# A tool for creating and parallelizing bioinformatics pipelines

Chenggang Yu, Paul A. Wilson

Biotechnology HPC Software Applications Institute,

Telemedicine and Advanced Technology Research Center, US Army Medical Research and Materiel Command, Ft. Detrick, MD 21702, USA

cyu@bioanalysis.org

## Introduction

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. Considerable programming effort is needed for both integrating individual programs into a pipeline and parallelizing them.

## Objective

The object of our Bioinformatics Pipeline Generation and Parallelization Toolkit (BioGent) is to reduce researchers' programming burden. A user only needs to create a pipeline definition file that describes the data processing sequence and input/output files. Program termed *schedpipe* in BioGent toolkit takes the definition file and executes the designed procedure.

## Method

*schedpipe* automatically parallelizes the pipeline execution through executing independent data processing steps on multiple CPUs, and through 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* is developed to effectively launch and monitor programs on multiple CPUs by using Message Passing Interface (MPI). The API provided by *PJM* is used to handle multiple CPUs without complicated message exchange among CPUs.

## Results

BioGent is demonstrated to be effective for quickly parallelizing bioinformatics pipelines. It's application for parallelizing a protein function analyzing pipeline (*InterProScan*), containing 12 programs, shows 10% to 50% savings in time compared to the indigenous parallelization through batch queuing system. BioGent has been used to create a protein structural domain prediction pipeline that integrates three programs. A test shows that the pipeline can process over 5,000 proteins within eight hours by using 64 processors on the JVN computer cluster at the Army Research Laboratory (ARL) Major Shared Resource Center (MSRC).

## Significance to DoD

BioGent helps DoD life scientists, who lack programming experience, to use DoD's high performance computing resources.

| Report Documentation Page | | Form Approved OMB No. 0704-0188 |
|---|---|---|

| 1. REPORT DATE **2007** | 2. REPORT TYPE | 3. DATES COVERED **00-00-2007 to 00-00-2007** |
|---|---|---|

| 4. TITLE AND SUBTITLE **A tool for creating and parallelizing bioinformatics pipelines** | 5a. CONTRACT NUMBER |
|---|---|
| | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |

| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **US Army Medical Research and Material Command,Biotechnology HPC Software Applications Institute,Telemedicine and Advanced Technology Research Center,Fort Detrick,MD,21702** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
|---|---|

| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
|---|---|
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES
**Proceedings of Department of Defense High Performance Computing Modernization Program Users Group Conference 2007, June 18-22, 2007, Pittsburgh, PA**

14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT **unclassified** | b. ABSTRACT **unclassified** | c. THIS PAGE **unclassified** | **Same as Report (SAR)** | **6** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

# 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 result 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 also through calling particular commands provided by the queuing system, for example, command 'bjobs' provided by Load Sharing Facility (LSF) or command 'qstat' provided by the Sun Grid Engine (SGE). Although this method is easy to implement, it impairs pipelines 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 slowdown the job-dispatching process and affect the pipelines 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 have direct control of multiple CPUs, launching and monitoring jobs will become faster and easier. This method is adopted by our Bioinformatics Pipeline Generation and Parallelization Toolkit (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 MPI to hand out jobs from a main pipeline program to multiple remote CPUs, and to monitor 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.
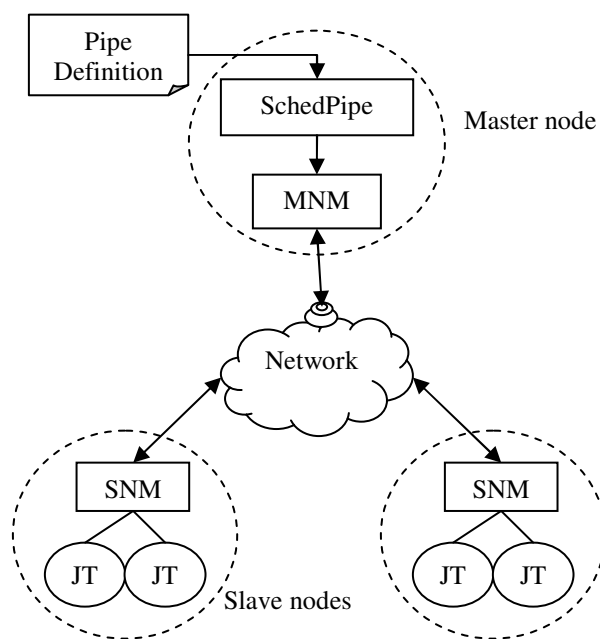


Figure 1. *schedpipe*'s control of multiple computer nodes via *PJM*.

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 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 report 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 continues to exist to spawn threads for new jobs, report job state, and track CPU status. It terminates when requested to do so

by MNM after the user's program is completed.

## 3 RESULTS AND DISCUSSION

BioGent's efficiency to manage parallel computation of multiple computer nodes was tested on ARL's Powell cluster, which have 128 nodes with dual CPUs running Red Hat Linux and using the SGE batch queuing system. BioGent was first used to create a simple pipeline that has only one data processing step and each job generated for that step requires 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 issue 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 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



Figure 2. BioGent's parallelization speedup.

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 1580 CPUs to run ten-second jobs with 90% efficiency (which is 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).
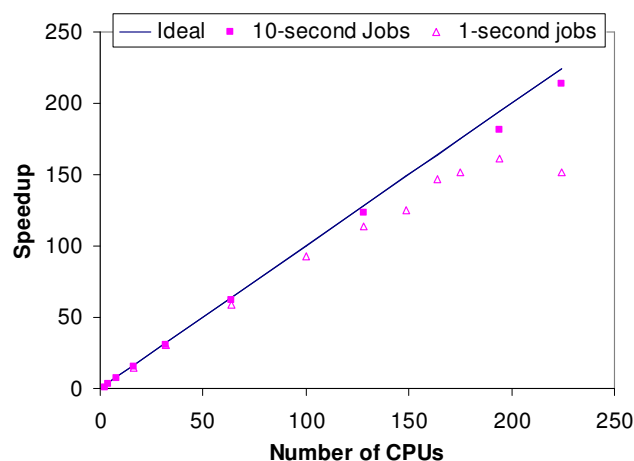
We compared an actual pipeline's parallelization using BioGent and a batch queuing system. The software *InterProScan* contains 12 programs to search 12 different databases for an input protein sequence. The software splits input sequences into chunks of sequences and submits jobs to the batch queuing 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 showed in Figure 3. The figure also shows the performance improvement obtained from 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.

Another application of BioGent was examined by using *schedpipe* to create and parallelize a pipeline for protein structure domain predictions using the PPRODO program (http://gene.kias.re.kr/~jlee/pprodo/). The pipeline predicts protein structure domains in four steps: 1) call the PSIBLAST program to search the non-redundant (nr) database, 2) call the 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 number 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 showed in the inner panel of Figure 4. The inferior speedup compared to the result in Figure 2 is mainly due to the disk I/O on the shared network file system. If needed, copying data files to the local disk on each node will improve the performance.
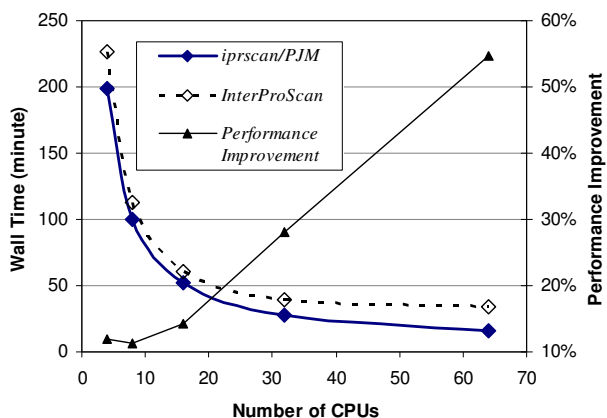


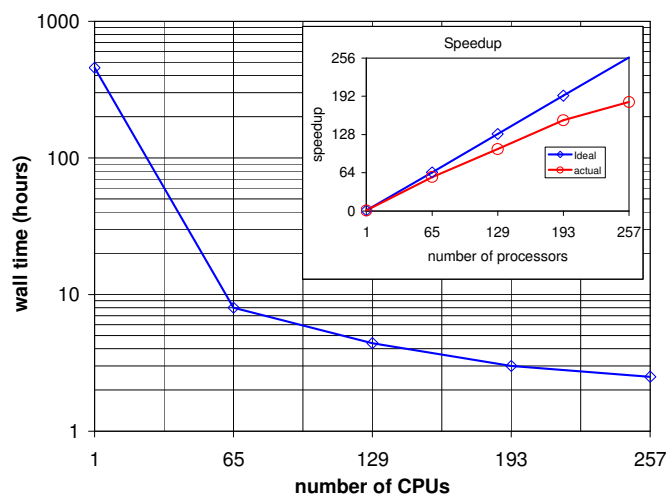Figure 3. Comparison of parallelization based on BioGent and a batch queuing system.

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* leverage the *PJM* to create parallelized BIPs without requiring any computer programming 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

## ACKNOWLEDGEMENT

## REFERENCES

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

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

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

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

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

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

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

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