by any non-preemptive schedule on an unbounded number of processors ignoring communications times (PRAM model [17]). **Communication volume** C_1 and **delay** C_∞ are evaluated from graph similarly to T_1 and T_∞ but taking into account only the weights (sizes) of data version nodes. C_1 is the sum of the weights over all data nodes; assuming that the shared memory is emulated in an auxiliary file, C_1 is then an upper bound on the total number of accesses performed in this file during a serial execution. C_∞ is the length of the critical communication path.

3.2.2 Execution time with HOMA

Let us assume that the communication cost for a message of size L is hL. The size of the data flow graph is noted by σ.

Proposition 1 *The time to execute a CORBA client on a p processors machine using* HOMA *is:* $T_p^{homa} = O(\frac{T_1}{p} + T_\infty + h(\frac{C_1}{p} + C_\infty)) + O(\sigma)$.

The proposition is deduced from the results on parallel execution of any ATHAPASCAN program [19]. This time includes the overhead to compute the schedule of the program. The result is valid on a homogeneous parallel architecture (processors must have the same speed). The authors [5] propose a work stealing algorithm to the case of heterogeneous systems with similar bounds but that does not take into account the communication. The execution cost model of HOMA is inherited from ATHAPASCAN [19], because HOMA translates remote blocking invocation into non-blocking invocations (see section 3.1.2) and it enables direct communication between servers (section 3.1.3). Such techniques

add the overhead of creating an extra task for the continuation and an extra communication in the case of *get mode*. But it does not increases asymptotically neither the work of the program nor the volume of communication. The above features of HOMA IDL compiler and runtime allow to execute any CORBA program as an ATHAPASCAN program.

Note that if standard CORBA environment is used and because of limitations explained in the introduction, the execution is sequential and data transit through the client. Because of non-direct communication, the cost of data transfer between servers is about double but remains in $O(h)$. Therefore, the complexity of any execution is: $T_p^{corba} = O(T_1+T_\infty+h(C_1+C_\infty))+O(\sigma)$. If the program is highly parallel ($T_\infty << T_1$) and involves large data ($C_\infty << C_1$), then the gain of using HOMA versus CORBA is nearly linear with respect to the number of processors ($\frac{T_p^{corba}}{T_p^{homa}} = \Theta(p)$).

3.2.3 Memory space with HOMA

Without any care, some schedules may not guarantee any bound concerning the space required. However, ATHAPASCAN [19] is based on scheduling algorithm which bound the memory space for executing program on a p-processors machine. Using notation of [8,19], let us note by S_1 the memory space required for the serial execution on 1 processor. S_p is the memory space required for the execution on p processors. For *fully strict computation*, [8] bounds the memory space by $S_p = O(pS1)$ while preserving good execution time.

The assumptions in [7] about the structure of the computation may be applied for ATHAPASCAN to bound memory space required for execution. Thus any ATHAPASCAN program has memory space bounded by $S_1+O(pT_\infty\ \log(pT_\infty))$ and the time bounded by $O(T_1/p + T_\infty \log p)$ on a p-processors machine [7]. Note that the running time is slightly increased with comparison to the bound given in the previous section. This leads to the following proposition showing that HOMA program have bounded memory space requirement during execution.

Proposition 2 *For any* HOMA *program which required S_1 memory space for serial execution, it exists a scheduling with parallel time $O(T_1/p + T_\infty \log p)$ and memory space $S_1 + O(pT_\infty\ \log(pT_\infty))$.*

3.3 Efficient Data Parallel Exploitation

There are several attempts to integrate data parallel applications in the CORBA object model [31,34,25]. Parallel CORBA object (PACO) [36,34] concept aims at providing an efficient technique to encapsulate parallel codes based on the use of message passing libraries (*i.e.* MPI). It defines the data parallel object as a collection of identical CORBA objects. Data Parallel CORBA Specifica-

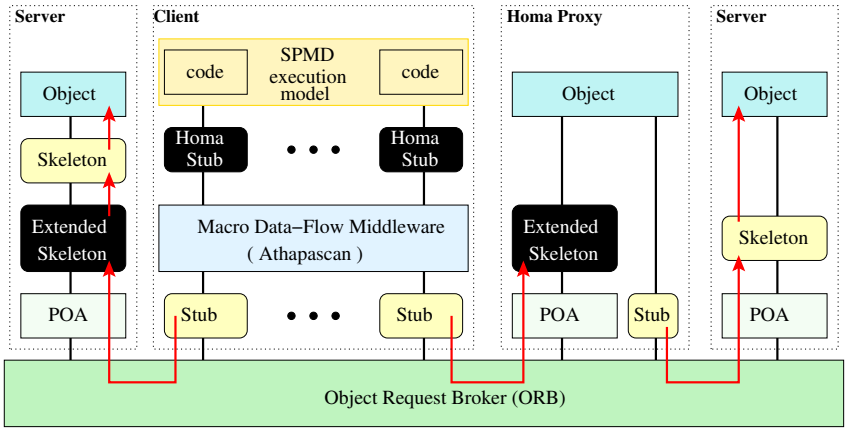


Fig. 2. HOMA's architecture.

tion [31] has been approved by OMG. It defines an additional approach for the implementation and use of CORBA objects that enables the object implementers to take advantage of parallel computing resources to achieve scalable, high performance. Parallel ORBs [31] exploit these advantages just only for one client-to-server invocation.

It is interesting to note that even if data parallel CORBA components [31,34] are used, the semantic of communication implies that parameters transit through the client, thus even if the work is parallelized by $(T_1/p+T_\infty)$ the communication bottleneck due to the client remains in order of $h(C_1+C_\infty)$. Hence even a parallel client could not overcome the lack of direct communication between individual parts of the parallel objects. Thanks to the communication on the demand approach, HOMA enables the parts of parallel objects to communicate in parallel (see section 3.1.3). The extended skeleton contains the extra methods which are the extensions of the user defined object methods. HOMA consider the same data partitioning and request distribution of original parallel objects for its proxy. Using Parallel ORBs, HOMA's data parallel proxies are able to communicate directly, so the communication remains in order of $h(C_1/p + C_\infty)$.

3.4 Implementation

The HOMA IDL compiler generates stub and skeleton in order support the abstract interpretation, the efficient non-preemptive execution and the automatic data movement described in section 3.1. The source code of the client and server does not change but it needs to be recompiled. The generated stub and skeleton relies on stubs and skeletons generated from a standard CORBA implementation. Figure 2 shows the interaction between the HOMA extended stub and skeleton and the native CORBA stub and skeleton. HOMA stub intercepts method invocations to generate the needed information for data flow graph unfolding and automatic data movement. This concerns the transformation of parameters to global references and the management of data into

caches. The cache is distributed among all the processus of the application and stores data as value of CORBA type `any`. The graph is managed and unfold at runtime using the kernel of ATHAPASCAN [19]. The whole compilation process is detailed in [22].

Two approaches are considered to serve legacy codes encapsulated in CORBA object: By recompilation of the server codes, HOMA inserts an extended skeleton which intercepts incoming requests (figure 2, left server). HOMA is also able to generate proxy object to intercept invocations to legacy code servers (figure 2, right server) which introduces extra inter-objects invocation.

4 Experiments

All experiments were made using the OmniORB3 implementation of CORBA. The programs were carried out on the iCluster of INRIA at Grenoble (PC 733Mhz, 256MByte, network 100Mbit/s). The timings are given using the current version of the HOMA environment, which implements only the *get mode* presented above. The scheduling algorithm is a simple greedy list-based centralized algorithm that executes the first ready task of the list.

4.1 Aggregate Bandwidth

In this experience we run the program of figure 3. N is the number of pairs, parameters of invocations are the sequences of size K and each of $2N$ objects is mapped onto a different processor (p=2N). The invocations `produce` on the

```
1. // Call the producters
2. for(int i=0; i < N; i++)
3.    // x[i] is output parameter
4.    Ps[i] -> produce ( x[i] );
5. // Call the consumers
6. for(int i=0; i < N; i++)
7.    // x[i] is input parameter
8.    Pc[i] -> consume ( x[i] );
```

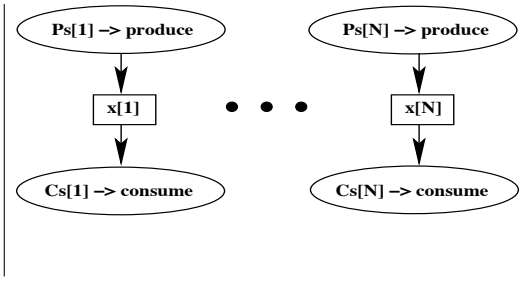


Fig. 3. Left: Client code. Right: Associated data flow graph.

`Ps[i]` write sequences of size K while invocations `consume` on the `Cs[i]` read all the entries of the input sequences. This program is characterized by $T_1 = O(N)$ and $T_\infty = O(1)$; $C_1 = O(NK)$ and $C_\infty = O(1)$. The figure 4, reports the execution time using CORBA and HOMA with two sizes of sequence ($K = 3.2MBytes$ and $K = 9.6MBytes$). The time for CORBA is linear in the number of processors as well as in K as predicted by $T_p^{corba} = O(NK)$.

With HOMA, we observe the time which is nearly constant: results fit our

predictive complexity $T_p^{homa} = O(K)$. Nevertheless, a finer analysis shows that time for HOMA is linear in N: this is due to the term $O(\sigma)$ in our cost model (see proposition 1 page 7). This term takes into account the time required to unfold the data flow graph which basically corresponds to sequentially unroll the loops to create $O(N)$ tasks with dependencies. This time is several orders of magnitude less than the time required to invoke a method on a remote object and has no impact when compared to standard CORBA, as shows in figure 4.

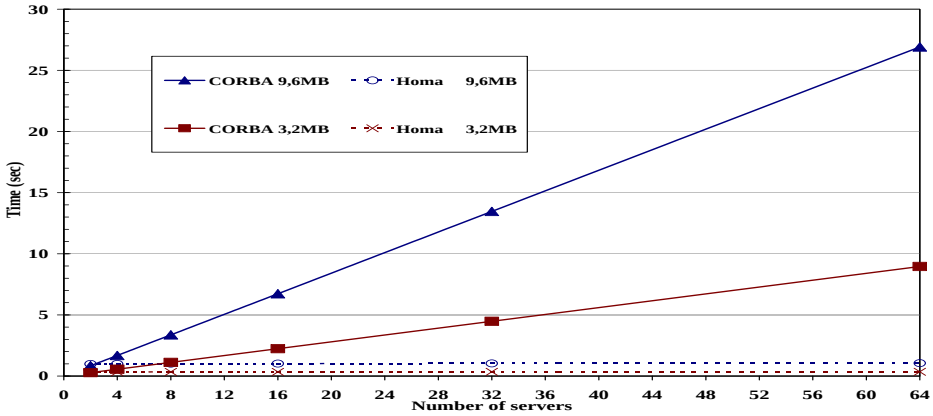


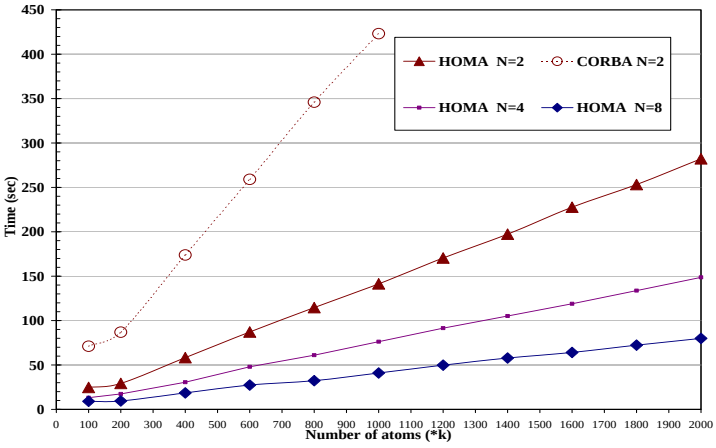
Fig. 4. Execution time of code figure 3.

Using HOMA, the aggregate bandwidth linearly grows as the number of pairs, the maximum aggregate bandwidth is about $304MBytes/s \simeq 3Gbits/s$ for 32 servers communicating to 32 servers (64 servers). The average point-to-point bandwidth between two servers is about[4] $9.5MBytes/s$. In an other experiment, scalability of HOMA up to 120 servers was observed with little degradation in the average point-to-point bandwidth. So HOMA is able to exploit parallelism between method invocations and to manage parallel communications. On similar experiments where parallel objects may be applied and on a same fast ethernet network, published results for PACO [36,34] relates point-to-point bandwidth which varies from $9.1MBytes/s$ to $11.3MBytes/s$ depending on the version of the implementation, the architecture and the number of servers. Nevertheless, loops cannot be handle as in figure 3 (see section 3.3).

4.2 SIMBIO

This experiment measures the ability of HOMA to make parallel execution of invocations with complex dependencies. Figure 1 page 4 sketches the algorithm and associated data flow graph of SIMBIO application. All parameters of invocations are sequences of size M (number of atoms).

[4] Maximum point-to-point bandwidth with respect to MPI on TCP is about



Atoms	p		
	3	6	12
	Speedup		
100k	2.72	5.09	7.41
200k	2.98	5.00	9.15
400k	2.99	5.69	9.39
600k	2.98	5.40	9.46
800k	2.98	5.60	10.57

Fig. 5. Left: SIMBIO execution time with several objects/application. Right: HOMA gains versus CORBA.

In the experiments MD object is distributed to N_1 and CM to N_2 sub-objects of the same interface: each invocation is dispatched to several invocations on each sub-objects. Note that the control flow depends on the evaluation of `cmstep` which is only known at runtime. In this experiment $N_1 = 2N_2 = N$ and we choose `cmstep` to be true for each two iterations. All objects are mapped onto different processors.

Let us note by T_1 the work of SIMBIO application, T_∞ the critical path, $C_1 = O(M)$ the communication volume and by C_∞ the communication delay. At first approximation we can consider that algorithms used in SIMBIO are linear with respect to the number of atoms, thus $T_1 = O(M)$ and $T_\infty = O(M(\frac{1}{N_1} + \frac{1}{N_2}))$. This assumption leads to $T_p^{homa} = O(\frac{M}{N}(1+h))$ using our predictive model. Left side of figure 5 reports the time of 10 iterations. As the cost model analysis expects, there is a linear relation between the execution time using HOMA and object parallelization: the time decreases linearly as the number of processors increases. Right side of figure 5 presents the gain of HOMA versus CORBA for different number of atoms and processors $p = N_1 + N_2$. There are two reasons for subsidency of efficiency in growth of p. First, different number of sub-objects ($N_1 \neq N_2$) causes scatter/gather of messages from/to an object (in this experiment two sub-objects of MD versus one sub-object of CM) serialising the communications. As the second reason, increasing of granularity decreases message size per sub-objects so HOMA overhead becomes important. Our cost model may benefit from finer characterization of the communication using a machine model such as $logP$.

$11 MBytes/s$ on iCluster.

5 Related Works

Lazy Approach and Cache Strategy: In order to achieve the necessary throughput and latency in CORBA, we observed two groups of research. First, the attempts to provide an object replication framework specially to used over Internet like CASCADE [13], ScaFDOCS system [26]. The object replication approach is not efficient for the methods with large size arguments which is our target applications, the numerical simulations. Second, the researches for caching results of method invocations so that the subsequent method invocations will return locally cached values without invoking the remote operation. A very limited solution is provided by *smart stub* mechanism in some existing CORBA implementations (e.g. Orbix [24] and VisiBroker). But the burden of maintaining coherency lies entirely on the application programmer. MinORB [28] is a research ORB that allows caching partial results of read method invocations at the client side. However, to our knowledge, there is no attempt to automatically optimize communication on a sequence of invocations.

In [12] and [3], the concept of future is used in order to generate parallel flow of control between Java's remote method invocations. In [12], the main problem solved is the reduction of communications between a client and one server by aggregating invocations. Our work targets scalable architecture with hundreds of processors with predictible performance model. The way the data are cached onto servers also differs. In [12] aggregate of invocations are cached. In [3] input or output data are cached as well as in HOMA. Nevertheless, the knowledge of the data flow allows us to anticipate communications and use parallel algorithm to broadcast or reduce data. Moreover it does not require a general purpose cache coherent protocol.

Predictive cost model and algorithmic skeleton: Algorithmic skeleton [14,33,35] was introduced in order to design high performance algorithm by providing basic building blocks, called skeletons, which can be optimized on various parallel or distributed architectures. In [2,1] predictive performance models are computed by estimating time required to execute basic instruction of Java virtual machine. Our approach differs in the parameters of the predictive cost model. We use the work and the critical path of parallel programs to predict the time of execution on a p processor computers. Moreover, such cost model may be applied to any parallel algorithm without using skeletons. Nevertheless, skeleton approaches seems to reach better performances when potential parallelism is roughly equivalent to the available parallelism: using our approaches the cost of computing the schedule may introduce an overhead. This is the reason why HOMA implementation uses a skeleton to scatter data to a given set of processors (see section 3.1.3).

Aggregate Object for Parallel Codes: In CORBA object model, PARDIS [25]

and PACO [34] have studied the integration of data parallel application. OMG has recently published a specification to handle data parallel application [31]. PARDIS reports to the user the concept of *future* while our approach makes it use implicit. Neither PARDIS nor PACO provides an efficient support robust to the composition without rewriting interface to access to data. These researches are orthogonal with HOMA where the optimization of a general sequence of invocations is shown to be efficient and scalable. Nevertheless, HOMA makes few assumptions about the distributed nature of the objects or parameters and such parallel object extension should be easily reused.

Service-based Metacomputing: NetSolve [11], Ninf [38] and DIET [10] are the RPC-based client/agent/server systems that enable users to solve complex scientific problems remotely. Netsolve and Ninf developed their own runtime support for communication but DIET is based on CORBA technology. In Netsolve and Ninf, Request Sequencing [16] enables programmers to demand optimization of communication on a sequence of requests. But it is explicit and not transparent. There is no support to efficiently exploit data parallel objects.

6 Conclusion

This paper presents our approach for an efficient and scalable composition of distributed applications viewed as composition of CORBA method invocations. To fulfill this goal we present how to build the data flow graph of the execution and how to exploit it to extract parallelism using ATHAPASCAN and to avoid unnecessary communication. Together these features permit to several servers to communicate in parallel.

Moreover we show that the cost model of ATHAPASCAN is inherited in HOMA and we present the theoretical gain over a standard CORBA environment. Some experiments validate our analysis. The techniques could be automatically used and we have developed an initial prototype. HOMA provides a way to program computational grid with CORBA as a standard from the industry, using automatic code transformation.

Ongoing works are to study scheduling heuristics in the case where a task in our graph could be dispatch on any server of a set in order to minimize the communication (the problem is NP-hard). The target application is numerical simulation in computer aided process engineering [4]. Moreover, technical work currently deals with trying to reduce the overhead of our implementation by an efficient cache; avoiding the extra copy needed for generic cache data type (CORBA `Any` data type).

To the knowledge of the authors, HOMA's work is the first attempt to con-

sider the efficiency and the scalability of the composition of CORBA method invocations for high performance applications in numerical simulation.

References

[1] M. Alt, H. Bischof, and S. Gorlatch. Algorithm Design and Performance Prediction in a Java-Based Grid System with Skeletons. In Burkhard Monien and Rainer Feldmann, editors, *Euro-Par 2002*, volume 2400 of *LNCS*, pages 899–906. Springer-Verlag, August 2002.

[2] M. Alt, H. Bischof, and S. Gorlatch. Program Development for Computational Grids Using Skeletons and Performance Prediction. *Parallel Processing Letters*, 12(3):157–174, 2002.

[3] M. Alt, H. Bischof, and S. Gorlatch. Future-Based RMI: Optimizing Compositions of Remote Method Calls on the Grid. In *Euro-Par 2003*, volume 2790 of *LNCS*. Springer-Verlarg, 2003.

[4] J.-P. Belaud, B. Braunschweig, and M. Pons. Open software architecture for process simulation : The current status of cape-open standard. In *European Symposium on Computer Aided Process Engineering, ESCAPE-12*, pages 847–852, The Hague, The Netherlands, 2002.

[5] M.O. Bender and M. O. Rabin. Online scheduling of parallel programs on heterogeneous systems with applications to cilk. *Theory of Computing Systems*, 35(3):289–304, May 2002.

[6] P.E. Bernard and O. Coulaud. Parallel constrained molecular dynamics. *INRIA Lorraine, Project NUMATH, Research report RR-3868*, January 2000.

[7] G.E. Blelloch, P.B. Gibbons, G.J. Narlikar, and Y. Matias. Space-efficient scheduling of parallelism with synchronization variables. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 12–23, 1997.

[8] R.D. Blumofe and C.E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 1(27):202–229, 1997.

[9] D. Caromel. Towards a method of object-oriented concurrent programming. *Communication of the ACM*, 36:90–102, 1993.

[10] E. Caron, F. Desprez, F. Lombard, J.M. Nicod, M. Quinson, and F. Suter. A Scalable Approach to Network Enabled Servers. In B. Monien and R. Feldmann, editors, *Proceedings of the 8th International EuroPar Conference*, volume 2400 of *LNCS*, pages 907–910, Paderborn, Germany, August 2002. Springer-Verlag.

[11] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. In *Workshop of Vector and Parallel computing*, Manno, Switzerland, March 1997. SPEEDUP Society.

[12] K. Cheung Yeung and P. K. Kelly. Optimising java rmi programs by communication restructuring. In *Middleware 2003: ACM/IFIP/USENIX International Middleware Conference*, volume 2790. ACM, 2003.

[13] G. V. Chockler, D. Dolev, R. Friedman, and R. Vitenberg. Implementing a caching service for distributed corba objects. In *Proc. of IFIP/ACM International Conf. on Distributed Systems Platforms and Open Distributed Processing*, NY, USA, 2000.

[14] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press, 1989. Available at homepages.inf.ed.ac.uk/mic/Pubs.

[15] A. Denis, C. Pèrez, and T. Priol. PadicoTM: An open integration framework for communication middleware and runtimes. *Future Generation Computer Systems*, 19(4):575–585, May 2003.

[16] C. Arnold Dorian, Bachmann Dieter, and J. Dongarra. Request sequencing: Optimizing communication for the grid. In *Proceedings of 6th International Euro-Par Conference*, volume 1900, pages 1213–1222, Germany, 2000. LNCS, Springer Verlag.

[17] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. of the tenth annual ACM symposium on Theory of computing*, pages 114–118. ACM Press, 1978.

[18] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. *Intl J. Supercomputer Applications*, 2001.

[19] F. Galilée, J.L. Roch, G.G.H Cavalheiro, and M. Doreille. Athapascan-1: On-line building data flow graph in a parallel language. In *Proceedings of the IEEE International Conference on Parallel Architectures and Compilation Techniques, PACT'98*, pages 88–95, Paris, France, October 1998.

[20] T. Gautier and H.R. Hamidi. Automatic re-scheduling of dependencies in a rpc-based grid. In *Proceedings of 18th annual ACM International Conference on Supercomputing (ICS'04)*, pages 89–94, Saint-Malo, France, June 2004. ACM Press.

[21] T. Gautier and H.R. Hamidi. High performance composition of services with data dependencies on a computational grid. In *Proceedings of the International Conference on Parellel and Distributed Processing Techniques and Applications (PDPTA'04)*, pages 809–814, Las Vegas, USA, June 2004. CSREA.

[22] T. Gautier and H.R. Hamidi. Homa: automatic re-scheduling of multiple invocations in corba. *INRIA Rhône-Alpes, projet APACHE, Research report RR-5191*, May 2004.

[23] T. Gautier, R. Revire, and J.-L. Roch. Athapascan: API for Asynchronous Parallel Programming. Technical Report RR-0276, INRIA Rhône-Alpes, projet APACHE, February 2003.

[24] IONA. *Orbix Programming Guide*. IONA Technology Ltd., 1995.

[25] K. Keahey and D. Gannon. Pardis: A parallel approach to corba. In *Proceedings of the 6th International Symposium on High Performance Distributed Computing (HPDC '97)*, page 31. IEEE Computer Society, 1997.

[26] R. Kordale, M. Ahamad, and M. Devarkonda. Object caching in a corba compliant system. *USENIX Computing Systems Journal*, 9(4), 1996.

[27] I. Lopez, G.J. Follen, R. Gutierrez, I. Foster, B. Ginsburg, O. Larsson, and S. Tuecke. Using corba and globus to coordinate multidisciplinary aeroscience applications. In *Proc. of the NASA HPCC/CAS Workshop*, pages 15–17, 2000.

[28] P. Martin, V. Callaghan, and A. Clark. High performance distributed objects using caching proxies for large scale applications. In *Proceeding of the IEEE International Symposium on Distributed Objects and Applications (DOA'99)*, Edinburgh, Scotland, September 1999.

[29] S. Matsuoka, H. Nakada, M. Sato, and S. Sekiguchi. Design issues of Network Enabled Server Systems for the Grid. In *Grid Computing – GRID 2000*, volume 1971 of *LNCS*, pages 4–17. Springer-Verlag, December 2000.

[30] OMG. Corba component model. Technical report, formal/2002-06-65, 2002.

[31] OMG. Data parallel object. Technical report, formal/2002-06-65, 2002.

[32] M. Parashar, G. von Laszewski, S. Verma, J. Gawor, K. Keahey, and N. Rehn. A corba commodity grid kit. In *Concurrency and Computation: Practice and Experience, John Wiley and Sons*, 2002.

[33] S. Pelagatti. *Structured Development of Parallel Programs*. Taylor & Francis, 1998.

[34] C. Pèrez, T. Priol, and A. Ribes. A parallel corba component model for numerical code coupling. In Craig A. Lee, editor, *Proceeding of the 3nd International Workshop on Grid Computing*, LNCS, Baltimore, Maryland, USA, November 2002. Springer-Verlag.

[35] F. A. Rabhi and S. Gorlatch, editors. *Patterns and Skeletons for Parallel and Distributed Computing*. Springer, 2003.

[36] C. René and T. Priol. MPI code encapsulating using parallel CORBA object. *Cluster Computing*, 3(4):255–263, 2000.

[37] R. Revire, F. Zara, and T. Gautier. Efficient and Easy Parallel Implementation of Large Numerical Simulations. In *Proceedings of EuroPVM/MPI, Parallel Simulation (ParSim 2003)*, Venice, Italy, September 2003.

[38] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A network based information library for a global world-wide computing infrastracture. In *HPCN'97 (LNCS-1225)*, pages 491–502, 1997.

[39] D. Schmidt, A. Gokhale, T. Harrison, D. Levine, and C. Cleeland. Tao: A high-performance endsystem architecture for real-time corba. In *IEEE Communications Magazine feature topic issue on Distributed Object Computing*, February 1997.