Metrum Research Group LLC
Phone: 860.735.7043
billg@metrumrg.com, yiz@metrumrg.com

2 Tunxis Road, Suite 112
Tariffville, CT 06081
www.metrumrg.com

Torsten report

Speed up population Bayesian inference by combining cross-chain warmup and within-chain parallelism

Based on:
(Torsten Version xxx, Stan version 2.23)

September 28, 2020

1. Introduction

With increasing adoption of Bayesian inference to pharmacometric(PMX) modeling, it has become evident that high-performance computing(HPC) must be utilized for large-scale models to be accessible, and inference framework based on Markov Chain Monte Carlo(MCMC) must improve efficiency through multiple channels. For example, probabilistic programming language Stan [2] uses efficient samplers such as the No-U-Turn Sampler(NUTS) [3], and provide `map_rect` functions to parallelize expensive likelihood evaluation.

The scope of the work presented here is to improve Bayesian inference efficiency of population models. The work is based on Torsten [5], a library of Stan functions that simplifies PMX modeling and extends the range of models that may be implemented. We address two aspects of the efficiency problem. First, we propose a dynamic warmup approach, as an alternative to current Stan's warmup where a fixed number(default 1000) of iterations are performed. Second, we combine the new warmup algorithm with existing within-chain parallelization functionality of Torsten [9] to formulate a *multilevel* parallel method that utilizes dynamic warmup *and* within-chain parallelism to speed up simulation.

Discussions related to the proposal can be found at Stan forum [1, 7]. Source code, model, and data used in this study can be found at acompanying repo [8].

2. Cross-chain warmup

2.1. **Algorithm & implementation.** The standard practice of Stan is to perform a fixed number of warmup iterations. With this practice, the efficacy of the warmup is unknown *a priori* and often warmup is unncessarily long as user oversubscribe warmup iterations. The proposed warmup algorithm tries to avoid this by checking potential scale reduction coefficients ($\hat{R}$) and effective sample sizes (ESS) [6] . Specifically, for warmup we propose(see also Figure 1).

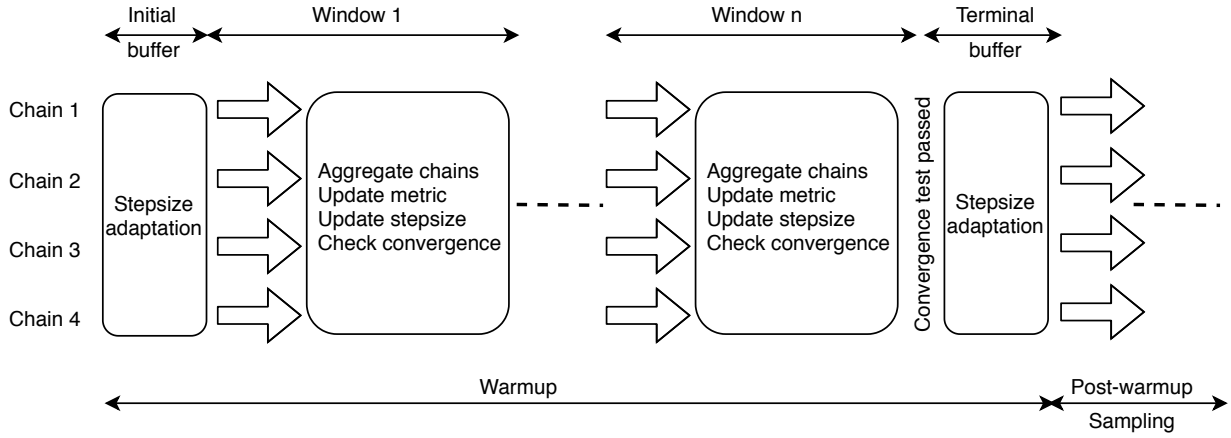

Figure 1. Cross-chain warmup algorithm

(1) Given a fixed window size w(default 100) and initial buffer size(default 75), the sampler iterates during warmup with stepsize adapted as in regular warmup runs.
(2) At the end of a window, we aggregate the joint posterior probability `lp__` from all the chains and calculate corresponding $\hat{R}$ as well as ESS. Specifically, when the warmup reaches the last iteration of window n, we calculate $\hat{R}^i$ and ESS^i, $i = 1 \ldots, n$. The superscript i indicates that the quantity is calculated based on `lp__` aggregated from all the chains, using the iterations from window i, $i+1$, $\ldots$, n. For example, with default window size

$w = 100$, when warmup reaches iteration 300, we calculate $\hat{R}^i$ and ESS^i for $i = 1, 2, 3$, so that

$\hat{R}^1$ and ESS^1 are based on warmup iteration 1 to 300;
$\hat{R}^2$ and ESS^2 are based on warmup iteration 101 to 300;
$\hat{R}^3$ and ESS^3 are based on warmup iteration 201 to 300.

(3) At the end of window n, with predefined target value $\hat{R}^0$ and ESS^0, from $1, \ldots, n$, we select j such that

$$\text{ESS}^j > \text{ESS}^i, \quad \forall i \neq j, \quad 1 \leq i \leq n. \tag{1}$$

Namely, we select j so that it has the maximum ESS. A new metric is calculated by aggregating samples from

$$\text{window } j, \text{window } j+1, ..., \text{window } n$$

from all the chains, and a new stepsize is calculated by taking geometric mean of chain stepsizes. The new metric and stepsize are used in future iterations for all the chains. If, in addition, j satisfies

$$\begin{aligned} \hat{R}^j &< \hat{R}^0, \\ \text{ESS}^j &> \text{ESS}^0, \end{aligned} \tag{2}$$

the warmup is considered complete((*converges*). Otherwise warmup continues until the end of the next window and step 2-3 are repeated.

(4) After convergences, the warmup continues into terminal buffer(50 iterations by default). As in standard Stan warmup, in this buffer the metric is no longer updated while stepsize is further adapted.

Unlike current warmup scheme, the above proposal requires communication among the chains, hence we call it *cross-chain warmup*. The implementation is based on Torsten's parallel setup using Message Passing Interface (MPI). In a cross-chain warmup run, all chains move forward independently except at the end of a window, where samples are aggregated from the chains to calculate $\hat{R}^i$ and ESS^i and new metric and stepsize are distributed to the chains. After warmup the sampler moves into independent post-warmup sampling stage with no more cross-chain communications.

The latest version of cross-chain warmup implementation can be found at

2.2. **Performance evaluation.** We compare the two warmup schemes by running several models from posteriordb and Torsten repo. For each model, we compare the effect of warmup on

- total number of leapfrog integration steps in warmup
- total number of leapfrog integration steps in sampling
- number of leapfrog integration steps in per each warmup iteration
- number of leapfrog integration steps in per each sampling iteration
- minimum ESS_{bulk} per iteration
- minimum ESS_{tail} per iteration
- minimum ESS_{bulk} per leapfrog step
- minimum ESS_{tail} per leapfrog step
- maximum wall time

We run each model runs with 10 random seeds

```
seed <- seq(8235121, 8235130)
```

and plot the above quantities' the average(barplot) and standard deviation(error bar).

For a `stanfit` object, we use the following R function evaluate cross-chain performance.

```
## Based on Aki's script to evaluate ESS
perf.cc <- function(stanfit) {
    (n_chain = stanfit@sim$chains)
    (n_warmup = stanfit@sim$warmup)
    n_iter = stanfit@sim$iter-n_warmup
    sampler_params <- rstan:::get_sampler_params(stanfit, inc_warmup = TRUE)
    leapfrogs = sapply(sampler_params, function(x) x[, "n_leapfrog__"])
    (sum_warmup_leapfrogs = sum(leapfrogs[1:n_warmup,]))
    (sum_leapfrogs = sum(leapfrogs[n_warmup+(1:n_iter),]))
    (mean_warmup_leapfrogs = sum_warmup_leapfrogs/n_warmup / n_chain)
    (mean_leapfrogs = sum_leapfrogs/n_iter / n_chain)
    mon = rstan::monitor(as.array(stanfit), warmup=0, print=FALSE)
    (maxrhat = max(mon[,'Rhat']))
    bulk_ess_per_iter = mon[,'Bulk_ESS']/n_iter / n_chain
    tail_ess_per_iter = mon[,'Tail_ESS']/n_iter / n_chain
    bulk_ess_per_leapfrog = mon[,'Bulk_ESS']/sum_leapfrogs
    tail_ess_per_leapfrog = mon[,'Tail_ESS']/sum_leapfrogs
    min(bulk_ess_per_iter)
    min(tail_ess_per_iter)
    min(bulk_ess_per_leapfrog)
    min(tail_ess_per_leapfrog)
    elapsed <- as.data.frame(rstan::get_elapsed_time(stanfit))
    (stepsizes = sapply(sampler_params, function(x) x[, "stepsize__"])[n_iter,])

    res <- data.frame(run = c(sum_warmup_leapfrogs / n_chain, sum_leapfrogs /
     ↪  n_chain,
                                  mean_warmup_leapfrogs, mean_leapfrogs,
                                  min(bulk_ess_per_iter),
                              min(tail_ess_per_iter),
                              min(bulk_ess_per_leapfrog),
                              min(tail_ess_per_leapfrog),
                              max(elapsed$warmup + elapsed$sample)))
    row.names(res) <- c("leapfrogs(warmup)", "leapfrogs(sampling)",
                        "leapfrogs(warmup)/iter", "leapfrogs(sampling)/iter",
                        "min(bulk_ESS/iter)", "min(tail_ESS/iter)",
                        "min(bulk_ESS/leapfrog)", "min(tail_ESS/leapfrog)",
                        "max(elapsed_time)")
    return(res)
}
```

For profiling cross-chain performance of a particular model, we compare fit results from different target ESS as well as regular runs(4 chains with 1000 warmup iterations in each chain). Figures in this section are generated by

```
multiple.run.ess("cmdstan/examples", "model-name", 4, 4, "hostfile", seq(8235121,
 ↪  8235130), c(100,200,400))
```

See `script/run_cc.R` for details of functions. Equivalently, one can run the model using accompanying `cmdstan`. To activate cross-chain feature, compile model with `cmdstan/make/local` set as

```
STANC2=true
MPI_ADAPTED_WARMUP=1
TBB_CXX_TYPE=clang
```

and run it with

```
# we use MPICH options in this report.
mpiexec -n 4 -l -f hostfile ./model-name sample save_warmup=1 adapt
 ↪  cross_chain_ess=target_ess data file=model-name.data.R init=init.R random
 ↪  seed=seed id=i
```

For regular run the model should be compiled with `cmdstan/make/local` set as

```
STANC2=true
```

All wall time in this report are measured in seconds.

3. Multilevel parallelism: combining cross-chain warmup and within-chain parallelization

Combining cross-chain warmup and within-chain parallelization, we are able to design a framework of *multilevel parallelism* for Bayesian inference of population models. Orthogonal to the above warmup algorithm, *within-chain* parallelization implemented in Stan and Torsten does not induce communication across chains but distributes a heavy-lifting modeling task to mutiple processes in a single chain. In Torsten this within-chain strategy focuses on solving ODEs in population model. The corresponding functions are [9]

```
pmx_solve_group_rk45
pmx_solve_group_adams
pmx_solve_group_bdf
```

3.1. **Algorithm & implementation.** Our multilevel framework has an upper and lower level of parallelization(Figure 9). The upper level handles the cross-chain warmup by running parallel chains and updating metric and stepsize. Chains exchange information only at the end of each window at this level.

The lower level of within-chain parallelization occurs more frequently: with every new set of parameter samples, NUTS updates the likelihhod by solving the ODEs in the population model, and Torsten's group solvers distribute the population to multiple processes, with each processe handling one or several subjects' ODE systems.

3.2. **Example.** To demonstrate the above multilevel method, we apply it to a time-to-event model for the time to the first grade 2+ peripheral neuropathy (PN) event in patients treated with an antibody-drug conjugate (ADC) delivering monomethyl auristatin E (MMAE). We call it Time-To-PN(TTPN) model, and analyze data using a simplified version of the model reported in [4]. We consider three treatment arms: fauxlatuzumab vedotin 1.2, 1.8 and 2.4 mg/kg IV boluses q3w x 6 doses, with 20 patients per treatment arm. In this model, each patient's PK is described by an effective compartment model(one-compartment), and PD by a linear model. The likelihood for time to first 2+ PN event is described by a hazard function that depends on the concentration effect through Weibull distribution. Two unknowns from PK model and the cumulative hazard form a three-component ODE system. Each evaluation of likelihood requires solving this 3-system for every patient.

In Torsten's model, ODEs corresponding to the entire population can be solved by a single call of `pmx_solve_group_rk45` function. The three parameters of the model are:

- k_{e0} in effective compartment model.
- α the coefficient of linear PD model.

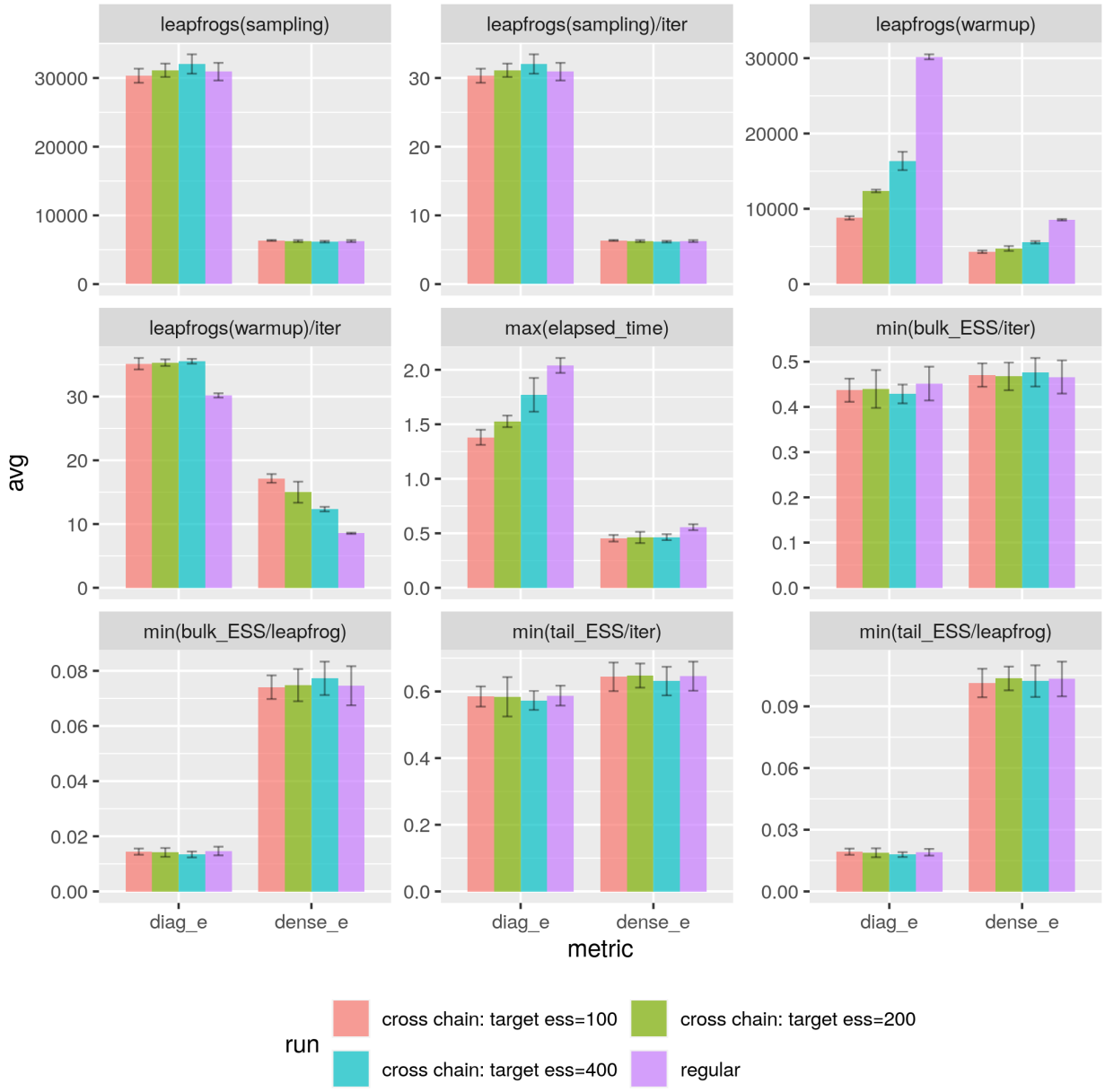

FIGURE 2. Cross-chain warmup performance comparison: arK model

- β Weibull distribution scale parameter.

To activate multilevel feature, compile model with `cmdstan/make/local` set as

```
STANC2=true
TORSTEN_MPI=1
MPI_ADAPTED_WARMUP=1
TBB_CXX_TYPE=clang
```

and run it with

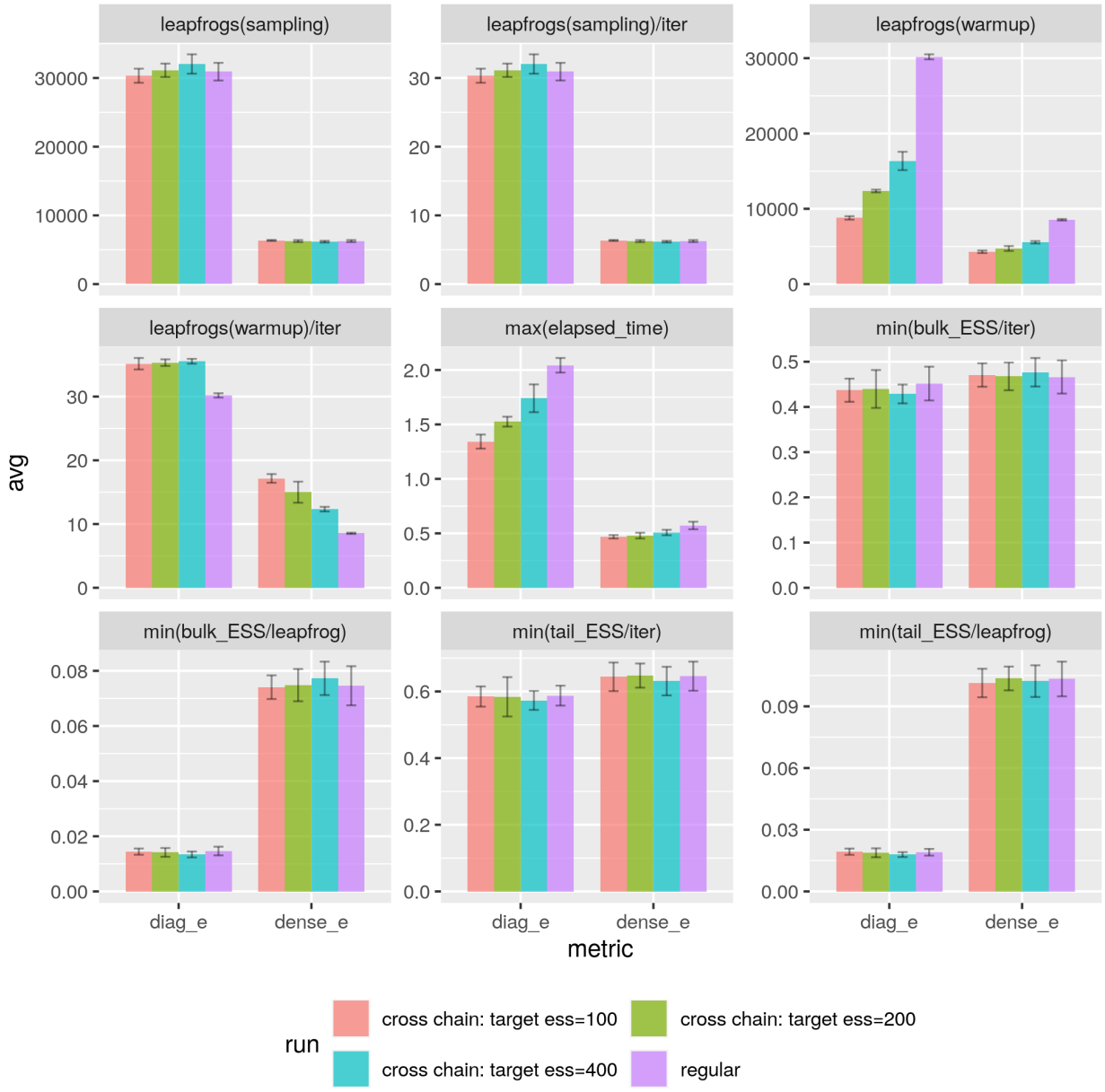

FIGURE 3. Cross-chain warmup performance comparison: arK-arK model

```
# we use MPICH options in this report.
mpiexec -n nproc -l -f hostfile ./model-name sample save_warmup=1 adapt
 ↪  cross_chain_ess=target_ess data file=model-name.data.R init=init.R random
 ↪  seed=seed id=i
```

Similar to previous section, Figure shows performance of cross-chain and regular runs based on target ESS = 400. Unlike in previous models, we did not performe runs with multiple seed or target ESS to avoid long computing time. One can make conclusion consistent with the other models, that the cross-chain warmup reduce total run time without compromising ESS.

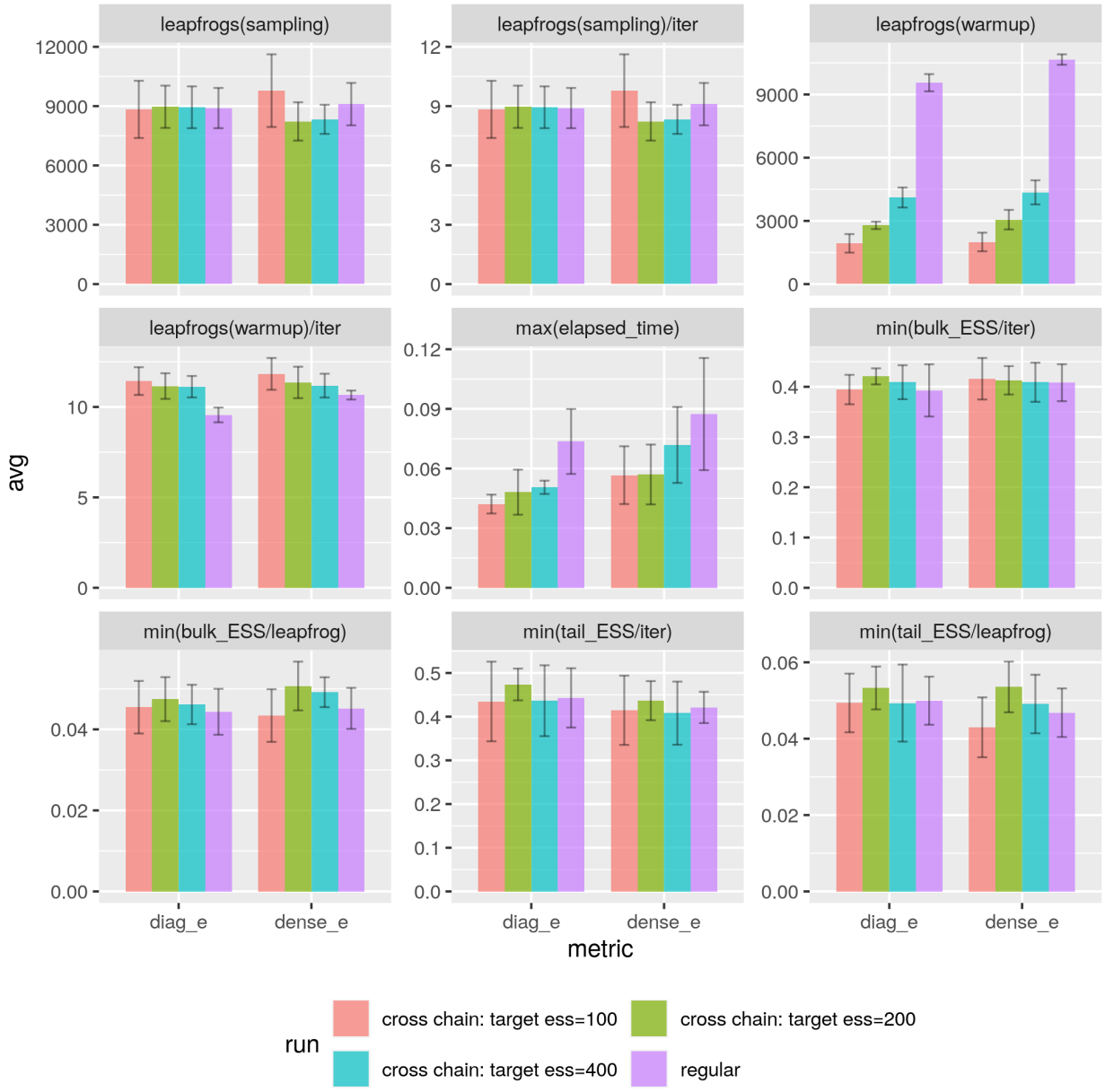

FIGURE 4. Cross-chain warmup performance comparison: eight schools model

Next, we apply multilevel method to TTPN model with a fixed target ESS = 400, by running the model with 4 chains using $n_{\text{proc}} = 8, 16, 32, 60, 80$ processes. Equivalently, there are $n_{\text{proc\_per\_chain}} = 2, 4, 8, 15, 20$ processes per chain so that within-chain parallelization can be utilized. With population size 60, each process handles solution of $n_{\text{id}} = 30, 15, 7, 4, 3$ subjects' ODE system, respectively.

To show parallel scaling performance, we collect `stanfit` objects of the benchmark runs and plot their wall time speedup against regular Stan runs. With all runs having 1000 post-warmup sampling iterations, in multilevel runs the number of warmup iterations is determinted at runtime, while both within-chain parallel runs and regular Stan runs have 1000 warmup iterations. Among 4 chains in a run, we use the one with maximum total walltime(in seconds) as performance measure, as in practice usually further model evaluation becomes accessible only after all chains finish.

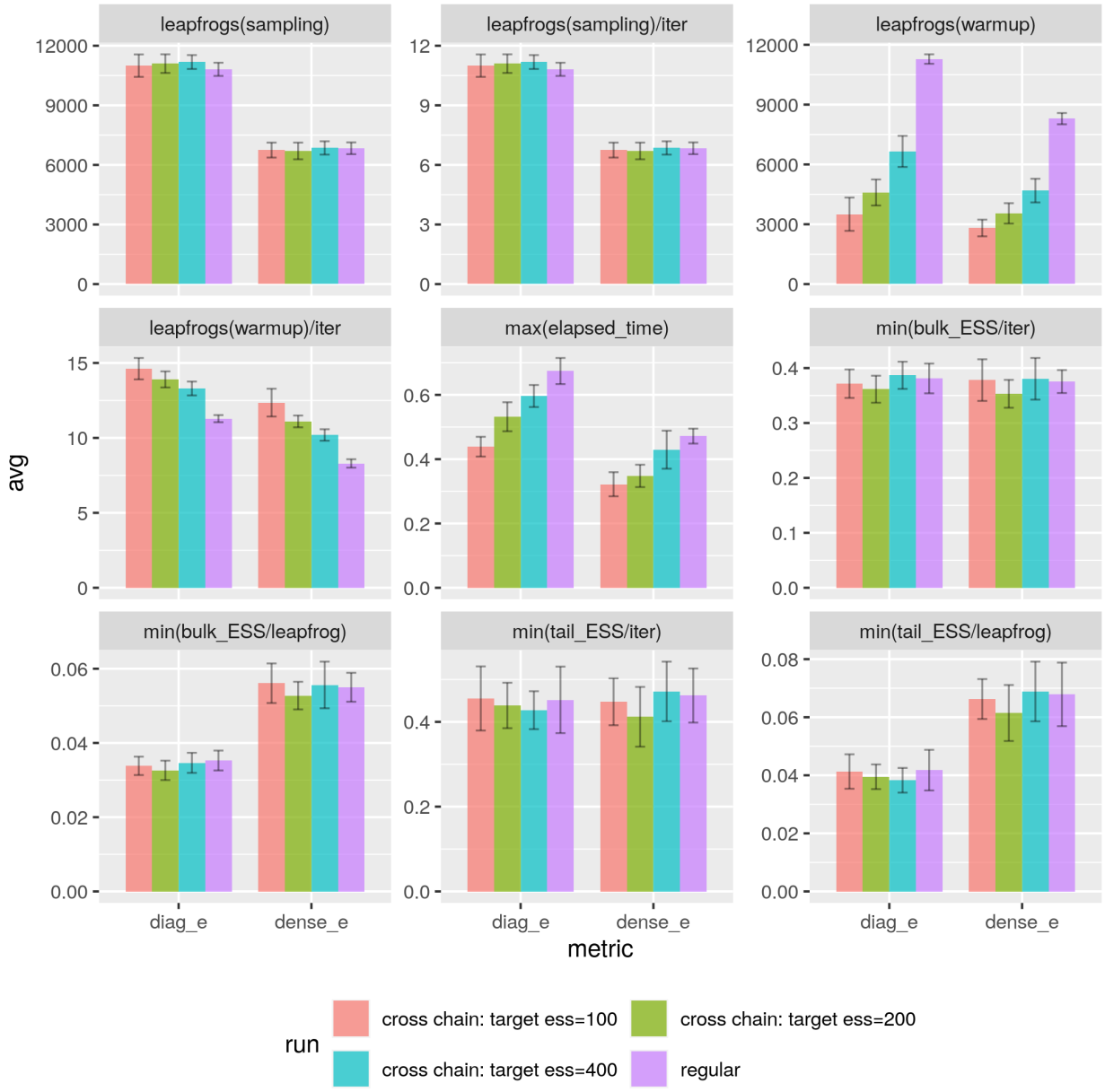

FIGURE 5. Cross-chain warmup performance comparison: garch-garch11 model

As shown in Figure 11, both muiltilevel and within-chain-only parallel runs exhibit good scaling up to 60 processes(15 processes per chain × 4 chains)[1]. In addition, cross-chain warmup enables multilevel runs to be more efficient, with a steady ~20% performance gain when 4+ processes per chain are applied. Since two parallel setup produce similar post-warmup sampling efficiency, this gain is entirely contributed by our new warmup algorithm.

4. Conclusion and future work

We note two benefits of the new multilevel algorithm:

[1]We did not apply more than 60 processes in benchmark in order to reduce cluster computing cost in this study. Previous study on within-chain parallelization [9] shows that reward of using more cores diminishes.

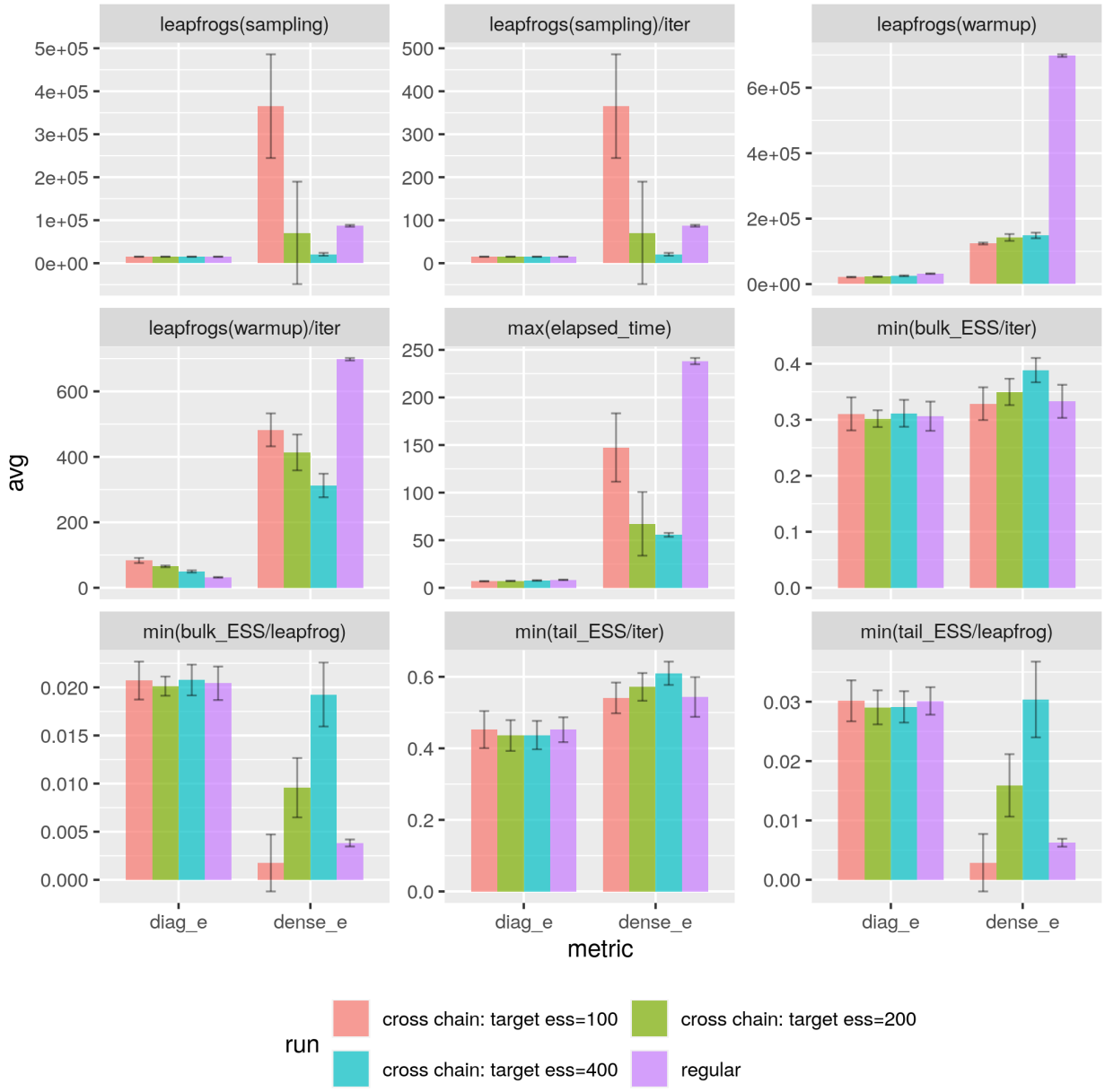

FIGURE 6. Cross-chain warmup performance comparison: radon model

(1) It significantly improves computational efficiency and extends the range of models that may be practically implemented.
(2) Cross-chain warmup quantifies warmup efficacy by providing informative runtime summary.

A natural followup study would be to increase the number of parallel chains in warmup, in the hope that we can maintain warmup quality while further reduce the number of iterations in each chain. Note that doing this in the multilevel setting induces burderns on computing resources: adding n more chains when each chain's within-chain parallel solver uses m processes requires adding $n \times m$ processes in computing.

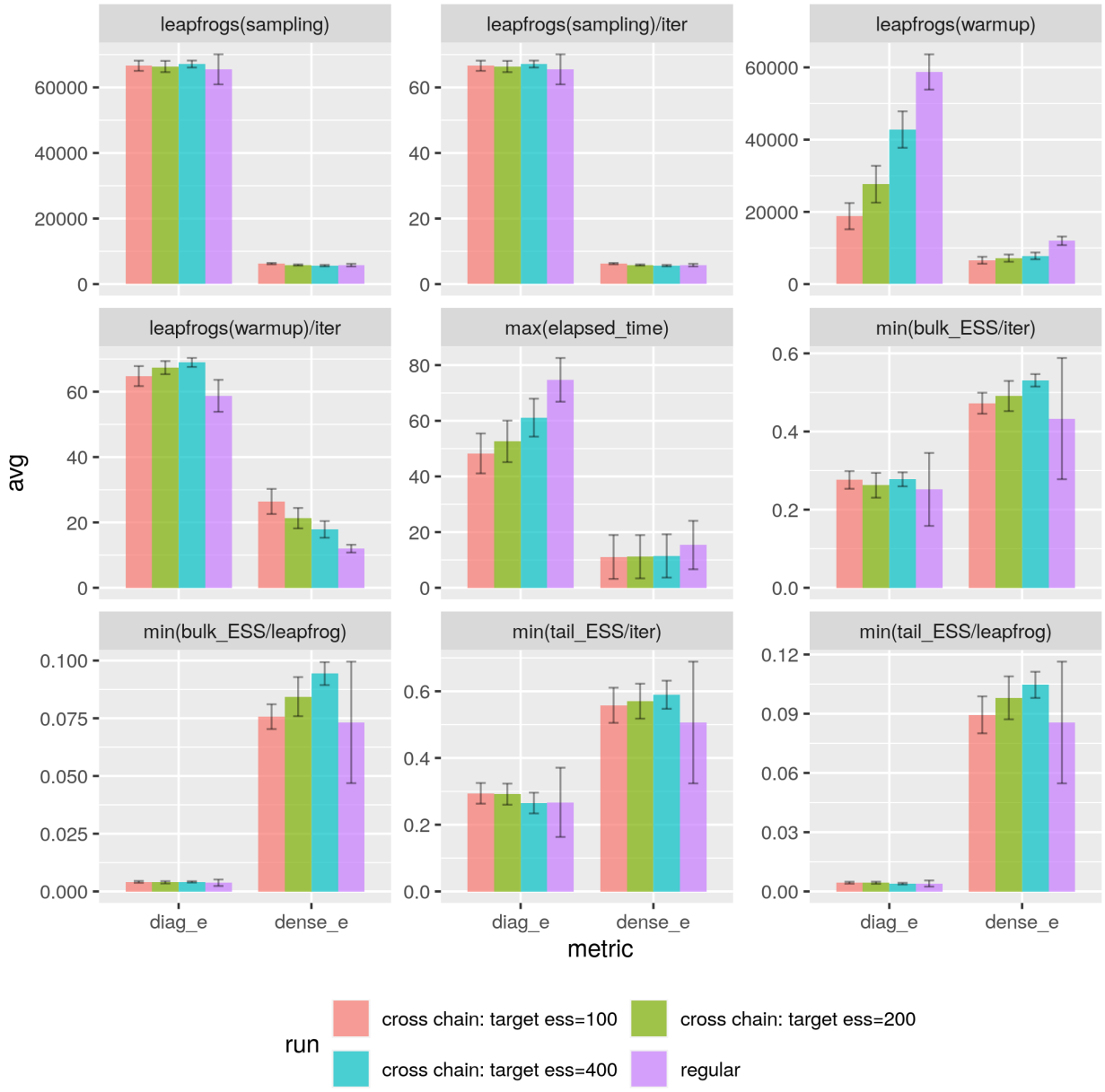

FIGURE 7. Cross-chain warmup performance comparison: SIR model

REFERENCES

[1] B. BALES ET AL., *New adaptive warmup proposal.* https://discourse.mc-stan.org/t/new-adaptive-warmup-proposal-looking-for-feedback/12039.

[2] B. CARPENTER, A. GELMAN, M. D. HOFFMAN, D. LEE, B. GOODRICH, M. BETANCOURT, M. BRUBAKER, J. GUO, P. LI, AND A. RIDDELL, *Stan: A Probabilistic Programming Language*, Journal of Statistical Software, 76 (2017), pp. 1–32.

[3] M. D. HOFFMAN AND A. GELMAN, *The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo*, Journal of Machine Learning Research, 15 (2014), pp. 1593–1623.

[4] D. LU, W. R. GILLESPIE, S. GIRISH, P. AGARWAL, C. LI, J. HIRATA, Y.-W. CHU, M. KAGEDAL, L. LEON, V. MAIYA, AND J. Y. JIN, *Time-to-Event Analysis of Polatuzumab Vedotin-Induced Peripheral Neuropathy to Assist in the Comparison of Clinical Dosing Regimens*, CPT: pharmacometrics & systems pharmacology, 6 (2017), pp. 401–408.

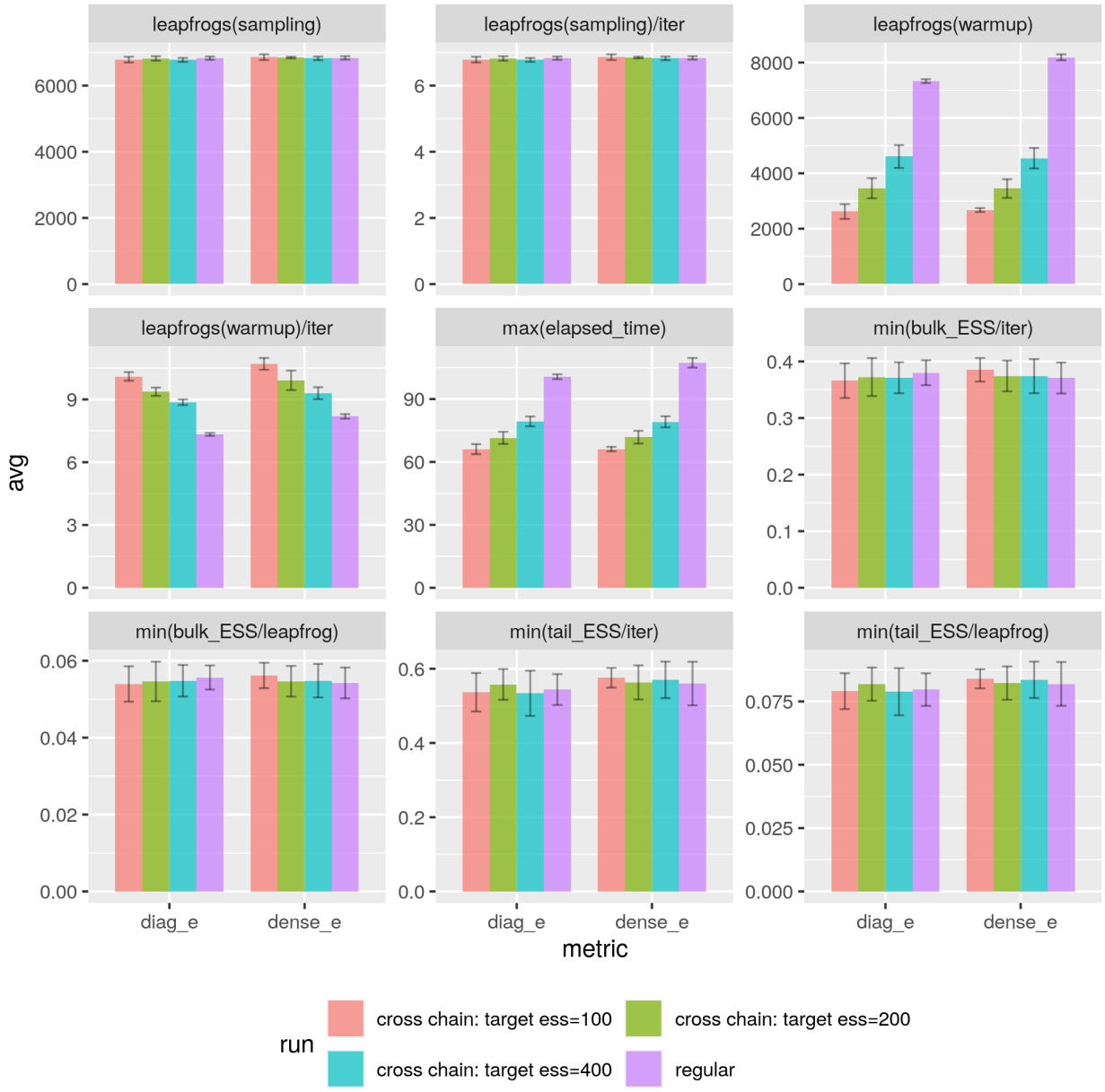

FIGURE 8. Cross-chain warmup performance comparison: chemical reaction model

[5] TORSTEN DEVELOPMENT TEAM, *Torsten: library of C++ functions that support applications of Stan in Pharmacometrics*. https://github.com/metrumresearchgroup/Torsten.

[6] A. VEHTARI, A. GELMAN, D. SIMPSON, B. CARPENTER, AND P.-C. BÜRKNER, *Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC*, arXiv:1903.08008 [stat], (2019). arXiv: 1903.08008.

[7] Y. ZHANG ET AL., *Cross-chain warmup adaptation using mpi*. https://discourse.mc-stan.org/t/cross-chain-warmup-adaptation-using-mpi/12912.

[8] ———, *github repo: Speed up population bayesian inference by combining cross-chain warmup and within-chain parallelism*. https://github.com/metrumresearchgroup/acop_2020_torsten_parallelization.

[9] Y. ZHANG AND W. R. GILLESPIE, *Poster: Speed up ode-based modeling using torstens population solvers*, in StanCon 2019, Cambridge, UK, August 2019.

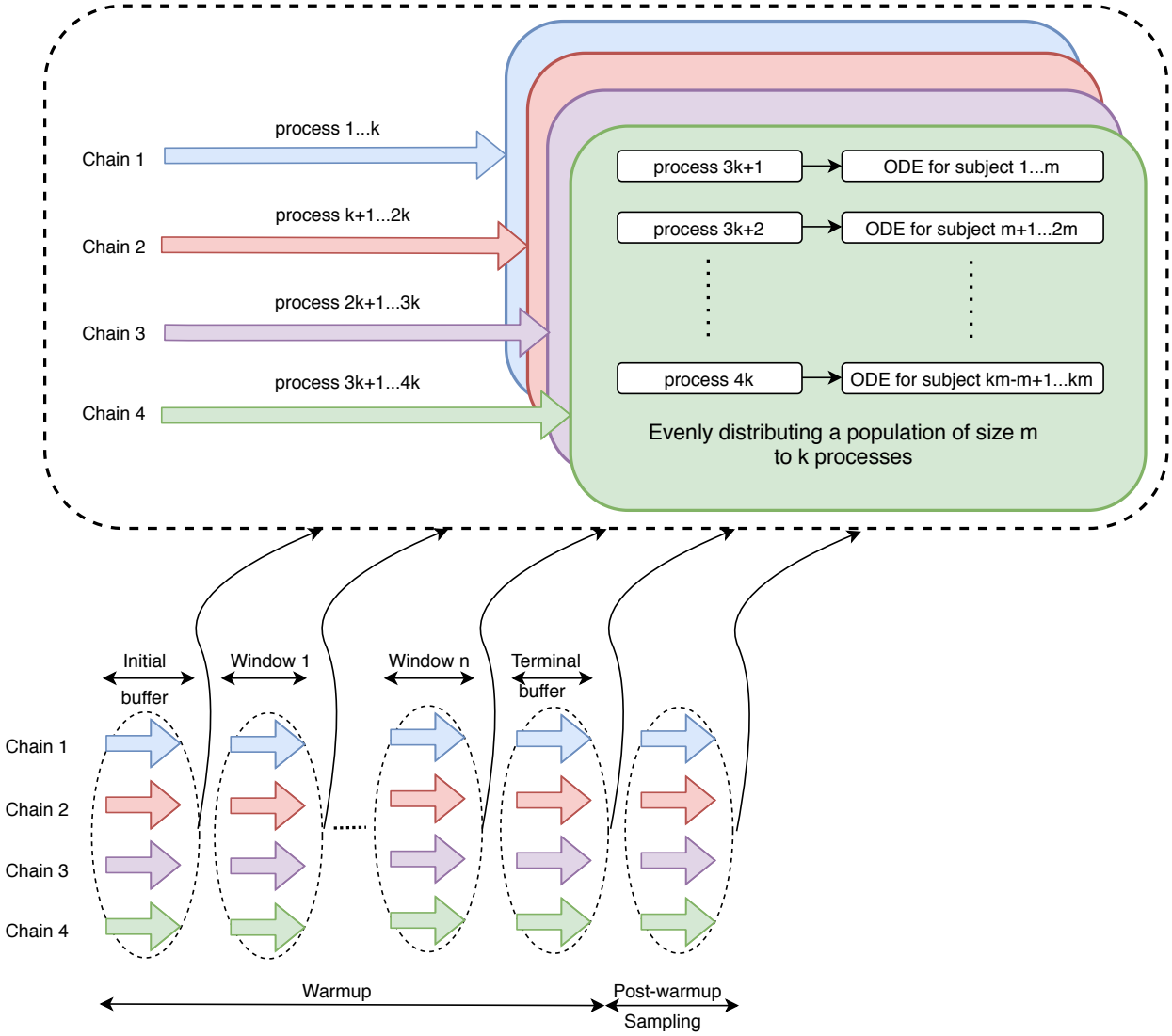

FIGURE 9. Multilevel parallelism for ODE-based population models. A simplified version of Figure 1, the lower diagram shows the cross-chain warmup through multiple windows. In within-chain parallelization, as shown in the upper diagram, each chain has its own parameter samples(indicated by different colors), and dedicated processes for solving the population model.

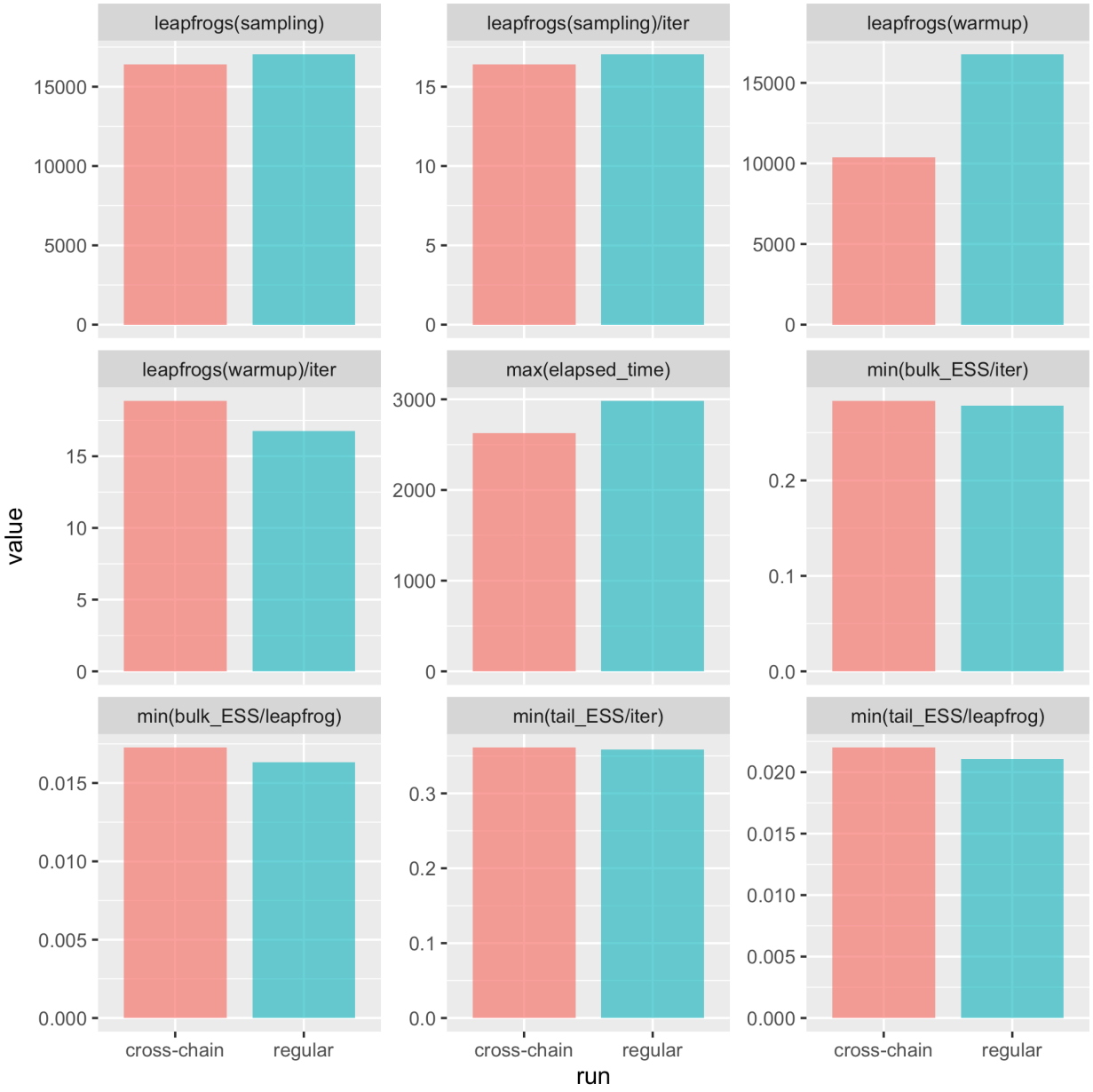

FIGURE 10. Cross-chain warmup performance comparison(Target ESS 400): TTPN model.

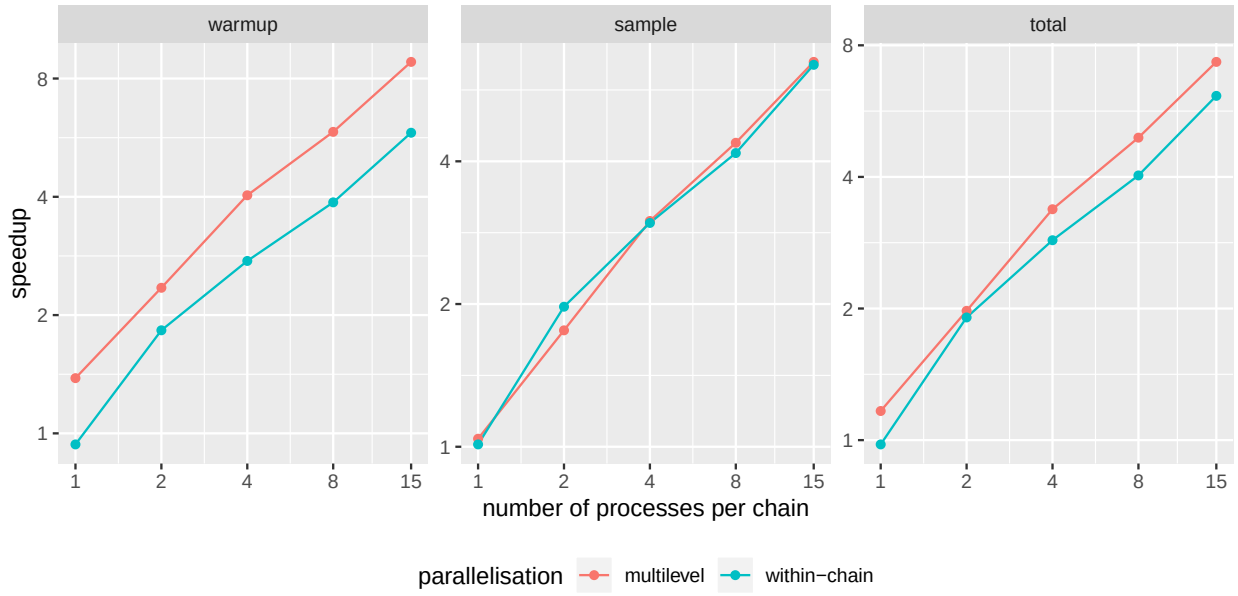

FIGURE 11. multilevel parallelisation performance of TTPN model(target ESS=400). Speedup for warmup, sampling, and total(warmup + sampling) are based on corresponding regular run wall time.