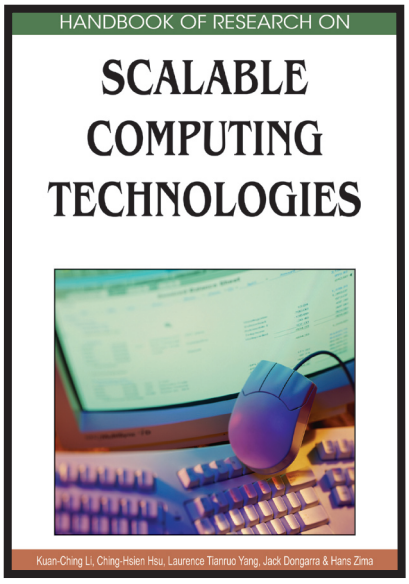
Information Science

REFERENCE

The premier reference source for computer science and information technology management

New Release July 2009

Handbook of Research on Scalable Computing Technologies (2 volumes)



Edited by: Kuan-Ching Li, Providence University, Taiwan; Ching-Hsien Hsu, Chung Hua University, Taiwan; Laurence Tianruo Yang, St. Francis Xavier University, Canada; Jack Dongarra, University of Tennessee, USA; and Hans Zima, University of Vienna, Austria

13-digit ISBN: 978-1-60566-661-7
1,093 pages; 2010 Copyright
Price: US $555.00 (hardcover*)
Perpetual Access: US $830.00
Print + Perpetual Access: US $1,110.00
Illustrations: figures, tables (8 1/2" x 11")
Translation Rights: World

*Paperback is not available.

"This publication is a valuable source targeted to those interested in the development of field of grid engineering for academic or enterprise computing, aimed for computer scientists, researchers and technical managers working all areas of science, engineering and economy from academia, research centers and industries."

- Jack Dongarra, University of Tennessee, USA

The past decade has witnessed a fruitful proliferation of increasingly high performance scalable computing systems mainly due to the availability of enabling technologies in hardware, software, and networks.

The **Handbook of Research on Scalable Computing Technologies** presents ideas, results, and experiences in significant advancements and future challenges of enabling technologies. A defining body of research on topics such as service-oriented computing, data-intensive computing, and cluster and grid computing, this Handbook of Research contains valuable findings for those involved with developing programming tools and environments in computing as well as those in related upper-level undergraduate and graduate courses.

Subject:
High Performance Computing; Software/Systems Design; Computer Engineering; Electronic Services; Networking/Telecommunications; Data Mining/Databases; Web Technologies; Mobile/Wireless Computing

Handbook of Research on Scalable Computing Technologies

Edited by: Kuan-Ching Li, Providence University, Taiwan; Ching-Hsien Hsu, Chung Hua University, Taiwan; Laurence Tianruo Yang, St. Francis Xavier University, Canada; Jack Dongarra, University of Tennessee, USA; Hans Zima, University of Vienna, Austria

Contributors

David Allenotor, University of Manitoba, Canada
Jörn Altmann, International University in Germany, Germany
C. E. R. Alves, Universidade Sao Judas Tadeu, Brazil
Alan A. Bertossi, University of Bologna, Italy
Rajkumar Buyya, The University of Melbourne, Australia
E. N. Cáceres, Universidade Federal de Mato Grosso do Sul, Brazil
Franck Cappello, Université Paris-Sud, France
Ruay-Shiung Chang, National Dong Hwa University, Taiwan
Jih-Sheng Chang, National Dong Hwa University, Taiwan
Zizhong Chen, Colorado School of Mines, USA
Jinjun Chen, Swinburne University of Technologies, Australia
Shang-Feng Chiang, National Taiwan University, Taiwan
Kuo Chiang, National Taiwan University, Taiwan
Kenneth Chiu, University at Binghamtom, State University of NY, USA
Yuan-Shun Dai, University of Electronics Science Technology of China, China
Marcos Dias de Assunção, The University of Melbourne, Australia
Rodrigo Fernandes de Mello, University of São Paulo – ICMC, Brazil
F. Dehne, Carleton University, Canada
Evgueni Dodonov, University of São Paulo – ICMC, Brazil
Jack Dongarra, University of Tennessee, USA
Daniel C. Doolan, Robert Gordon University, UK
Wanchun Dou, Nanjing University P.R. China,
Jörg Dümmler, Chemnitz University of Technology, Germany
Rudolf Eigenmann, Purdue University, USA
Rasit Eskicioglu, University of Manitoba, Canada
Thomas Fahringer, University of Innsbruck, Austria
Gilles Fedak, ENS Lyon, France
Tore Ferm, Sydney University, Australia
Edgar Gabriel, University of Houston, USA
Jean-Luc Gaudiot, University of California, Irvine, USA
Wolfgang Gentzsch, EU Project DEISA and Board of Directors of the Open Grid Forum, Germany
Peter Graham, University of Manitoba, Canada
Alan Grigg, Loughborough University, UK
Dan Grigoras, University College Cork, Ireland
Lin Guan, Loughborough University, UK
Sudha Gunturu, Oklahoma State University, USA
Minyi Guo, Shanghai Jiao Tong University, China
Phalguni Gupta, Indian Institute of Technology Kanpur, India
Xiangjian He, Australia University of Technology, Australia
Yong J. Jang, Yonsei University, Korea
Yanqing Ji, Gonzaga University, USA
Hai Jiang, Arkansas State University, USA
Hong Jiang, University of Nebraska at Lincoln, USA
Derrick Kondo, INRIA Rhônes-Alpes, France
King Tin Lam, The University of Hong Kong, Hong Kong
Xiaobin Li, Intel® Corporation, USA
Xiaolin Li, Oklahoma State University, USA
Chen Liu, Florida International University, USA
Shaoshan Liu, University of California, Irvine, USA
Paul Malécot, Université Paris-Sud, France
V.E. Malyshkin, Russian Academy of Sciences, Russia
Verdi March, National University of Singapore, Singapore
Marian Mihailescu, National University of Singapore, Singapore
Farrukh Nadeem, University of Innsbruck, Austria
Priyadarsi Nanda, University of Technology, Australia
Doohwan Oh, Yonsei University, Korea
Zhonghong Ou, University of Oulu, Finland
Manish Parashar, The State University of New Jersey, USA
Jean-Marc Pierson, Paul Sabatier University, France
M. Cristina Pinotti, University of Perugia, Italy
Radu Prodan, University of Innsbruck, Austria
Dang Minh Quan, International University in Germany, Germany
Rajiv Ranjan, The University of Melbourne, Australia
Thomas Rauber, Chemnitz University of Technology, Germany
Mika Rautiainen, University of Oulu, Finland
Ala Rezmerita, Université Paris-Sud, France
Romeo Rizzi, University of Udine, Italy
Won W. Ro, Yonsei University, Korea
Gudula Rünger, Chemnitz University of Technology, Germany
Haiying Shen, University of Arkansas, USA
Wei Shen, University of Cincinnati, USA
Mohammad Shorfuzzaman, University of Manitoba, Canada
S. W. Song, Universidade de Sao Paulo, Brazil
Junzhao Sun, University of Oulu, Finland
Sabin Tabirca, University College Cork, Ireland
Feilong Tang, Shanghai Jiao Tong University, China
Yong Meng Teo, National University of Singapore, Singapore
Ruppa K. Thulasiram, University of Manitoba, Canada
Parimala Thulasiraman, University of Manitoba, Canada
Daxin Tian, Tianjin University, China
Sameer Tilak, University of California, San Diego, USA
Cho-Li Wang, The University of Hong Kong, Hong Kong
Sheng-De Wang, National Taiwan University, Taiwan
Qiang Wu, University of Technology, Australia
Yang Xiang, Central Queensland University, Australia
Meilian Xu, University of Manitoba, Canada
Laurence T. Yang, St. Francis Xavier University, Canada
Jaeyoung Yi, Yonsei University, Korea
Mika Ylianttila, University of Oulu, Finland
Ruo-Jian Yu, National Taiwan University, Taiwan
Qing-An Zeng, University of Cincinnati, USA
Jiehan Zhou, University of Oulu, Finland
Yifeng Zhu, University of Maine, USA
Albert Y. Zomaya, Sydney University, Australia

About the Editors:

Kuan-Ching Li received PhD and MS degrees in electrical engineering and Licenciatura in mathematics from the University of São Paulo (Brazil). After he received his PhD, he was a postdoctoral scholar at the University of California – Irvine (UCI) and University of Southern California (USC). His main research interests include cluster and grid computing, parallel software design, and life science applications. He has authored over 60 research papers and book chapters, and is the co-editor of the *Handbook of Research on Scalable Computing Technologies* (IGI Global) and volumes of *LNCS* and *LNAI* (Springer). He has served as a guest editor of a number of journal special issues, including the *Journal of Supercomputing* (TJS), the *International Journal of Ad Hoc and Ubiquitous Computing* (IJAHUC), and the *International Journal of Computer Applications in Technology* (IJCAT). In addition, he has served on the steering, organizing, and program committees of several conferences and workshops, including conference co-chair of CSE'2008 (Sao Paulo, Brazil) and program co-chair of APSCC'2008 (Yilan, Taiwan) and AINA'2008 (Okinawa, Japan). He is a senior member of the IEEE.

Ching-Hsien Hsu received his BS and PhD degrees in computer science from Tung Hai University and Feng Chia University (Taiwan, 1995 and 1999, respectively). He is currently an associate professor with the Department of Computer Science and Information Engineering at Chung Hua University (Taiwan). Dr. Hsu's research interests are primarily in parallel and distributed computing, grid computing, P2P computing, RFID, and services computing. Dr. Hsu has published more than 80 academic papers in journals, books, and conference proceedings. He was awarded annual outstanding researcher by Chung Hua University (2005, 2006 and 2007) and got the excellent research award (2008). He is serving on a number of editorial boards for various journals, including the *International Journal of Communication Systems*, the *International Journal of Computer Science*, the *International Journal of Grid and High Performance Computing*, the *International Journal of Smart Home*, and the *International Journal of Multimedia and Ubiquitous Engineering*.

Laurence T. Yang is a professor with the Department of Computer Science at St Francis Xavier University (Canada). His research includes high performance computing and networking, embedded systems, ubiquitous/pervasive computing, and intelligence. He has published around 300 papers (including over 80 international journal papers such as *IEEE* and *ACM Transactions*) in refereed journals, conference proceedings, and book chapters in these areas. He has been involved in more than 100 conferences and workshops as a program/general/steering conference chair and more than 300 conference and workshops as a program committee member. He served as the vice-chair of IEEE Technical Committee of Supercomputing Applications (TCSA) until 2004, currently is the chair of IEEE Technical Committee of Scalable Computing (TCSC), and the chair of IEEE Task force on Ubiquitous Computing and Intelligence. In addition, he is the editor-in-chief of several international journals and a few book series. He is serving as an editor for numerous international journals. He has been acting as an author/co-author or an editor/co-editor of 25 books from Kluwer, Springer, IGI Global, Nova Science, American Scientific Publishers, and John Wiley & Sons. He has won 5 Best Paper Awards (including the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA-06)) and 1 Best Paper Nomination in 2007; as well as a Distinguished Achievement Award, 2005; and Canada Foundation for Innovation Award, 2003.

Jack Dongarra holds an appointment at the University of Tennessee and holds the title of distinguished research staff at Oak Ridge National Laboratory (ORNL), Turing fellow at the University of Manchester. He was awarded the IEEE Sid Fernbach Award (2004) for his contributions in the application of high performance computers using innovative approaches and in 2008 he was the recipient of the IEEE Medal of Excellence in scalable computing. He is a fellow of the AAAS, ACM, and the IEEE and a member of the National Academy of Engineering.

Hans P. Zima is a principal scientist at the Jet Propulsion Laboratory, California Institute of Technology, and a professor emeritus of the University of Vienna (Austria). He received his PhD degree in mathematics and astronomy from the University of Vienna (1964). His major research interests have been in the fields of high-level programming languages, compilers, and advanced software tools. In the early 1970s he designed and implemented one of the first high-level real-time languages for the German Air Traffic Control Agency. During his tenure as a professor of computer science at the University of Bonn (Germany), he contributed to the German supercomputer project "SUPRENUM", leading the design of the first Fortran-based compilation system for distributed-memory architectures (1989). After his move to the University of Vienna, he became the chief designer of the Vienna Fortran language (1992) that provided a major input for the high performance Fortran de-facto standard. From 1997 to 2007, Dr. Zima headed the priority research program "Aurora", a ten-year program funded by the Austrian Science Foundation. His research over the past years focused on the design of the "Chapel" programming language in the framework of the DARPA-sponsored HPCS project "Cascade". More recently, Dr. Zima has become involved in the design of space-borne fault-tolerant high capability computing systems. Dr. Zima is the author or co-author of about 200 publications, including 4 books.

Chapter 17
Communication Issues in Scalable Parallel Computing[1]

C. E. R. Alves
Universidade Sao Judas Tadeu, Brasil

E. N. Cáceres
Universidade Federal de Mato Grosso do Sul, Brasil

F. Dehne
Carleton University, Canada

S. W. Song
Universidade de Sao Paulo, Brasil

ABSTRACT

In this book chapter, the authors discuss some important communication issues to obtain a highly scalable computing system. They consider the CGM (Coarse-Grained Multicomputer) model, a realistic computing model to obtain scalable parallel algorithms. The communication cost is modeled by the number of communication rounds and the objective is to design algorithms that require the minimum number of communication rounds. They discuss some important issues and make considerations of practical importance, based on our previous experience in the design and implementation of parallel algorithms. The first issue is the amount of data transmitted in a communication round. For a practical implementation to be successful they should attempt to minimize this amount, even when it is already within the limit allowed by the CGM model. The second issue concerns the trade-off between the number of communication rounds which the CGM attempts to minimize and the overall communication time taken in the communication rounds. Sometimes a larger number of communication rounds may actually reduce the total amount of data transmitted in the communications rounds. These two issues have guided us to present efficient parallel algorithms for the string similarity problem, used as an illustration.

1. INTRODUCTION

In this book chapter, we discuss some important communication issues to obtain a highly scalable com-

DOI: 10.4018/978-1-60566-661-7.ch017

puting system. Scalability is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged. We consider the CGM (Coarse-Grained Multicomputer) model, a realistic computing model to obtain scalable parallel algorithms. A CGM algorithm that solves a problem of size n with p processors each with $O(n/p)$ memory consists of an alternating sequence of computation rounds and communication rounds. In one communication round, we allow the exchange of $O(n/p)$ data among the processors. The communication cost is modeled by the number of communication rounds and the objective is to design algorithms that require the minimum number of communication rounds. We discuss some important issues and make considerations of practical importance, based on our previous experience in the design and implementation of several parallel algorithms.

The first issue is the amount of data transmitted in a communication round. For a practical implementation to be successful we should attempt to minimize this amount, even when it is already within the maximum allowed by the CGM model which is $O(n/p)$.

The second issue concerns the trade-off between the number of communication rounds which the CGM attempts to minimize and the overall communication time taken in the communication rounds. Under the CGM model we want to minimize the number of communication rounds so that we do not have to care about the particular interconnection network. In a practical implementation, we do have more information concerning the hardware utilized and the communication times in a particular interconnection network. Sometimes a larger number of communication rounds may actually reduce the total amount of data transmitted in the communications rounds. Although the goal of the CGM model is to minimize the number of communication rounds, ultimately the main objective is to minimize the overall running time that includes the computation and the communication times.

These two issues have guided us to present efficient parallel algorithms for the string similarity problem, used as an illustration. By using the wavefront-based algorithms we present in this book chapter to illustrate these two issues, we also address a third issue, the desirability of avoiding costly global communication such as broadcast and all-to-all primitives. This is obtained by using wavefront or systolic parallel algorithms where each processor communicates with only a few other processors.

The string similarity problem is presented here as an illustration. This problem is interesting in its own right. Together with many other important string processing problems (Alves et al., 2006), string similarity is a fundamental problem in Computational Biology that appears in more complex problems (Setubal & Meidanis, 1997), such as the search of similarities between bio-sequences (Needleman & Wunsch, 1970; Sellers, 1980; Smith & Waterman, 1981). We show two wavefront parallel algorithms to solve the string similarity problem. We implement both the basic algorithm (Alves et al., 2002) and the improved algorithm (Alves et al., 2003) by taking into consideration the communication issues discussed in this book chapter and obtain very efficient and scalable solutions.

2. PARALLEL COMPUTATION MODEL

Valiant (1990) introduced a simple *coarse grained* parallel computing model, called *Bulk Synchronous Parallel Model – BSP*. It gives reasonable predictions on the performance of the algorithms when implemented on existing, mainly distributed memory, parallel machines. It is also one of the earliest models to consider communication costs and to abstract the characteristics of parallel machines with a few parameters. The main objective of BSP is to serve a bridging model between the hardware and

software necessities. This is one of the fundamental characteristics for the success of the von Neumann model. In the BSP model, parallel computation is modeled by a series of *super-steps*. In this model, p processors with local memory communicate through some interconnection network managed by a router with global synchronization. A BSP algorithm consists of a sequence of super-steps separated by *synchronization barriers*. In a super-step, each processor executes a set of independent operations using local data available in each processor at the start of the super-step, as well as communication consisting of send and receive of messages. An h-relation in a super-step corresponds to sending or receiving at most h messages in each processor. The response to a message sent in one super-step can only be used in the next super-step.

In this paper we use a similar model called the *Coarse Grained Multicomputers – (denoted by BSP/CGM)*, proposed by Dehne et al. (1993). A BSP/CGM consists of a set of p processors P_1, P_2,…,P_p with $O(n/p)$ local memory per processor and each processor is connected through any interconnection network. The term *coarse granularity* comes from the fact that the problem size in each processor n/p is considerably larger than the number of processors, that is, $n/p>>p$. A BSP/CGM algorithm consists of alternating local computation and global communication rounds separated by a barrier synchronization. The BSP/CGM model uses only two parameters: the input size n and the number of processors p. In a computing round, each processor runs a sequential algorithm to process its data locally. A communication round consists of sending and receiving messages, in such a way that each processor sends at most $O(n/p)$ data and receives at most $O(n/p)$ data. We require that all information sent from a given processor to another processor in one communication round is packed into one long message, thereby minimizing the message overhead.

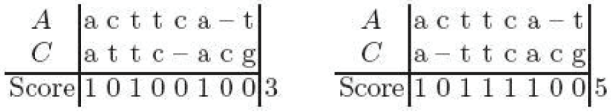
In the BSP/CGM model, the communication cost is modeled by the number of communication rounds which we wish to minimize. In a good BSP/CGM algorithm the number of communication rounds does not depend on the input size n. The ideal algorithm requires a constant number of communication rounds. If this is not possible, we attempt to get an algorithm for which this number is independent on n but depends on p. This is the case of the present chapter.

The BSP/CGM model has the advantage of producing results are close to the actual performance of commercially available parallel machines. Some algorithms for computational geometry and graph problems require a constant number or $O(\log p)$ communication rounds (e.g. see Dehne et al. (1993)). The BSP/CGM model is particularly suitable for current parallel machines in which the global computing speed is considerably greater than the global communication speed.

One way to explore the use of parallel computation can be through the use of clusters of workstations or Fast/Gigabit Ethernet connected Linux-based Beowulf machines, with *Parallel Virtual Machine - PVM* or *Message Passing Interface - MPI* libraries. The latency in such clusters or Beowulf machines of 1Gb/s is currently less than 10 μs and programming using these resources is today a major trend in parallel and distributed computing.

Though much effort has been expended to deal with the problems of interconnection of clusters or *Beowulfs* and the programming environment, there is still few works on methodologies to design and analyze algorithms for scalable parallel computing systems.

Figure 1. String alignment examples



3. THE STRING SIMILARITY PROBLEM

In Molecular Biology, the search for tools that identify, store, compare and analyze very long bio-sequences is becoming a major research area in Computational Biology. In particular, sequence comparison is a fundamental problem that appears in more complex problems (Setubal & Meidanis, 1997), such as the search of similarities between bio-sequences (Needleman & Wunsch, 1970; Sellers, 1980; Smith & Waterman, 1981), as well as in the solution of several other problems such as approximate string matching, file comparison, and text searching with errors (Hall & Dowling, 1980; Hunt & Szymansky, 1977; Wu & Manber, 1992).

One main motivation for biological sequence comparison, in particular proteins, comes from the fact that proteins that have similar tri-dimensional forms usually have the same functionality. The tri-dimensional form is given by the sequence of symbols that constitute the protein. In this way, we can guess a functionality of a new protein by searching a known protein that is similar to it.

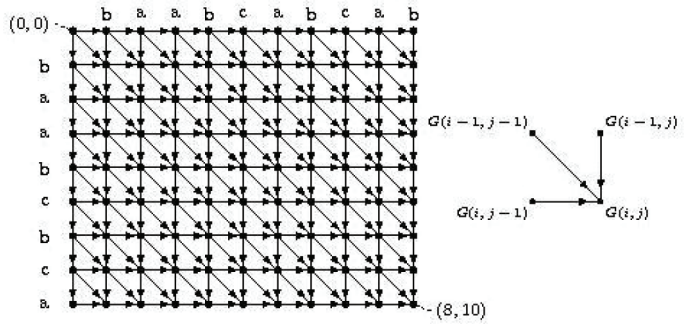
In this section we present the string similarity problem. One way to identify similarities between sequences is to align them, with the insertion of spaces in the two sequences, in such way that the two sequences become equal in length. We expect that the alignment of two sequences that are similar will show the parts where they match, and different parts where spaces are inserted. We are interested in the best alignment between two strings, and the score of such an alignment gives a measure of how much the strings are similar.

The similarity problem is defined as follows. Let A = $a_1a_2\ldots a_m$ and C = $c_1c_2\ldots c_n$ be two strings over some alphabet.

To align the two strings, we insert spaces in the two sequences in such way that they become equal in length. See Figure 1 where each column consists of a symbol of *A* (or a space) and a symbol of *C* (or a space). An *alignment* between *A* and *C* is a matching of the symbols $a \in A$ and $c \in C$ in such way that if we draw lines between the corresponding matched symbols, these lines cannot cross each other. The alignment shows the similarities between the two strings. Figure 1 shows two simple alignment examples where we assign a score of 1 when the aligned symbols in a column match and 0 otherwise. The alignment on the right has a higher score (5) than that on the left (3).

A more general score assignment for a given alignment between strings is done as follows. Each column of the alignment receives a certain value depending on its contents and the total score for the alignment is the sum of the values assigned to its columns. Consider a column consisting of symbols *r* and *s*. If *r* = *s* (i.e. a *match*), it will receive a value $p(r, s) > 0$. If $r \neq s$ (a *mismatch*), the column will receive a value $p(r,s) < 0$. Finally, a column with a space in it receives a value $-k$, where $k \in N$. We look for the alignment (*optimal alignment*) that gives the maximum score. This maximum score is called the *similarity measure* between the two strings to be denoted by *sim*(*A*,*C*) for strings *A* and *C*. There may

Figure 2. Grid DAG G for A= baabcbca and B = baabcabcab



be more than one alignment with maximum score (Setubal & Meidanis, 1997).

Dynamic programming is a technique used in the solution of many optimization and decision problems. It decomposes the problem into a sequence of optimization or decision steps that are interconnected and are solved one after another. The optimal solution of the problem is obtained by the decomposition of the problem in sub-problems, and computing the optimal solution for each sub-problem. By combining these solutions we obtain the optimal solution of the global problem.

Differently from the other optimization methods, such as *linear programming* and *branch and bound*, dynamic programming is not a general technique. Optimization problems should be translated into a more specific form before dynamic programming can be used. This translation can be very difficult. This constitutes a further difficulty in addition to the need of formulating the problem to be solved efficiently by the dynamic programming approach.

Consider two strings A and C, where $|A| = m$ and $|C| = n$. We can solve the string similarity problem by computing all the similarities between arbitrary prefixes of the two strings starting with the shorter prefixes and use previously computed results to solve the problem for larger prefixes. There are $m + 1$ possible prefixes of A and $n + 1$ prefixes of C. Thus, we can arrange our calculations in an $(m + 1) \times (n + 1)$ matrix S where each $S(r,s)$ represents the similarity between $A[1..r]$ and $C[1..s]$, that denote the prefixes $a_1a_2\ldots a_r$ and $c_1c_2\ldots c_s$, respectively.

Observe that we can compute the values of $S(r,s)$ by using the three previous values $S(r-1,s)$, $S(r-1,s-1)$ and $S(r, s-1)$, because there are only three ways to compute an alignment between $A[1..r]$ and $C[1..s]$. We can align $A[1..r]$ with $C[1..s-1]$ and match a space with $C[s]$, or align A[1..r – 1] with $C[1..s-1]$ and match $A[r]$ with $B[s]$, or align A[1..r – 1] with $C[1..s]$ and match a space with $A[r]$. (Figure 2)

Figure 3. The recursive definition of the similarity score

$$S(r,s) = \max \begin{cases} S[r, s-1] - k \\ S[r-1, s-1] + p(r,s) \\ S[r-1, s] - k \end{cases}$$

The similarity score S of the alignment between strings A and C can be computed as in Figure 3.

An $l_1 \times l_2$ grid DAG (Figure 2) is a directed acyclic graph whose vertices are the $l_1 l_2$ points of an $l_1 \times l_2$ grid, with edges from grid point $G(i, j)$ to the grid points $G(i, j + 1)$, $G(i + 1, j)$ and $G(i + 1, j + 1)$.

Let A and C be two strings with $|A| = m$ and $|C| = n$ symbols, respectively. We associate an (m + 1) × (n + 1) grid DAG G with the similarity problem in the natural way: the $(m + 1)(n + 1)$ vertices of G are in one-to-one correspondence with the $(m + 1)(n + 1)$ entries of the S-matrix, and the cost of an edge from vertex (t, l) to vertex (i, j) is equal to k if $t = i$ and $l = j - 1$ or if $t = i - 1$ and $l = j$; and to $p(i, j)$ if $t = i - 1$ and $l = j - 1$.

It is easy to see that the string similarity problem can be viewed as computing the minimum source-sink path in a grid DAG. In Figure 2 the problem is to find the minimum path from (0,0) to (8,10).

A sequential algorithm to compute the similarity between two strings of lengths m and n uses a technique called *dynamic programming*. The complexity of this algorithm is $O(mn)$. The construction of the optimal alignment can be done in sequential time $O(m + n)$ (Setubal & Meidanis, 1997).

PRAM (*Parallel Random Access Machine*) algorithms for the dynamic programming problem have been obtained by Galil and Park (1991). PRAM algorithms for the string editing problem have been proposed by Apostolico et al. (1990). A more general study of parallel algorithms for dynamic programming can be seen in (Gengler, 1996).

We present two algorithms that use the realistic BSP/CGM model. A characteristic and advantage of the wavefront or systolic algorithm is the modest communication requirement, with each processor communicating with few other processors. This makes it very suitable as a potential application for grid computing where we wish to avoid costly global communication operations such as broadcast and all-to-all operations.

4. THE BASIC SIMILARITY ALGORITHM

The basic similarity algorithm is due to Alves et al. (2002). It is a BSP/CGM algorithm and attempts to minimize the number of communication rounds.

Consider two given strings $A = a_1 a_2 \ldots a_m$ and $C = c_1 c_2 \ldots c_n$. The basic similarity algorithm computes the similarity between A and C on a CGM/BSP with p processors and mn/p local memory in each processor.

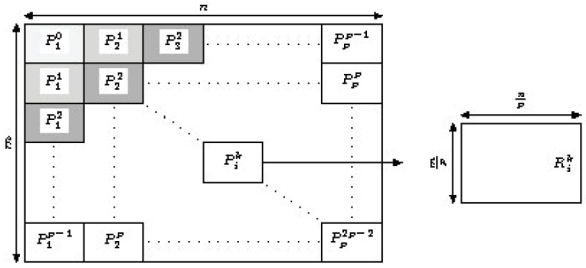
We divide C into p pieces, of size n/p, and each processor P_i, $1 \leq i \leq p$, receives the string A and the i-th piece of C ($c_{(i-1)n/p+1}, \ldots, c_{in/p}$).

Each processor P_i computes the elements $S_i(r, s)$ of the submatrix S_i, where $1 \leq r \leq m$ and $(i-1)n/p+1 \leq s \leq in/p$ using the three previous elements $S_i(r-1,s)$, $S_i(r-1, s-1)$ and $S_i(r, s-1)$, because, as mentioned before, there are only three ways of computing an alignment between $A[1..r]$ and $C[1..s]$. We can align $A[1..r]$ with $C[1..(s-1)]$ and match a space with $C[s]$, or align $A[1..(r-1)]$ with $C[1..(s-1)]$ and match $A[r]$ with $B[s]$, or align $A[1..(r-1)]$ with $C[1..s]$ and match a space with $A[r]$.

To compute the submatrix S_i, each processor P_i uses the best sequential algorithm locally. It is easy to see that processor P_i, $i > 1$, can only start computing the elements $S_i(r, s)$ after the processor P_{i-1} has computed part of the submatrix $S_{i-1}(r, s)$.

Denote by R_i^k, $1 \leq i, k \leq p$, all the elements of the right boundary (rightmost column) of the k-th part of the submatrix S_i. More precisely, $R_i^k = \{S_i(r, in/p, (k-1)m/p + 1 \leq r \leq km/p\}$.

Figure 4. An O(p) communication rounds scheduling used in the basic algorithm



The idea of the algorithm is the following: After computing the k-th part of the submatrix S_i, processor P_i sends to processor P_{i+1} the elements of R_i^k . Using R_i^k, processor P_{i+1} can compute the k-th part of the submatrix S_{i+1}. After $p - 1$ rounds, processor P_p receives $R_{p-1}^{\ 1}$ and computes the first part of the submatrix S_p. At round $2p - 2$, processor P_p receives $R_{p-1}^{\ p}$ and computes the p-th part of the submatrix S_p and finishes the computation.

Using this schedule (Figure 4), we can see that in the first round, only processor P_1 works. In the second round, processors P_1 and P_2 work. It is easy to see that in round k, all processors P_i work, where $1 \le i \le k$.

We now present the basic string similarity algorithm.Basic Similarity Algorithm (see Figure 5).

Theorem 1.

The basic similarity algorithm uses $2p - 2$ communication rounds with $O(mn/p)$ sequential computing time in each processor.

Proof.

Processor P_1 sends $R_1^{\ k}$ to processor P_2 after computing the k-th block of m/p rows of the mn/p sub-

Figure 5. The basic similarity algorithm

Input: (1) The number p of processors; (2) The number i of the processor, where $1 \le i \le p$; and (3) The string A and the substring C_i of size m and $\frac{n}{p}$, respectively.
Output: $S(r,s) = \max\{S[r,s-1]-k, S[r-1,s-1]+p(r,s), S[r-1,s]-k\}$, where $(i-1)\frac{m}{\sqrt{p}}+1 \le r \le i\frac{m}{\sqrt{p}}$ and $(j-1)\frac{n}{p}+1 \le s \le j\frac{n}{p}$.

(1) **for** $1 \le k \le p$
  (1.1) **if** $i = 1$ **then**
    (1.1.1) **for** $(k-1)\frac{m}{p}+1 \le r \le k\frac{m}{p}$ **and** $1 \le s \le \frac{n}{p}$
      `compute` $S(r,s)$;
    (1.1.2) `send`(R_i^k, P_{i+1});
  (1.2) **if** $i \ne 1$ **then**
    (1.2.1) **receive**(R_{i-1}^k, P_{i-1});
    (1.2.2) **for** $(k-1)\frac{m}{p}+1 \le r \le k\frac{m}{p}$ **and** $1 \le s \le \frac{n}{p}$
      `compute` $S(r,s)$;
    (1.2.3) **if** $i \ne p$ **then**
      `send`(R_i^k, P_{i+1});
— End of Algorithm —

Figure 6. Table of running times of the basic algorithm for various string lengths

$m \times n$	1	2	4	8	16	32	64
512×512	0.0558	0.0377	0.0220	0.0138	0.0436	0.0459	0.0483
512×1024	0.1135	0.0787	0.0549	0.0541	0.0449	0.0527	0.0574
1024×1024	0.2246	0.1546	0.0894	0.0705	0.0629	0.0502	0.0621
1024×2048	0.4556	0.3355	0.2032	0.1208	0.1040	0.0926	0.0730
2048×2048	0.9057	0.6588	0.4005	0.2365	0.1526	0.1129	0.1040
2048×4096	1.9459	1.3030	0.7691	0.4316	0.2508	0.1633	0.1272
4096×4096	3.7533	2.5544	1.5285	0.8479	0.5058	0.4569	0.2387
4096×8192	7.5440	5.1274	2.9868	1.6403	0.8875	0.5091	0.3312
8192×8192			5.8492	3.2054	1.7659	1.0096	0.6237
8192×16384				6.3080	3.3550	1.8139	1.0178

matrix S_1. After $p - 1$ communication rounds, processor P_1 finishes its work. Similarly, processor P_2 finishes its work after p communication rounds. Then, after $p - 2 + i$ communication rounds, processor P_i finishes its work. Since we have p processors, after $2p - 2$ communication rounds, all the p processors have finished their work.

Each processor uses a sequential algorithm to compute the similarity submatrix S_i. Thus this algorithm takes $O(mn/p)$ computing time.

Theorem 2.

At the end of the basic similarity algorithm, $S(m, n)$ will store the score of the similarity between the strings A and C.

Proof.

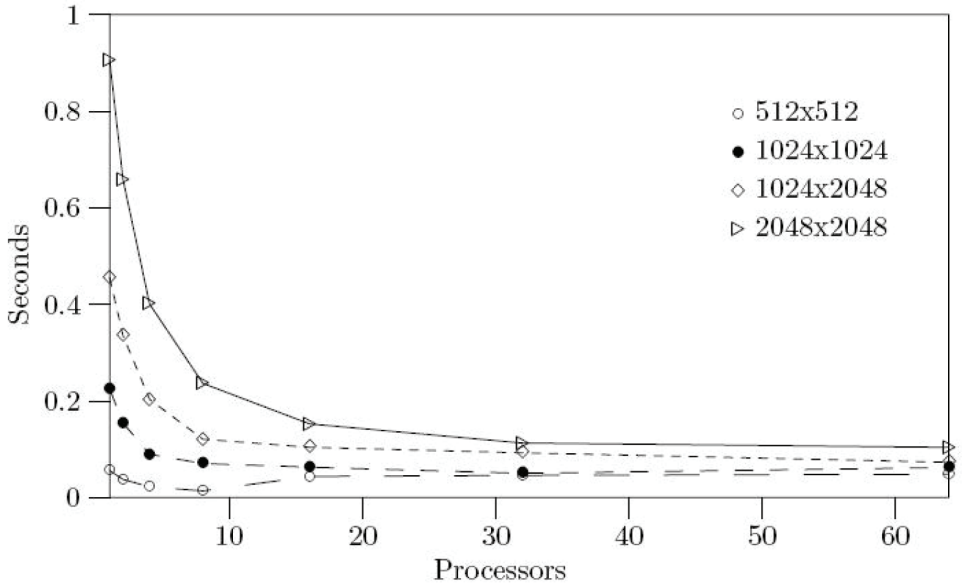
By Theorem 1, after $2p - 2$ communication rounds, processor P_p finishes its work. Since we are essentially computing the similarity sequentially in each processor and sending the boundaries to the right processor, the correctness of the algorithm comes naturally from the correctness of the sequential algorithm. Then, after $2p - 2$ communication rounds, $S(m, n)$ will store the similarity between the strings A and C.

4.1. Experimental Results of the Basic Algorithm

In this section we present the experimental results of the basic similarity algorithm. The following figures give running time curves.

We have implemented the $O(p)$ rounds basic similarity algorithm on a Beowulf with 64 nodes. Each node has 256 MB of RAM memory and more 256 MB for swap. The nodes are connected through a 100 MB interconnection network.

Figure 7. Curves of the observed times for various string lengths



The obtained times (Figures 6, 7 and 8) show that with *small* sequences, the communication time is

Figure 8. Curves of the observed times for various string lengths

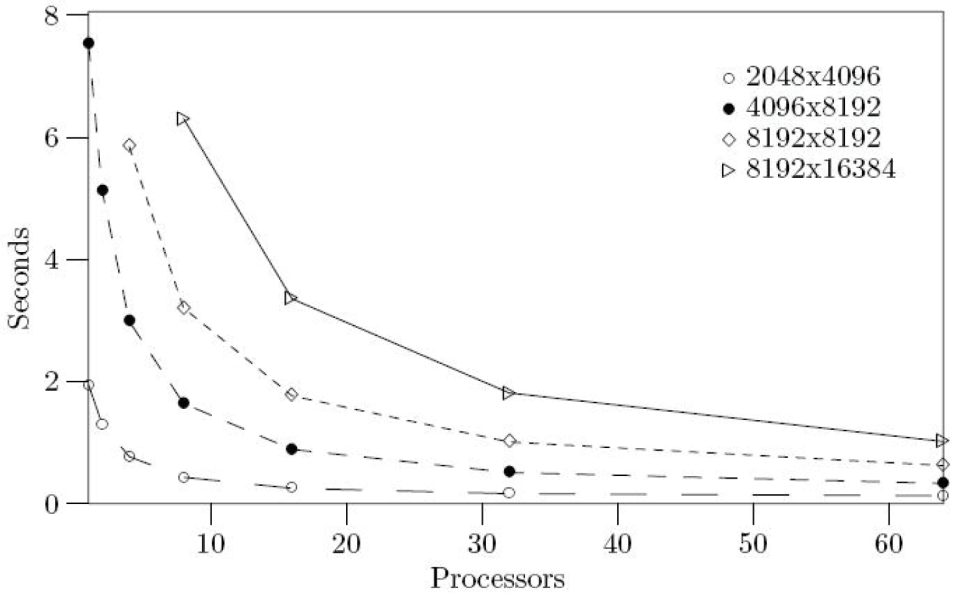
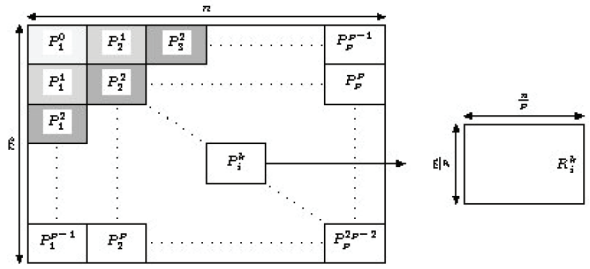


Figure 9. An O(p) communication rounds scheduling with α= 1



significant when compared to the computation time with more than 8 and 16 processors, respectively (512 × 512 and 512 × 1024). When we apply the algorithm to sequences greater than 8192, using one or two processors, the main memory is not enough to solve the problem. The utilization of swap gives us meaningless resulting times. This would not occur if the nodes have more main memory. Thus we have suppressed these times.

In general, the implementation of the CGM/BSP algorithm shows that the theoretical results are confirmed in the implementation.

The basic similarity algorithm requires $O(p)$ communication rounds to compute the score of the similarity between two strings. We have worked with a fixed block size of $m/p \times n/p$. Another good alternative is to work with *adaptative choice* of the optimal block size to further decrease the running time of the algorithm.

The alignment between the two strings can be obtained with $O(p)$ communication rounds backtracking from the lower right corner of the grid graph in $O(m + n)$ time (Setubal & Meidanis, 1997). For this, $S(r, s)$ for all points of the grid graph must be stored during the computation (requiring $O(mn)$ space).

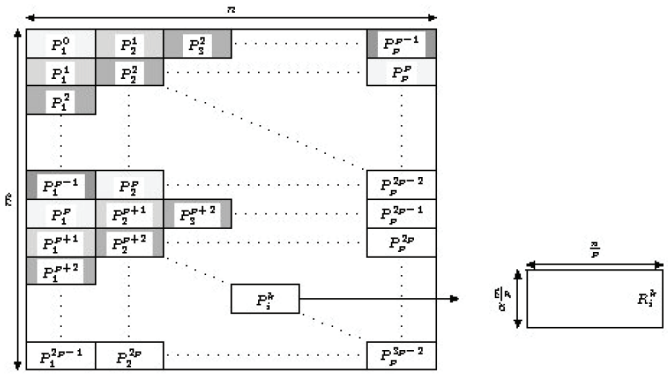
5. THE IMPROVED SIMILARITY ALGORITHM

Alves et al. (2003) extend and improve the basic similarity algorithm (Alves et al., 2002) for computing an alignment between two strings A and C, with $A = |m|$ and $C = |n|$. On a distributed memory parallel computer of p processors each with $O((m + n) / p)$ memory, the improved algorithm also requires $O(p)$ communication rounds, more precisely $(1 + 1 / \alpha)p - 2$ communication rounds where α is a parameter to be presented shortly, and $O(mn / p)$ local computing time. As in the basic algorithm, the processors communicate in a wavefront or systolic manner, such that each processor communicates with few other processors. Actually each processor sends data to only two other processors.

The novelty of the improved similarity algorithm is based on a compromise between the workload of each processor and the number of communication rounds required, expressed by a parameter called α. The proposed algorithm is expressed in terms of this parameter that can be tuned to obtain the best overall parallel time in a given implementation. In addition to showing theoretic complexity we confirm the efficiency of the proposed algorithm through implementation. As will be seen shortly, very promising experimental results are obtained on a 64-node Beowulf machine.

We present a parameterized $O(p)$ communication rounds parallel algorithm for computing the similarity between two strings A and C, over some alphabet, with $|A| = m$ and $|C| = n$. We use the CGM/BSP

Figure 10. An O(p) communication rounds scheduling with α = 1/2



model with p processors, where each processor has $O(mn / p)$ local memory. As will be seen later, this can be reduced to $O((m + n) / p)$.

Let us first give the main idea to compute the similarity matrix S by p processors. The string A is broadcasted to all processors, and the string C is divided into p pieces, of size n / p, and each processor P_i, $1 \le l \le p$, receives the i-th piece of C ($c_{(i-1)n/p+1} \ldots c_{in/p}$).

The scheduling scheme is illustrated in Figure 9. The notation P_i^k denotes the work of Processor P_i at round k. Thus initially P_1 starts computing at round 0. Then P_1 and P_2 can work at round 1, P_1, P_2 and P_3 at round 2, and so on. In other words, after computing the k-th part of the sub-matrix S_i (denoted S_i^k), processor P_i sends to processor P_{i+1} the elements of the right boundary (rightmost column) of S_i^k. These elements are denoted by R_i^k. Using R_i^k, processor P_{i+1} can compute the k-th part of the sub-matrix S_{i+1}. After $p - 1$ rounds, processor P_p receives R_{p-1}^1 and computes the first part of the sub-matrix S_p. In round $2p - 2$, processor P_p receives R_{p-1}^p and computes the p-th part of the sub-matrix S_p and finishes the computation.

It is easy to see that with this scheduling, processor P_p only initiates its work when processor P_1 is

Figure 11. The improved similarity algorithm

Input: (1) The number p of processors; (2) The number i of the processor, where $1 \le i \le p$; and (3) The string A and the substring C_i of size m and $\frac{n}{p}$, respectively; (4) The constant α.
Output: $S(r,s) = \max\{S[r,s-1]-k, S[r-1,s-1]+p(r,s), S[r-1,s]-k\}$, where $(i-1)\frac{m}{\sqrt{p}}+1 \le r \le i\frac{m}{\sqrt{p}}$ and $(j-1)\frac{n}{p}+1 \le s \le j\frac{n}{p}$.

(1) **for** $1 \le k \le \frac{p}{\alpha}$
  (1.1) **if** $i = 1$ **then**
    (1.1.1) **for** $\alpha(k-1)\frac{m}{p}+1 \le r \le \alpha k\frac{m}{p}$ **and** $1 \le s \le \frac{n}{p}$
      `compute` $S(r,s)$;
    (1.1.2) `send`(R_i^k, P_{i+1});
  (1.2) **if** $i \neq 1$ **then**
    (1.2.1) **receive**(R_{i-1}^k, P_{i-1});
    (1.2.2) **for** $\alpha(k-1)\frac{m}{p}+1 \le r \le \alpha k\frac{m}{p}$ **and** $1 \le s \le \frac{n}{p}$
      `compute` $S(r,s)$;
    (1.2.3) **if** $i \neq p$ **then**
      `send`(R_i^k, P_{i+1});
— End of Algorithm —

finishing its computation, at round $p - 1$. Therefore, we have a very poor load balancing.

In the following we attempt to assign work to the processors as soon as possible. This can be done by decreasing the size of the messages that processor P_i sends to processors P_{i+1}. Instead of message size m / p we consider sizes $\alpha\, m / p$ and explore several sizes of α. In our work, we make the assumption that the sizes of the messages $\alpha\, m / p$ divides m. Therefore, S_i^k (the similarity sub-matrix computed by processor P_i at round k) represents $k\, \alpha\, m / p + 1$ to $(k + 1)\, \alpha\, m / p$ rows of S_i that are computed at the k-th round.

We now present the improved similarity algorithm.

The improved algorithm works as follow: After computing S_i^k, processor P_i sends R_i^k to processor P_{i+1}. Processor P_{i+1} receives R_i^k from P_i and computes S_{i+1}^{k+1}. After $p - 2$ rounds, processor P_p receives R_{p-1}^{p-2} and computes S_p^{p-1}. If we use $\alpha < 1$ all the processors will work simultaneously after the $p - 2$-th round. We explore several values for α trying to find a balance between the workload of the processors and the number of rounds of the algorithms. Figure 10 shows how the algorithm works when $\alpha = 1/2$. In this case, processor P_p receives R_{p-1}^{3p-3}, computes S_p^{3p-2} and finishes the computation.

Improved Similarity Algorithm (see Figure 11).

Using the schedule of Figure 10, we can see that in the first round, only processor P_1 works. In the second round, processors P_1 and P_2 work. It is easy to see that at the k-th round, all processors P_i work, where $1 \leq i \leq k$. Since the total number of rounds is increased with smaller values of α the processors start working earlier.

Theorem 3

The improved algorithm uses $(1 + 1 / \alpha)p - 2$ communication rounds with mn / p sequential computing time in each processor.

Proof:

Processor P_1 sends R_1^k to processor P_2 after computing the k-th block of $\alpha\, m / p$ rows of the mn / p sub-matrix S_1. After $p / \alpha - 1$ communication rounds, processor P_1 finishes its work. Similarly, processor P_2 finishes its work after p / α communication rounds. Then, after $p / \alpha - 2 + i$ communication rounds, processor P_i finishes its work. Since we have p processors, after $(1 + 1 / \alpha)p - 2$ communication rounds, all the p processors have finished their work.

Each processor uses a sequential algorithm to compute the similarity sub-matrix S_i. Thus this algorithm takes $O(mn / p)$ computing time.

Theorem 4

At the end of the improved algorithm, $S(m, n)$ will store the score of the similarity between the strings A and C.

Proof:

Figure 12. Table showing running times for various values of αwith m=8K and n=16K

$8K \times 16K$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$
$\alpha = 2$	16.000	8.6738	4.7117	2.5494	1.4172
$\alpha = 1$	11.622	6.2732	3.3209	1.8213	1.0718
$\alpha = 1/2$	9.5802	5.0730	2.6848	1.4811	0.9023
$\alpha = 1/4$	8.5727	4.4721	2.3604	1.3726	0.8306
$\alpha = 1/8$	8.0455	4.1770	2.2151	1.3349	0.8107
$\alpha = 1/16$	7.7996	4.0530	2.1522	1.2794	0.8681
$\alpha = 1/32$	7.7079	3.9948	2.1469	1.3295	0.9656
$\alpha = 1/64$	7.6800	3.9857	2.1891	1.4127	1.1525

Theorem 3 proves that after $(1 + 1 / \alpha)p - 2$ communication rounds, processor P_p finishes its work. Since we are essentially computing the similarity sequentially in each processor and sending the boundaries to the right processor, the correctness of the algorithm comes naturally from the correctness of the sequential algorithm. Then, after $(1 + 1 / \alpha)p - 2$ communication rounds, $S(m, n)$ will store the similarity between the strings A and C.

5.1. Experimental Results of the Improved Similarity Algorithm

In this section we present the experimental results of the improved similarity algorithm. We have implemented the improved similarity algorithm on a Beowulf with 64 nodes. Each node has 256 MB of RAM

Figure 13. Time curves vs. number of processors with m=8K and n=16K

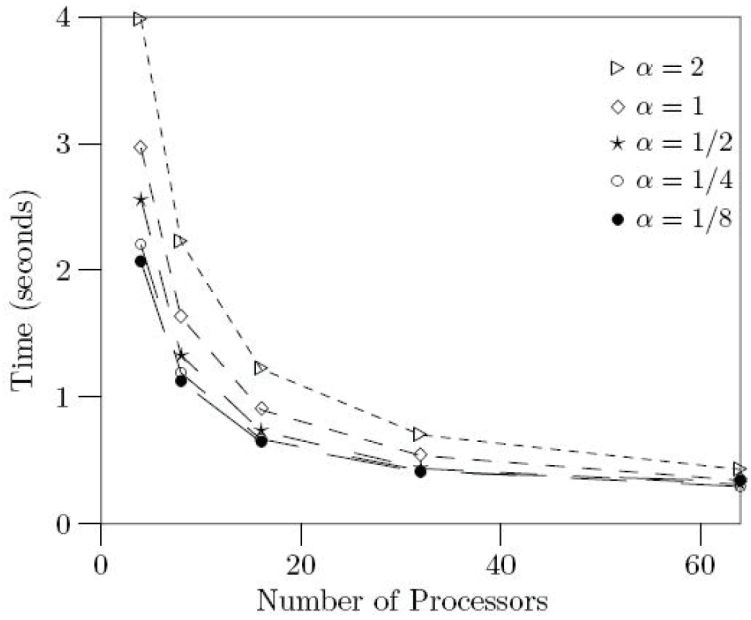


Figure 14. Time curves vs. values of α with m=8K and n=16K

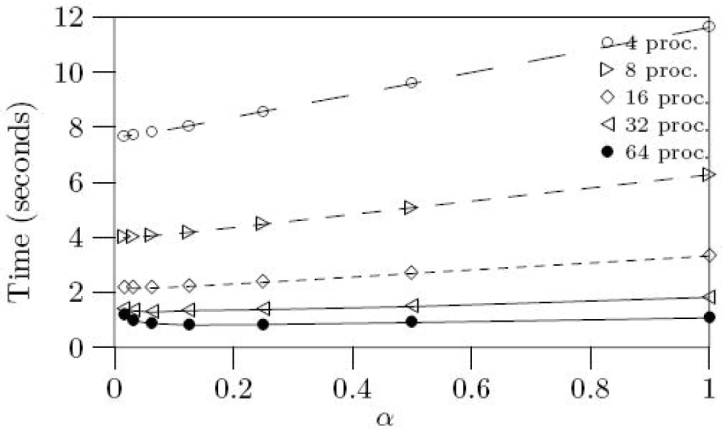


Figure 15. Table showing running times for various values of α with m=4K and n=8K

$4K \times 8K$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$
$\alpha = 2$	3.9790	2.2244	1.2223	0.7007	0.4281
$\alpha = 1$	2.9637	1.6294	0.8986	0.5373	0.3426
$\alpha = 0.5$	2.5599	1.3295	0.7320	0.4353	0.3112
$\alpha = 0.25$	2.1977	1.1891	0.6680	0.4199	0.2938
$\alpha = 0.125$	2.0660	1.1224	0.6452	0.4067	0.3367
$\alpha = 0.0625$	2.0197	1.0857	0.6310	0.4298	0.3637
$\alpha = 0.03125$	1.9956	1.0841	0.6493	0.4632	0.4852
$\alpha = 0.015625$	1.9840	1.0964	0.6996	0.5668	
$\alpha = 1/2$	2.5599	1.3295	0.7320	0.4353	0.3112
$\alpha = 1/4$	2.1977	1.1891	0.6680	0.4199	0.2938
$\alpha = 1/8$	2.0660	1.1224	0.6452	0.4067	0.3367
$\alpha = 1/16$	2.0197	1.0857	0.6310	0.4298	0.3637
$\alpha = 1/32$	1.9956	1.0841	0.6493	0.4632	0.4852
$\alpha = 1/64$	1.9840	1.0964	0.6996	0.5668	

Figure 16. Time curves versus number of processors with m=4K and n=8K

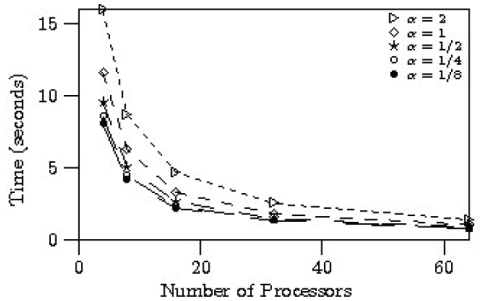
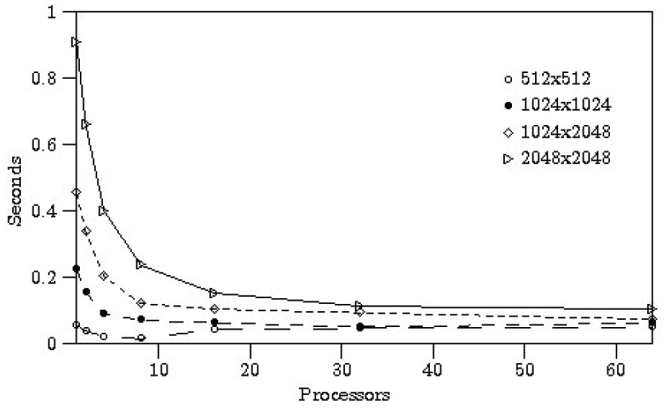


Figure 17. Curves of the observed times - quadratic space



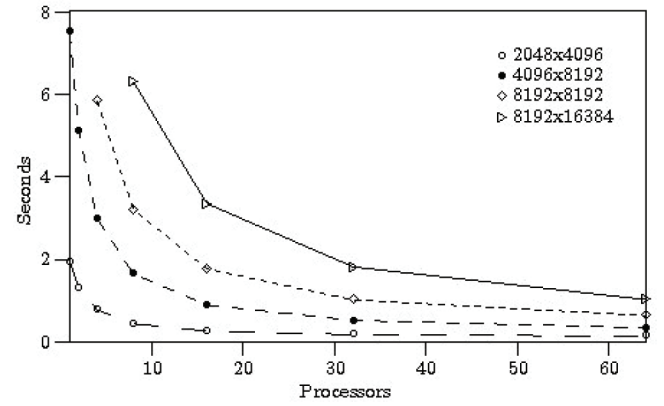
memory in addition to 256 MB for swap. The nodes are connected through a 100 MB interconnection network.

Figures 12, 13 and 14 show the running times of the improved similarity algorithm for different values of α for string lengths of m=8K and n=16K. For a given experiment and hardware platform a parameter tuning phase is required to obtain the best value for α.

Figures 12, 13 and 14 show running times for string sizes m =8K and n =16K where K=1024. It can be seen that, for *very small* α, the communication time is significant when compared to the computation time. We have analyzed the behavior of α to estimate the optimal block size. The observed times show that when $\alpha\, m / p$ decreases from 16 to 8 (the number of rows of the sub-matrix $S_i(k)$), we have an increase on the total time. The best times are obtained for α between 1/4 and 1/8.

Figures 15 and 16 show the running times of the improved similarity algorithm for different values of α for string lengths of m=4K and n=8K. Again, for a given experiment and hardware platform a parameter tuning phase is required to obtain the best value for α.

Figure 18. Curves of the observed times - linear space



5.2. Quadratic vs. Linear Space Implementation

We can further improve our results by exploring a linear space implementation, by storing a vector instead of the entire matrix. In the usual quadratic space implementation, each processor uses $O(mn / p)$ space, while in the linear space implementation each processor requires only $O((m + n) / p)$ space. The results are impressive, as shown in Figures 17 and 18. With less demand on the swap of disk space, we get an almost 50% improvement. We have used $\alpha=1$.

6. CONCLUSION

We have presented a basic and an improved parameterized BSP/CGM parallel algorithm to compute the score of the similarity between two strings. On a distributed memory parallel computer of p processors each with $O((m + n) / p)$ memory, the proposed algorithm requires $O(p)$ communication rounds and $O(mn / p)$ local computing time. The novelty of the improved similarity algorithm is based on a compromise between the workload of each processor and the number of communication rounds required, expressed by a new parameter called α. We have worked with a variable block size of $\alpha\, m / p \times n / p$ and studied the behavior of the block size. We show how this parameter can be tuned to obtain the best overall parallel time in a given implementation. Very promising experimental results are shown.

Though we dedicated considerable space to present the two string similarity algorithms, these algorithms serve the purpose of illustrating two main issues. The first issue is the amount of data transmitted in a communication round. For a practical implementation to be successful we should attempt to minimize this amount, even when it is already within the limit allowed by the CGM model. The second issue concerns the trade-off between the number of communication rounds which the CGM attempts to minimize and the overall communication time taken in the communication rounds. Sometimes a larger number of communication rounds may actually reduce the total amount of data transmitted in the communications rounds. To this end the parameter α is introduced in the improved similarity algorithm. By adjusting the proper value of α, we can actually require more communication rounds while diminishing the total amount of data transmitted in the communication rounds, thus resulting in a more efficient solution.

As a final observation notice that a characteristic of the wavefront communication requirement is that each processor communicates with few other processors. This makes it very suitable as a potential application for grid computing.

REFERENCES

Alves, C. E. R., Caceres, E. N., Dehne, F., & Song, S. W. (2002). A CGM/BSP Parallel Similarity Algorithm. In Proceedings *I Brazilian Workshop on Bioinformatics* (pp. 1-8). Porto Alegre: SBC Computer Society.

Alves, C. E. R., Caceres, E. N., Dehne, F., & Song, S. W. (2003). A Parallel Wavefront Algorithm for Efficient Biological Sequence Comparison. In Kumar, M. L. Gavrilva, C. J. K. Tan, & P. L'Ecuyer (Eds.). *The 2003 International Conference on Computational Science and its Applications.* (LNCS Vol. 2668, pp. 249-258). Berlin: Springer Verlag.

Alves, C. E. R., Caceres, E. N., & Song, S. W. (2006). A coarse-grained parallel algorithm for the all-substrings longest common subsequence problem. *Algorithmica*, *45*(3), 301–335. doi:10.1007/s00453-006-1216-z

Apostolico, A., Atallah, M. J., Larmore, L. L., & Macfaddin, S. (1990). Efficient parallel algorithms for string editing and related problems. *SIAM Journal on Computing*, *19*(5), 968–988. doi:10.1137/0219066

Dehne, F. (1999). Coarse grained parallel algorithms. *Algorithmica*, *24*(3/4), 173–176.

Dehne, F., Fabri, A., & Rau-Chaplin, A. (1993). Scalable parallel geometric algorithms for coarse grained multicomputers. In Proceedings *ACM 9th Annual Computational Geometry* (pp. 298-307).

Galil, Z., & Park, K. (1991). *Parallel dynamic programming* (Tech. Rep. CUCS-040-91). New York: Columbia University, Computer Science Department.

Gengler, M. (1996). An introduction to parallel dynamic programming. In *Solving Combinatorial Optimization Problems in Parallel*. (LNCS Vol. 1054 pp. 87-114). Berlin: Springer Verlag.

Hall, P. A., & Dowling, G. R. (1980). Approximate string matching. *Comput. Surveys*, *12*(4), 381–402. doi:10.1145/356827.356830

Hunt, J. W., & Szymansky, T. (1977). An algorithm for differential file comparison. *Communications of the ACM*, *20*(5), 350–353. doi:10.1145/359581.359603

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453. doi:10.1016/0022-2836(70)90057-4

Sellers, P. H. (1980). The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, (4): 359–373. doi:10.1016/0196-6774(80)90016-4

Setubal, J., & Meidanis, J. (1997). *Introduction to computational molecular biology*. Boston: PWS Publishing Company.

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *J. Mol. Bio.* (147), 195-197.

Valiant, L. (1990). A bridging model for parallel computation. *Communications of the ACM*, *33*(8), 103–111. doi:10.1145/79173.79181

Wu, S., & Manber, U. (1992). Fast text searching allowing errors. *Communications of the ACM*, *35*(10), 83–91. doi:10.1145/135239.135244

KEY TERMS AND DEFINITIONS

Coarse-Grained Multicomputer: A simple and realistic parallel computing model, characterized by two parameters (input size n and number of processors p), in which local computation rounds alternate with global communication rounds, with the goal of minimizing the number of communication

rounds. **Granularity:** A measure of the size of the components, or descriptions of components, that make up a system. In parallel computing, granularity refers to the amount of computation that can be performed by the processors before requiring a communication stepto exchange data. **Scalability:** A desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged.

String Similarity Metrics: Textual based metrics resulting in a similarity or dissimilarity (distance) score between two pairs of text strings for approximate matching or comparison. **Systolic Algorithm:** An algorithm that has the characteristics of a systolic array.

Systolic Array: A pipelined network of processing elements called cells, used in parallel computing, where cells compute data and store it independently of each other and passes the computed data to neighbor cells. **Wavefront Algorithm:** An algorithm that has the characteristics of a systolic array, also known as systolic algorithm.

ENDNOTE

[1] Partially supported by FAPESP Proc. No. 2004/08928-3, CNPq Proc. No. 55.0094/05-9, 55.0895/07-8, 30.5362/06-2, 30.2942/04-1, 62.0123/04-4, 48.5460/06-8, FUNDECT 41/100.115/2006, and the Natural Sciences and Engineering Resarch of Canada.