Sindh Univ. Res. Jour. (Sci. Ser.) Vol. 43 (2) 149-156 (2011)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Performance Evaluation of Protein StructureComparison Algorithms Under Integrated Resource Management Environment for MPI Jobs

A.A. SHAH, I.A. ISMAILI, L.D. DHOMEJA, G. FOLINO* AND N. KRASNOGOR**

(imdad.ismaili,lachhman) @usindh.edu.pk , folino@icar.cnr.it, natalio.krasnogor@nottingham.ac.uk

Institute of ICT, University of Sindh, Pakistan

+Corresponding author: A. A. SHAH, (azhar.shah, @ usindh.edu.pk) Cell. 92-3473707691

Received 28rth June 2011 and Revised 11th July 2011)

Abstract: The comparison of protein tertiary structures is a key milestone in many structural bioinformatics activities that rely in comparing very large structure datasets. As the number of proteins in the dataset increases, the corresponding computational time taken by the protein structure comparison algorithms also increases, squarely for an all-against-all comparison and linearly for an all-against-target assessment. Thus ever larger proteomics problems call for the distribution of pairwise comparison jobs in the form of well granulated subsets/packages to be run in parallel on a pool of networked processors/workstations under the coordination of a Message Passing Interface (MPI) environment. This paper evaluates the effect on the performance of such jobs when the MPI environment is integrated with a Local Resource Management System (LRMS) such as Sun Grid Engine (SGE). From our experiments with different ways of integration we draw a comparative picture of all possible approaches with the description of resource usage information for each parallel job on each processor. Understanding of different ways of integration sheds light on the most promising routes for setting up an efficient environment for very large scale protein structure comparisons.

Keywords: Protein Structure Comparison, Protein Similarity, ProCKSI

1. INTRODUCTION

Protein Structure Comparison (PSC) is at the core of many important structural biology problems. PSC is used to infer the evolutionary history of distantly related proteins; it can also help in the identification of the biological function of a new protein by comparing it with other proteins whose function has already been annotated; PSC is also a key step in protein structure prediction, because one needs to reliably and efficiently compare tens or hundreds of thousands of decoys (predicted structures) for '*native-like*' candidates (e.g. *Critical Assessment of Techniques for Protein Structure Prediction* (CASP) experiment). There is a variety of methods and tools used for PSC such as the *Universal Similarity Metric* (USM) (Krasnogor and Pelta, 2004), *Fast Alignment and Search Tool* (FAST) (Zhu , 2005), *Maximum Contact Map Overlap* (MaxCMO-MSVNS) (Pelta *et al.*, 2008), *Distance Alignment method* (DaliLite) (Holm and Park, 2000), *Combinatorial Extension* (CE) method (Shindyalov and Bourne, 1998), and TMAlign (Zhang and Skolnick, 2005) to Name but a few. It has recently been shown (Barthel *et al.*, 2007) that, as each comparison method offers a specific view of what *"similarity"* means, a consensus of various methods should be computed to obtain a robust and unbiased assessment of similarity. Thus not only large datasets must be handled but also they need to be efficiently processed through several (potentially expensive) algorithms. In order to cope with the resource-hungry nature of such an integrated system a grid-styled distributed framework for *Protein (Structure) Comparison Knowledge Similarity and Information* (ProCKSI) has been proposed in (Folino *et al.*, 2008).

This paper builds on recent work presented in (Barthel *et al.*, 2007) and (Folino *et al.*, 2008), by investigating the effect of allowing a *Local Resource Management System* (LRMS) such as the *Sun Grid Engine* (SGE) (Sistare *et al.*, 2001) to take control over distributed PSC jobs that are run with *Message Passing Interface* (MPI). To this aim we use an integrated architecture for MPI jobs similar to the one proposed in (Sistare *et al.*, 2002). This type of integrated architecture enables the LRMS to have full control over the jobs and

* Institute of High Performance Computing and Networking, Italian National Research Council, Cosenza 87036, Italy

**School of Computer Science, University of Nottingham, NG81BB, UK

+An earlier version of this article was presented at ISPA08 workshop, Sydney, Australia

hence provide the user with better monitoring and resource usage information. We evaluate the performance of PSC jobs running under such an integrated architecture configured with different approaches (as described in section 3) in order to provide a scientific basis for decision making while building a large scale e-Science infrastructure that meets with the requirements of today high-impact scientific applications.

The remainder of this paper is organized as follows: section 2.1 provides a succinct introduction to protein structure data and the preparation of datasets that have been used for this study; section 2.2 briefly introduces the testbed setup consisting of SGE, parallel libraries and the additional scripts used for their integration; section 3 provides the results and discussions of protein structure comparison jobs when run with different integration methods; finally section 4 concludes the paper.

2. MATERIALS AND METHODS

2.1 Preparation of Protein Tertiary Structure Datasets

Structural similarity of proteins is obtained by means of the comparison (alignment) of tertiary (3D) structure data. In many cases this data is publicly available through databases such as the *Protein Data Bank* (PDB) (Berman *et al.*, 2000), the *Macromolecular Structure Database* (MSD) and the *Research Collaboratory for Structural Bioinformatics* (RCSB). At the time of writing this paper, there are *46,730* proteins, 1847 nucleic acids (NAs) and 2013 protein/NA complexes making a total of *50623* entries in the PDB. According to PDB statistics, the number of entries in PDB grows exponentially. Each protein structure file in PDB format, can be composed of several *models* of the same structure determined with several different techniques such as X-ray crystallography, NMR, Electron Microscopy etc. Each of these models, in turn, consist of several different *chains*. For all those structure comparison methods listed in the introduction, the original PDB files need to be parsed and re-written as more simple files consisting of only one chain of any particular model to be used as input to these algorithms. This simplification however increases the number of files. For example, the extraction of 1000 randomly selected PDB files resulted in as many as 4007 single chain based files. These single chain files have been used to prepare three datasets with varying number of proteins as listed in **(Table 1. Fig. 1)** shows the effect of the increasing number of protein chains on the computational time taken by FAST (Zhu, 2005), one of the structure comparison methods we used in this study.

Table 1. Datasets used in the experiments. *SFBK* is an abbreviation consisting of author surname initials. Hash sign (#) represents the word *Number of'*

Dataset	# of Chains	# of comparisons	Size in MB
SFBK-250	250	62,500	33.1
SFBK-500	500	250,000	66.5
SFBK-1000	1000	1,000,000	137.5

2.2 Integrated Resource Management Environment for MPI jobs

In order to achieve the high-end computational power needed for compute-intensive applications such as Protein Structure Comparison, the operation of these applications is normally decomposed using some parallel programming libraries that are based on the implementations of Message Passing Interface (MPI) model. Commonly used MPI libraries include MPICH/MPICH2 (Gropp and Lusk, 1996) (Gropp *et al.*, 1996) and LAM/MPI (Burns *et al.*, 1994) (Squyres and Lumsdaine, 2003) which has recently been merged into a global project

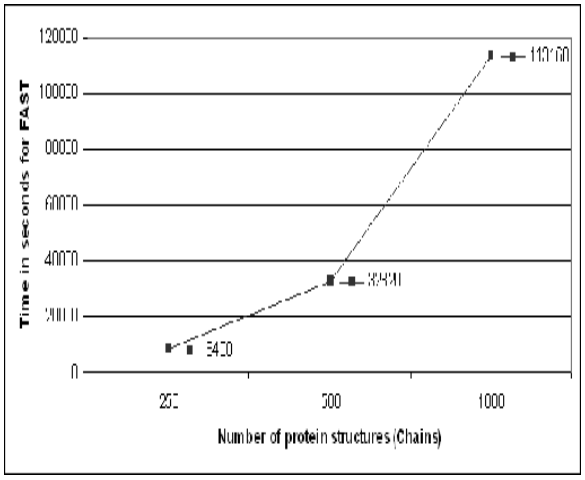


Fig. 1. Computational time for PSC algorithm (FAST) on P4-2.4GHz uni-processor machine with increasing number of proteins (single chain based PDB files) in the dataset.

under the name of OpenMPI (Gabriel *et al.*, 2004). Furthermore, the scope of parallelism itself is extending from supercomputers to

personal computers, workstations and networks (Foster, 1995). This extension has created a *converging* effect on the fields of *parallel* and *distributed* computing. It is in this perspective that the operation (submission, execution and monitoring) of MPI based jobs needs to be automated through some suitable resource management middleware **(see Fig. 2)** such as Sun Grid Engine (SGE), Portable Batch System (PBS), Load Sharing Facility (LSF) etc. The specialized literature refers to the functionality provided by these software using various names for instance *Local Resource Management System* (LRMS), *Queuing System*, *Batching System*, *Workload Manager*, *Job Management System* and *Distributed Resource Manager* etc. For detailed survey of these and related issues please refer to (Shah *et al.*, 2008, 2007).

In Fig.2, each newly submitted request for the execution of user jobs is taken care by the Queue Manager (e.g. *sge qmaster daemon of SGE*), which stores these jobs in a queue and requests the Scheduler (e.g. *sge schedd daemon* of SGE) for their execution. The Scheduler gets job information from the Queue Manager and optimally decides when and where to execute these jobs based on the information obtained from the Resource Manger (e.g. *sge execd/sge shepherd daemons* of SGE) and system administrator's policies. This type of automated system frees the user from manually looking if the required resources for the execution of a job are available and to manually start the job on each node both of the problems then it is said to be a *Tight Integration*, which enables the LRMS to run the MPI jobs under the control of its *Job Controlling Agent* so as to avoid the problem of resource overloading and getting full resource usage information.

Normally, when MPI based parallel jobs are submitted to LRMS, it calls some external framework to launch the MPI environment for the execution of jobs (Sistare *et al.*, 2002). If there is no appropriate coordination between the external framework and LRMS, then the launched jobs may get dispatched on somewhat different resources than what had been allocated by the LRMS, creating, on the one hand, resource overloading and conflicting problems, on the other hand, providing inaccurate or no resource usage and monitoring information. Both of these problems could be eliminated if certain means are used to integrate the operation of LRMS and MPI environment (Sistare *et al.*, 2002). If the integration is achieved in such a way that it removes only the first problem namely the launching of parallel jobs doesn't create overloading and resource conflicts, then the integration is said to be *Loose Integration*. In Loose Integration, the LRMS only sends information to MPI environment regarding which resources a particular job should run on but the job runs in isolation from LRMS and hence no resource usage and monitoring information is obtained by the LRMS. However, if the integration is achieved in such a way that it removes

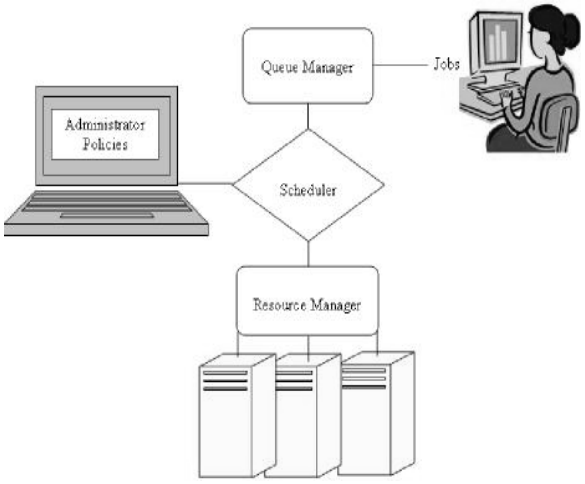


Fig. 2. Three main components of a Local Resource Management System (LRMS).

In order to evaluate the performance of PSC jobs under both (*i.e.* Loose and Tight) ways of integration, we have developed a testbed consisting of a *heterogeneous* Linux cluster (**resources shown in Table 2)** for the integration of SGE 6.1u3 and MPICH2-1.0.7rc2 with different approaches as described in (SGE, 2011). According to (SGE, 2011), though MPICH2 provides different process management methods such as *gFoker*, *MPD*, and *SMPD*, only the later could be used for tight integration as it satisfies the requirement of a *single process group* on each node of the cluster. The SMPD method by itself could be used in two different ways i.e. *SMPD daemon-based* (SMPD-DB) and *SMPD daemon-less* (SMPD-DL). With SMPD-DB a single daemon per node is used to manage all the processes running at that node. In case of multi-processor node the resource usage information of all other

processors is aggregated to the one and same daemon and hence the SGE would record the overall resource usage information per node through a single *Queue Remote Shell* (qrsh) invocation when the job

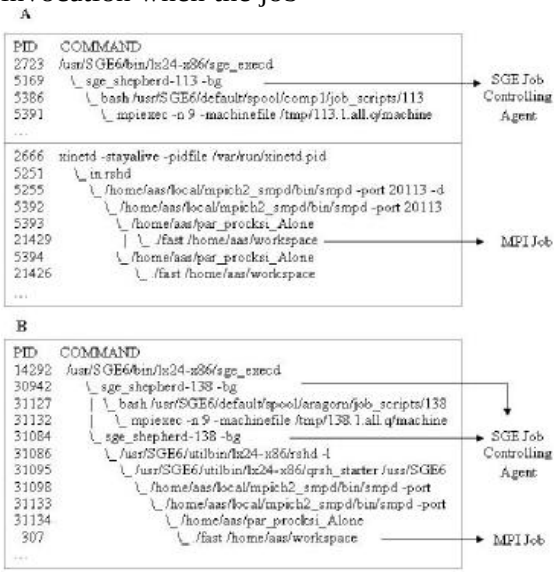


Fig.3. Parallel jobs running under integrated resource management environment for SGE and MPICH2.

3. RESULTS AND DISCUSSIONS

In order to evaluate the goodness and weakness of the integrated environment described in the previous section, we run the FAST algorithm using three datasets listed in Table 1. The experiments were performed on 6 heterogeneous nodes (9 processors) of a Linux Cluster with the characteristics described in **(Table 2. Fig. 3A)**, gives an illustration of a PSC job running FAST algorithm under loose integration, whereas Figure 3B illustrates the operation of the same MPI job running under tightly integrated environment.

In Fig.3A, Loose Integration; SGE launches the parallel job through its job controlling agent *sge shepherd* using *mpiexec* command by specifying the resources to be used (through *machine* file) and hence eliminating the problem of resource conflicting. However, the actual MPI job (FAST algorithm) doesn't run under the control of *sge shepherd* and hence no resource usage (accounting) information could be achieved. In Fig.3**B**: Tight Integration; SGE uses two instances of its job controlling agent *sge shepherd*, one for launching the parallel job by specifying the resources to be used and other for controlling (monitoring) the operation of the running MPI job (FAST algorithm) and hence providing the full resource usage (accounting) information. finishes its execution. Whereas, with SMPD-DL each process starts on its own on each node and SGE needs to record each *qrsh* invocation for each process separately. For the sake of comparison we used both SMPD-DB and SMPD-DL to perform Loose and Tight Integration using the distinct configuration parameters as listed in **(Table 3)**.

Table 2. Hardware Configuration of the Cluster Grid. All hosts communicate through 100 Mbps Ethernet cards

Host	CPU(GHz)	Cache (KB)	RAM(GB)	HD(GB)
comp1	2x P4-1.86	4096	2.0	250
comp2	2x P4- 3.0	512	1.0	80
comp3	P4-2.4	512	0.5	20
comp4	P4-1.7	256	1.0	40
comp5	2x P4-3.6	1024	2.0	80
comp6	P4-2.4	512	0.5	40

Table 3. Distinct configuration parameters for Loose Vs Tight integration of SGE and MPICH2.

Parameter	Loose Integration	Tight Integration
start proc arg	/sge/mpich2 smpd/startmpich2.sh $pe hostfile /local/mpich2 smpd	/sge/mpich2 smpd/startmpich2.sh $pe hostfile -catch rsh /local/mpich2 smpd
stop proc arg	/sge/mpich2 smpd/stopmpich2.sh /local/mpich2 smpd	/sge/mpich2 smpd/stopmpich2.sh /local/mpich2 smpd -catch rsh /local/mpich2 smpd
control slaves	FALSE	TRUE
job is first task	TRUE	FALSE

Table 4 shows the computational time in terms of *wall clock* time, *user* time, *system* time and CPU time required for the execution of one of the protein structure comparison algorithms named FAST (Zhu, 2005) with different datasets under the integrated resource management environment for MPI jobs that has been configured with different approaches. The brief description of each of these time components is given bellow:

– Wall Clock time: is the real time taken by the algorithm as perceived by the user and includes CPU time, I/O time and communication time e.g. between different nodes,
– User time: is part of the cpu time taken by the user application (i.e. FAST algorithm) itself,
– System time: is the part of cpu time taken by the system software; and **– CPU time:** is the sum of the *user* and *system* times.

It can be seen in Table 4, that in terms of wall clock time, the algorithm finishes earlier under Loose integration as compared to Tight integration. Possible reasons for this performance degradation might be that of *Inter Process Communication* (IPC), context switching, record keeping overhead and the large amount of output forwarded by the slave processes to the master process via stdout etc. However, with Loose integration the *user*, *system* and *cpu* times (accounting information) can not be noted as the mpi job doesn't run under the control of job controlling agent *sge-shepherd*. Furthermore, in Loose integration the SGE is unable to get proper monitoring (state) information in case if the mpi job goes through any problems. An example of such job that SGE was unable to get information about is one suffering with an mpi related error:
unable to read the cmd header on the pmi context, socket connection closed, error stack: MPIDU Socki handle read(607): connection closed by peer (set=3,sock=50331655).

This job took far beyond its expected execution time and SGE did not notify the user and the job was killed manually afterwards. Being able to keep track of which pairs of structures have been (un)successfully compared and by which method is an essential feature of modern robust comparison servers such as (Barthel *et al.,* 2007). Hence, notwithstanding its speed, this is a potential weakness of *Loose* integration.

Referring to **(Table 4)** again, it could also be observed that for a small number of proteins (e.g. *SFBK-250 dataset*) the performance of both SMPD-DB and SMPD-DL types of tight integrations is almost same, however, for larger datasets (e.g. *SFBK-500* and *SFBK-1000*), the former outperforms the later one. This is because in case of SMPD-DL, SGE needs to record each *qrsh* invocation and hence incurs an extra overhead. **(Tables 5, 6)** illustrate some interesting differences between *SMPD-DB* and *SMPD-DL* types of tight integrations. Table 5 shows that SMPD-DB integration doesn't expose the dual-processor nature of some of the nodes (e.g. *comp1*, *comp2*, and *comp4*; Table 2). However, it utilizes both of the processors of all dual-processor nodes implicitly as can be seen from the values of CPU times for comp2 and comp5 in Table 5, which are even greater than the value of their corresponding *wall clock* times. Table 6 shows that the overall performance of the SMPD-DL tight integration gets degraded because of the longer CPU time taken by the slowest node (*comp4*; Table 2). This longer CPU time could be because of the slowest speed of certain nodes in subject as well as because of the longer lengths of protein chains being compared by FAST algorithm at that node. It is because of this long running node that faster CPUs also go through long idling cycles.

4. CONCLUSIONS

Performance of the distributed framework for *Protein (Structure) Comparison, Knowledge, Similarity and Information* (ProCKSI) has been evaluated (using FAST algorithm as an example) under integrated resource management environment for MPI jobs. Results of the evaluation indicate that though *Loose Integration* method provides efficient results but it is not much reliable in terms of accounting and monitoring information. Furthermore, for larger datasets, Tight Integration with SMPD daemon-based method outperforms its counterpart i.e. SMPD daemon-less method. It has also been learnt that in a heterogeneous cluster where some nodes have double the performance as that of others, the slowest node could become a bottleneck for the overall performance of the system. This performance degradation could deteriorate further when that slowest node is assigned PSC jobs with longer protein chains.

Table 4. Resource usage comparison for FAST algorithm (Zhu and Z., 2005) with different datasets using Loose and Tight integration of SGE and MPICH2 with SMPD daemon-based (SMPD-DB) and daemon-less (SMPD-DL) startup methods.

Dataset	Time [hh:mm:ss]	LOOSE SMPD-DB	SMPD-DL	TIGHT SMPD:DB	SMPD:DL
SFBK-250	Wall clock	00:10:43	00:13:51	00:59:18	00:52:27
	User	-	-	00:50:18	00:45:46
	System	-	-	00:08:05	00:06:25
	CPU	-	-	00:58:23	00:52:11
SFBK-500	Wall clock	00:51:12	01:01:29	01:38:28	02:11:48
	User	-	-	01:22:11	01:50:25
	System	-	-	00:13:18	00:18:51
	CPU	-	-	01:35:39	02:09:16
SFBK-1000	Wall clock	02:22:25	02:22:36	05:39:36	07:44:04
	User	-	-	04:34:06	06:28:56
	System	-	-	00:49:15	01:08:34
	CPU	-	-	05:23:22	07:37:30

Table 5. Time distribution for FAST algorithm with *SFBK-1000* dataset over nine processor using tight integration with SMPD daemon-based (SMPD-DB) method.

	comp1	comp2	comp3	comp4	comp5	comp6
Wall Clock [hh:mm:ss]	05:39:37	05:39:37	05:39:37	05:39:37	05:39:37	05:39:37
User [hh:mm:ss]	03:43:17	05:05:48	02:56:23	03:40:22	07:21:00	04:37:37
System [hh:mm:ss]	00:33:37	01:02:10	00:32:29	00:41:13	01:09:55	00:56:36
CPU [hh:mm:ss]	04:16:28	06:08:07	03:28:52	04:21:35	08:30:55	05:34:13
Mem [MB]	31.06	66.08	60.87	43.53	95.83	104.53

Table 6. Time distribution for FAST algorithm with *SFBK-1000* dataset over nine processor using tight integration with SMPD daemon-less (SMPD-DL) method.

	comp1 cpu-1	comp1 cpu-2	comp2 cpu-1	comp2 cpu-2	comp3	comp4	comp5 cpu-1	comp5 cpu-2	comp6
Wall Clock [hh:mm:ss]	07:44:05	07:44:05	07:44:05	07:44:05	07:44:05	07:44:05	07:44:05	07:44:05	07:44:05
User [hh:mm:ss]	01:29:22	02:01:50	03:55:15	03:00:35	02:31:28	06:28:56	02:57:52	02:14:24	03:20:42
System [hh:mm:ss]	00:14:33	00:17:54	00:40:19	00:29:51	00:29:50	01:08:34	00:30:41	00:24:15	00:45:00
CPU [hh:mm:ss]	01:43:56	02:19:44	04:35:34	03:30:26	03:01:18	07:37:30	03:28:33	02:38:39	04:05:42
Mem [MB]	10.66	29.42	63.66	41.75	22.69	161.77	24.65	30.56	48.44

5. ACKNOWLEDGEMENTS

N. Krasnogor acknowledges EPSRC for project EP/DO61571B/1. A.A Shah acknowledges University of Sindh for scholarship FSCH/SU/PLAN/F.SCH/794.

REFERENCES:

Barthel, D., J. Hirst, J. Blazewicz, E.K. Burke, and N. Krasnogor (2007) 'The ProCKSI Server: a decision support system for Protein (Structure) Comparison, Knowledge, Similarity and Information', *BMC Bioinformatics* (**8**): 416Pp.

Berman, H., J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne, (2000), 'The protein data bank', *Nucleic Acids Res* (**28**)**:** 235–242.

Burns, G., R. Daoud, and J. Vaigl, (1994) LAM: An Open Cluster Environment for MPI, *in* 'Proceedings of Supercomputing Symposium', 379–386.

Folino, G., A. Shah, D. Barthel, and N. Krasnogor, (2008) Towards a Distributed

Framework for Protein (Structure) Comparison, Knowledge, Similarity and Information (ProCKSI), *in* 'Proceedings of the UK All Hands Meeting (AHM2008)'.

Foster, I. (1995) Parallel computers and computation, *in* 'Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering'.

Foster, I. (2005) Globus toolkit version 4: Software for service-oriented systems, *in* 'Proceedings, IFIP International Conference on Network and Parallel Computing', 2–13.

Gabriel, E., G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, (2004) Open MPI: Goals, concept, and design of a next generation MPI implementation, *in* 'Proceedings, 11th European PVM/MPI Users' Group Meeting', Budapest, Hungary, 97–104.

Gregoretti, F., G. Laccetti, A. Murli, G. Oliva, and U. Scafuri, (2008) 'Mgf: A grid-enabled mpi library', *Future Generation Computer Systems* 24 (**2**): 158– 165.

Gropp, W. D. and E. Lusk, (1996) *User's Guide for* mpich, *a Portable Implementation of MPI*, Mathematics and Computer Science Division, Argonne National Laboratory. ANL-96/6.

Gropp, W., E. Lusk, N. Doss, and A. Skjellum, (1996) 'A high-performance, portable implementation of the MPI message passing interface standard', *Parallel Computing* 22 (**6**): 789–828.

Holm, L. and J. Park, (2000) 'Dalilite workbench for protein structure comparison', *Bioinformatics* (**16**)**:** 566–567.

Krasnogor, N. and D. A. Pelta, (2004) 'Measuring the similarity of protein structures by means of the universal similarity metric', *Bioinformatics* (**20**)**:** 1015– 1021.

Pelta, D. A., J. R. Gonzalez, and M., M.V. (2008) 'A simple and fast heuristic for protein structure comparison', *BMC Bioinformatics* (**9**): 161Pp.

SGE (2011) 'Tight Integration of the MPICH2 library into SGE, http://gridengine.sunsource.net/howto/mpich2-integration/mpich2-integration.html; Retrieved: May 5, 2011 '.

Shah, A., D. Barthel, and N. Krasnogor, (2007) Grid and distributed public computing schemes for structural proteomics: a short overiew, *in* P. Thulasiraman, X. He, T. Li-Xu, M. Denko, R. Thulasiram and L. Yang, eds, 'Lecture Notes in Computer Science', Springer, 424–434.

Shah, A., D. Barthel, P. Lukasiak, J. Blazewicz, and N. Krasnogor, (2008) 'Web & grid technologies in bioinformatics, computational and systems biology: A review', *Current Bioinformatics* 3 (**1**): 10–31(22).

Shindyalov, I. and P. Bourne, (1998) 'Protein structure alignment by incremental combinatorial extension (ce) of the optimal path', *Protein Eng* (**11**)**:** 739–747.

Sistare, S., J. Test, and D. Plauger, (2001) Sun grid engine: towards creating a compute power grid, *in* 'Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid', IEEE Computer Society, 35–36.

Sistare, S., J. Test, and D. Plauger, (2002) An architecture for integrated resource management of mpi jobs, *in* 'Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER02)', IEEE Computer Society, 370–377.

Squyres, J. M. and A. Lumsdaine, (2003) A Component Architecture for LAM/MPI, *in* 'Proceedings, 10th European PVM/MPI Users' Group Meeting', number 2840 *in* 'Lecture Notes in Computer Science', Springer-Verlag, Venice, Italy, 379–387.

Zhang, Y. and J. Skolnick, (2005) 'Tm-align: A protein structure alignment algorithm based on tm-score', *Nucleic Acids Res* (**33**): 2302–2309.

Zhu, J. and Z., W. (2005) 'Fast: A novel protein structure alignment algorithm', *Proteins Struct Funct Bioin*