A Tool for Creating and Parallelizing Bioinformatics Pipelines

Chenggang Yu and Paul A. Wilson
*US Army Medical Research and Materiel Command (MRMC), Biotechnology HPC Software
Applications Institute, Telemedicine and Advanced Technology Research Center, Ft. Detrick, MD*
cyu@bioanalysis.org

Abstract

Bioinformatics pipelines enable life scientists to effectively analyze biological data through automated multi-step processes constructed by individual programs and databases. The huge amount of data and time consuming computations require effectively parallelized pipelines to provide results within a reasonable time. To reduce researchers' programming burden for pipeline creation and parallelization, we developed the Bioinformatics Pipeline Generation and Parallelization Toolkit (BioGent). A user needs only to create a pipeline definition file that describes the data processing sequence and input/output files. A program termed schedpipe in the BioGent toolkit takes the definition file and executes the designed procedure. Schedpipe automatically parallelizes the pipeline execution by performing independent data processing steps on multiple CPUs, and by decomposing big datasets into small chunks and processing them in parallel. Schedpipe controls program execution on multiple CPUs through a simple application programming interface (API) of the Parallel Job Manager (PJM) library. As a part of the BioGent toolkit, PJM was developed to effectively launch and monitor programs on multiple CPUs using a Message Passing Interface (MPI) protocol. The PJM API can also be used to parallelize other serial programs. A demonstration using PJM for parallelization shows 10% to 50% savings in time compared to an indigenous parallelization through a batch queuing system.

1. Introduction

Bioinformatics pipelines (BIPs) enable life scientists to effectively analyze biological data through automated multi-step processes constructed by individual programs and databases. For example, InterProScan (Quevillon, et al., 2005) was designed to use multiple applications to search 12 independently-developed proteomics databases that are incorporated into InterPro (Mulder, et al., 2005). PUMA2 (Maltsev, et al., 2006) incorporates more than 20 public databases for genome analysis and annotation. The huge amount of data and time consuming computations require effective parallelization for a pipeline to provide results within a reasonable time. Therefore, considerable programming effort is needed for both integration of individual programs into a pipeline and parallelization of the pipeline. This has led to the development of software tools to simplify pipeline generation. Examples of such tools include Biopipe (Hoon, et al., 2003), Pegasys (Shah, et al., 2004), BOD (Qiao, et al., 2004), EGene (Durham, et al., 2005), Pipeline Pilot (Hassan, et al., 2006), and Ergatis (ergatis.sf.net). One computational aspect of these tools is the decomposition of the data processing workflow into individual jobs, each consisting of one program (e.g., BLAST) and the necessary input data (e.g., FASTA file). This decomposition provides a general, multiple-program multiple-data model for parallelization.

The importance of parallelization increases when considering genome-wide research, where the large amount of data employed necessitates high throughput capabilities. Moreover, parallelization becomes attractive as the number of programs and databases integrated into a single BIP increases. Many pipeline generation tools simply submit decomposed individual jobs to a batch queuing system for parallel execution. The monitoring of the job status is performed by calling particular commands provided by the queuing system, for example, command 'bjobs', provided by Load Sharing Facility or command 'qstat', provided by the Sun Grid Engine. Although this method is easy to implement, it impairs pipeline portability by tying it to a particular queuing system. Moreover, the method is not effective in handling dependencies among individual jobs. Some queuing systems have provided dependency options in their job submission commands. However, the options are system dependent and are not easily handled in pipeline generation tools. When numerous jobs are submitted by a pipeline program to a queuing system, efficiency could be another issue. A large number of dependant jobs may significantly slow down the job-dispatching process and affect the pipeline as well as other users' work.

An alternative to using batch queuing systems for parallelization is to write parallel code to directly use multiple CPUs to run individual jobs. This will reduce the batch queuing system's burden. Since a user's program will have direct control of multiple CPUs, launching and monitoring jobs will become faster and easier. This method is adopted by our BioGent, which reduces the programming burden for both integration and parallelization of multiple bioinformatics programs. A pipeline can be generated and automatically parallelized through a user-provided pipeline description file. The parallelization is based on a MPI protocol that hands out jobs from a main pipeline program to multiple remote CPUs, and monitors the progress of these jobs.

2. Methods

The BioGent package has two main components: a pipeline control program called *schedpipe* and a binary library called *Parallel Job Manager* (*PJM*). They provide two tiers of solutions for quick parallelization of BIP programs.

The first tier doesn't need any programming effort. Users simply write a text file to describe a pipeline's data processing flow as multiple independent or dependant steps. Each step consists of a program and its input and output. The input can be a chunk of data in a large data file, or the output of previous processing steps. Similarly, the output can be the final result of the pipeline, or the intermediate result that becomes the input to the next processing step. The way to split a large data file into chunks is set in the pipeline definition file.

When s*chedpipe* is executed, it takes control of multiple computer nodes. The program reads in the pipeline definition file, creates multiple jobs for data processing steps and sends each job to a different CPU for execution. When one job is done, *schedpipe* sends the next job until all jobs are completed. It determines the order of job execution by dependency. A job is sent out for execution only when all jobs that it depends on are finished.

The second tier of quick parallelization is through calling the *PJM* library in user programs. An API for parallel job control is provided by the library that uses MPI for communication among processes on different computer nodes. Actually, *schedpipe* also uses *PJM* for the control of multiple computer nodes. Figure 1 depicts the application of *PJM* in *schedpipe*. When *PJM's* function *parallel_int* is called, a multi-nodes manager (MNM) thread is created on the same node (equivalent to a master node) running the user's programs and single node manager (SNM) threads are created on each remote node (i.e., slave nodes). The MNM communicates with a user program through shared memory and with SNM using MPI.

However, this is transparent to the user's programs. A user's program can call the *getIdleNodes* function to get all available slave nodes and use the *setSimpleJob* function to assign a job for execution on a specific node. The MNM thread relays the information to the SNM thread on the designated node, which then spawns a new job thread (JT) to execute the job. The SNM monitors the job's execution and constantly reports to the MNM. The user program can call the *getAllDone* function to find out which jobs are completed. The job thread terminates when the job finishes. The SNM thread continuously spawns threads for new jobs, reports job state, and tracks CPU status. It terminates when requested to do so by MNM after the user's program is completed.
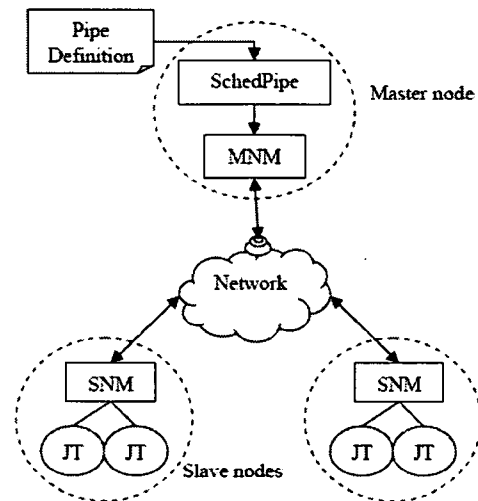

Figure 1. *Schedpipe's* control of multiple computer nodes via *PJM*

3. Results And Discussion

BioGent's efficiency to manage parallel computation of multiple computer nodes was tested on the Army Research Laboratory's (ARL's) Powell cluster, which has 128 nodes with dual CPUs running Red Hat Linux and using the Sun Grid Engine batch queuing system. BioGent was first used to create a simple pipeline that had only one data processing step and each job generated for that step required the same CPU time to complete. Figure 2 shows the speedup of parallel execution of 50,000 one-second jobs and 50,000 ten-second jobs. The speedup represents the efficiency of BioGent's job management of multiple computer nodes without concern for pragmatic issues, such as competition for shared resources like the network file system. The figure indicates that BioGent produces a nearly ideal speedup curve for ten-second jobs running on 220 CPUs, while the curve for one-second jobs drops when the number exceeds 200 CPUs. This drop in speedup is

caused by the CPUs quickly finishing a short job and waiting for their next job. The fraction of waiting time increases when more CPUs are used to execute very short jobs. In practice, jobs for a bioinformatics pipeline require longer times to run, while splitting input data into larger chunks also increases the execution time. Therefore, BioGent is efficient as a quick parallelization tool for bioinformatics pipelines. In fact, our test on Powell showed that BioGent needs only 7 milliseconds to start a new job on a remote node, which means that it can manage as many as 1,580 CPUs to run ten-second jobs with 90% efficiency (measured as the time to run the pipeline in sequential mode divided by the product of the time to run the pipeline in parallel and the number of CPUs used).
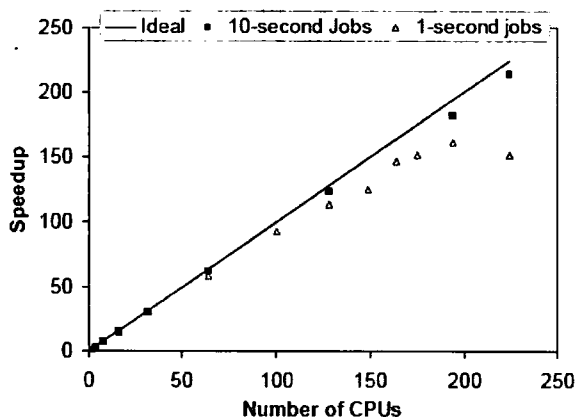

Figure 2. BioGent's parallelization

We compared the execution of a pipeline parallelized with BioGent and a batch queuing system. The software *InterProScan* contains 12 programs to search 12 different databases for an input protein sequence. The software splits input sequences into chunks of sequences and submits jobs to the batch queuing system for each program to process each chunk of data. All results are assembled in one output file at the end. We wrote a wrapper program *iprscan_PJM* to perform the same work but using *PJM* to send jobs directly to available CPUs and monitor their execution. A dataset of 200 proteins was used in the comparison. The output from both programs was exactly the same except that *iprscan_PJM* runs faster than *InterProScan*, as showen in Figure 3. The figure also shows the performance improvement obtained by using BioGent. The improvement is measured as the percentage of time saved by *iprscan PJM* as compare to *InterProScan*. The figure indicates that the parallelization of *InterProScan* using BioGent (*iprscan PJM*) saves 10% to 54% in wall-clock time and the time saving increases with the number of CPUs used.
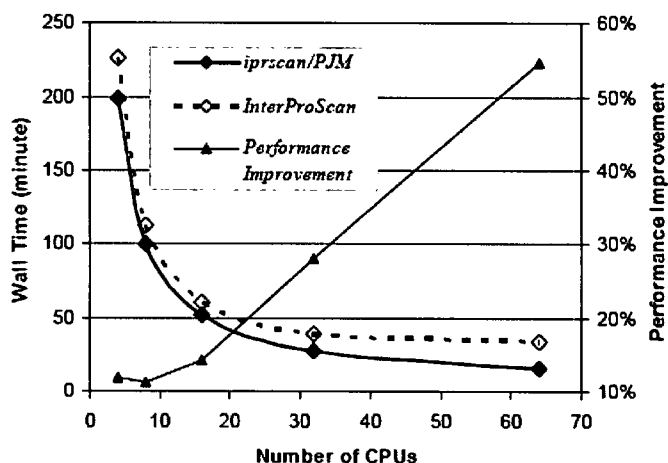

Figure 3. Comparison of parallelization based on BioGent and a batch queuing system

Another application of BioGent was examined by using *schedpipe* to create and parallelize a pipeline for protein structure domain predictions using the prediction of Protein Domain Boundaries Using Neural Networks (PPRODO) program (http://gene.kias.re.kr/~jlee/pprodo/). The pipeline predicts protein structure domains in four steps: 1) call the Protein Specific Iterated BLAST (PSIBLAST) program to search the non-redundant (nr) database, 2) call the Protein Structure Prediction Server (PSIPRED) program to predict secondary structure, 3) call the PSIBLAST program again to search the nr database but with different parameters, and 4) perform the PPRODO prediction of the domain boundary. Figure 4 shows the wall-clock time for the parallelized program when predicting domains for 5,138 proteins using different numbers of CPUs on ARL's JVN cluster. The parallelization effectively reduces the computation time from nearly 20 days on a single CPU to a few hours on ~100 CPUs. The speedup curve is showen in the inner panel of Figure 4. The inferior speedup compared to the results in Figure 2 is due mainly to the disk input/output on the shared network file system. If needed, copying data files to the local disk on each node improves performance.
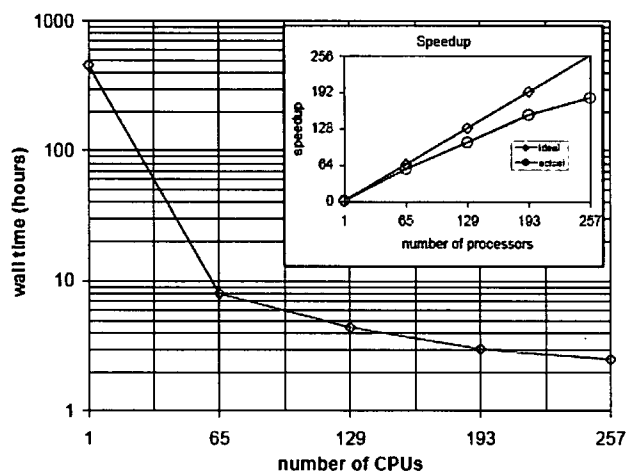
Figure 4. Performance of BioGent for a parallelized PPRODO pipeline

4. Conclusions

BioGent is a compact, portable package that currently contains the *schedpipe* program and the *PJM* library. BioGent is independent of third-party programs, database management systems, and specific batch queuing systems. These attributes allow for ease of installation and use.

Schedpipe leverages the *PJM* to create parallelized BIPs without requiring any extensive computer programming expertise from the user. The *PJM* can also be employed in user programs to execute jobs on multiple CPUs. This methodology is more effective than using a batch queuing system when parallelizing bioinformatics pipelines.

Disclaimer

The opinions or assertions contained herein are the private views of the authors and are not to be construed as official or as reflecting the views of the US Army or the US Department of Defense.

Acknowledgements

References

Durham, A.M. and A.Y. Kashivabara, et al., "EGene: a configurable pipeline generation system for automated sequence analysis." *Bioinformatics*, 21(12), pp. 2812–2813, 2005.

Hassan, M. and R.D. Brown, et al., "Cheminformatics analysis and learning in a data pipeline environment." *Molecular Diversity*, 10(3), pp. 283–299, 2006.

Hoon, S. and K.J. Ratnapu, et al., "Biopipe: a flexible framework for protocol-based bioinformatics analysis." *Genome Res.*, 13(8), pp. 1904–1915, 2003.

Maltsev, N. and E. Glass, et al., "PUMA2--grid-based high-throughput analysis of genomes and metabolic pathways." *Nucleic Acids Res.*, 34 (Database Issue), D369-37, 2006.

Mulder, N.J. and R. Apweiler, et al., "InterPro, progress and status in 2005." *Nucleic Acids Res.*, 33 (Database Issue), pp. D201–205, 2005.

Qiao, L. and J. Zhu, et al., "BOD: a customizable bioinformatics on demand system accommodating multiple steps and parallel tasks." *Nucleic Acids Res.*, 32(14), pp. 4175–4181, 2004.

Quevillon, E. and V. Silventoinen, et al., " InterProScan: protein domains identifier." *Nucleic Acids Res.*, 33 (Web Server issue), pp. W116–120, 2005.

Shah, S.P. and D. He, et al., " Pegasys: software for executing and integrating analyses of biological sequences." *BMC Bioinformatics*, 5(40), 2004.