BLASTScanner: Fast BLAST data processing with database output

Detlef Groth (1), Joachim Selbig (1), Albert J. Poustka (2), Georgia Panopoulou(2)

(1): Potsdam University, Bioinformatics Group, c/o Max Planck Institute of Molecular Plant Physiology, Am Mühlenberg 1, 14476 Potsdam, Germany
(2): Max Planck Institute for Molecular Genetics, Ihnestr. 63/73, D-14195 Berlin, Germany

Abstract

Motivation: One of the most widely used bioinformatic tools to compare protein or nucleotide sequences is BLAST. Although a large number of applications and frameworks exists for parsing and analyzing BLAST output files, none of them completely fulfills the requirements of an easy and platform independent installation, fast processing speed, and storage of the parsing results in a manner suitable for downstream processing.
Results: We developed BLASTScanner, a small cross-platform console application which translates BLAST text files into standard database code suitable for loading into modern relational databases like SQLite, PostgreSQL and MySQL. BLASTScanner can be easily compiled, requires no installation, and was faster and easier to use than other commonly used BLAST parsers.
Availability: The source code and binaries for various platforms are freely available at the sourceforge project page http://bioscanners.sf.net
Contact: dgroth@uni-potsdam.de

Introduction

Although biologists and bioinformaticians are permanently challenged with parsing and analysing output of various biological tools, often utilising different formats, a standard method for solving such tasks has not been found, yet. Standalone applications connected by UNIX pipes are used quite often but the integration of their outputs is in many cases a tedious and error prone task of manual inspection. The competing Bio-framwork approach also has disadvantages: They are difficult to install and to maintain, and as they do both both: data parsing and data integration, they require sophisticated programming skills and a lot of knowledge concerning the framework specific programming interface. Furthermore, their data parsing procedures are often very slow.

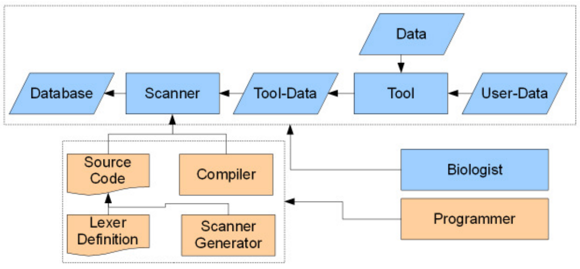

Figure 1: Data-Analysis pipeline using a scanner generator.

To overcome those problems we generated the Bioscanners project at sourceforge. The usage of scanner generators ensures simple programming, a small code base, thereof easy maintenance, and very high processing speed. The steps for a programmer to create a working scanner are outlined in the lower part of Figure 1. It has been shown that Flex based BLAST-file scanners were more than 200 times faster than BioPerl ones (Paquola et al. 2003). However, the usage of this scanner require coding knowledge in C or Perl and the output format is not standardized.

As, besides easy and fast data scanning, the storage of the scanning result in a format that can be parsed to selectively extract information at any time is important for the user. For that reason the output of our scanners is standard database code, suitable to be used for SQL (Structured Query Language) compliant databases like PostgreSQL, MySQL or SQLite (Figure 2).

```
BLASTScanner-Linux-x86 --infile sample.blastn --prefix sam1 \
                  | sqlite3 sample.sql
```

Figure 2: Invocation of BLASTScanner with a redirect to a database.

Our BLASTScanner, created with the scanner generator re2c, was a proof of principle implementation of a Bio-data scanner. As BLAST is one of the most often used tools in Bioinformatics, this application might be of interest for a broader audience of researchers as well.

Table 1. Scanners, Compilers and WC binary sizes

scanner	language	reference	compiler	wc-sizes
Flex	C	Paxson (1995)	GNU gcc 4.1.2	16 / 489
Flexpp	C++	Paxson (1995)	GNU g++ 4.1.2	21 / 1002
RE2C	C	Bumbulis and Cowan (1993)	GNU gcc 4.1.2	7 / 7
TPLex	Pascal	Graef (1998)	Free Pascalfpc 2.0.4	109 / 109
JFlex	Java	Klein (2008)	SUN javac 1.6.0	7 / NA
GPLEX	C#	Gough (2008)	Mono gmcs 2.0.1	13 / 6525

Methods

The used scanner generators and compilers are shown in Table 1. Platforms used for testing were 32-bit computers with Intel(R) processors having the operating systems Windows-XP Professional with SP 2 and Linux Fedora core 8, 64-bit computers with Intel(R) processors with Mac-OSX Darwin kernel 8.11, CentOS 5.1. Additionally a Sparc computer with Solaris 9 and an Alpha computer with OSF-1 V4.0 were used. All time tests were done on the 32-bit Linux (2 GB RAM, 2 Intel(R) Xeon(TM) CPUs, 2.8 GHz) machine using Tcl-scripts for measuring time and memory parameters.

Results & Discussion

We evaluated the usability of various scanner generators to produce small cross platform executables. We used six scanners generators that accept Flex-like input files (Table 1) and where the generated source code can be compiled on the platforms Windows, Linux, and Mac-OSX. Our scanner generators support five programming languages. We used only non-scripting languages in our analysis, since it has recently been shown that scripting languages are much slower than semi- or fully compiled languages (Fourment 2008).

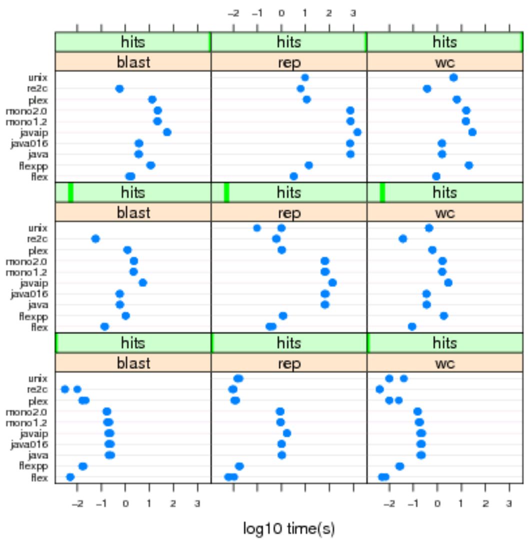

Figure 3: Time usage for different scanners and tasks. wc=word counter, rep=simple character replacer, blast=simple blast scabber. Input were BLAST files with 1 (left), 100 (center) and 1000 (right) hits. The latter file was around 40Mb large.

Our scanners were quite small in size. With the exception of the Pascal executables (around 100kb in size) all executables were smaller than 30kb. The RE2C and the Pascal executables did not depend on installed libraries or a programming language runtime on the target platform, simplifying the installation of the application. The RE2C executables were around two times faster than the Flex based, and around ten times faster than the Java ones in scanning large data files (Figure 3). Flexpp and GPLEX scanners were even slower than Java scanners. As the RE2C applications also required only low amounts of memory, we favoured RE2C as the scanner generator used for our BLASTScanner implementation.

Conclusions

Our BLASTScanner, available from the sourceforge project page, is a single C source file which can be compiled on any modern computer platform to a small 20Kb executable not depending on any external library or runtime. To our knowledge BLASTScanner is the only available console application which directly translates standard BLAST-files into database code. The processing time required for scanning the BLAST files is even shorter than the time required by the database application to import the data into the database. Thereof there is no need for further improvements in scanning speed. The use of an optional prefix argument ensures the possibility to save results of different BLAST runs in the same database. The output of our scanners was tested successfully with freely available database engines like MySQL, PostgreSQL and SQLite.

References

- Paquola et al. (2003). Bioinformatics 19:1035
- Fourment and Gillings (2008). BMC Bioinformatics 9:82