Chapter Number

Using Open Source Desktop Grids in Scientific Computing and Visualization

Zoran Constantinescu[1], Monica Vladoiu[2]
[1]*ZealSoft Ltd.*,
[2]*PG University of Ploiesti*
Romania

1. Introduction

Scientific Computing is the collection of tools, techniques, and theories required to develop and solve on a computer, mathematical models of problems in science and engineering, and its main goal is to gain insight of such problems (Heat, 2002; Steeb et al., 2004; Hamming, 1987). Generally, it is difficult to understand or communicate information from complex or large datasets generated by scientific computing methods and techniques (computational simulations, complex experiments, observational instruments etc.). Therefore, support of Scientific Visualization is needed, to provide techniques, algorithms, and software tools that are necessary to extract and display properly important information from numerical data. Complex computational and visualization algorithms normally require large amounts of computational power. The computing power of a single desktop computer is not sufficient for running such complex algorithms, and, traditionally, large parallel supercomputers or dedicated clusters were used for this job. However, very high initial investments and maintenance costs limit the large-scale availability of such systems. A more convenient solution, which is becoming more and more popular, is based on the use of non-dedicated desktop PCs in a desktop grid computing environment. Harnessing idle CPU cycles, storage space and other resources of networked computers, to work together, on a particularly computational intensive application, perform this job. Increasing power and communication bandwidth of desktop computers provides for this solution as well.

In a Desktop Grid (DG) system, the execution of an application is orchestrated by a central scheduler node, which distributes the tasks amongst the worker nodes and awaits workers' results. It is important to note that an application only finishes when all tasks have been completed. The attractiveness of exploiting desktop grid systems is further reinforced by the fact that costs are highly distributed: every volunteer supports her resources (hardware, power costs and internet connections), while the benefited entity provides management infrastructures, namely network bandwidth, servers and management services, receiving in exchange a massive and otherwise unaffordable computing power. The typical and most appropriate application for desktop grid comprises independent tasks (no communication exists amongst tasks) with a high computation to communication ratio (Domingues, Silva & Silva, 2006; Constantinescu, 2008). The usefulness of desktop grid computing is not limited to major high throughput public computing projects. Many institutions, ranging from

academics to enterprises, hold vast number of desktop machines and could benefit from exploiting the idle cycles of their local machines. In fact, several studies confirm that CPU idleness in desktop machines averages 95% (Domingues, Silva & Silva, 2006; Constantinescu, 2008; Vladoiu & Constantinescu, 2009b).

In the work presented in this chapter, the central idea has been to build a desktop grid computing framework and to prove its viability by testing it in some scientific computing and visualization experiments. We present here QADPZ, an open source system for desktop grid computing, which enables users from a local network or even Internet to share their resources. It is a multi-platform, heterogeneous system, where different computing resources from inside an organization can be used. It can also be used for volunteer computing, where the communication infrastructure is the Internet. QADPZ supports the following native operating systems: Linux, Windows, MacOS and Unix variants. The reason behind the native support for multiple operating systems, and not only for one (Unix or Windows, as other systems do), is that, often in real life, this kind of limitation restricts very much the usability of dekstop grid computing.

QADPZ provides a flexible object-oriented software framework that makes it easy for programmers to write various applications, and for researchers to address issues such as adaptive parallelism, fault-tolerance, and scalability. The framework supports also the execution of legacy applications, which for different reasons could not be rewritten, and that makes it also suitable for other domains as business. It also supports either low-level programming languages as C and C++ or high-level language applications, like for example Lisp, Python, and Java, providing the necessary mechanisms to use such applications in a computation. Consequently, users with various backgrounds can benefit from using QADPZ. The flexible, object oriented structure and the modularity of the system allows improvements and further extensions to other programming languages to be made easily.

We have developed a general-purpose runtime and an API to support new kind of high performance computing applications, and therefore to benefit from the advantages offered by desktop grid computing. We show how distributed computing grid extends beyond the master-worker paradigm, typical for such systems, and provide QADPZ with an extended API which supports in addition lightweight tasks creation and parallel computing, using the Message Passing Interface paradigm (MPI). The C/C++ programming language is directly supported by the API. QADPZ supports parallel programs running on the desktop grid, by providing and API in the C/C++ language, which implements a subset of the MPI standard. This extends the range of applications that can be used in the system to already existing MPI based applications, like for example parallel numerical solvers, from computational science, or parallel visualization algorithms. Another restriction of existing systems, especially middleware based, is that each resource provider needs to install a runtime module with administrator priviledges. This poses some issues regarding data integrity and accessibility on providers computers. The QADPZ system tries to overcome this by allowing the middleware module to run as a non-privileged user, even with restricted access, to the local system (Constantinescu, 2008; Vladoiu & Constantinescu, 2008a).

QADPZ provides also low-level optimizations, such as on-the-fly compression and encryption for communication. The user can choose from different algorithms, depending on the application, improving both the communication overhead imposed by large data transfers and keeping privacy of the data. The system goes further, by providing an experimental, adaptive compression algorithm, which can transparently choose different

algorithms to improve the application. QADPZ support two different protocols (UDP and TCP/IP) in order to improve the efficiency of communication (Constantinescu, 2008; Constantinescu & Vladoiu, 2009a). Free availability of the source code allows its flexible installations and modifications based on the individual needs of research projects and institutions. In addition to being a very powerful tool for computationally-intensive research, the open-source availability makes QADPZ a flexible educational platform for numerous small-size student projects in the areas of operating systems, distributed systems, mobile agents, parallel algorithms, and others. Moreover, free or open source software constitutes a natural choice for modern research, as well, because it encourages integration, cooperation and boosting of new ideas, in a very effective way (Cassens & Constantinescu, 2003; Constantinescu, 2008).

QADPZ has been built in top of an extended master-worker conceptual model. The extensions and refinements to the original master-worker paradigm concern mainly both the reduction of the time consumed for delays in communications and the increase of the period of time in which the workers perform computations. This goal is achieved by making use of different methods and techniques. The most important of them are as follows: pulling or pushing of work units, pipelining of the work-units at the worker, sending more work-units at a time, adaptive number of workers, adaptive timeout interval for work units, multithreading, redundant computation of the last work-units to decrease the time to finish, overlapped communication and computation at both the workers and the master, and use of compression to trim down the dimension of messages (Constantinescu 2008; Vladoiu & Constantinescu 2008b). Our work has also shown that that the use of desktop grid computing should not be limited to only master-worker type of application, but it can be used also for more fine-grained parallel applications, in the field of scientific computing and visualization, by performing some experiments in those domains. Thus, we have used QADPZ in several experiments: geophysical circulation modelling, fluid flow around a cylinder, both simulation and visualisation and so on (Constantinescu, 2008; Constantinescu & Vladoiu, 2009a). It is worth to mention that to the present QADPZ has over a thousand four hundred users who have download it, and that many of them use it for their daily tasks. They constantly give feedback on the system, ask for support in using it for their research and development tasks, or discuss about it in specialized discussion forums. QADPZ is also actively used in universities where students get work assignments based on its features (Constantinescu, 2008; QADPZ, 2010).

The chapter structure is as follows: the second section reveals other major desktop grid environments, such as BOINC, distributed.net, Condor and XtremWeb, along with examples of both desktop grids generated with their support and projects that use these environments. The third section describes briefly the QADPZ system. It starts with a justification for our endeavour to build a new DG system, and continues with the main capabilities of QADPZ, and with the improvements of the master-worker model implemented in the system. Within the next section, some scientific and visualization experiments performed with QADPZ's support are presented: geophysical circulation modelling within the Trondheim fjord, fluid flow around a cylinder – simulation and fluid flow around a cylinder – visualization. In the last section we will present some conclusions and some future work ideas that aim to improve both the conceptual model and the QADPZ system. We also invite the interested enthusiastic developers in the open-source community to join our development team and we appreciate any kind of feedback.

2. Related work

In high-throughput computing or desktop grid systems, pooled resources are used to achieve high throughput on a set of compute jobs. Such systems permit large numbers of machines to be added as a single resource in a higher-level grid system, achieving this way significant benefits in reduced management effort and grid complexity (Foster & Kesselman, 2003). Desktop Grids (DGs) evolve in two major directions: institution- or enterprise-wide desktop grid computing environment, and volunteer computing. The former, usually called simply desktop grid (or local DG, or LAN-based DG), refers to a grid infrastructure that is confined to an institutional boundary, where the spare processing capacity of an enterprise's desktop PCs are used to support the execution of the enterprise's applications. User participation in such a grid is not usually voluntary and is governed by the enterprise's policy. Applications like CONDOR, Platform LSF, DCGrid and GridMP are all such examples (Mustafee & Taylor, 2006).

The later is an arrangement in which volunteers provide computing resources to various projects that use those resources to perform distributed computationally intensive applications and/or storage. Volunteers are typically members of the general public who own Internet-connected PCs. Organizations such as universities and businesses may also volunteer the use of their computers. Projects are generally academic and are concerned with scientific research. Though, there are notable exceptions, such as GIMPS (GIMPS, 2010) and distributed.net (distributed.net, 2010), two major projects that are not academic. In fact, GIMPS (Great Internet Mersenne Prime Search), which started in 1996, has been the first volunteer computing project ever (BOINC, 2010). Other early projects include distributed.net, SETI@home (Seti@home Classic, 2010), and Folding@home (Folding@home, 2010).

Several aspects of the project/volunteer relationship are worth to be noticed (BOINC, 2010): (1) volunteers are truly anonymous - even though they may be asked to register and supply information as the email address, there is no way for a project to link them to a real-world identity; (2) due to their anonymity, volunteers are not accountable to projects – for example, if a volunteer misbehaves in some way, the project cannot take any action against that volunteer; (3) volunteers are ought to trust that projects will provide applications that will not damage their computers or invade their privacy, will be truthful about both work that is performed by its applications and use of the resulting intellectual property, and will follow proper security practices, so that the projects will not be used as vehicles for malicious activities. Desktop grid differs from volunteer computing in several aspects. First, the computing resources can be trusted, namely one can assume that the workstations do not usually return results that are wrong either intentionally or due to hardware malfunction, and that they do not falsify credit. Hence there is typically no need for redundant computing. Moreover, there is no need for using screensaver graphics and, in fact, it may be desirable that the computation is completely invisible and out of the control of the PCs' users. Furthermore, client deployment is typically automated (BOINC, 2010).

2.1 SETI@home - BOINC

SETI, or the Search for Extraterrestrial Intelligence, is a scientific effort seeking to determine if there is intelligent life outside Earth. One popular method that the SETI researchers use is *radio SETI*, which involves listening for artificial radio signals coming from other stars. Previous radio SETI projects have used special-purpose supercomputers, located at the telescope, to do the bulk of data analysis. In 1995, a new idea was proposed to do radio SETI

using a virtual supercomputer composed of large numbers of Internet-connected computers (SETI@home Classic, 2010). SETI@home, developed at the University of California in Berkley, is a radio SETI project that allows anyone with a computer and an Internet connection to participate. The method they use to do this is based on a screen saver that can go get a chunk of data from a central server over the Internet, analyse that data, and then report the results back. When the computer is needed back by its user, the screen saver instantly gets out of the way and it only continues its analysis when the computer is not used anymore. The program that runs on each client computer looks and behaves like a captivating screen saver. It runs only when the machine is idle, and the user can choose from several different colourful and dynamic "visualizations" of the SETI process. The data analysis task can be easily broken up into little pieces that can all be worked on separately and in parallel. None of the pieces depends on the other pieces, which makes large deployment of clients and computations over the Internet very easy.
The SETI@home project has not been evolving without problems. For all the media attention and public interest that it has got, funding has not been forthcoming. The SETI@home project has been devised for a very specific problem, as described above. There was no general framework for the system, which could have been used by other types of applications, and, therefore it became SETI@home Classic. When new funding has become available for the BOINC project, the SETI@home project was rewritten for the new framework and it became the new SETI@home/BOINC in 2005. BOINC is open-source software that can be used for both volunteer computing and desktop grid computing. It presents the following features: project autonomy, volunteer flexibility (flexible application framework, security, server performance and scalability), source code availability, support for large data, multiple participant platforms, open and extensible software architecture, and volunteer community related characteristics. BOINC has been designed to support applications that have large computation requirements, large storage requirements, or both. The main requirement of the application is that it shall be divisible into a large number (thousands or millions) of jobs that can be executed independently. In addition, to benefit from volunteered resources, a project needs to be appealing for the general public and needs to have a low data/compute ratio (BOINC, 2010).
As a *quasi-supercomputing platform*, BOINC has about 495,000 active host computers worldwide that process on average about 3 PetaFLOPS per day as of October 2010 (Wikipedia, 2010a; BOINC stats, 2010), which is more than the processing power of the fastest existing supercomputer system (Jaguar - Cray XT5), with a sustained processing rate of 1.759 PetaFLOPS (Wikipedia, 2010; Meuer, 2010).
There are many BOINC-based projects that are active around the world. Some are based at universities and research laboratories, while others are run by companies or individuals. The research areas of the problems to be solved vary from mathematics (e. g. ABC@home, NFS@home, SZTAKI Desktop Grid, Rectilinear Crossing Number, primaboinca etc.), cryptography (e. g. Enigma@home and PrimeGrid), biology and medicine (e. g. GPUGrid.net, POEM@HOME, SIMAP, docking@home, Malariacontrol.net etc.), to earth sciences (e. g. Climateprediction.net and Virtualprairie), astronomy (e.g. SETI@home, Cosmology@Home, Orbit@home, Milkyway@home etc.) and so on (BOINC, 2010).

2.2 distributed.net

A very similar project is the distributed.net project (Constantinescu, 2008; distributed.net, 2010). It takes up challenges and run projects which require a lot of computing power.

The collective computing projects that have attracted the most participants have been attempts to decipher encrypted messages. RSA Security, a commercial company has made public a number of cryptographic puzzles, and has offered cash prizes for those who would have solved them. The company's aim has been to test the security of their own products and to demonstrate the vulnerability of the encryption schemes they considered inadequate (Hayes, 1998).
Another type of project, which involves a lot of computing power, is the Optimal Golomb Ruler (OGL). Essentially, a Golomb Ruler is a mathematical term given to a set of whole numbers where no two pairs of numbers have the same difference. An Optimal Golomb Ruler is just like an everyday ruler, except that the marks are placed so that no two pairs of marks measure the same distance. OGRs can be used in various real world situations, including sensor placements for X-ray crystallography and radio astronomy. Golomb rulers can also play an important role in combinatorics, coding theory and communications. The search for OGRs becomes exponentially more difficult as the number of marks increases, being a *NP complete* problem (distributed.net, 2010).
The focus of the distributed.net project is on very few specialized computing challenges. Furthermore, the project only releases the clients' binary code and not the server code, making impossible the adaptation of this to other types of projects.

2.3 Condor

Condor is a High Throughput Computing (HTC) environment that can manage very large collections of distributively owned available computing resources. It is being developed at the department of Computer Science, University of Wisconsin, Madison. Condor delivers large amounts of computational power over a long period of time, usually weeks or months. In contrast, High Performance Computing (HPC) environments deliver a tremendous amount of compute power over a short period of time. In a high throughput environment, scientists and engineers are more interested in how many jobs they can complete over a long period of time instead of how quick an individual job can complete. Hence, HTC is more concerned to efficiently harness the use of all available resources (Condor, 2010).
The Condor environment has a layered architecture that provide for a powerful and flexible suite of resource management services for sequential and parallel applications. Condor is a workload management system that is focused on computationally intensive jobs, and offers a job queuing mechanism, a scheduling policy, a priority scheme, and resource monitoring and management. Users can submit their serial or parallel jobs to the system, which places them into a queue, schedule them to be run based on a policy, carefully monitors their progress, and finally informs the user upon completion (Constantinescu, 2008; Condor, 2010).
Condor provides a powerful resource management mechanism by matchmaking resource owners with resource consumers. This is crucial to any successful HTC environment. This mechanism implements a very elegant design, called ClassAds, that works similarly to the newspaper classified advertising want-ads. All the machines in the system's pool advertise in a resource offer ad their resource properties, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, current load average etc. When submitting a job, a user needs to specify a resource request ad, which defines both the required and a desired set of properties that are needed to run that job. Hence, Condor acts as a broker by matching and ranking resource offer ads with resource request ads, and by making certain that all the requirements in both ads are fulfilled. During this process,

Condor also considers several layers of priority values: the priority assigned to the resource request ad by the user, the priority of the user which submitted that ad, and the desire of the machines in the pool to prefer certain types of ads over others (Condor, 2010).
There are various Condor-based grids around the world, for example Greedy@EFPL, a desktop grid platform (called Greedy Pool) that runs at Federal Polytechnic School of Lausanne, and supports various mono-processors jobs (Grid@EFPL, 2010; Thiemard, 2008; Jermini, 2010), and a very large condor pool available at University College of London, which is used for several UK eScience projects, such as eminerals within molecular simulations of environmental issues domain (Wilson, 2004; eMinerals, 2010). Condor is used also at his "home university" from Wisconsin in several scientific projects such as genomics, IceCube Neutrino Observatory, Collider Detector at Fermilab etc. (Paradyn/Condor Week Presentations, 2008). Another interesting approach consists in using Condor inside a Blackboard learning management system for archiving Blackboard courses, reducing this way the time needed by 65% (Hoover, Dobrenen & Trauner, 2009).

2.4 XtremWeb, a open source platform for desktop grids

XtremWeb is a open source software that provide for build lightweight desktop grids by gathering the unused resources of desktop computers (CPU, storage, network), which is developed by IN2P3 (CNRS), INRIA and Universisty Paris XI. The working principle is that any participant can volunteer his computing resources, as in BOINC, but can also use other participants' available resources. Thus, participants may register new applications and data and may submit new computational jobs, provided that they are entitled to (advances in DGs, 2010). XtremWeb allows multi-users, multi-applications and cross-domains deployments, by harnessing volatile resources spread over LAN or Internet and putting them to work as a runtime environment that executes highly parallel applications. XtremWeb may be customised for a wide range of applications (bag-of tasks, master/worker), of computing requirements (data, CPU or network-intensive) and of computing infrastructures (clusters, desktop PCs, multi-LAN) (XtremWeb, 2010).
The designers of XtremWeb place the system somewhere between *pure desktop grid system, a la Condor and pure Volunteer Computing system, a la BOINC* (XtremWeb, 2010). XtremWeb relies on a pull model (workers pull tasks from the server), whilst Condor relies on a push model (the matchmaker selects the best machine to run the job and push the job on it). Moreover, with XtremWeb, each user or node has the ability (if authorized) to submit a job. When using BOINC, only the projects are able to create tasks and insert new applications (XtremWeb, 2010).
Several deployments are available for XtremWeb, for instance the XtremWeb-CH project that aims at building an effective Peer-To-Peer system for high performance computing, based on a completely symmetric model where nodes are both providers and consumers, or XWHEP (XtremWeb for High Energy Physics), a desktop grid computing platform capable of deploying and run user applications on the available computinf resource. Various projects use these desktop grid platforms. For instance, NeuroWeb and Virtual EZ Grid in medical domain, and From-P2P in public computing are all based on XtremWeb-CH (XtremWeb-CH, 2010). XWHEP middleware is currently used by the DGHEP, EDGI, DEGISCO, IDGF, and EDGeS projects (XWHEP, 2010). It is worth to mention that there is also a constant endeavour to create an integrate grid infrastructure that seamlessly put together a variety of desktop grids based either on BOINC, or on XtremWeb, around the

world, within the EDGeS (Enabling Desktop Grids for e-Science) umbrella (EDGeS, 2010; Cardenas-Montes et al., 2008; He et al., 2010).

3. The QADPZ system

Using the idle computational resources from the existent desktop computers in a organization or connected to the internet is not a new idea, though the use of such distributed systems, especially in a research environment, has been limited because both the lack of supporting applications and the numerous challenges regarding security, management, and standardization. This section includes a brief description of the QADPZ system, starting with a justification for our endeavour to develop a new DG platform, and continuing with both the main QADPZ capabilities and the improvements of the master-worker conceptual model that is implemented in the system.

3.1 Why to develop a new desktop grid system?

A fair question to ask is why we have taken into consideration the development of a new desktop grid system instead of using an existent one. One reason is that many of the existing systems at the time when the QADPZ project started were highly specialized in a very limited set of computationally challenging problems, and hence they did not allow the execution of a general application. For example, SETI@home has been developed having one specific goal: the analysis of data from telescopes; similarly, distributed.net aimed to test the vulnerability of some particular encryption schemes. Moreover, at the time of the development, the source code of the available desktop grid systems was generally not available, therefore making difficult the extension or analysis of any new, non-standard application. Commercial systems such as Entropia, Office Grid and United Devices offered a lot of useful features, but they were not free. On the other hand, the available open source systems, like XtremWeb, BOINC, Condor, were limited in functionality. Furthermore, very few existing systems provided for making specific considerations with respect to the challenges of computationally intensive applications, especially those of scientific computing and visualization. Systems like BOINC and Bayanihan allowed only task parallelism, where there was no communication whatsoever between the running tasks during a computation, nevertheless most computationally intensive applications need such communication (Constantinescu, 2008; Vladoiu & Constantinescu, 2008a).

Many of today's networks are heterogeneous, thus requiring a distributed computing system with both support for various architectures and different type of operating systems. The Java language provides the incentives for such requirements, and many Java-based systems emerged: JXTA, Bayanihan, XtremWeb, Javelin. There were very few systems that supported different architectures and operating systems in native mode – e.g. Condor and BOINC. There were also systems, which run only on one type of operating system, either Windows or Unix, therefore limiting their usability in heterogeneous environments – for instance, Entropia. In addition, most of the existing systems usually had a complicated deployment procedure, requiring high-level, privileged access to the desktop computers and that made very hard to use such systems on a larger scale, and also complicated the further maintenance of computers – e.g. Condor and Globus Toolkit (Constantinescu, 2008; Vladoiu & Constantinescu, 2008a; Vladoiu & Constantinescu, 2009a).

3.2 QADPZ capabilities

When the QADPZ project has been started we had to come up with a set of capabilities that a desktop grid computing system should provide in order to successfully support computationally intensive applications. Back then, QADPZ has been foreseen as being friendly, flexible and suitable to a variety of needs. Therefore, we have created an open architecture able to evolve in pace with the needs and challenges of the continuously changing real world. The main functional capabilities of the system concern resource sharing and management, job management, heterogeneity, simple installation and maintenance, support for parallel programming, network support, autonomy, performance measurements, multi-project use, and on-line/off-line support (Constantinescu, 2008; Vladoiu & Constantinescu, 2009a).

The most important resources that need to be shared are the idle computational cycles of the desktop machines that contribute to the system. Other kind of resources, like storage space, may be shared as well. The system is able to manage efficiently the available shared resources; though, the owners of the desktop computers that share resources keep control of them, and are able to both define use policies and retract some resources on their will. The users may submit computational jobs to the system, which will be executed using the shared computational cycles; there is also possible to monitor and control job executions. The system may be deployed on a network of heterogeneous desktop computers, with different architectures (Intel, RISC, etc.) and different operating systems (UNIX type, Windows, Mac OS). It is the responsibility of the user who submits the jobs to provide the appropriate binary files for execution on different platforms. QADPZ is able to work both in a LAN environment and in the Internet. Two families of protocols are available for communication: TCP/IP and UDP/IP. The system supports different parallel programming paradigms, for example both task- and data-parallelism.

The system is easy to install on a large number of computers in a network, and further maintenance of the installed programs it is be minimal. QADPZ is able to deal with its own complexity, by supporting different autonomic features: self-healing, self-management, self-knowledge, self-configuration, self-optimisation (Constantinescu, 2003). QADPZ is able to provide information about its performances, which could be used for better usage of the available resources. It also provide for multi-project use. Many projects have downtime and shortage of tasks, and participation in multiple projects helps to cope with projects' downtime. The system provides support for both batch and interactive type of applications. In a batch setting, the user submits jobs that will be executed at a later time, when resources become available. In contrast, interactive jobs provide real-time feedback of the execution, and the user can inspect the partial result, and interact with the execution of the application.

The QADPZ user interface provides for monitoring and controlling of the behaviour of the system, and the programming interface allows different user applications to interact with the system. Both interfaces satisfy requirements with regard to personalization (different levels of access for various users, according to their personal skills and preferences), job management (a simple, platform independent, graphical user interface that allows submission, monitoring and control of different computational jobs), and resource sharing (a simple, intuitive graphical user interface, which allows the control of shared resources).

QADPZ complies also with non-functional requirements that mainly concern object oriented programming, for its well-known advantages, and open source development, which is a natural choice for modern research, as it encourages integration, cooperation and boosts new ideas (Cassens & Constantinescu, 2003; Constantinescu, 2008).

3.3 QADPZ architecture

The QADPZ system has a centralized architecture, based on the client-server model, which is the most common paradigm used in distributed computing. The paradigm describes an asymmetric relationship between two processes, one of which is the client, and the other is the server. Generally, applications based on this paradigm involve multiple clients, however they can involve one or multiple servers. In our case, the server manages the computational resources available from the desktop computers, by offering services, which can be used by other processes. *The client* is a process that needs those services in order to accomplish a certain work. It sends a request to the server, in which it asks for the execution of a concrete task that is covered by some particular service. Usually, the server carries out the task and sends back the result to the client (Constantinescu, 2008; Vladoiu & Constantinescu, 2009a).
In our situation, the server has two parts: a single master that accepts new requests from the clients, and multiple slaves, which handle those requests. The system consists of three types of entities: master, client, and slave (Fig. 1). Each computer that contributes with computing power to the system is called *a slave*, and runs a small background process in the form of a UNIX daemon or as a Windows system service. The process can be run with the privileges of an ordinary user, it does not need administrative rights. This process is responsible for reporting the computer's resources and the status to a central server, *the master*. It also accepts computational requests from the master, downloads the corresponding binaries and data files for the tasks, executes the task, and then uploads the result files when finished. The slave also downloads the task to be executed together with the input data, and starts the computation. The presence of a user logging into a slave computer is automatically detected and the task is killed or moved to another slave to minimize the disturbance of the regular computer users (Constantinescu, 2008; Vladoiu & Constantinescu, 2009a).

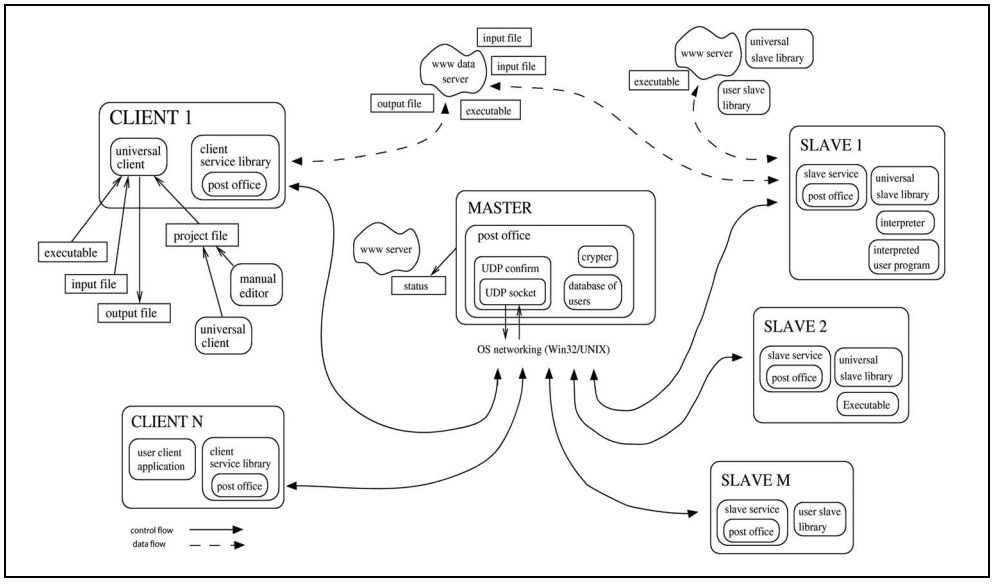


Fig. 1. The QADPZ close-up architecture

The control and data flow in the system are separated. The data files (represented by binary, input, and output files) that are necessary to run the applications are not sent to the master. They are stored on one or more data servers. The smallest independent execution unit of the QADPZ is called *a task*. To facilitate the management, multiple tasks can be grouped into *jobs*. Different types of jobs can be submitted to the system: programs written in scripting languages (e.g. LISP, Java, or Python), legacy applications and parallel programs (MPI). A job can consist of independent tasks, which do not require any kind of communication between them. This is usually called task parallelism. Jobs can also consist of parallel tasks, where different tasks running on different computers can communicate with each other. Inter-slave communication is accomplished using a MPI subset.
The current implementation of QADPZ considers only one central master node. This can be an inconvenience in certain situations, when computers located in different networks are used together. However, our high-level communication protocol between the entities allows a master to act as a client to another master, thus making possible to create *a virtual master* that consists of independent master nodes, which communicate with each other.

3.4 Improved master worker model

Our conceptual model is based on the master-worker paradigm that is built on the observation that many computational problems can be broken into smaller pieces that can be computed as one or more processes in parallel. That is, the computations are fairly simple consisting of a loop over a common, usually compute-intensive, region of code. The size of this loop is usually considered to be long. In this model, a number of worker processes are available, each being capable of performing any one of the steps in a particular computation. The computation is divided into a set of mutually independent work units by a master node. Worker nodes then execute these work units in parallel. A worker repeatedly gets a work unit from its master, carries it out and sends back the result. The master keeps a record of all the work units of the computation that it is assigned to perform. As each work unit is completed by one of the workers, the master records the result. Finally when all the work units have been completed, the master produces the complete result. The program works in the same way irrespective of the number of workers available - the master just gives out a new work unit to any worker who has completed the previous one (Constantinescu, 2008; Vladoiu & Constantinescu, 2008b).
The most important part of parallel programming is to map out a particular problem on a multiprocessor environment. The problem must be broken down into a set of tasks that can be solved concurrently. There are two different ways of decomposing a parallel problem, based on the way and time when the work-units are created: static or dynamic. In the case of s*tatic decomposition* the master generates all the work-units in the beginning of the computation, while in *dynamic decomposition* the computation starts with a small number of work-units, and later new work-units are created, depending on the results of already executed work-units. Moreover, the master can create or delete dynamically work-units. After decomposing the problem, the work-units need to be distributed to the work-units, or scheduled for execution. The key to making a parallel program work well is to schedule their execution so that the load is balanced between the processing elements. Distribution of work-units to the workers can be of two types: static or dynamic. In the former case, the master processor decides on the distribution of work at the start of the computation, by assigning the work-units to the workers, and in the latter the distribution of work-units varies between workers as the computation proceeds.

QADPZ uses an improved version of the master-worker model, which is based on an algorithm with dynamic decomposition of the problem and dynamic number of workers. The improvements regard the performance of the original model by increasing the time workers are doing computations, and decreasing the time used for communication delays. This is achieved by using different techniques, such as using pulling or pushing of work units, pipelining of the work-units at the worker, sending more work-units at a time, adaptive number of workers, adaptive timeout interval for work units, multithreading, redundant computation of the last work-units to decrease the time to finish, overlapped communication and computation at the workers and the master, and use of compression to reduce the size of messages (Constantinescu, 2008; Vladoiu & Constantinescu, 2008b).

In the original master-worker model, each time a worker finishes a work-unit, it has to wait until it receives the next work-unit for processing. If this communication time is comparable with the time needed for executing a work-unit, the efficiency of the worker is reduced very much. The master-worker model uses the pull technology, which is based on the *request/response paradigm*. The user (the worker) is requesting data from the publisher (the master). The user is the initiator of the transaction. In contrast, a push technology relies on the *publish/subscribe/distribute paradigm*. The user subscribes once to the publisher, and the publisher will initiate all further data transfers to the user. In this model, the worker doesn't send any more requests for work-units. Instead, it first announces its availability to the master when it starts, and the master is responsible for sending further work-units. The workers just wait for work-units, and process them when received. At the end of each work-unit, it sends back the results to the master. The master will further send more work-units to the worker. This moves all decisions about initiating work-units transfers to the master, allowing a better control and monitoring of the overall computation.

The worker efficiency increases if we reduce the total time spent in waiting for communication. One way to do that is to use work-units pipelining at the worker, thus making sure that the worker has a new work-unit available when it finishes the processing of the current work-unit. In the beginning, the master sends more than one work-units to the worker, then after each received result, sends another work-unit to be queued on the worker. The worker does not need to wait again for a new work-unit from the master after sending the result, the next work-unit being already available for processing. If the communication time is too large, the pipeline will eventually become empty and the worker will need to wait for new work-units. To overcome this situation, the master needs to send more than one work-units per each message. The master starts by sending a message containing more than one work-unit, and then keeps sending messages as long as the pipeline is not full. Each time it receives a result, it sends another work-unit, to compensate the decreasing number of work-units from the pipe. If the worker sends only one result per message back to the master, and the master sends only one new work-unit, then, eventually the pipeline will become empty. In order to prevent that, the worker will need to send back more results at a time.

In a heterogeneous environment based on idle desktop computers, the total number of workers available could be changing during the computation. New workers can register to the master, and other can become temporarily unavailable. The master controls the total number of workers used for computation, since it is the one sending out work-units to the workers. If necessary, the master can choose not to use all the available workers for computation, only a few of them. The master can become a bottleneck, either when there are a lot of workers, which connect to get work-units and send results back or it could be caused

by too much communication. QADPZ uses an adaptive algorithm to choose the number of workers, based on performance measures and estimates of execution times for work-units and communication times for sending and receiving messages. The number of workers is automatically reduced if the efficiency of the computation decreases. We employ a heuristic-based method that uses historical data about the behavior of the application. It dynamically collects statistical data about the average execution times on each worker.

4. QADPZ-supported scientific computing and visualization experiments

Scientific Computing (or Computational Science) is concerned with using computers to analyze and solve scientific and engineering problems, by constructing mathematical models and numeric methods and techniques to tackle the involved issues. The Scientific Computing's approach is to gain understanding, mainly through the analysis of mathematical models implemented on computers (Wikipedia, 2010b). As Richard Hamming has observed many year ago, "the purpose of computing is insight, not numbers" (Hamming, 1987). Scientific Computing programs often model real-world changing conditions, such as weather, air flow around a plane, automobile body distortions in a crash, the motion of stars in a galaxy, an explosive device, etc. Such programs might create a 'logical mesh' in computer memory where each item corresponds to an area in space and contains information about that space that is relevant to the model. For example in weather models, each item might be a square kilometre, with land elevation, current wind direction, humidity, temperature, pressure, etc. The program calculates the likely next state based on the current one, in simulated time steps, solving equations that describe how the system operates, and then it repeats the process to calculate the next state (wiki). Scientists and engineers develop software systems that implement the mathematical models of the studied systems and run these programs with various sets of input parameters. These models require massive amounts of calculations, usually floating-point, and need huge computing power to be run. Supercomputers, computer clusters, grid computing, or desktop grids can supply the necessary computational and storage resources.

Following further the idea of Hamming, Scientific Visualization could help overcome the dilemma of having information, but not the right interpretation for it. Interactive computing and visualization could provide an invaluable support during the scientific discovery process, as well as a useful tool for gaining insight into scientific anomalies or computational errors. Scientists and engineers need a more helpful alternative to numbers. Paraphrasing another well-known idea, *a image is worth a thousand .. numbers*, the use of images has become the needed alternative. The ability of scientists to visualize complex computations and simulations is essential to ensure the integrity of the analysis, to give insights, and to communicate about them with others. Scientific Visualization is concerned with presenting data to users by means of images, and seeks ways to help users explore, make sense of, and communicate about data. It is an active research areas, drawing on theory in information graphics, computer graphics, human-computer interaction and cognitive science (McCormick, 1987).

Some of the domains and directions in which Scientific Computation and Visualization are able to give valuable insight are listed here: engineering, computational fluid dynamics, finite element analysis, electronic design automation, simulation, medical imaging, geospatial, RF propagation, meteorology, hydrology, data fusion, ground water modeling, oil and gas exploration and production, finance, data mining/OLAP, numerical simulations,

orbiting satellites returning earth resource, military intelligence, astronomical data, spacecraft sending planetary and interplanetary data, earthbound radio astronomy arrays, instrumental arrays recording geophysical entities, such as ocean temperatures, ocean floor features, tectonic plate and volcanic movements, and seismic reflections from geological strata, medical scanners employing various imaging modalities, such as computed transmission and emission tomography, and magnetic resonance imagery.
Further on in this section, we presents some of the scientific and visualization experiments we have performed to prove the viability of the QADPZ system: a real world problem: geophysical circulation modeling within the Trondheim fjord, fluid flow around a cylinder – simulation and fluid flow around a cylinder – visualization.

4.1 Real world problem: Trondheim fjord

With the growing concern for environmental and ecological issues, and the increasing number of occurrences of serious pollution episodes, geophysical circulation modelling is a more and more important area of research. This relates to problems of different scales, from global issues to more local ones about water pollution in coastal areas, estuaries, fjords, lakes, etc. Analysis of these specific problems is based on the prediction of the flow circulation and transport of different materials, either suspended in water or moving along the free surface or the bottom. The numerical model of this complex phenomenon is based on a finite element formulation, because the finite element flexibility is beneficial for applications in restricted waters, where the topography is usually complex. The Navier-Stokes equations give the basic mathematical formulation (Utnes & Brors, 1993).
A map of Trondheimsfjorden, a typical Norwegian fjord that is located on the coast of central Norway, is shown in Fig. 3a. Detailed topographical data are used to interpolate the depth data to the element mesh. The horizontal element mesh is shown in Fig. 3b. It consists of 813 biquadratic elements with 3683 nodes, and there are 17 levels in the vertical direction with fine grading close to the bottom boundary. This grid is assumed to be detailed enough to describe the main flow field of the Trondheim fjord (Constantinescu 2008; Constantinescu & Vladoiu, 2009b). Shading the discretized cells according to the value of the scalar data

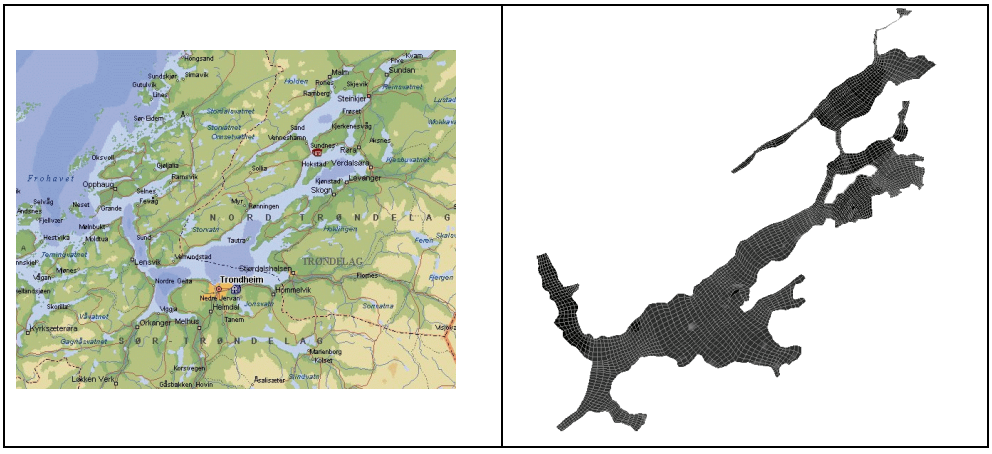


Fig. 3. a) Map of the Trondheim fjord b) Grid of the Trondheim fjord

field does colour coding. The colour allocation is linearly graduated for improved appreciation of continuum data. Using directed arrows also represents the velocity vector field (Figure 4 to Figure 5).

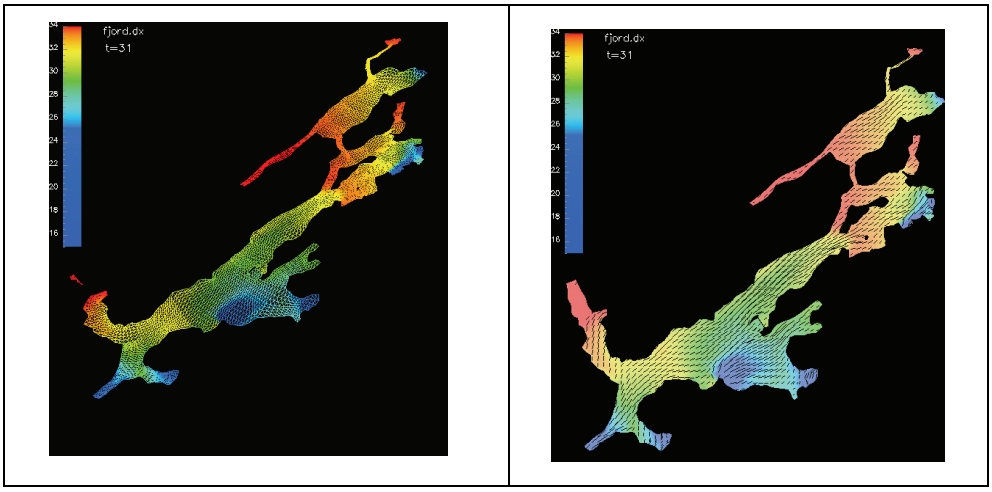


Fig. 4. Grid colored by salinity concentration (max 34 ppt - parts per thousand - mg/l)

Note. Color by salinity concentration - 3D model with isosurface representation

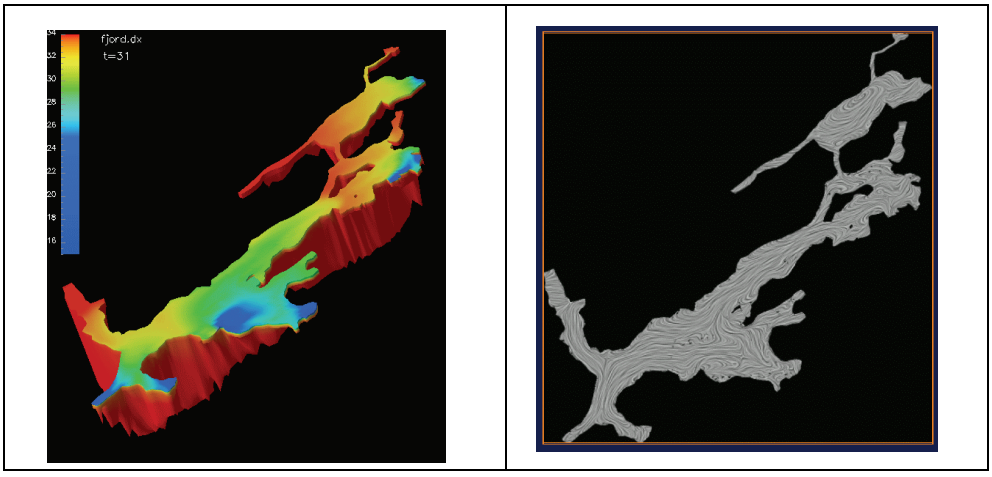


Fig. 5. a) Trondheim fjord model b) Velocity vector field - LIC representation

Large vector fields, vector fields with broad dynamic ranges in magnitude, and vector fields representing turbulent flows can be hard to visualize efficiently using common techniques such as drawing arrows or other icons at each data point or drawing streamlines. Drawing arrows of length proportional to vector magnitude at every data point can generate cluttered and confusing images. In areas of turbulence, arrows and streamlines can

be difficult to interpret (Merzkirch, W. 1987; Forsell, 1994; Smits & Lim, 2000). Line Integral Convolution (LIC) is a powerful technique for imaging and animating vector fields. The image is created beginning with a white noise that is then convoluted along integral lines of the given vector field. That creates a visible correlation between image pixels that lie on the same integral line. The local nature of the LIC algorithm recommends a parallel implementation, which could, in principle, compute all pixels simultaneously. This would provide interactive generation of periodic motion animations and special effects (Cabral & Leedom, 1993; Forssell & Cohen 1995).

4.2 Fluid Flow Around a Cylinder - Simulation

In this sub-section we present some experiences with running a typical Computational Fluid Dynamics problem in QADPZ environment. The numerical method for solving the incompressible Navier–Stokes equations is used in a test case for the system. An evaluation of the performance of the system is presented, by comparing it with running the same simulation on a typical dedicated cluster environment.

To solve the incompressible Navier–Stokes equations we have used a version of the well–known projection method, in which equations for the velocity components and pressure are solved sequentially at each time step. Solving the discretized, fully coupled equations can be very expensive due to the nested iterations, and this decoupling procedure is found to be computationally efficient for transient problems, particularly for higher Reynolds numbers. In general, the separation of pressure and velocity can be performed on both the continuous equations and the discretized equations. While the latter is attractive due to the straightforward interpretation of boundary conditions, these methods give a significantly more complicated pressure equation and we therefore prefer a splitting at the differential level. The Galerkin finite element method is used to discretize in space. The pressure is fixed in one node to ensure a unique solution, and has homogeneous Neumann conditions on all boundaries.

Implementation of the flow solver has been done in C++ using the object oriented numerical library Diffpack. In fact, a parallel version of Diffpack was used, which is based on the MPI standard for communication. Linux has been used as our development platform. An MPI library with a subset of the most used MPI calls has been implemented on top of QADPZ's communication protocol. This QADPZ-MPI library allows us to use QADPZ as a straightforward replacement communication system for the MPI based Diffpack library. A series of tests have been performed on a dedicated cluster of 40 PCs, with Athlon XP 1.46GHz CPU, 1 GByte memory, and 100 MBps network interconnection between nodes, running a Linux distribution. First, we used the original implementation of the solver's software (which uses MPICH as a parallel communication protocol) to run the cluster version of the simulation. Second, we recompiled the solver using the QADPZ-MPI library to create the distributed computing version of the simulation. The solver was run using exactly the same computers of the cluster (i.e. identical hardware setup). A third test case used a pool of 8 computers with similar hardware specifications, that were ordinary desktop PCs from our labs together with other computers connected to our LAN. Simulations were done in two different times of the day: during the night, when network traffic in the LAN is minimal, and during working hours, when the LAN traffic is significant.

The first set of results shows the simulation of some time steps of an oscillating flow around a fixed cylinder in three dimensions. The grid had 81600 nodes and was made of 8-node isoparametric elements, while the coarse mesh (used for pressure preconditions) had approximately 2000 nodes. The resulted execution times are presented in Fig. 6. We run the same simulation in three different parallel settings for the underlying MPI library: (1) the MPICH library from the Argone National Laboratory, (2) the QADPZ MPI library using LZO based compression for communication, and (3) the QADPZ MPI library using bzip2 based compression for communication. The second set of results presented here is from corresponds to simulating some time steps of an oscillating flow around a fixed cylinder in three dimensions. The grid had 307296 nodes and was made of 294912 elements (8-node isoparametric). A coarse grid was used for pressure preconditioning, which had 2184 nodes and 1728 elements. Three sets of simulations were done, using respectively, MPICH, PVM and QADPZ-MPI as communication library. For each set of simulation, a maximum of 16 processors (Athlon AMD 1.466 GHz) have been used. Running times were between around 240 minutes (1 processor) and 18 minutes (16 processors). Speedup results from these simulations, presented in Fig. 7, show that the performance of our system is comparable with other similar systems based on the message-passing interface.
The advantages of using our system are as follow: the installation of the slave on the computational nodes is extremely easy, only one executable file and one configuration file are needed; moreover, no root or administrator access is necessary. Upgrade of the slaves is done automatically from the master, without any administrator intervention. Also, there is no need for a shared file system, since each of the slaves is downloading by itself the necessary files. The slaves systems may have different operating systems, and there is no need for remote access to them (by using the rsh/ssh type of protocols).

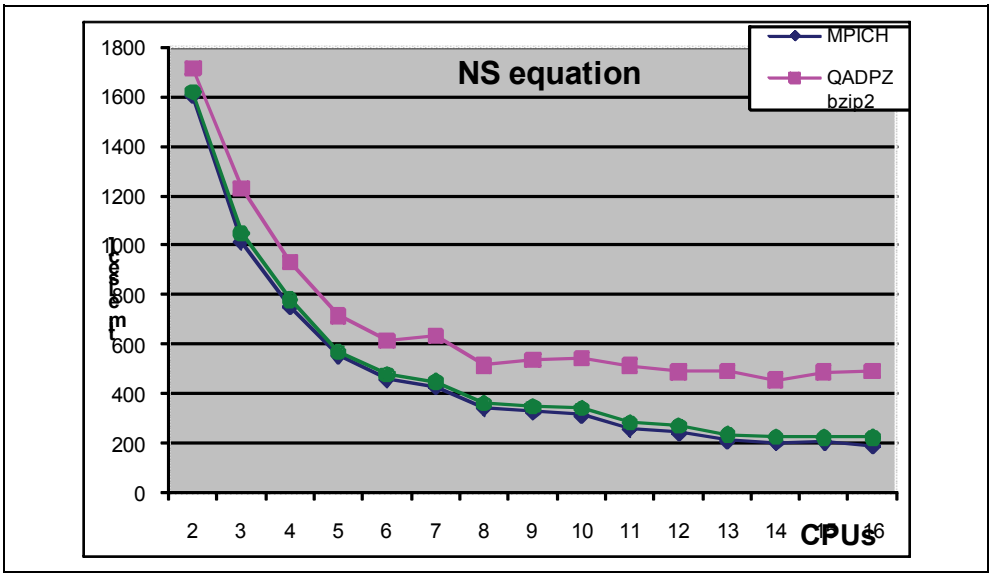
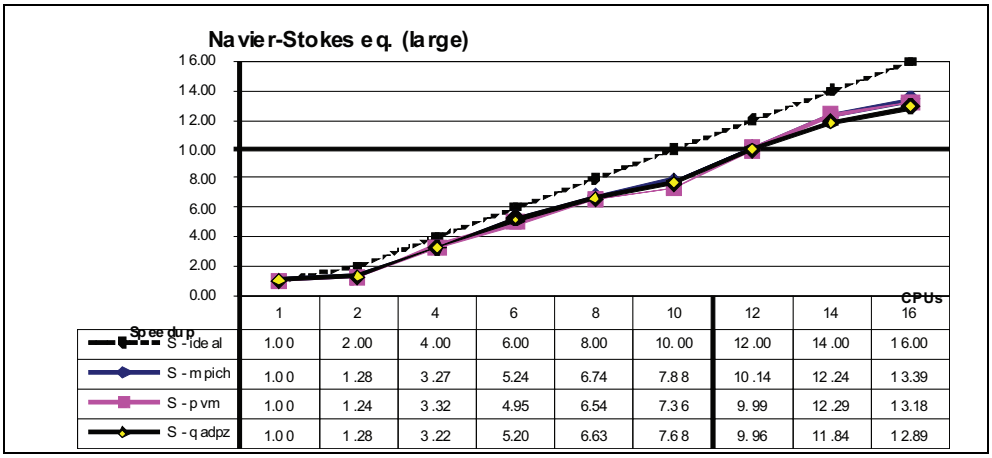


Fig. 6. Execution times for the solver



Speedup	1	2	4	6	8	10	12	14	16
S - ideal	1.00	2.00	4.00	6.00	8.00	10.00	12.00	14.00	16.00
S - mpich	1.00	1.28	3.27	5.24	6.74	7.88	10.14	12.24	13.39
S - pvm	1.00	1.24	3.32	4.95	6.54	7.36	9.99	12.29	13.18
S - qadpz	1.00	1.28	3.22	5.20	6.63	7.68	9.96	11.84	12.89

Fig. 7. Speedup for the simulation

4.3 Fluid flow around a cylinder - visualization

The results correspond to some time steps of an oscillating flow around a fixed cylinder in three dimensions. The second experiment is with three cylinders. The grid had 81600 nodes and was made of 8-node isoparametric elements, while the coarse mesh (used for pressure preconditioning) had approximately 2000 nodes. As in the previous case, numerical simulation was done by using the Navier Stokes equations. In the Fig. 8a the 2D domain grid is represented using triangular elements, and the colouring is done using the pressure scalar field. Fig. 8b is a real life experimental image for Re=26 (Van Dyke, 1982), and Fig. 8c is a simulation image using LIC vector field representation for Re=20. Two simulations with one and three cylinders are presented, respectively, in Fig. 9 and Fig. 10.

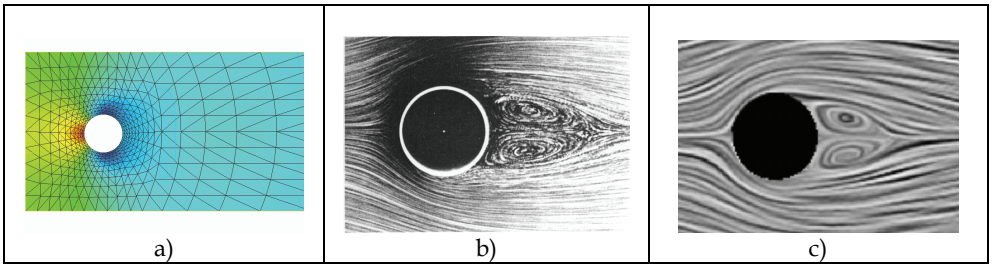


Fig. 8. Flow around cylinder: a) grid b) measurement c) simulation

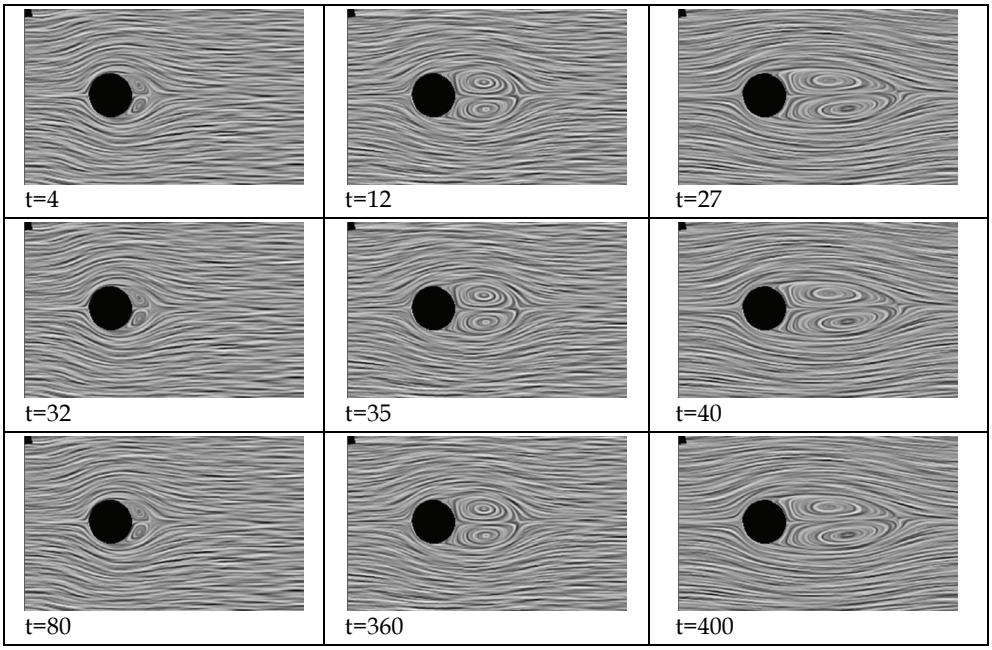


Fig. 9. Flow around cylinder: Simulation experiment for Re=100 at different time steps.

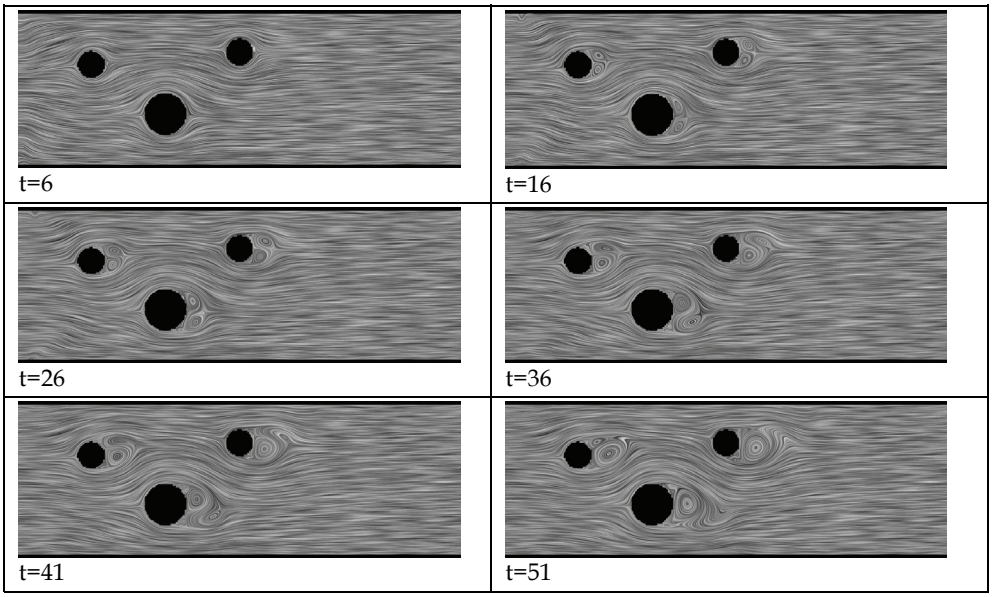


Fig. 10. Flow around cylinder: Simulation experiment for Re=100 at different time steps, with 3 cylinders

5. Conclusions and future work

The nowadays advances in information and communication technology, especially commodity computing, supercomputing and grid computing enable every scientist or engineer to have access to an advanced simulation kit of tools that makes analysis, product development, and design more optimal and cost effective. Further, that will make possible the investigation of incredibly complex dynamics by means of ever more realistic simulations. However, this brings with it huge amounts of data. To analyze these data it is crucial to use software tools, which can visualize multi-dimensional data sets, despite the difficulties inherent to this process. Scientific breakthroughs are enabled by insight, and, better visualization of an issue provides for a better understanding of the underlying science, and often for the discovery of something profoundly new and unexpected. The most exciting potential of visualization is the insight gained and the mistakes caught by spotting visual inconsistencies while computing. Advanced capabilities for visualization may prove to be as critical as the existence of the supercomputers themselves for scientists and engineers, and also for specialists in other domains. Better visualization tools might enhance human productivity and improve efficiency in several areas of science, industry, business, medicine, government and society in general. Visualization has the potential to *put the scientist into the computing loop and change the way the science is done* (McCormick, 1988).

Moreover, today everyone can produce their own computer graphics, with easy-to-use software integrated into various computer applications that makes charts and plots an obligatory element of many electronic documents. More professional packages offer in turn complex techniques for visualization of higher dimensional data. Visualization has thus become ubiquitous. Scientists are becoming familiar with desktop programs capable of presenting interactive molecular graphics. Bioinformatics, cheminformatics, and medical imaging use heavily such visualization engines for both interpreting lab data and training purposes (Wright, 2007). Furthermore, nowadays data visualization techniques are commonly used to provide business intelligence, for instance performance metrics and indicators may be displayed on interactive digital dashboards, and business executives rely on these software applications to monitor the status of business results and activities.

Therefore, all users of Scientific Computing and Visualization have an interest in better hardware, software and integrated systems, and much of what has being developed so far has been shared by a number of scientific and engineering projects up to a point, and with very large costs that were accessible only to large research facilities (e.g. SGI visualization servers and large PC clusters). Then the gaming industry has made a breakthrough, under the pressure of the gamers who have requested more and more graphical power, by developing very high performance graphics cards, at very low costs (commodity hardware). The visualization community has shifted to the use of these low-priced resources for their visualization tasks and progressively more PC-based visualization results have been obtained. However, gaming graphics hardware is not well suited for scientific computing and visualization, leading to a fundamental rethinking of how high-end systems are built as designers attempt to apply to large scale (interactive) rendering the clustered computing techniques that have revolutionized HPC (Constantinescu, 2008).

By now, desktop grid and volunteer computing systems have demonstrated their potential to supply more computing power to science and other in need domains than any other type of computing, and have therefore proved themselves as a viable solution to this problem. The benefits will increase over time due to the laws of economics, which states that

consumer electronics (PCs and game consoles) will progress faster than any specialized products, and that there will be an abundance of them. Nevertheless, desktop grid computing and volunteer computing are still under heavy conceptualization, research and development. There are still many aspects to clarify and solve: security issues, scheduling, volatile environment, sabotage-tolerance, integration with Grid, decentralization, peer to peer aspects etc.

The core idea of the work presented in this chapter has been to provide a desktop grid computing framework and to prove its viability by testing it in some Scientific Computing and Visualization experiments. We presented here QADPZ, an open source system for desktop grid computing, which enables users from a local network or Internet to share their resources. It is a multi-platform, heterogeneous system, where different computing resources inside an organization can be used. It can also be used for volunteer computing, where the communication infrastructure is the Internet. The system is currently used for research tasks in the areas of large-scale scientific visualization, evolutionary computation, simulation of complex neural network models, and other computationally intensive applications. It is also in use for educational purposes in universities around the world.

Further on we present some future work ideas that aim to improve both the conceptual model and the QADPZ system, as they are meant to make QADPZ more compliant with the requirements for a successful desktop grid and, therefore, more useful for its users. First, we consider inclusion of checkpointing, as many large-scale parallel problems require such support, because running a parallel application for hours or even days and loosing all results due to one failing node is unacceptable. Another feature to be added with high priority is handling of the restart of the master computer. In the current version of the system, each job needs a different client process, and currently we consider extending the client functionality to allow a single client instance to optionally connect to multiple masters and handle multiple jobs. Improved support for user data security is envisaged as well; computation results data can be encrypted and/or signed so that the user will be ensured that the received data is correct. Data integrity is another important issue, which is especially useful in an open environment (such as Internet). For faster performance, slave libraries will be cached at slave computers – in the current version, they are downloaded before each task is started. Moreover, slave computers will provide a flexible data storage available to other computers in QADPZ. Users are ought to be provided with different scheduling algorithms to choose from, according to the type of their problem.

Future developments of the system include more complete implementation of the MPI layer and development of a complete library of the collective communication routines. Our current implementation does not support collective communication, only the MPI_COMM_WORLD communicator. However, a complete library of such communication routines can be written entirely using the point-to-point communication functions and a few auxiliary functions. Current QADPZ's implementation is limited to a small subset of the MPI standard. It contains only the most used functions, as it has been developed to support only testing and evaluation of parallel communications. Another future work idea we consider is adding a set of transparent profiling tools for evaluating performance of different components. This is an important issue in compute-intensive applications, especially when running parallel applications. Dynamic balancing of the workload can be used for tackling this issue. The current user interface to the system is based on C++. Possible extensions of the system would be different interfaces for other languages, e.g. Java, Perl, Tcl or Python.

This can easily be done, since the message exchanges between different components of the system are based on an open XML specification.
The current implementation of the system is made considering only one central master node. This can be an inconvenience in certain situations, where computers located in different networks are used together. The master node can also be subject to failures, both software and hardware, and therefore a more decentralized approach is necessary. However, the basic elements are already in place as our high-level communication protocol between the entities, especially between the client and master, allows a master to act as a client to another master, thus making possible to create a distributed master that consists of independent master nodes, which communicate with each other, i.e. some sort of *virtual master*. Approaches from peer-to-peer computing will be used for implementing such a decentralized approach. Future desktop grid infrastructure are ought to be *decentralized, robust, highly available, and scalable*, while efficiently mapping application instances to available resources in the system (Berman et al., 2003). However, current desktop grid computing platforms are typically based on a client-server architecture, which has inherent shortcomings with respect to robustness, reliability and scalability. Fortunately, these problems can be addressed as well through the capabilities promised by new paradigms and techniques in P2P systems. By employing P2P services, our system could allow users to submit jobs to be run in the system and to run jobs submitted by other users on any resources available in the system, essentially allowing a group of users to form an ad-hoc set of shared resources, moving this way towards a P2P architecture. Interconnection with a grid computing environment is another goal, as desktop grid computing is not to evolve outside the Grid, but connected closely with it, inside it.
During the last decades, technologies have become commoditized and will continue to evolve in this direction to such a degree that we will be able to afford to have a tremendous number of them in the environment, providing for a commensurately powerful interconnected infrastructure to support for e-science in particular, and for e-society in general. The quality of provided services will improve constantly, as the infrastructure will become more autonomous and will strengthen its capabilities of self-configuring, self-optimizing, self-healing, self-protecting, and, why not, self-programming. In our opinion, desktop grids have revealed their huge potential to support computationally intensive applications, and to solve other significant real-world problems, being an invaluable part of this future infrastructure. In the near future we will find them enabling millions of users to collaborate and solve unimaginably sophisticated problems, having this way an important contribution to scientific research, and, even more important, to the directions of this research. Finally, improvements of their day to day life may and will come up as a result of this active participation of people to science.

6. References

Berman F., Fox G. & Hey A.J.G. (2003). *Grid computing: making the global infrastructure a reality*, Wiley, ISBN 978-0470853191, New York

BOINC. (2010). Open Source Software for Volunteer Computing and Grid Computing, available at http://boinc.berkeley.edu, accessed September 2010

BOINCstats. (2010). Statistics for BOINC, available at http://www.boincstats.com, accessed September 2010

Cabral, B. & Leedom, L. C. (1993). Imaging Vector Fields Using Line Integral Convolution. *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH 1993,* pp. 263-270, ISBN 0-89791-601-8, August, 1993, Anaheim, CA, ACM Press, New York

Cardenas-Montes, M., Emmen, A., Marosi, A. C., et al. (2008). EDGeS: bridging Desktop and Service Grids, *Proceedings of the 2nd Iberian Grid Infrastructure Conference (IBERGRID 2008)*, pp. 212-224, ISBN 978-84-9745-288-5, Porto, Portugal, May, 2008, NETBIBLO, A Coruna, Spain

Cassens J. & Constantinescu Z. (2003). Free Software: An Adequate Form of Software for Research and Education in Informatics?, *Proceedings of LinuxTag 2003 Conference*, Karlsruhe, Germany, July, 2003

Condor. (2010). Condor High Throughput Computing, available at http://www.cs.wisc.edu/condor, accessed September 2010

Constantinescu, Z. (2003). Towards an Autonomic Distributed Computing System, *IEEE DEXA Workshop on Autonomic Computing Systems (ACS'2003)*, Prague, Czech Republic, September, 2003, IEEE Computer Society Press

Constantinescu, Z. (2008). *A Desktop Grid Computing Approach for Scientific Computing and Visualization*, PhD Thesis, Norwegian Univ. of Science and Technology, ISBN 978-82-471-9158-3 Trondheim, Norway

Constantinescu Z., Vladoiu M. (2009a). A Solution for Improving Communications in Desktop Grid Systems, *BMIF Journal - Bulletin of PG University of Ploieşti, Series Mathematics, Informatics, Physics*, Vol. LXI, No. 2, December, 2009, pp. 27-32, ISSN 1224-4899, e-ISSN 2067-242X

Constantinescu Z., Vladoiu M. (2009b). Experimenting with Visualization on Desktop Grids, *BMIF Journal - Bulletin of PG University of Ploieşti, Technical Series*, Vol. LXI, No. 3, June 2009, pp. 255-260, ISSN 1224-8495

Distributed.net. (2010). Distributed.net available at http://boinc.berkeley.edu, accessed September 2010

Domingues, P.; Silva, J. G. & Silva, L. (2006). Sharing Checkpoints to Improve Turnaround Time in Desktop Grid Computing, *Proceedings of 20th Int. Conf. on Advanced Information Networking and Applications (AINA 2006)*, Vol. 1, pp. 301-306, Viena, Austria, April 2006, IEEE Computer Society Press, Los Alamitos, California

EDGeS. (2010). Enabling Desktop Grids for eScience, available at http://www.edges-grid.eu, accessed September 2010

eMinerals. (2010). Environment from the Molecular Level – a NERC eScience testbed project, available at http://www.eminerals.org, accessed September 2010

Folding@home. (2010). Folding@home distributed computing, available at http://folding.stanford.edu, accessed September 2010

Forssell, L. K. (1994). Visualizing Flow Over Curvilinear Grid Surfaces Using Line Integral Convolution, *Proceedings of IEEE Visualization '94*, pp. 240-247, ISBN 0-8186-6626-9, Washington D.C., October, 1994, IEEE Computer Society Press

Forssell, L. K. & Cohen, S. D. (1995). Using line integral convolution for flow visualization: curvilinear grids, variable-speed animation, and unsteady flows, *IEEE Transactions*

on Visualization and Computer Graphics, Vol. 1, No. 2, June, 1995, pp. 133-141, ISSN 1077-2626

Foster, I. & Kesselman, C. (2004). *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, ISBN 978-1558609334, Boston, USA

GIMPS. (2010). Great Internet Mersenne Prime Search, available at http://www.mersenne.org, accessed September 2010

Grid@EFPL. (2010). Greedy Pool desktop Grid, available at http://greedy.epfl.ch, accessed September 2010

He H., Fedak G., Kacsuk P. et al. (2010). Extending the EGEE Grid with XtremWeb-HEP Desktop Grids, *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 685-690, ISBN 978-0-7695-4039-9, Melbourne, Australia, May 2010, IEEE Computer Society Press Washington, DC, USA

Hoover, S., Dobrenen, K. & Trauner, M. (2009). Project Blackbird: Deploying Condor in a Blackboard Environment, *EQ EDUCAUSE Quarterly Magazine*, Vol. 32, No. 3, 2009, *ISSN* 1528-5324, available online at http://www.educause.edu/EDUCAUSE+Quarterly/EDUCAUSEQuarterlyMagazineVolum/ProjectBlackbirdDeployingCondo/182625, accessed September 2010

Jermini, P. (2010). Building a Condor Desktop Grid and its exercises, *Swiss Grid School 2010*, June, 2010, Lugano, Switzerland, available online at http://greedy.epfl.ch/articles/building-condor-grid-sgs10.pdf, accessed September 2010

Hamming, R. W. (1987). *Numerical Methods for Scientists and Engineers*, Dover Publications, 2nd edition, ISBN 978-0486652412 , Mineola, New York

Hayes, B. (1998). Collective Wisdom, *American Scientist*, Vol. 86, No. 2, March, April, 1998, pp. 118-122, ISSN 0003-0996

Heath, M.T. (2002). *Scientific Computing: an Introductory Survey*, McGraw-Hill, ISBN 978-0072399103, Boston

McCormick, B. H. (1988). Visualization in Scientific Computing, *ACM SIGBIO Newsletter*, Vol. 10, No. 1, March, 1998, pp. 15-21, ISSN 0163-5697

Merzkirch, W. (1987) *Flow visualization*, 2nd Edition, Academic Press, ISBN 978-0124913516, Orlando, Florida, USA

Meuer, H.W. (2010), Top500 Supercomputer Sites Report, 35th Edition, June 2010, available at http://www.top500.org, accessed September 2010

Mustafee, N. & Taylor, S. J. E. (2006). Using a desktop grid to support simulation modelling, *Proceedings of 28th International Conference on Information Technology Interfaces (ITI 2006)*, pp. 557-562, ISBN 953-7138-05-4, Dubrovnik, Croatia, June, 2006, Zagreb's University Computing Centre SRCE, Zagreb, Croatia

Paradyn/Condor Week. (2008). Paradyn/Condor Week – Condor Presentations, Madison, Wisconsin, United States of America, April-May, 2008, available at http://www.cs.wisc.edu/condor/PCW2008/condor_presentations.html, accessed September 2010

Seti@home Classic. (2010). The Search for Extraterrestrial Intelligence, available at http://seticlassic.ssl.berkeley.edu, accessed September 2010

Seti@home. (2010). The Search for Extraterrestrial Intelligence - the current Seti@home project, available at http://setiathome.ssl.berkeley.edu, accessed September 2010

Smits, A. J. & Lim, T. T. (2000). *Flow visualization: techniques and examples*, Imperial College Press, ISBN 978-1860941931, London, UK

Steeb, W.-H.; Hardy, Y.; Hardy, A. & Stoop, R. (2004). *Problems & Solutions In Scientific Computing With C++ And Java Simulations*, World Scientific Publishing Company, ISBN 978-9812561121, Singapore

Thiemard, M. (2008). On the Usage of a Desktop Grid - Greedy@EPFL, *Symposium on High Performance Computing Methods (HPCM)*, June, 2008, Lausanne, Switzerland, available online at http://greedy.epfl.ch/articles/hpcm-0608.pdf, accessed September 2010

QADPZ. (2010). Quite Advanced Distributed Parallel Zystem, QADPZ's homepage on sourceforge, available at http://qadpz.sourceforge.net, accessed September 2010

Utnes, T. & Brors, B. (1993). Numerical modelling of 3-D circulation in restricted waters, *Applied Mathematics Modeling*, Vol. 17, No. 10, October, 1993, pp. 522-535

Van Dyke, M. (1982). *Album of Fluid Motion*, Parabolic Press, ISBN 0-915760-03-7, Stanford, California, USA

Vladoiu, M. & Constantinescu Z. (2008a). A Taxonomy for Desktop Grids from Users Perspective, *Proceedings of ICPDC 2008 - the 2008 International Conference of Parallel and Distributed Computing, a Conference of World Congress on Engineering 2008 (WCE 2008)*, Vol. I., pp. 599-604, ISBN 978-988-98671-9-5, London, UK, July, IAENG

Vladoiu, M. & Constantinescu Z. (2008b). An Extended Master-Worker Model for a Desktop Grid Computing Platform (QADPZ), *Proceedings of 3rd International Conference on Software and Data Technologies (ICSOFT 2008)*, Vol. I, pp. 169-174, ISBN 978-989-8111-51-7, Porto, Portugal, July, 2008, INSTICC, Setubal, Portugal

Vladoiu, M. & Constantinescu Z. (2009a). Development Journey of QADPZ - A Desktop Grid Computing Platform, *International Journal of Computers, Communications & Control (IJCCC)*, Vol IV, No. 1/2009, pp 82-91, ISSN 1841–9836; E-ISSN 1841–9844

Vladoiu, M. & Constantinescu Z. (2009b). Availability of Computational Resources for Desktop Grid Computing, *BMIF Journal - Bulletin of PG University of Ploieşti, Series Mathematics, Informatics, Physics*, Vol. LXI, No. 1, June, 2009, pp. 43-48, ISSN 1224-4899

Wikipedia. (2010a). Berkeley Open Infrastructure for Network Computing, available at http://en.wikipedia.org/wiki/Berkeley_Open_Infrastructure_for_Network_Computing, accessed September 2010

Wikipedia. (2010b). Wikipedia - The Free Encyclopedia's page on Computational Science, available at http://en.wikipedia.org/wiki/Computational_science, accessed September 2010

Wilson, P., Emmerich, W. & Brodholt, J. (2004). Leveraging HTC for UK eScience with Very Large Condor Pools: Demand for Transforming Untapped Power into Results, *Proceedings of the UK e-Science Programme All Hands Meeting (AHM 2004)*, pp. 308-315, ISBN 1-904425-21-6, Nottingham, UK, August-September, 2004, EPSRC, Swindon, UK

Wright H. (2007). *Introduction to Scientific Visualization*, Springer, ISBN 978-1846284946

XtremWeb. (2010). The Open Source Platform for Desktop Grids, available at http://www.xtremweb.net, accessed September 2010
XtremWeb-CH. (2010). The Volunteer Computing Middleware, available at http://www.xtremwebch.net, accessed September 2010
XtremWeb-HEP. (2010). Desktop Grid for High Energy Physics, available at http://www.xwhep.org, accessed September 2010