Distributed Optimization of Fiber Optic Network Layout using MATLAB

Roman Pfarrhofer[1], Markus Kelz[1], Peter Bachhiesl[1], Herbert Stögner[1], and Andreas Uhl[1,2]

[1] Carinthia Tech Institute, School of Telematics & Network Engineering
Primoschgasse 8, A-9020 Klagenfurt, Austria
[2] Salzburg University, Department of Scientific Computing
Jakob-Haringerstr.2, A-5020 Salzburg, Austria

Abstract. A MATLAB-based planning tool for the computation of cost optimized laying for fiber optic access networks is employed on homogenous and heterogenous Windows PC networks. The approach is based on a custom MATLAB toolbox which does not require a MATLAB client installed at the participating machines. Experiments demonstrate the efficiency and real-world usability of the approach which facilitates an interactive network planning process.

1 Introduction

The demand for broadband capacities has entailed an increasing changeover from conventional transmission techniques to the fiber optic technology [1]. During the last two years European network-carriers have invested 7.5 billion Euros in the expansion of the core and the distribution net domain (backbones, city backbones and metropolitan area networks). However, investigations have shown that about 95% of the total costs for the implementation may be expected for the area-wide realization of the last mile (access networks). In order to achieve a return on investment, carriers will be forced to link access net nodes, like corporate clients, private customers or communication nodes for modern mobile services (e.g. UMTS) to their city backbones.

As a consequence, the cost optimized laying of underground networks (e.g. fiber optical networks) with respect to the given spatial topologies and under consideration of usable infrastructures represents a challenging task of practical interest. Whereas a variety of strategic planning tools focus on problems of laying optimization, discussion of reliability, and traffic optimization in the core net domain [2], carriers hardly use any computational planning tools for cost estimation and optimization in the access net domain. The planning process is performed manually based on expert knowledge and therefore often yield sub-optimal decisions. However, since the telecommunication market runs through a period of consolidation the neglection of economization potentials has to be considered as a critical competitive disadvantage.

In recent work [3], we have introduced the methodologies behind the MATLAB-based planning tool NETQUEST-OPT for the computation of cost optimized and real world laying for fiber optic access networks. Real-world topological geometries are represented by detailed geoinformation data. The optimization kernel is based on cluster strategies, exact and approximative graph theoretical algorithms, combinatorial optimization, and ring closure heuristics. It includes subproblems which are polynomially or even exponentially growing with respect to the complexity of the underlying geometries. As a consequence, NETQUEST-OPT currently requires computation times of several hours on a single workstation even for medium sized access domains. For an efficient planning process interactivity is a desired property, e.g. for rapid prototyping of different clustering strategies. This goal can only be achieved by using high performance computing systems.

In this work, we focus on the efficient use of NETQUEST-OPT in a distributed environment using a custom distributed MATLAB approach based on the PARMATLAB and TCPIP toolboxes. In Section 2, we describe previous approaches to use MATLAB in parallel and distributed environments and shortly review our own development MDICE. Section 3 describes the basic principles and the structure of NETQUEST-OPT and subsequently discusses a computing intensive part of the tool in some detail. In Section 4 we present experimental results of execution in homogenous and heterogenous environments.

2 MATLAB in Parallel and Distributed Computing

MATLAB has established itself as the numerical computing environment of choice on uniprocessors for a large number of engineers and scientists.

For many scientific applications, the desired levels of performance are only obtainable on parallel or distributed computing platforms. Therefore, a lot of work has been done, both in industry and in academia, to develop high performance MATLAB computing environments.

C. Moler stated in "Why there isn't a parallel MATLAB" (Mathworks newsletter in 1995) that due to the lack of widespread availability of high performance computing (HPC) systems there would be no demand for parallel MATLAB. With the emerge of cluster computing and the potential availability of HPC systems in many universities and companies, the demand for such a software system is obvious. A comprehensive and up-to-date overview on high performance MATLAB systems is given by the "Parallel MATLAB Survey"[1]. Several systems may be downloaded from the MATHWORKS FTP[2] and WWW[3] servers. There are basically three distinct ways to use MATLAB on HPC architectures:

1. Developing a high performance interpreter
   (a) Message passing: communication routines usually based on MPI or PVM are provided. These systems normally require users to add parallel instructions to MATLAB code [4, 5, 6].

[1] http://supertech.lcs.mit.edu/~cly/survey.html
[2] ftp://ftp.mathworks.com/pub/contrib/v5/tools/
[3] http://www.mathtools.net/MATLAB/Parallel/

   (b) "Embarrassingly parallel": routines to split up work among multiple MATLAB sessions are provided in order to support coarse grained parallelization. Note that the PARMATLAB and TCPIP toolboxes and our own development MDICE [7] fall under this category.
2. Calling high performance numerical libraries: parallelizing libraries like e.g. SCALAPACK are called by the MATLAB code [8]. Note that parallelism is restricted within the library and higher level parallelism present at algorithm level cannot be exploited with this approach.
3. Compiling MATLAB to another language (e.g. C, HPF) which executes on HPC systems: the idea is to compile MATLAB scripts to native parallel code [9, 10, 11]. This approach often suffers from complex type/shape analysis issues

Note that using a high performance interpreter usually requires multiple MATLAB clients whereas the use of numerical libraries only requires one MATLAB client. The compiling approach often does not require even a single MATLAB client. On the other hand, the use of numerical libraries and compiling to native parallel code is often restricted to dedicated parallel architectures like multicomputers or multiprocessors, whereas high performance interpreters can be easily used in any kind of HPC environment. This situation also motivates the development of our custom high performance MATLAB environment: since our target HPC systems are (heterogenous) PC clusters running a Windows system based on the NT architecture, we are restricted to the high performance interpreter approach. However, running a MATLAB client on each PC is expensive in terms of licensing fees and computational resources. Consequently, our aim was to develop a high performance interpreter which requires one MATLAB client for distributed execution only.

MATLAB **DI**stributed **C**omputing **E**nvironment (MDICE [7]) is based on the PARMATLAB and TCPIP toolboxes. The PARMATLAB toolbox supports coarse grained parallelization and distributes processes among MATLAB clients over the intranet/internet. Note that each of these clients must be running a MATLAB daemon to be accessed. The communication within PARMATLAB is performed via the TCPIP toolbox. Both toolboxes may be accessed at the Mathworks ftp-server (referenced in the last Section) in the directories `parmatlab` and `tcpip`, respectively.

However, in order to meet the goal to get along with a single MATLAB client the PARMATLAB toolbox needed to be significantly revised. The main idea was to change the client in a way that it can be compiled to a standalone application. At the server, jobs are created and the solve routine is compiled to a program library (*.dll). The program library and the datasets for the job are sent to the client. The client is running as background service on a computer with low priority. For this reason the involved client machines may be used as desktop machines by other users during the computation (however, this causes the need for a dynamic load balancing approach of course). This client calls over a predefined standard routine the program library with the variables sent by the server and sends the output of the routine back to the server. After the receipt of all solutions the server defragments them to an overall solution.

For details on MDICE and corresponding performance results for a Monte Carlo simulation application see [7].

3 Optimization of Fiber Optic Network Layout in the Access Net Domain

3.1 Sequential Approach

In order to consider real world conditions, detailed geoinformation data, usually supplied by the network carriers, are essential. We map the real world geometries to a set of nodes V of a graph $G = (V, E)$. In the simplest case the optimization of network layings means to find an optimal subset E^* of the set of edges E such that all network users $R \subset V$ (R is the set of access nodes) are reachable.

According to the methods in [3] the following additional information is extracted from the GIS databases:

Penalty Grid: A spatially balanced score card combines all relevant land classes with typical implementation costs. The access net domain is regarded as a regular grid $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$, i and j are indices of finite index sets. Each entry of the penalty matrix P describes the specific implementation costs for the grid pixel $[x_i,x_{i+1}] \times [y_j, y_{j+1}]$.

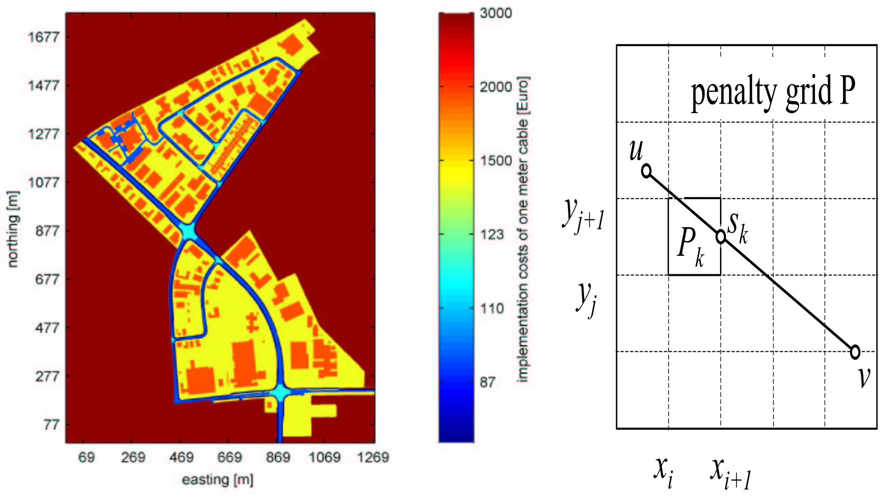
Auxiliary geometries: Plane geometries of all relevant land classes are exported. The shapes of these geometries are determined by a set S of auxiliary nodes.

For a detailed review of some elaborate laying problems treated by NETQUEST-OPT (e.g. consideration of redundancy requirements) we refer to [3]. In all cases of optimization we have to construct a weighted graph $G = (V, E; W)$. Thus we penalize the edges in E with respect to the underlying penalty grid P for the considered network domain. Fig. 1.a shows the corresponding implementation costs for the different land classes of the penalty grid. Figs. 1 and 2 correspond to an industrial benchmark project based on data provided by NetCologne, one of the major private city net carriers in Germany.

For an edge $[u, v] \in E$, $u \in V$, $v \in V$, $u \neq v$, we define the entry $W_{u,v}$ of the symmetric cost matrix $W \in Mat\,(|V| \times |V|)$ according to

$$W_{u,v} := \sum_{k=0}^{K-1} \|s_{k+1} - s_k\|_2 \, P_{k+1} \ . \tag{1}$$

As depicted in Figure 1.b, s_k denotes the (x, y) -coordinates of the k-th intersection point of the edge $[u, v]$ with the grid lines of the cost grid P. P_k represents the specific implementation costs corresponding to the predecessor pixel for the k-th intersection point s_k of an edge $[u, v]$ with the grid lines of the cost grid. The predecessor notation corresponds to the direction from u to v.



(a) Implementation costs

(b) Derivation of the costs

Fig. 1. Penalty grid in the context of network laying

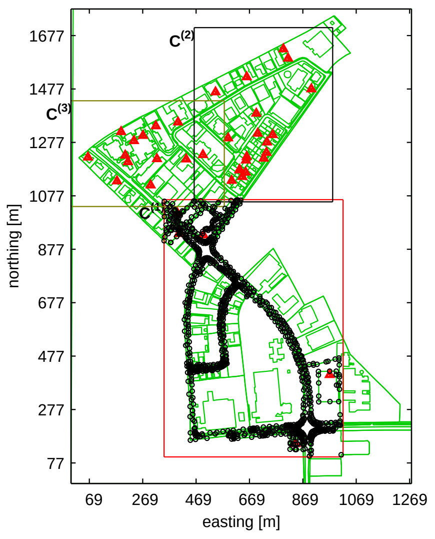
The computation of W is one of the most time demanding procedures in the entire laying optimization process and requires $(|V| - 1)\,|V|\,/2$ calls of (1). In real world geometries, the network planning problem leads to dimensions up to $|V| \cong 20000$ nodes. Due to memory restrictions on our target architectures, these problem dimensions required us to segment the entire optimization procedure into three hierarchical steps: determination of local clusters, local solution in each cluster and computing cluster shortest spanning trees [3].

Fig. 2.a shows the required access nodes for the domain shown in Fig. 2.b. The problem is partitioned into three clusters. The black circles depict the auxiliary nodes of the first cluster based on the underlying geometries, gridding is done with a resolution of one meter in square.

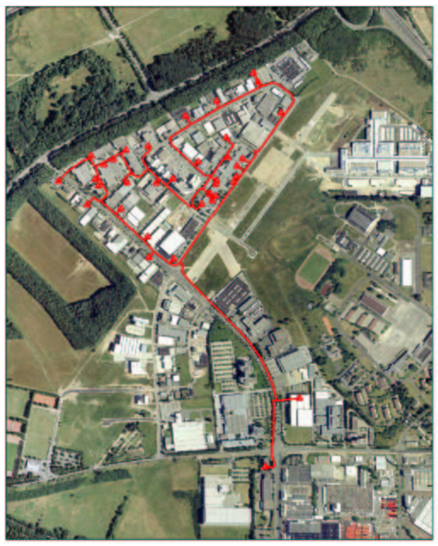
The solution depicted in Fig. 2.b was computed in 223 minutes on three PCs. The result saves 22% distance and 20% costs with respect to the original NetCologne laying.

3.2 Distribution Strategy

The segmentation of the spatial domain into independent clusters for an efficient solution of the network laying problem as described above allows also straightforward and computationally efficient distributed processing at first hand. However, this clustering procedure (inter-cluster parallelism) leads to suboptimal global umbrella networks only, especially in case of a larger number of clusters which



(a) 37 access nodes (triangles) and three clusters



(b) Arial view of target area with overlayed computed solution

Fig. 2. Required access nodes and computed solution.

would be required for scalable parallel or distributed execution. Therefore, clustering must not be used as a means to provide parallelism. For this reason, we propose a "intra-cluster parallelism" approach which does not exploit the parallelism introduced by the clustering process at all. As a consequence, an optimal global solution may be obtained if clustering can be avoided (i.e. the target architecture allows a single-cluster approach).

The computations associated with the evaluation of the cost matrix W (which are covered by our experiments) may be distributed in a simple fashion by domain decomposition without the requirement of inter-process communication. This simple partitioning approach leads to entirely deterministic behaviour since the computational effort of evaluating parts of the matrix W is not data dependent. Therefore, load balancing functionalities seem not to be necessary at first hand. However, due to the possible interference of other applications (recall that the clients run with low priority on desktop machines) and the potential employment of the application in heterogeneous environments, we use a simple load balancing approach (see below). We emphasize that MDICE is also employed for other, more sophisticated subproblems of NETQUEST-OPT – we exemplarily mention the solution of Steiner Network Problems [12], the solution of cluster interconnecting minimal spanning tree problems or the computation of ring closure matrices for redundant networks [3]. However, due to the clarity of the results with respect to the use of MDICE we cover the evaluation of W as exemplary part of NETQUEST-OPT.

4 Experiments

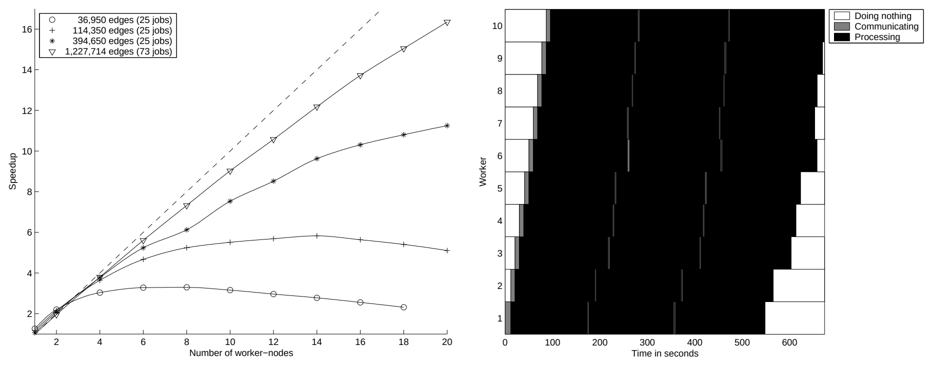
4.1 Experimental Settings

The data required for computing the cost matrix W is split into a certain number of equally sized jobs N to be distributed among the clients M by the server ($N \geq M$). Whenever a client has sent back its result to the server after the initial distribution of M jobs to M clients, the server assigns another job to this idle client until all N jobs are computed. This approach is denoted “asynchronous single task pool method” [13]. In case $N \gg M$ this strategy is a dynamic load balancing approach which can also cope with heterogeneous environments. When the server contacts a client machine the first time, the compiled client software is sent in addition to the required data to perform the first computational job. For all subsequent jobs to be executed on this machine, only data has to be sent (which is much less demanding of course). Note that since MDICE does not provide tools for automatic parallelization, the N jobs (i.e. the decomposition of W) need to be specified before the entire computation starts.

We consider cost matrices W from real-world problems associated with a different number of edges (i.e. 36950, 114350, 394650, and 1227714) in order to show the impact of varying the problemsize on the performance of our solution. Additionally, we vary the number of jobs distributed among the worker clients after fixing the problemsize to 394650 edges to show the tradeoff between good load distribution versus communication effort. The computing infrastructure consists of the server machine (1.99 GHz Intel Pentium 4, 504 MB RAM, Windows XP Prof.) and two types of client machines (996 MHz Intel Pentium 3, 128 MB RAM, and 730 MHz Intel Pentium 3, 128 MB RAM, both types under Windows XP Prof.). The Network is 100 MBit/s Ethernet. In order to demonstrate the flexibility of our approach, we present results in “homogenous” and “heterogenous” environments. In the case of the homogenous environment, we use client machines of the faster type only, the results of the heterogenous environment correspond to six 996 MHz and four 730 MHz clients, respectively. Note that the sequential reference execution time used to compute speedup (see Fig. 3.a) has been achieved on a 996 MHz client machine with a compiled (not interpreted) version of the application to allow a fair comparison since the client code is compiled as well in the distributed application. We use MATLAB 6.5.0 with the MATLAB compiler 3.0 and the LCC C compiler 2.4.

4.2 Experimental Results

First we discuss the homogenous environment. In Fig. 3.a we display speedup results for differently sized matrices W. It is clearly exhibited that speedup saturates for the smaller problems at 4 or 8 clients, respectively. On the other hand, reasonable scalability is shown for the larger problems up to 20 clients.



(a) Speedup for varying problem-size (b) Visualization of execution

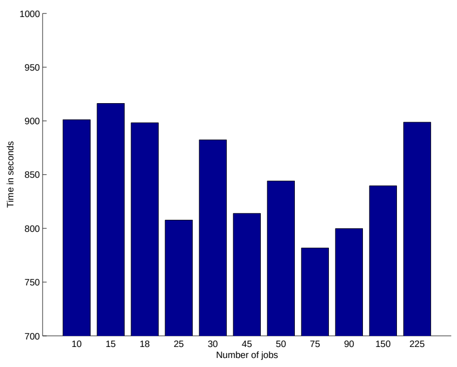
Fig. 3. Results in the homogenous environment.

Fig. 3.b shows a visualization of the distributed execution, where black areas represent computation time-intervals and gray areas communication events. The number of edges has been fixed to 394650, the number of clients is set to 10, and 30 jobs are used. We clearly notice the staggered start of computation at different clients which is due to the expensive initial communication phase, where the server has to send the compiled code and input data to each of the clients. This behaviour is the reason why the settings $N = M$ or $N = n \times M$ (with small n, where $n = 3$ in our case) do not perform as well as expected. The solution is to increase the number of jobs in order to balance the load at the end of the application, however, the delayed start at several clients can not be avoided.

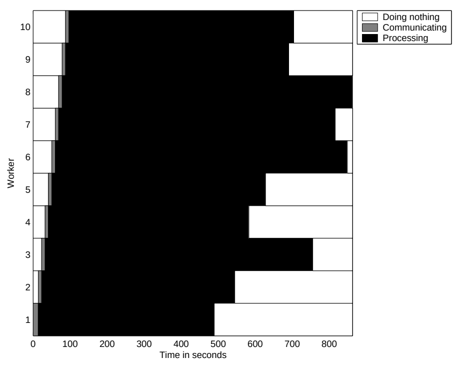
In the following we discuss the heterogenous environment, again the number of edges has been fixed to 394650 and the number of clients is set to 10 as specified in the last Section. In Fig. 4.a we vary the number of jobs distributed among the clients. Clearly, a high number of jobs leads to an excellent load distribution, but on the other hand the communication effort is increased thereby reducing the overall efficiency.

For our target system, the optimal number of jobs is identified to be 75. A further increase leads to an increase of execution time. Fig. 4.b shows an execution visualization where $N = M = 10$. Of course, the performance is worse as compared to the analogous homogenous case and the slower clients (numbers 3,6,7,8) are immediately identified.

Finally, in Figs. 5.a and 5.b we show the execution behaviour for $N = 25$ and $N = 75$, respectively. Although the structure and the visual appearance of the two cases is quite different, we result in almost identical execution times. Whereas for $N = 25$ we have little communication effort but highly unbalanced load, the opposite is true for $N = 75$. The tradeoff between communication and load distribution is inherent in the single pool of task approach which means that the optimal configuration needs to be found for each target environment.

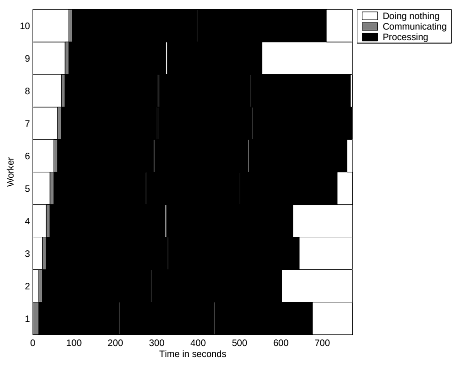


(a) Execution time with a varying number of jobs

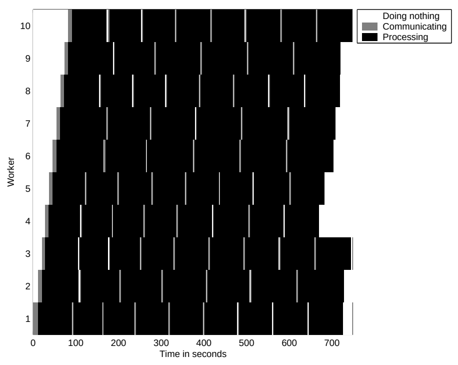


(b) Visualization of execution with 10 jobs

Fig. 4. Results in the heterogenous environment.



(a) 25 jobs



(b) 75 jobs

Fig. 5. Visualization of execution behaviour in the heterogeneous environment.

5 Conclusion

In this work, we employ a MATLAB-based planning tool for the computation of cost optimized laying for fiber optic access networks on homogenous and heterogenous Windows PC networks. The custom MATLAB toolbox MDICE may take advantage of the large number of Windows NT-architecture based machines available in companies and universities. It proves to be efficient in terms of low licencing fees and satisfying execution behaviour. The distributed approach suggested in this work allows for an interactive network planning process provided enough PCs are available. Additionally, our approach may lead to a better quality of the computed network solution as compared to a sequential clustering approach in case the clustering can be avoided in the distributed execution.

References

[1] G. Gilder. *Telecosm: The world after bandwidth abundance.* Touchstone Books, Charmichael, 2002.

[2] D. Bertsekas. *Network Optimization: Continous and Discrete Models.* Athena Scientific, Belmont, 1998.

[3] P. Bachhiesl, G. Paulus, M. Prossegger, J. Werner, and H. Stögner. Cost optimized layout of fibre optic networks in the access net domain. In U. Leopold-Wildburger, F.Rendl, and G. Wäscher, editors, *Proceedings of Operations Research OR'02*, Springer Series on Operations Research, pages 229–234. Springer Verlag, 2002.

[4] J.F. Baldomero. PVMTB: Parallel Virtual Machine Toolbox. In S. Dormido, editor, *Proceedings of II Congreso de Usarios MATLAB'99*, pages 523–532. UNED, Madrid, Spain, 1999.

[5] V.S. Menon and A.E. Trefethen. MultiMATLAB: integrating MATLAB with high performance parallel computing. In *Proceedings of 11th ACM International Confernce on Supercomputing.* ACM SIGARCH and IEEE Computer Society, 1997.

[6] S. Pawletta, T. Pawletta, and W. Drewelow. Comparison of parallel simulation techniques - MATLAB/PSI. *Simulation News Europe*, 13:38–39, 1995.

[7] R. Pfarrhofer, P. Bachhiesl, M. Kelz, H. Stögner, and A. Uhl. MDICE – a MATLAB toolbox for efficient cluster computing. In *Proceedings of Parallel Computing 2003 (ParCo 2003).* Elsevier Science B.V., 2004. To appear.

[8] S. Ramaswamy, E.W. Hodges, and P. Banerjee. Compiling MATLAB programs to SCALAPACK: Exploiting task and data parallelism. In *Proceedings of the International Parallel Processing Symposium (IPPS)*, pages 814–821. IEEE Computer Society Press, 1996.

[9] L. DeRose and D. Padua. A MATLAB to Fortran 90 translator and its effectiveness. In *Proceedings of 10th ACM International Conference on Supercomputing.* ACM SIGARCH and IEEE Computer Society, 1996.

[10] P. Drakenberg, P. Jakobson, and B. Kagström. A CONLAB compiler for a distributed memory multicomputer. In *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, volume 2, pages 814–821, 1993.

[11] M. Quinn, A. Malishevsky, N. Seelam, and Y. Zhao. Preliminary results from a parallel MATLAB compiler. In *Proceedings of the International Parallel Processing Symposium (IPPS)*, pages 81–87. IEEE Computer Society Press, 1998.

[12] A.Z. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9:463–470, 1993.

[13] A.R. Krommer and C.W. Überhuber. Dynamic load balancing – an overview. Technical Report ACPC/TR92-18, Austrian Center for Parallel Computation, 1992.