

General Parallel Computations on Desktop Grid and P2P Systems

James C. Browne, Madulika Yalamanchi, Kevin Kane and Karthikeyan Sankaralingam
Department of Computer Sciences
University of Texas
Austin, TX 78712
{browne,madhu,kane,karu}@cs.utexas.edu
1-512-471-9579

ABSTRACT

This paper defines the requirements for effective execution of iterative computations requiring communication on a desktop grid. It then proposes a combination of a p2p communication model, an algorithmic approach (asynchronous iterations) and a programming model which have promise for satisfying those requirements. Experimental results from an implementation of asynchronous algorithms in pure p2p system will be given. The integration of the p2p communication model with a commercially supported implementation of the [SETI@Home](#) desktop grid system to yield a prototype implementation of a desktop grid implementing communication among its hosts is described and some very preliminary results from application of the extended desktop grid for computation of Google pageranks and solution of linear systems. The desktop grid prototype preserves anonymity among desktop hosts, supports dynamic communication set membership and enables transparent replication of computations for fault tolerance.

1. Introduction

The motivation for this research is the potential of desktop grids for low cost execution of large scale computations. Desktop grids based on the [SETI@Home](#) [29] model have been very successful in cost effective computation of fully partitionable computations. But it is also clear that desktop grids cannot be applied to general parallel computations as long as communication is restricted to the master-slave model of parallelism and communication and current parallel computational infrastructures which largely rely on synchronous algorithms executing in a fully reliable resource environment.

The research reported in this paper is based upon integration of results from three threads of research:

- P2P networks based upon several different types of communication models have been widely applied for file sharing.

- Successful applications [2,10,27] of asynchronous iteration [7] to computation of the lowest eigenvalue and eigenvector of stochastic matrices. (Google pageranks [22]).
- Studies suggesting good convergence properties for asynchronous iterative algorithms in application to linear systems.

into desktop grids.

The first step is to establish the requirements for effective execution of general communicating parallel computations on desktop grids. The next steps are to identify possible means of meeting these requirements, and to evaluate at least some of the possible solutions. This paper reports on combining a p2p communication model with asynchronous iteration for solution of linear systems. The results of these experiments suggest potential of this combination of communication model and algorithmic model for distributed solution of linear systems on desktop grids.

Finally it must be shown that p2p communication model used in these studies can be combined with desktop grid technology and asynchronous iterative solution methods to enable a potentially scalable execution platform for general parallel computations requiring communication among units of computation.

Section 2 sketches the requirements for successful application of p2p systems and desktop grids for general parallel computations including linear system solution. Section 3 describes the communication model, the p2p system built upon it and the programming model for the p2p system. Section 4 describes the experiments and results for solution of linear systems. Section 5 defines and describes a potentially scalable implementation of a desktop grid system where the desktop systems are coupled by the p2p communication network.

2. Effective Execution of Iterative Parallel Computations on Desktop Grids

2.1 Requirements

The requirements for desktop grids which can effectively execute iterative parallel computations requiring communication are anonymous, scalable and fault-tolerant communication among the hosts of a scalable desktop grid systems and fault-tolerant computational algorithms which are insensitive to heterogeneity in processing power of hosts and communication speeds among hosts. The requirements on the computational algorithms are obvious from the nature of desktop grids as are the requirements for scalable and fault-tolerant communication. Anonymity is required of the communication mechanism among the hosts in a desktop grid because the software executed on hosts is written by users and poses risk even when encapsulated by desktop grid client agents. Anonymity among the desktop resources minimizes security and privacy violation.

2.2 Computational Algorithms

Computations formulated in parallel *asynchronous iterative* algorithms [9] are much less sensitive to heterogeneity of communication and computational power than are conventional synchronous parallel iterative algorithms. The hosts for each partition of the computation can continue to execute without waiting for satisfaction of all of its boundary condition dependencies.

Fault tolerance through replication of application components is straightforward for iterative processes since asynchronous iteration is essentially idempotent and can be made rigorously idempotent through associative communication as will be shown following.

There are, however, many research issues which remain to be resolved. General convergence properties and convergence rates for asynchronous algorithms, are more difficult to determine than for synchronous iterations [three books, two papers]. There has been considerable research on convergence of asynchronous iterative algorithms but relatively little of which applies directly to solution of linear systems on desktop grids which is the class of computations considered in this research. In particular there has been little or no research on the convergence and performance properties of asynchronous formulations of many of the efficient algorithms for iterative solution of linear systems such as conjugate gradient algorithms. (See Section 7 on Related Research for a summary.) In the case of determination of the dominant eigenvector and the associated eigenvector for the matrices defined Google page ranks, it seems well established, however, that the asynchronous iterative computation of the power method of eigenvalue computations converge more rapidly than synchronous iterative algorithms. [2,10,27].

Another issue requiring attention is distributed computation of convergence metrics for asynchronous computations.

Finally, choice of pre-conditioners for linear system solvers is still a domain specific issue.

2.3 Communication Model

Communication will be implemented through an overlay network based implementation of *associative broadcast* [3,4] coupling the resources in a desktop grid. Associative broadcast is a multicast with a dynamically varying set of participants. Components interacting through associative broadcast maintain descriptive names¹ (*Profiles*, see Section 3.1 for the definition of the descriptive names as profiles.) specifying their current local state in terms of a globally known set of attributes. Associatively broadcast messages are addressed (See Section 3.1 for details.) to potential receivers in terms of the descriptive names of the receivers rather than the network or physical addresses of the potential receivers. The sender of a message does not know and does not need to know the network address of the receivers of a message and the receiver does not know the network address of the sender so that anonymity is intrinsic.

Replication is synchronization-free under associative interactions so that fault-tolerance can be obtained through replication of computations. If there are multiple replicas of a source component for a given message and the computation implemented by the target component is not idempotent, then the target component can modify its profile to prevent receipt of multiple inputs of the same result from replicated computations.

Combining associative interactions with parallel asynchronous iterative computation algorithms therefore yields one basis for computational processes appropriate for the heterogeneous and dynamic structure of desktop grids for those computations which can be cast as iterative processes.

Scalability of associative communication is the most critical unresolved issue for the communication model. One possibly scalable implementation of associative broadcast specific to matrix computations based on space filling curves [reference] is sketched in Section 7 on Future Research.

3. P2P Communication and Programming Model

3.1 Associative Broadcast

Associative broadcast [3,4] is so-called because the model is similar to content addressing of associative memory. A target set specification travels with each message that is broadcast onto the network. The target set is determined for

¹ Descriptive names are also called intentional names in the network literature [1]

each message by the set of local recipients whose local states match that of the target specification. Therefore, the sender does not need to know the identities of the targets in order to broadcast a message; the identities and Internet addresses of the target set is completely unknown to the sender. The only global knowledge required is that all participants share a common domain vocabulary for defining local states of components and target set specification for messages.

A component which is to participate in an associative broadcast communication set is encapsulated with associative broadcast interface which has two elements: an *accepts* interface and a *requests* interface. *Accepts* and *requests* interfaces are extensions of associative broadcast to include additional specifications for the interaction which results from a successful match [14].

The *accepts* interface for a component is a set of three-tuples (profile, transaction, state machine). Multiple members of the set may be associated with each instance of a component model and new instances instantiated at runtime.

- A profile is a set of attribute/value pairs which specify the current local state of a component.
- A transaction specifies the functionality and the parameters of the units of work, which are executed on a given interaction by this instance of the component.
- A state machine defines the sequence of simple interactions necessary to complete the interaction specified by the profile.

The *requests* interface is a set of three-tuples (selector, transaction, state machine). Multiple members of the set may be associated with each instance of an object and new instances instantiated at runtime.

A selector is a conditional expression over the attributes of the objects in its domain and the other domains with which it has interactions. The conditional expression of a selector is a template with slots for attribute name/value pairs specified in the profiles of other object instances. A selector is said to match a profile whenever the conditional expression of the selector evaluates to true when the attribute values from the profile are inserted into or compared with the slots in the selector expression. A selector thus specifies a (dynamic) target set for a message.

3.2 Programming Model

The CoorSet [13] programming model is an extended version of asynchronous state machines interacting through messages. A program consists of a set of components each of which is encapsulated by an associative interface and communicates by associative broadcast. A component is a program in a sequential programming language which implements one or more actions which may result in further interactions with other components and a change of state for the component. A component modifies its *accepts* and *requests* interfaces to conform to its current state.

The programming model extends the simple interacting state machine model to incorporate complex conditions for enabling execution of a component and replication for both representation of SPMD parallelism and fault-tolerance.

The conditions for executing and action of a component commonly include receipt of multiple messages. The concept of a “firing rule” is introduced into the associative interface. A firing rule is an expression over some set of messages which must be received to initiate some action of a component. Additionally, since components may and often will have persistent state, there may be precedence relations among possible enabling message sequences. These extensions are accomplished by adding types to messages and incorporating a conditional expression over message types and local state into the associative interface.

The definition of firing rules used in the extended coordination model is taken from a data flow programming model [20], where rather than waiting on a single input, a node in a data flow graph waits on multiple inputs, possibly in a particular order, before becoming enabled for execution. Firing rules are specified with a Java-like logical syntax. Specifying reception of *either* of two message types *R*, *S* is done with a rule “*R* || *S*”. Reception of *both* of two message types is specified with a rule “*R* && *S*”. Reception of *R* followed by *S* is specified with a rule “*R* < *S*”. These rules can be compounded and grouped with parentheses, such as “(*R* < *S*) || (*R* < *T*)”. The ‘<’ operator has the lowest precedence, followed by ‘||’, and ‘&&’ has the highest precedence.

Replication is another feature that must be included in associative interaction specifications to enable facile specification of parallelism and fault-tolerance. SPMD parallelism can be readily implemented by replication of components. Replication of functionality for fault-tolerance can be made transparent and synchronization-free after initialization. If an initiating component starts several replicas of a given component to insure success in an unreliable environment and each of the replicated components responses by associative broadcast then the initiating component can safely proceed after the first successful result and set its profile to ignore the other completions. A component can be replicated by adding an index attribute to its profile and instantiating replicas in conformance to the index range. Figure 1 below is the CoorSet main program for the iterative linear solver described in Section 4.

```

component 8{
  class AsyncIterLib
  target AsyncIter execute startup
  include ("AIS.AsyncIterLib", "AsyncIterLib.class")
  profile ("AsyncIter")
  accepts
  "SolutionUpdate" ReceivedUpdate (Integer, Double)
  requests

```

```
"SolutionUpdate" SendUpdateRequest "AsyncIter"
(Integer, Double)
```

Figure 1 – CoorSet Program for Asynchronous Iteration

This CoorSet program is particularly simple. The state machine has only a single state so is not required. The line “Component 8” specifies that what follows is the defines the initial conditions for the component and that there are to be 8 replicas of the component instantiated in the network. The component is a program in Java which uses the runtime system to implement communication. The runtime system discovers hosts, downloads the code and initiates execution of the component. The three lines

```
class AsyncIterLib
target AsyncIter execute startup
include("AIS.AsyncIterLib", "AsyncIterLib.class")
```

give the classes to be loaded on the host and that the method to be executed is “startup.” The lines

```
profile ("AsyncIter")
accepts
"SolutionUpdate" ReceivedUpdate (Integer, Double)
requests
"SolutionUpdate" SendUpdateRequest "AsyncIter"
(Integer, Double)
```

specify the initial profile of the component AsyncIter.

The interested reader can find example programs which illustrate more of the capabilities of CoorSet can be found in Kane and Browne [13]

4. P2P Solution of Linear Systems

The experimental evaluation of p2p system solution of linear systems by asynchronous iteration was formulated as a CoorSet Program. Each peer hosts a component which has

- An associative interface
- A partition of the matrix to be solved
- Implementations of simple, unoptimized versions of Jacobi & Gauss Seidel relaxation procedures.

A more detailed report can be found in Yalamanchi [35].

4.1 Data Structures

Let ‘m’ be the number of equations a peer owns. (total number of documents/Number of peers) . Each peer maintains a matrix of size $m * (N + 1)$ to store the equation set and an array of size ‘N’ to store the values of the variables.

4.2 Algorithm

The algorithm for implementing the solution of linear system of ‘n’ equations in ‘n’ variables ($Ax=b$, where A is a 2D

matrix, x is the vector of unknowns and b is a constant vector.) is detailed below. The equations are numbered 1 to ‘n’ and the variables are number 1 to ‘n’. Each peer in the system is responsible for computing the values of a subset of the solution. If the peer owns equations ‘k’ to ‘m’, it is responsible for computing variables ‘k’ to ‘m’.

Each peer

1. Reads its equation set (N/m entries offset by its peer id) from an input file
2. Initializes the values of variables to random values.
3. Starts computing the values of N/m variables the peer is responsible for using either the Jacobi or the Gauss-Seidel rule. The initial implementations were done with Jacobi and Gauss-Seidel because of the simplicity of the asynchronous implementations.
4. Computes the difference in variable values in successive iterations and uses this as a measure for convergence. If the difference is greater than a threshold, broadcasts “variable update” messages to all other peers. (Element by element convergence is a very stringent convergence criterion. See Section 4.5)
5. Upon receiving a “variable update” message for a variable, updates its entry in the variable array
6. Update of a variable results in the generation of more variable update messages which propagate till the system stabilizes (till the variables converge)
7. When the relative difference between successive values for a variable is less than a threshold, the variable’s value is considered to be stabilized and no more messages are sent for that variable
8. The values of different variables converge at different times

There are many possible optimizations of this simple algorithm. Some of these optimizations are described in Sections 4.5 and 7.

4.3 Data Sets

Some of the data sets used for the experiments were generated and some were obtained from Matrix Market [19]. Only the results from the matrices obtained from MatrixMarket are reported here. The constant matrix b has been randomly generated in all cases. It always had values between 0 and 1.

These matrices are derived from finite-difference and finite element representations physical processes and systems. The properties they all had in common were

- a. Block tri-diagonal
- b. Symmetric, positive definite

4.4 Experiments

In all cases the solutions were checked against Matlab. The convergence factor was 10^{-6} . That is, each variable in the solution was judged to be converged when its value changed by less than 10^{-6} on successive iterations.

All of the results reported in the table were executed on eight processors in a local area network.

The abbreviations used in the Tables are : MM: Matrix Market; Gen: Generated; LD: Lower diagonal; G-S: Gauss-Seidel; J: Jacobi. Table 1 summarizes the properties of matrices used and the convergence results obtained. The initial solution vector was set to $1/a_{ii}$. Experiments are reported here only for 3 cases for Gauss-Seidel method. More examples will be given in the talk and many more in the final paper.

Table 1 - Convergence results for experiment 2

Matrix Size	No. entries	SR(J)	SR(G-S)	Convergence (min)
100	594	1.0627	0.9959	18.7479
468	5172	1.0489	0.9979	155.093
900	7744	0.9923	0.9847	27.2894

4.5 Discussion

The convergence for solution of these linear systems was much slower than has been previously reported for computation of pageranks. The synchronous algorithm converged much more rapidly than the asynchronous algorithm. But all of the matrices have spectral radii very close to unity and with the exception of the 900x900 matrix which was derived from a finite difference Laplacian, all have condition numbers greater than 10,000. Preliminary results are currently being obtained for a preconditioned conjugate gradient solver. The initial results are that the convergence time for asynchronous conjugate gradient solution is about a factor of 50 less than for Gauss-Seidel which is in line with the results from comparisons of the synchronous formulations of Gauss-Seidel and conjugate gradient.

There is a vast range of possible optimizations ranging from partial asynchrony, message "chunking," choice of preconditioners, choice of convergence metrics, etc. A systematic but rather slow study of these possible algorithmic improvements is in progress.

5. Integration of P2P Communication with [SETI@Home](#) Desktop Grid.

Desktop grid platforms which are effective at executing completely parallelizable applications are available both as commercially supported software [32] and open source software [7]. (United Devices has licensed source code access to its system by UT-Austin.) These desktop grid systems already provide robust solutions for difficult and complex problems such as secure encapsulation of application program components on the desktop grid hosts, partitioning and distribution of both program components and data and monitoring of progress. Robust re-implementation of all of these capabilities would be an undertaking far beyond the resources available to us for this research. The approach taken here is to extend existing desktop grid platforms with anonymous and fault-tolerant communication based on associative broadcast. An initial prototype integration of associative communication with the United Devices system has shown that addition of communication servers is straightforward. A set of communication servers coupled by an overlay network implementation of has been added to the application management and data servers of United Devices. A compiler which maps the CoorSet programs to the input format of the United Devices application server has been written so that programs written in CoorSet are submitted through the United Devices application server.

The challenge is scalability of associative communication. The current implementation of the overlay network coupling the communication servers is a generic associative broadcast runtime system based on multicast and tunneling which is demonstrably not scalable. The communication servers can, however, be application specific servers which are started by the application manager on hosts chosen for their locations and capabilities or they may, in the case of intranet-based desktop grids, be dedicated communication servers which are customized for specific applications. Using the locality properties intrinsic to partitioning of matrices coupled with descriptive names which include partition identification will in concept enable scalability on an application by application basis. (See Section 7.)

A feasibility demonstration prototype implementation of associative interactions on the United Devices platform which connects all communication servers via the LRMP [17] implemented multicast has been developed. This implementation, which was done to determine the feasibility of associative communication with the United Devices platform, has been tested for simple programs including the pagerank computation and the linear solvers reported here.

6. Related Research

Related Research includes studies of asynchronous iteration, systems implementing computational algorithms in peer to peer systems and previous research on computation of pageranks in p2p systems.

Bertsekas and Tsitsiklis give an extension discussion of asynchronous iterative algorithms in their 1989 book [5]. Frommer and Szyld [11] give an excellent survey of asynchronous iteration and the mathematical

theory of the several models of asynchronous iteration including necessary and sufficient conditions for convergence. Strikwerda [28] gives weaker conditions for convergence based on a probabilistic analysis. There have been several experimental studies of the performance of asynchronous iterative solvers for linear system solution (and non-linear system solution) on clusters and multiprocessors. See for example, [6].

Chen and Zhang (10) had previously found that a parallel asynchronous iterative computation of pageranks converged more rapidly than a synchronous computation when both were executed on a cluster multicomputer. Sankaralingam, Sethumadhavan and Browne [27] report very rapid convergence for the computation of pageranks via asynchronous iteration. Bahi, Domas and Mazouzi [2] also report that asynchronous computations of pagerank converge more quickly than synchronous computation.

The programming systems related to those reported herein include the P³ [21], the Consumer Grid project [31], ICENI [12], G2 [16] and the work of Butt, Zhang and Hu on “A Self-Organizing Flock of Condors” [8]. Each is developing a capability for executing applications on distributed desktops. In “A Self-Organizing Flock of Condors” the central managers of flocks are coupled by a distributed hash table implementation of p2p communication to implement flocking. Fault-tolerance to failure of a central manager within a pool or flock is obtained by superimposing a distributed hash table on the hosts within a flock and constructing a protocol to elect a new central manager when failure of a central manager is detected. The Consumer Grid Project, ICENI, and G2 do not address the issues of fault-tolerance. ICENI does have a performance modeling capability which could be used to partially accommodate computations to heterogeneous computation and communication resources. Oliveira et al. [21] propose to build a parallel peer-to-peer environment called P³. They propose to implement P³ in Java. P³ provides a distributed file system stored across all compute nodes that provide the support necessary for interprocess communication and synchronization. Fault-tolerance and heterogeneity issues are not addressed. The implementation of the P³ network, in Java, is still ongoing research as far as we can discern.

Jace [2], a system designed specifically for implementation of the communication needs of asynchronous iterative computations, is the research system most closely related to that reported here. The Jace paper reports faster convergence for asynchronous Jacobi iteration than for synchronous Jacobi iterations for the solution of linear systems.

7. Future Research

Future research includes optimizations for asynchronous iterations executing under distributed control and optimization of multicast communication as the underlying substrate for associative communication. The research issues for asynchronous computations include those

mentioned in Section 2.2. There are many possible approaches to optimization of communication in p2p networks. Integration of topological and physical locality is a subject of study in p2p networks. We plan to focus on domain specific optimizations for matrix computations. Associative broadcast need not be an actual broadcast. The communication servers will keep track of the network addresses of the hosts in the grid. The hosts will send their current profiles to a local communication server. The communication servers will maintain maps of profiles to network addresses which can then be used to route messages near optimally. One most promising approach and one which takes advantage of the natural locality of matrix partitions is to use an overlay network among the communication servers with routing directed by space-filling curves [25]. The space-filling curve based partitioning of matrices reported by Parashar, et. al. [22] are an initial basis for system design. Schmidt and Parashar [30] give an implementation of p2p communication based on space filling curves.

8. Acknowledgements

This work was partially supported by the National Science Foundation under grants EIA-0127857 and EIA-0305644. The authors are grateful to the reviewers of the original submission for reminding them of several neglected topics including an important reference.

9. References

- [1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *17th ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC, December 1999.
- [2] J.M. Bahi, S. Domas, K. Mazoui “Jace: A Java Environment for Distributed Asynchronous Iterative Computations” *Proceedings of 12th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (PDP 2004)*, 11-13 February 2004, A Coruna, Spain
- [3] B. Bayerdorffer: Associative broadcast and the communication semantics of naming in concurrent systems. Ph.D. dissertation, Department of Computer Sciences, The University of Texas at Austin (1993).
- [4] B. Bayerdorffer, : Distributed computing with associative broadcast. *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences* (1995).
- [5] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and distributed computation*. Prentice Hall, Englewood Cliffs, NJ, 1989.
- [6] Blathras, K., D. B. Szyld, and Y. Shi: 1999, ‘Timing Models and Local Stopping Criteria for Asynchronous Iterative Algorithms’. *Journal of Parallel and Distributed Computing* **58**, 446–465.
- [7] Boinc, <http://boinc.berkeley.edu/>

- [8] A. Butt, R. Zhang and Y. Hu “A Self-Organizing Flock of Condors” In Proceedings of SuperComputing 2003 (Phoenix, Arizona, November 2003)
- [9] D. Chazan and W. Miranker, Chaotic relaxation, *Linear Algebra and its Applications*, 1969, pp. 199-222
- [10] Y. Chen, Y. and H. Zhang: 2003, ‘Parallelization of the page ranking in the Google search engine’. <http://manip.crhc.uiuc.edu/chen/pagerank.ps>.
- [11] A. Frommer and D. B. Szyld: 2000, “On asynchronous iterations”, *Journal of Computational and Applied Mathematics* 123(1), 201 – 216
- [12] ICENI: <http://www.lesc.ic.ac.uk/iceni/>
- [13] K. Kane and J.C. Browne: 2004, ‘CoorSet: A Development Environment for Associatively Coordinated Components’, In: *Proceedings of Coordination 2004*, (Also to appear in a volume in the Springer-Verlag LNCS Series).
- [14] K. Kane, J.C. Browne and H. Tian, “An Associative Broadcast Based Coordination Model for Distributed Processes”, *Proceedings of Coordination 2002*, LNCS 2315, Springer-Verlag, 2002, pp. 96-110
- [15] K. Kane and Browne, J.C.: The Component Starting Component: an environment for distributed systems and peer to peer research. Department of Computer Sciences Technical Report TR-03-42, University of Texas at Austin (2003).
- [16] W. Kelly, P. Roe, et al., An Enhanced Programming Model for Internet Based Cycle Stealing, in Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 1649-1655, June 2003.
- [17] T. Liao: Light-weight Reliable Multicast Protocol. INRIA Technical Report (1998), http://webcanal.inria.fr/lrmp/lrmp_paper.ps
- [18] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations", *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104-111, June, 1988.
- [19] Matrix Market, <http://math.nist.gov/MatrixMarket>
- [20] P. Newton and Browne, J. C.: The CODE 2.0 Graphical Parallel Programming Language. *Proceedings of the ACM International Conference on Supercomputing* (1992) 167-177.
- [21] L. Oliveira, L. Lopes, and F. Silva, “P3: Parallel Peer to Peer, An Internet Parallel Programming Environment” *Proceedings of the Workshop on Web Engineering and Peer-to-Peer Computing* (Pisa, Italy , 2002)
- [22] Pagerank: 2002, ‘PageRank Explained’, <http://www.webrankinfo.com/english/pagerank>
- [23] M. Parashar, C. Edwards, K. Klimkowski and J. C. Browne) “A Common Data Management Infrastructure for Adaptive Algorithms for PDE Solutions,” *Proceedings of SC '97*, San Jose, CA, November 1997, pp. 1-22.
- [24] A. Rowstron and P. Druschel, Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer systems, *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms* (Middleware 2001), November 2001, pp. 329-350
- [25] Rowstron, A., Kermarrec, A-M., Castro, M., and Druschel, P.: SCRIBE: “The design of a large-scale event notification infrastructure”. *Proceedings of NGC2001*, UCL, London (2001).
- [26] Hans Sagan, Space-Filling Curves, (Springer-Verlag, New York, 1994.) ISBN: 0-387-94265-3.
- [27] Sankalingam. K, S.Sethumadhvan and J.C.Browne: “Distributed PageRank in Peer to Peer systems”, In: *Proceedings of the 12th International Symposium on High Performance Distributed Computing*. Pp. 58-68
- [28] Strikwerda, J. C.: 2002, ‘A Probabilistic Analysis of Asynchronous Iteration’. *Linear Algebra and its Applications* 349, 125–154.
- [29] SETI@home <http://setiathome.ssl.berkeley.edu>
- [30] C. Schmidt and M. Parashar “Flexible Information Discovery in Decentralized Distributed Systems” *Proceedings HPDC 12* (Seattle, June 2003) pp.266-235.
- [31] I. Taylor, O. F. Rana, R. Philp , Ian Wang and M. Shields “Supporting Peer-2-Peer Interactions in the Consumer Grid” *Proceedings of HIPS'03* (Nice, France, April 22-26, 2003) pp.3-14).
- [32] United Devices <http://www.ud.com>
- [33] J. N.Verbeke, Nadgir, G. Ruetsch, and I. Sharapov: 2002, ‘Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment’. In: *Proceedings of the 3rd International Workshop on Grid Computing*.
- [34] WestLake, J.R., A Handbook of Numerical Matrix Inversion and Solution of Linear Equations, (John Wiley & Sons, INC., 1968.)
- [35] M. Yalamanchi, “Experimental Evaluation of Peer to Peer Networks” (Masters Report, Department of Electrical and Computer Engineering, UT-Austin, May 2004)