Bioinformatics, 33(9), 2017, 1421–1423
doi: 10.1093/bioinformatics/btw838
Advance Access Publication Date: 16 January 2017
Applications Note

Systems biology

DistributedFBA.jl: high-level, high-performance flux balance analysis in Julia

Laurent Heirendt, Ines Thiele and Ronan M. T. Fleming*

University of Luxembourg, Luxembourg Centre for Systems Biomedicine, Esch-sur-Alzette, Luxembourg

*To whom correspondence should be addressed.

Associate Editor: Jonathan Wren

Received on November 10, 2016; revised on December 16, 2016; editorial decision on December 23, 2016; accepted on January 10, 2017

Abstract

Motivation: Flux balance analysis and its variants are widely used methods for predicting steady-state reaction rates in biochemical reaction networks. The exploration of high dimensional networks with such methods is currently hampered by software performance limitations.
Results: *DistributedFBA.jl* is a high-level, high-performance, open-source implementation of flux balance analysis in Julia. It is tailored to solve multiple flux balance analyses on a subset or all the reactions of large and huge-scale networks, on any number of threads or nodes.
Availability and Implementation: The code is freely available on github.com/opencobra/COBRA.jl. The documentation can be found at opencobra.github.io/COBRA.jl.
Contact: ronan.mt.fleming@gmail.com
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Constraint-based reconstruction and analysis (COBRA) (Palsson *et al.*, 2015) is a widely used approach for modeling genome-scale biochemical networks and integrative analysis of omics data in a network context. All COBRA predictions are derived from optimization problems, typically formulated in the form

$$\begin{array}{ll} \min\limits_{v \in R^n} & \psi(v) \\ \text{s.t.} & Sv = b \\ & Cv \le d \\ & l \le v \le u, \end{array} \tag{1}$$

where $v \in R^n$ represents the rate of each biochemical reaction, $\psi : R^n \to R$ is a lower semi-continuous and convex function, $S \in \mathrm{R}^{m\times n}$ is a stoichiometric matrix for m molecular species and n reactions, and b is a vector of known metabolic exchanges. Additional linear inequalities (expressed as a system of equations with matrix C and vector d) may be used to constrain combinations of reaction rates and keep reactions between upper and lower bounds, u and l, respectively.

In flux balance analysis (FBA), one obtains a steady-state by choosing a coefficient vector $c \in R^n$ and letting $\psi(v) := c^T v$ and $b := 0$. However, the biologically correct coefficient vector is usually not known, so exploration of the set of steady states relies on the embarrassingly parallel problem of solving (1) for many c. Moreover, while $c^T v^*$ is unique for an optimal flux vector v^*, there may be alternate optimal solutions. In flux variability analysis (FVA), one finds the extremes for each optimal reaction rate by choosing a coefficient vector $d \in R^n$ with one non-zero entry, then minimizing and maximizing $\psi(v) := d^T v$, subject to the additional constraint $d^T v \ge \gamma \cdot c^T v^*$ for each reaction in turn ($\gamma \in]0, 1[$).

For kilo-scale models ($n \simeq 1000$), the $2n$ linear optimization problems required for FVA can currently be solved efficiently using existing methods, e.g. *FVA* of the COBRA Toolbox, *fastFVA*, or the *COBRApy* implementation (Schellenberger *et al.*, 2011; Gudmundsson *et al.*, 2010; Ebrahim *et al.*, 2013). However, these implementations perform best when using only one computing node with a few cores, which becomes a temporal limiting factor when exploring the steady state solution space of larger models. Julia is a high-level, high-performance dynamic programming language for technical computing (Bezanson *et al.*, 2014). Here, we exploit Julia to distribute sets of FBA problems and compare its performance to existing implementations.

2 Overview and implementation

DistributedFBA.jl, part of a novel *COBRA.jl* package, is implemented in Julia and makes use of the high-level interface *MathProgBase.jl*

Table 1. Sizes of *S* for benchmark models

#	Model name	Metabolites *m*	Reactions *n*	References
1	Recon1	2785	3820	Duarte *et al.* (2007)
2	Recon2	5063	7440	Thiele *et al.* (2013)
3	Recon3[a]	7866	12 566	
4	Recon2 + 11M	19 714	28 199	Heinken *et al.* (2015)
5	Multi-organ[b]	47 123	61 230	
6	SRS064645	89 756	99 104	Magnusdottir *et al.* (2016)
7	SRS011061	126 682	139 420	Magnusdottir *et al.* (2016)
8	SRS012273	186 662	208 714	Magnusdottir *et al.* (2016)

[a]Brunk, E. *et al.* (2016) Recon 3d: a three-dimensional view of human metabolism and disease (in revision).

[b]Thiele, I. *et al.* (2016) Multi-organ model (prototype model) (*in preparation*).

(Lubin *et al.*, 2015; see Supplementary Material). A key feature is the integrated capability of spawning synchronously any number of processes to local and remote workers. Parallelization is primarily achieved through distribution of FBA problems (outer layer), while parallelization of the solution algorithm is solver based (inner layer). *COBRA.jl* extends the COBRA Toolbox (Schellenberger *et al.*, 2011) while existing COBRA models (Orth *et al.*, 2010) can be input.

3 Benchmark results

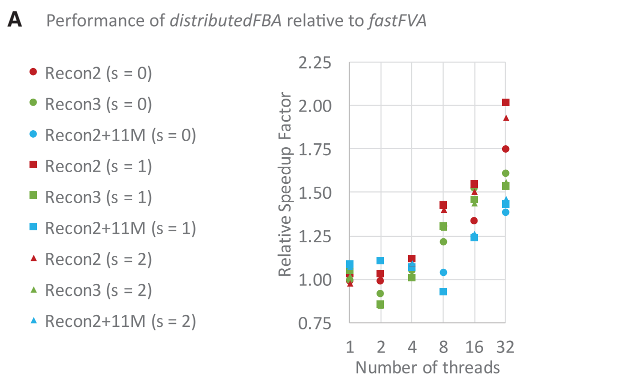
DistributedFBA.jl and *fastFVA* (Gudmundsson *et al.*, 2010) were benchmarked on a set of models of varying dimension (Table 1). All experiments were run on several DELL R630 computing nodes with 2 × 36 threads and 768GB RAM running Linux. As Julia is a just-in-time language, pre-compilation (warm-up) was done on a small-scale model before benchmarking (Orth *et al.*, 2010). The creation of a parallel pool of workers and the time to spawn the processes are not considered in the reported times.

The serial performance of both implementations is within 10%. The uninodal performance of *fastFVA* is slightly higher on a few threads, but the performance of *distributedFBA.jl* is superior for a higher number of threads on a single node (Fig. 1A). The way the FBA problems are distributed among workers (distribution strategy *s*, see Supplementary Material) yields an additional speedup of 10–20% on a larger number of threads.

According to Amdahl's law, the theoretical speedup factor is $\left(1-p+\frac{p}{N}\right)^{-1}$, where *N* is the number of threads and *p* is the fraction of the code (including the model) that can be parallelized. The fraction *p* increases with an increasing model size (Fig. 1B). The maximum speedup factor for a very large number of threads *N* is $(1-p)^{-1}$. All reactions of models 6–8 given in Table 1 have been optimized (with full output, $s=0$) on 4 nodes/256 threads in only 4094 *s*, 11 458 *s*, and 32 900 *s*, respectively. This demonstrates that for high-dimensional models, it is critical to have a large number of threads on multiple high-memory nodes to accrue a significant speedup.

4 Discussion

The multi-nodal performance of *distributedFBA.jl* is unparalleled: the scalability of *distributedFBA.jl* matches theoretical predictions, and resources are optimally used. Key advantages are that the present implementation is open-source, platform independent, and that no pool size limits, memory, or node/thread limitations exist. Its uninodal performance is similar to the performance of *fastFVA* on a few threads and about 2–3 times higher on a larger number of threads. A key reason is the direct parallelization capabilities of Julia and the wrapper-free interface to the solver. The unilingual and easy-to-use implementation relies on solvers written in other languages, allows the analysis of large and huge-scale biochemical networks in a timely manner, and lifts the analysis possibilities in the COBRA community to another level.



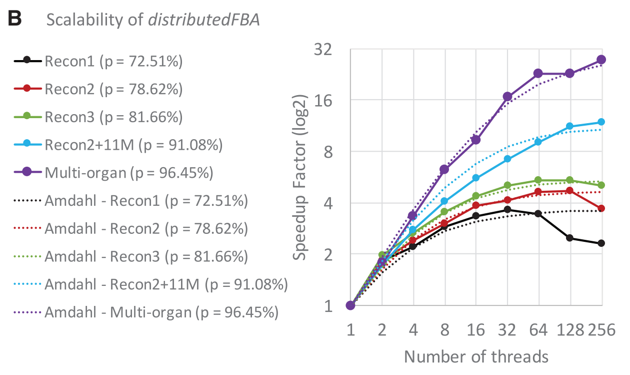


Fig. 1. Performance of *distributedFBA* for selected benchmark models given in Table 1. (**A**) Speedup factor relative to *fastFVA* as a function of threads and distribution strategy *s* (1 node). (**B**) Multi-nodal speedup in latency and Amdahl's law ($s=0$)

Funding

This study was funded by the National Centre of Excellence in Research (NCER) on Parkinson's disease, the U.S. Department of Energy, Offices of Advanced Scientific Computing Research and the Biological and Environmental Research as part of the Scientific Discovery Through Advanced Computing program, grant no. DE-SC0010429, and the Luxembourg National Research Fund (FNR) ATTRACT program grant (FNR/A12/01).

Conflict of Interest: none declared.

References

Bezanson,J. *et al.* (2014) Julia: a fresh approach to numerical computing, arXiv:1411.1607 [cs.MS].

Duarte,N.C. *et al.* (2007) Global reconstruction of the human metabolic network based on genomic and bibliomic data. *PNAS*, **104**, 1777–1782. doi: 10.1073/pnas.0610772104.

Ebrahim,A. *et al.* (2013) COBRApy: COnstraints-Based Reconstruction and Analysis for Python. *BMC Syst. Biol.*, **7**, 74.

Gudmundsson,S. *et al.* (2010) Computationally efficient flux variability analysis. *BMC Bioinformatics*, **11**, 489.

Heinken,A. *et al.* (2015) Systematic prediction of health-relevant human-microbial co-metabolism through a computational framework. *Gut Microbes*, **6**, 120–130.

Lubin,M. *et al.* (2015) Computing in operations research using Julia. *INFORMS J. Comput.*, **27**, 238–248. doi:10.1287/ijoc.2014.0623.

Magnusdottir,S. *et al.* (2016) Generation of genome-scale metabolic reconstructions for 773 members of the human gut microbiota. *Nat. Biotechnol.*, advanced access, doi: 10.1038/nbt.3703.

Orth,J.D. *et al.* (2010) Reconstruction and use of microbial metabolic networks: the core Escherichia coli metabolic model as an educational guide. *EcoSal plus*, **4**, doi:10.1128/ecosalplus.10.2.1.

Palsson,B. *et al.* (2015) *Systems Biology: Constraint-Based Reconstruction and Analysis*. 1st edn. Cambridge University Press, Cambridge, UK.

Schellenberger,J. *et al.* (2011) Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0. *Nat. Protocols*, **6**, 1290–1307.

Thiele,I. *et al.* (2013) A community-driven global reconstruction of human metabolism. *Nat. Biotechnol.*, **31**, 419–425. doi:10.1038/nbt.2488.