A lightweight Java Taskspaces framework for scientific computing on computational grids

H. De Sterck[†], R.S. Markel[+], T. Pohl[‡], and U. Rüde[‡]

[†]*Department of Applied Mathematics, University of Colorado at Boulder, Campus Box 526, Boulder, CO 80309-0526, USA. Phone: +1-303-735-4165, fax: +1-303-735-6355, email: desterck@colorado.edu.*
[+]*Advansys, Inc., 4950 Meredith Way #202, Boulder, CO 80303, USA. Phone: +1-303-413-8695, email: Advansys@attbi.com.*
[‡]*Lehrstuhl für Informatik 10 (Systemsimulation), Department of Computer Science, University of Erlangen-Nürnberg, Cauerstraße 6, D-91058 Erlangen, Germany. Phone: +49-9131-85-28924, fax: +49-9131-85-28928, email: Thomas.Pohl@informatik.uni-erlangen.de, Ulrich.Ruede@informatik.uni-erlangen.de.*

Submitted to SAC03-PDSN03.
(Special track on Parallel and Distributed Computing and Networking, at the Eighteenth Annual ACM Symposium on Applied Computing, March 2003, Melbourne, Florida, USA.)

A lightweight Java Taskspaces framework for scientific computing on computational grids

Abstract

A prototype Taskspaces framework for grid computing of scientific computing problems that require intertask communication is presented. The Taskspaces framework is characterized by three major design choices: decentralization provided by an underlying tuple space concept, enhanced direct communication between tasks by means of a communication tuple space distributed over the worker hosts, and object orientation and platform independence realized by implementation in Java. Grid administration tasks, for example resetting worker nodes, are performed by mobile agent objects. We report on large–scale grid computing experiments for iterative linear algebra applications showing that our prototype framework scales well for scientific computing problems that require neighbor–neighbor intertask communication. It is shown in a computational fluid dynamics simulation using a Lattice Boltzmann method that the Taskspaces framework can be used naturally in interactive collaboration mode. The scalable Taskspaces framework runs fully transparently on heterogeneous grids while maintaining a low complexity in terms of installation, maintenance, application programming and grid operation. It thus offers a promising roadway to push scientific grid computing with intertask communication beyond the experimental research setting.

1 Introduction

Distributed computing on heterogeneous computational grids over intranets or over the internet is evolving fast from an experimental research topic into a competitive commercial endeavor. Especially for large computational problems of the 'task farming' type, for which no intertask communication is required, many effective software environments with varying degree of general applicability have been developed. SETI@home [15], Condor [4] and Entropia [5] are three prominent examples. For scientific computing problems that do require intertask communication, however, the situation is different: several concepts for grid computing of this type of applications, including the MPICH-G2/Globus approach [11, 8], are being investigated, but experiments remain largely confined within a research setting [1] because efficient grid software solutions that run fully transparently on heterogeneous grids while maintaining a reasonable complexity in terms of installation, maintenance, application programming and grid operation, have not been realized yet.

In this paper the design and prototype implementation is described of a Java–based lightweight Taskspaces framework for scientific computing on computational grids that addresses these needs.

Our Taskspaces framework is designed around three main concepts.

First, our framework is based on an underlying tuple space concept. Tuple spaces were pioneered in the late 1970s and were first realized in the Linda system and language [7]. In a tuple space distributed computing environment, processes communicate solely by adding tuples

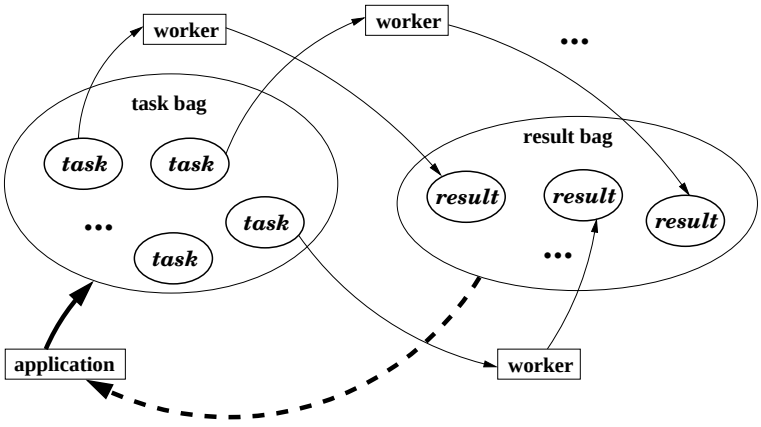

Figure 1: The tuple space paradigm for distributed computing.

to and taking them from a tuple space, a form of independent associative memory. A tuple is a sequence of fields, each of which has a type and contains a value. The tuple space model provides associative lookup: processes find relevant tuples by matching patterns in tuple field values. Fig. 1 shows conceptually how in such an environment an application program places subtasks, which result from the partitioning of a large computational problem, as tuples into a tuple space (which in the Bag-of-Tasks paradigm is called a 'task bag' [2]). 'Worker processes' take the task objects from the task space, execute the tasks, and place the result in a 'result bag'. The distributed computing process is decoupled in space, as the application, task and results bags, and the various worker processes may reside on a heterogeneous collection of machines that are connected by a network but that are otherwise widely geographically distributed. This allows for a flexible topology for the computation resulting from automatic configuration based on the availability of worker processes. Since spaces are persistent, tuples are persistent while resident in the space, and processes can access tuples long after the depositing process has completed execution, which provides for decoupling in time. The flexibility of topology and the decoupling largely eliminate the need for central control mechanisms and naturally provide for automatic scheduling and resource monitoring scenarios driven by the interchangeable worker nodes. Tuple spaces also naturally offer elegant solutions for fault tolerance, scalability and load balancing.

The second characteristic feature of our design is that we extend the tuple space concept in its pure form in three ways in order to significantly enhance grid applications. First, the access of information in our tuple spaces is governed by an event notification model instead of a polling mechanism. Worker processes register with the task bag which acts as a resource monitor, and as tasks get deposited in the task bag, they are sent to the registered workers for processing. Second, tuple space lookup proceeds via efficient direct indexing using a limited number of keys instead of by a full associative lookup mechanism. Third, a 'communication tuple space' is automatically distributed over the hosts on which worker processes reside, thus allowing for direct communication between worker hosts, while retaining the flexible network topology and automatic configurability of the tuple space paradigm. It has often been argued

that tuple space–based environments cannot be expected to be useful for scientific computing problems with intertask communication, because in a pure tuple space environment all the communication would pass through a central 'communication tuple space' with slow associative lookup mechanisms, which forms an inherent bottleneck that impedes scalability [12]. We show in the present paper how this communication bottleneck can effectively be removed by simply distributing the communication tuple space and thus allowing for direct communication between the worker nodes.

The third pillar of our design is to make our framework implementation object oriented and platform independent. We chose a Java implementation for our prototype implementation. Object orientation has many advantages. In our Taskspaces framework all application code is downloaded from a code server by 'dumb' generic workers. In our prototype implementation it suffices to simply augment the Java installation with a security policy and install and execute a Java bytecode executable of size $<$ 1kb to make any host participate in the grid. Task objects contain both data and code, and can be easily duplicated for checkpointing and fault tolerance purposes. Together with Java's built-in networking and security capabilities, this makes that our framework is truly lightweight compared to other approaches. Legacy code developed in other programming languages can be called. The decoupling provided by the tuple space largely eliminates the need for central control mechanisms. Some form of central control capabilities is desirable though, and is implemented through mobile agent Java objects that, e.g., may reset worker nodes. To enable the use of Java's built in security tools, the framework and application code are embedded within digitally signed Jar files. Using a combination of security policy files that limit local resource access and digitally signed Jar files enables a secure system to be constructed. Java's platform independence makes that our framework can run fully transparently on heterogeneous grids composed of any networked device on which Java is available.

Our framework is novel in that it combines three design concepts in a lightweight environment: the decentralized tuple space concept, efficient direct communication capabilities, and Java's platform independence and object orientation. We show in the applications presented in this paper that these design criteria are a good choice for grid computing of scientific computing problems with intertask communication. Some other existing grid computing frameworks may employ one or two of these three concepts, but we have not encountered any framework that combines all three concepts in an efficient way. For example, Entropia [5] and Condor [4] do not run fully transparently on heterogeneous grids and lack efficient direct communication mechanisms. MPICH-G2/Globus [11, 8] lacks the flexibility in topology provided by the decentralized tuple space concept, and is overly complex to install and operate fully transparently on heterogeneous grids. Parabon [14] and Ubero [17] exploit Java's networking, security and object capabilities, but lack the decentralization of the tuple space paradigm and lack efficient direct communication mechanisms. Sun's JavaSpaces [6] combines Java with tuple spaces. Experiments using JavaSpaces in a distributed scientific computing context are described in [12, 9]. We have found JavaSpaces to be overly complex and lacking efficient direct communication mechanisms. Similarly, JParadise/Linda [10] combines tuple spaces with Java, but does not address efficient direct communication.

In this paper we show that our design results in a lightweight prototype grid computing framework that is simple in terms of installation, maintenance, application programming and grid operation, and that provides an efficient grid computing solution for scientific computing applications that require intertask communication. It is demonstrated in large–scale grid computing experiments on parallel supercomputers and clusters that this distributed tuple space architecture with direct communication scales well for problems with neighbor-neighbor communication.

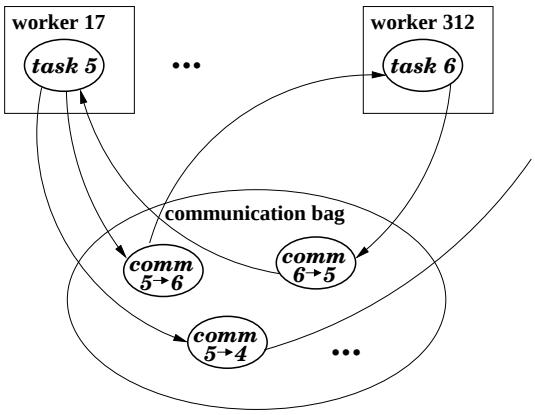

Figure 2: Communication through a central communication tuple space.

2 Intertask communication in the Taskspaces framework

2.1 Neighbor-neighbor communication through a communication bag

The most straightforward way to extend the concept of tuple space–based distributed computing as depicted in Fig. 1 such that it allows for communication between subtasks, is to add a global tuple space that acts as 'communication bag', see Fig. 2. Subtasks residing on different worker hosts can then exchange communication objects through the communication bag. This solution is faithful to the tuple space concept, but it is clear that communication through a central communication bag will be a bottleneck for parallel scalability when the number of tasks that need to communicate becomes large.

2.2 Direct communication between tasks

It is intuitively clear that by distributing the communication tuple space over the hosts on which worker processes reside in a way that allows for direct communication between worker hosts, this communication bottleneck can be removed. In our Taskspaces framework scalability is achieved by starting a local communication bag for every worker process, see Fig. 3. Communication is performed in two phases. In phase (1) IP addresses and port numbers that are required for direct communication in phase (2) are transmitted using a global communication

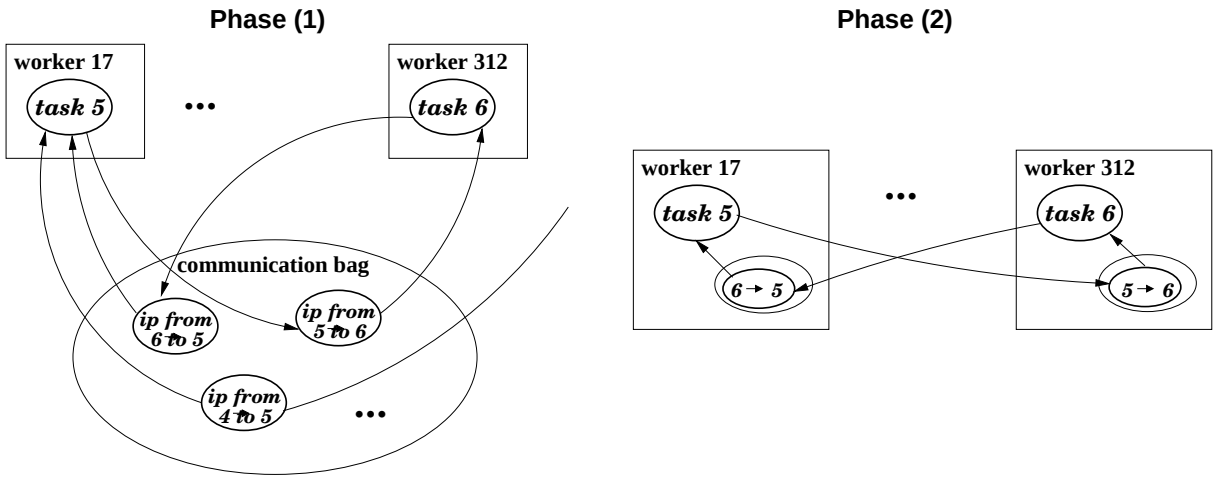

Figure 3: Tuple Spaces with direct communication. In phase (1) the communication pattern is set up by exchanging IP addresses via a central communication bag. In phase (2) direct communication ensues.

bag. For the applications envisaged, if task x needs to send a message to task y, then this will be known for both tasks at the time of program implementation, so a send–receive paradigm can be used for the communication. For messages to be sent in phase (2) from task x to task y, direct communication is set up in phase (1) as follows (Fig. 3). Task y sends a message object to the global communication bag containing the IP address of the host on which task y resides, along with the port on which task y's local communication bag listens. Task x reads this message object from the global communication bag, and thus receives the IP address and port of task y's communication bag, which are in phase (2) used by task x for direct communication. For applications of an iterative nature the communication pattern needs to be set up only once, and can then be used for direct communication in all the ensuing iteration steps. The way in which the communication pattern is set up in phase (1) retains the flexible network topology and automatic configurability of the tuple space paradigm, while the resulting direct communication pattern of phase (2) is in principle as efficient as the communication patterns used in MPI applications. Scaling results for the Taskspaces environment with direct communication are reported below in Sec. 4. It is important to mention here that global communication operations are not considered in this paper. For global communication, which intrinsically is more complicated and does not scale as well as neighbor-neighbor communication, a hierarchical multilevel communication pattern needs to be employed in order to achieve a useful degree of scalability [13]. Work on multilevel global communication for the Taskspaces framework is in progress.

2.3 The Taskspaces framework

A prototype Taskspaces framework has been implemented according to the above described design. Fig. 4 shows a simplified class diagram. All classes extend the Communicator class,

which contains the methods implementing communication. Communication between subtasks is performed by sending generic Objects over ObjectStreams associated with Sockets. Fig. 5 shows a deployment diagram. The Java classpath is extended to an http server from which framework and application classes are downloaded in digitally signed Jar files.

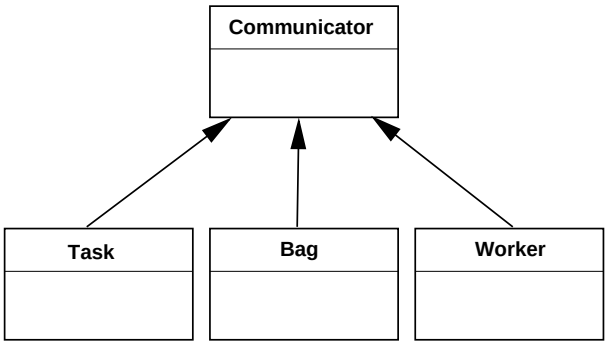

Figure 4: Taskspaces framework class diagram.

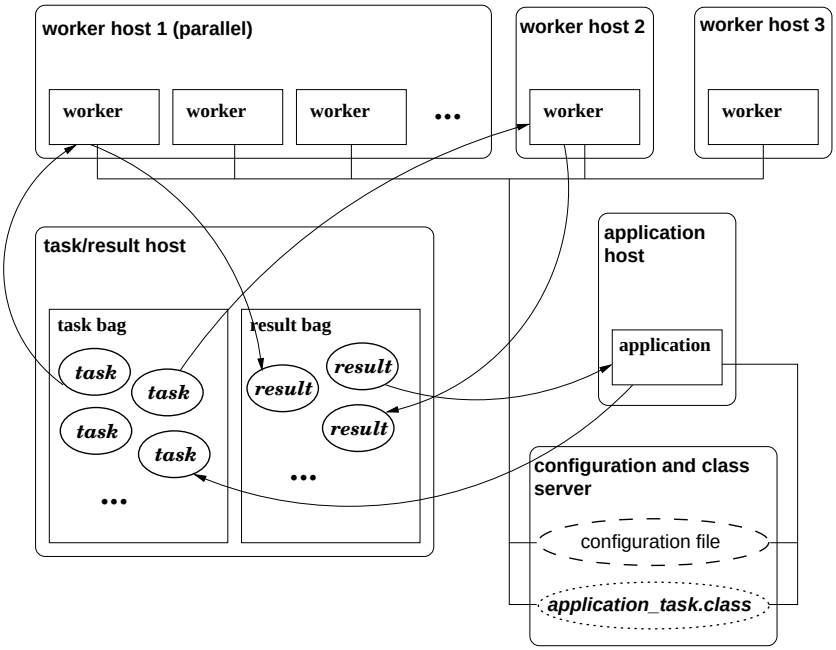

Figure 5: Taskspaces framework deployment diagram.

3 Scientific computing applications with intertask communication

The Taskspaces framework is tested for two scientific computing problems. In the first problem a linear system of algebraic equations deriving from finite difference discretization of a Laplace partial differential equation on a square two-dimensional domain is solved by the iterative Jacobi method [3]. The square domain with regular Cartesian mesh is divided into n^2 square subdomains of equal size that are assigned to n^2 tasks. Every task stores the unknowns that are part of its assigned subdomain, and every unknown is updated using its nearest neighbors. In every iteration step every task has to exchange the layers of unknowns that are adjoining the boundaries of the local domain in the four directions with its four direct neighbor tasks respectively. When the number of iterations is specified in advance, the Jacobi algorithm does not require any global communication. The Jacobi algorithm is not very useful by itself because it has complexity $O(N^2)$, with N the number of unknowns, but testing the scalability of the Jacobi algorithm is relevant because fixed numbers of iterations of similar algorithms are used as building blocks in multigrid methods for linear systems that achieve optimal $O(N)$ complexity [3]. The second problem is Lattice Boltzmann simulation of the Navier-Stokes equations for metal foam applications. In particular we have implemented the D2Q9-LBM algorithm as described in [18]. This is a real-life scientific computing problem that demands very high resolutions and high numbers of iterations, and thus requires the use of the vast resources available in computational grids. Our implementation of the D2Q9-LBM algorithm requires exactly the same partition and communication patterns as the Jacobi iterative method.

4 Scaling results

# of processors	# of grid points	direct communication	communication bag
1	1536^2	58s	58s
2x2	3072^2	74s	87s
4x4	6144^2	83s	216s
8x8	12288^2	89s	1504s

Table 1: Direct communication versus communication through a central communication bag (200 Jacobi iterations on Blue Horizon).

Table 1 compares total execution times using the central communication bag and using direct communication, for 200 iterations of the Jacobi algorithm with a problem size of 1536^2 unknowns per processor. These results were obtained on Blue Horizon, the IBM SP at the San Diego Supercomputer Center (SDSC), which features 375 MHz Power3 processors. For

the experiments reported in this paper the task, result and communication bags (Figs. 3 and 5) were located on login nodes of Blue Horizon, while the http server was located on a Linux PC at the University of Colorado at Boulder. It can be seen in Table 1 that communication through a central tuple space does not scale. Distributed tuple spaces that allow for direct communication, however, are scalable as the total execution time increases only slowly as a function of the number of processors. This is an important technical conclusion of this paper. As communication through a central communication bag does not scale on a single parallel computer, it will *a fortiori* not be useful in a grid computing environment, and will thus not be considered further in the grid computing experiments reported below.

# of processors	# of grid points	phase (1)	phase (2)
1	1500^2	–	20.59s
2x2	3000^2	0.35s	27.90s
3x3	4500^2	0.65s	30.20s
4x4	6000^2	1.29s	33.22s
5x5	7500^2	1.70s	34.12s
6x6	9000^2	2.49s	34.85s
7x7	10500^2	3.21s	34.66s
8x8	12000^2	3.63s	35.39s

Table 2: Direct communication (100 Jacobi iterations on Blue Horizon).

In Table 2 the scaling behavior of the direct communication solution on Blue Horizon is investigated in more detail. The tests described in Tables 1 and 2 were performed on non–interactive queues with dedicated access to processors. Setting up the communication in phase (1) does not scale, as this phase involves a central communication bag. For phase (2), the jump in execution time between 1 processor and 2x2 processors occurs because there is no communication required for execution on 1 processor. This increase can be reduced substantially by simply allowing for overlap between calculation and communication. The figures from the table allow to estimate that the calculation takes about 0.2s per iteration, while the communication takes about 0.07s per iteration. Every message contains 1500 doubles. In scaling up from 2x2 processors to 8x8 processors, one should theoretically expect that the execution time of phase (2) remains constant: indeed, the number of messages every task sends and receives is always four, the size of the messages remains constant, and the latency and bandwidth for direct communication can be assumed constant as well. Only slow growth of the phase (2) execution time is indeed observed. It is emphasized that the nearly optimal scaling behavior reported in Table 2 for phase (2) is expected to hold up for much larger numbers of processors, and also on heterogeneous computational grids (see below) as long as communication bandwidth does not become saturated.

Blue Horizon, SDSC, San Diego, CA	32	32	32	32	30	30	30	30
P4 Linux, CU Boulder, CO	–	–	–	–	2	2	2	2
Total number of workers	32	32	32	32	32	32	32	32
Total execution time	110s	104s	109s	106s	112s	119s	114s	116s

Table 3: Grid experiment (200 Jacobi iterations, 1500^2 grid points per processor). Number of processors (1 worker process per processor) and total execution times are shown.

In Table 3 it is investigated how the performance results obtained on a single parallel machine degrade when multiple machines are employed over the internet. For the Jacobi application, 200 iterations with 1500^2 grid points per processor are first performed on 32 Blue Horizon processors at SDSC in the interactive queue, followed by simulation of the same problem on 30 Blue Horizon processors connected over the internet with two 2.0 GHz Intel P4 processors on Linux workstations located at Boulder, CO. The P4 processors are faster than the Blue Horizon processors. Use of the San Diego-Boulder internet connection in stead of Blue Horizon's internal network results in an average performance degradation of only 7%. In general, performance results like this will depend on internet traffic conditions and connection bandwidth details. These particular experiments were performed under quiet internet traffic conditions (6:30am CO time on a Tuesday morning), but the P4 workstations are connected to the internet with only a low–bandwidth connection (10Mbit/s). Overall these test results show that it is feasible to perform simulations of this class of scientific computing problems on a grid with nodes connected by the internet. It can be expected that on grids with fast dedicated networks like the TeraGrid [16], even better results would be obtained.

Blue Horizon, SDSC, San Diego, CA (4 workers/processor)	64	128	240
P4 Linux, CU Boulder, CO (2 workers/processor)	4	4	4
Itanium Linux, CU Boulder, CO (2 workers/processor)	4	4	4
forseti1, NCSA, Urbana, IL (1 worker/processor)	16	16	16
hermod, NCSA, Urbana, IL (1 worker/processor)	16	16	16
Total number of workers	104	168	280
Total execution time	105s	103s	101s

Table 4: High throughput grid experiment (50 Jacobi iterations, 500^2 grid points per worker). Number of worker processes and total execution times are shown.

Table 4 shows large–scale experiments on a computational grid in a typical high throughput setting. The grid is composed of a variety of parallel computers and clustered workstations, including Blue Horizon at SDSC (interactive queue), two SGI Origin2000s at the National Center for Supercomputer Applications (NCSA) in Illinois (hermod and forseti1, with 250

MHz processors, dedicated queue), two Linux PCs with 2.0 GHz Intel P4 processors, and two Linux servers with 733 MHz Intel Itanium processors, located at the University of Colorado at Boulder. In the high throughput setting the tasks are divided in smaller parts (500^2 grid points per task), but several worker processes may be assigned to each processor in accordance with the CPU speed and workload of each processor. For the experiments in Table 4 we have chosen such a distribution by hand to demonstrate the concept. Work is in progress to automate this process based on processor capacity and load. The table shows optimal scaling, as the total execution time does not increase when the number of worker processes is increased from 104 workers over 168 workers to 280 workers. These results also suggest that optimal scaling would hold up for far more worker processes, but this is hard to test because access to large numbers of processors on different machines at the same time is hard to obtain within the context of queueing strategies presently employed at SDSC and NCSA. The results in Table 4 represent single experiment runs without averaging that takes into account network and server load fluctuations. These particular experiments were performed under quiet internet traffic conditions (Sunday afternoon). Surprisingly, the total execution time even goes down slightly when scaling up, which must be attributed to small network fluctuations. Here too it has to be noted that in general performance results will depend on internet traffic conditions and connection bandwidth details. See [1] for details on the internet path between SDSC and NCSA. From Table 2 we learned that the Taskspaces framework scales well on a single parallel machine. Table 4 shows that the Taskspaces framework also scales well for problems with neighbor–neighbor intertask communication on a geographically distributed heterogeneous computational grid connected by the Internet. *A fortiori,* it can be expected that on grids with fast dedicated networks like the TeraGrid [16], these scaling results would hold as well.

5 Interactive collaboration mode of the Taskspaces framework

Fig. 6 shows how the Taskspaces framework can be used in interactive collaboration mode. The Figure shows a Lattice Boltzmann Navier-Stokes simulation of gas flow in a driven cavity with obstacles [18] (the black dot in the W1,2 window is an obstacle). The full cavity domain is distributed over four workers that can reside on workstations anywhere in the world. While the simulation is running, human operators at the workstations can interactively add obstacles in the window on their workstation. The influence of all the added obstacles on the whole flow field is communicated to all the domains residing on the various workstations by the framework simulation, in the process of exchanging boundary information between neighboring processes in every iteration step.

Due to the decentralized and flexible design of our framework combined with the interactive capabilities of the Java language, it is simple and natural to set up such collaborative environments for parallel computing tasks with communication. This test example shows that our framework design is versatile and well suited to set up interactive collaborative environments. It is easy to imagine broad applications, e.g. engineers located at various locations can

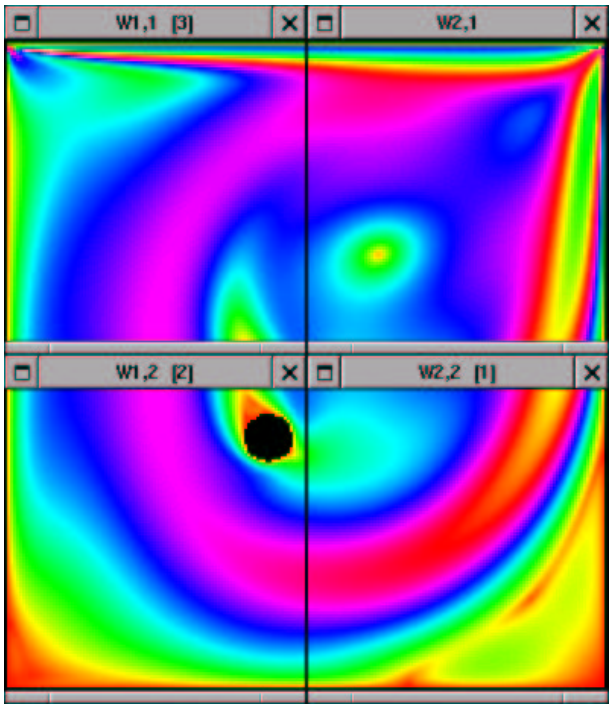

Figure 6: Density variations in a Lattice Boltzmann Navier-Stokes simulation of gas flow in a driven cavity with obstacles. While the flow field is being calculated by the framework, human operators can interactively add obstacles on every worker machine. The influence of all the added obstacles on the whole flow field is naturally communicated by boundary information exchange in every time step.

collaborate on aircraft design by interactively changing their assigned components while the framework calculates the aerodynamic flow around the entire aircraft.

6 Conclusions

In this paper it was demonstrated by large–scale grid computing experiments with up to 280 worker processes that our prototype platform independent and object oriented Taskspaces framework, based on tuple space concepts enhanced with efficient direct communication, scales well for scientific computing problems that require neighbor–neighbor intertask communication, while retaining the flexible network topology and automatic configurability of the tuple space paradigm. This shows that the Taskspaces architecture is promising for scientific grid computing problems that require intertask communication. Work is in progress on adding full fault tolerance, checkpointing, and automatic scalability and load balancing capabilities to our prototype framework, as well as efficient full message passing capabilities with multilevel global communication [13]. A fully implemented Taskspaces architecture has the potential to become a simple, elegant and lightweight realization of the functionality offered by more elaborate ap-

proaches for grid computing, e.g. the use of MPICH-G2 with Globus [11, 8, 1]. In addition, our Java tuple space design for grid computing has several important advantages over those other approaches, including a flexible network topology, automatic configurability, code downloading, ease of installation and maintenance, platform independence, and interactive collaboration capabilities. While the Taskspaces concept is by design not expected to deliver optimal performance on separate parts of computational grids, it may offer an attractive alternative to other ways of doing grid computing, especially in the context of user transparent high throughput grid computing for scientific computing problems that require communication. Our scalable Taskspaces framework runs fully transparently on heterogeneous grids while maintaining a low complexity in terms of installation, maintenance, application programming and grid operation. It thus offers a promising roadway to push scientific grid computing with intertask communication beyond the experimental research setting.

References

[1] G. Allen, T. Dramlitsch, I. Foster, T. Goodale, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In *Proceedings of SC 2001, November 10-16, 2001, Denver, Colorado*, 2001.

[2] G. R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison Wesley, Boston, 2000.

[3] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2000.

[4] Condor project homepage, http://www.cs.wisc.edu/condor.

[5] Entropia homepage, http://www.entropia.com.

[6] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces: Principles, Patterns, and Practice*. Addison-Wesley, New York, 1999.

[7] D. Gelertner. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[8] Globus project homepage, http://www.globus.com.

[9] K. A. Hawick and H. A. James. Dynamic cluster configuration and management using JavaSpaces. In *Proc. IEEE Cluster*, 2001.

[10] JParadise/Linda. Scientific Computing Associates, Inc. homepage, http://www.lindaspaces.com.

[11] MPICH-G2 homepage, http://www.hpclab.niu.edu/mpi/g2_body.html.

[12] M. Noble and S. Slateva. Scientific computation with JavaSpaces. Technical report, Harvard-Smithsonian Center for Astrophysics, Boston University, Boston, 2001.

[13] P. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, San Francisco, 1996.

[14] Parabon homepage, http://www.parabon.com.

[15] SETI@home homepage, http://setiathome.ssl.berkeley.edu.

[16] TeraGrid project homepage, http://www.teragrid.org.

[17] Ubero homepage, http://www.ubero.com.

[18] D. A. Wolf-Gladrow. *Lattice-gas cellular automata and lattice Boltzmann models*. Springer-Verlag, Berlin, 2000.