MultiBUGS: A parallel implementation of the BUGS modelling framework for faster Bayesian inference

Robert Goudie

with Andrew Thomas,
Rebecca Turner & Daniela De Angelis

9 Jan 2019

MRC Biostatistics Unit, University of Cambridge

Background

BUGS is general-purpose Bayesian modelling software that implements Markov chain Monte Carlo (MCMC).

Started in 1989: ClassicBUGS, then WinBUGS, then OpenBUGS

Ideas from BUGS widely adopted

- JAGS (Plummer, 2017)
- NIMBLE (Valpine *et al.*, 2017)
- Related ideas are used in Stan (Carpenter *et al.*, 2017)

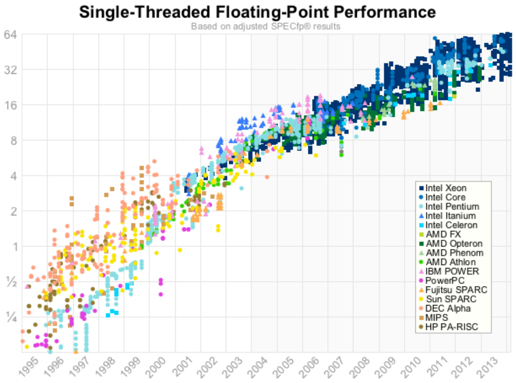
Latest version is MultiBUGS:

- Available to download from `https://www.multibugs.org`
- This talk is based on Goudie *et al.* (?2019)

Plummer, M. (2017). *JAGS Version 4.2.0 User Manual.*
Valpine, P. de *et al.* (2017). "Programming with Models: Writing Statistical Algorithms for General Model Structures with NIMBLE". *Journal of Computational and Graphical Statistics* **26**, 403–413.
Carpenter, B. *et al.* (2017). "Stan: A Probabilistic Programming Language". *Journal of Statistical Software* **76**, 1–32.
Goudie, R. J. B. *et al.* (?2019). "MultiBUGS: A Parallel Implementation of the BUGS Modelling Framework for Faster Bayesian Inference". *Journal of Statistical Software.* https://arxiv.org/abs/1704.03216.

Background & motivation

Impossible or extremely time-consuming to use OpenBUGS with a huge amount of data.



- OpenBUGS uses only a single CPU/core/thread
- Increases in single-thread performances slowing
- Number of cores available increasing

Aim: to make the speed-ups of multi-core computation available to applied statisticians using BUGS for general models, without requiring any knowledge of parallel programming

Note: not aiming to improve mixing properties of the Markov chain, simply to run it faster

Parallelisation in MultiBUGS

MultiBUGS implements two levels of parallelisation.

Simple approach – run each of multiple, independent MCMC chains on a separate CPU or core (Bradford and Thomas, 1996)

- Useful for assessing convergence e.g. the Brooks-Gelman-Rubin diagnostic
- Burn-in time isn't shortened

More complicated approach – use multiple CPUs/cores for a single MCMC chain

- Aim to shorten the per-iteration computation time by identifying tasks that can be calculated in parallel
- MultiBUGS parallelises the following tasks:
  1. "Likelihood" computation
  2. Sampling of conditionally-independent components

Bradford, R. and Thomas, A. (1996). "Markov Chain Monte Carlo Methods for Family Trees Using a Parallel Processor". *Statistics and Computing* **6**, 67–75.

Selected other parallelisation approaches

1. Speculatively consider sequence of MCMC steps, evaluate each on a separate core.

   e.g. Brockwell (2006)

2. Modify Metropolis-Hastings algorithm by proposing a sequence of candidate points in parallel.

   e.g. Calderhead (2014).

3. Run parts of the model on separate cores and then combine

   e.g. Scott *et al.* (2016), Goudie *et al.* (2019)

Brockwell, A. E. (2006). "Parallel Markov Chain Monte Carlo Simulation by Pre-Fetching". *Journal of Computational and Graphical Statistics* **15**, 246–261.
Calderhead, B. (2014). "A General Construction for Parallelizing Metropolis-Hastings Algorithms". *Proceedings of the National Academy of Sciences of the United States of America* **111**, 17408–17413.
Scott, S. L. *et al.* (2016). "Bayes and Big Data: The Consensus Monte Carlo Algorithm". *International Journal of Management Science and Engineering Management* **11**, 78–88.
Goudie, R. J. B. *et al.* (2019). "Joining And Splitting Models with Markov Melding". *Bayesian Analysis* **14**, 81–109.

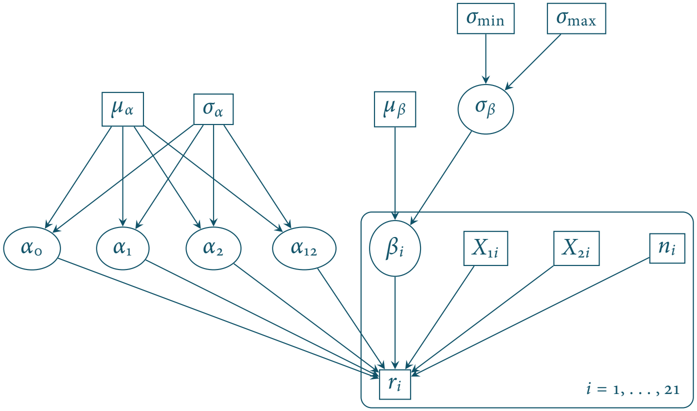
Trivial illustrative example ("seeds")

A random-effects logistic regression without outcome r_i and covariates X_{1i} and X_{2i} (21 observations)

$$r_i \sim \text{Bin}(p_i, n_i)$$

$$\text{logit}(p_i) = \alpha_0 + \alpha_1 X_{1i} + \alpha_2 X_{2i} + \alpha_{12} X_{1i} X_{2i} + \beta_i \qquad \alpha_0, \alpha_1, \alpha_2, \alpha_{12} \sim \text{N}(\mu_\alpha, \sigma_\alpha^2)$$

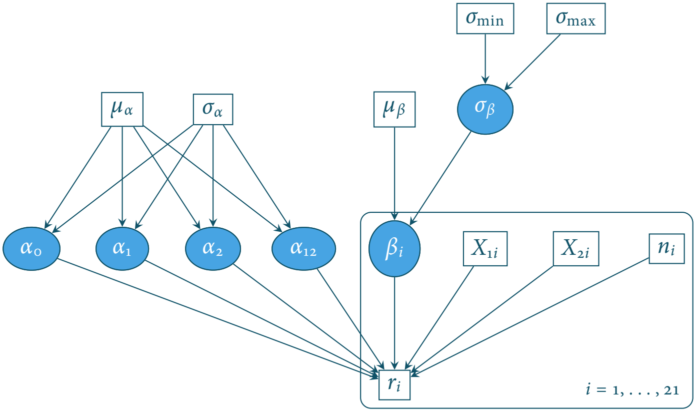
$$\beta_i \sim \text{N}(\mu_\beta, \sigma_\beta^2) \qquad \sigma_\beta \sim \text{Unif}(\sigma_{\min}, \sigma_{\max})$$



Trivial illustrative example ("seeds")

A random-effects logistic regression without outcome r_i and covariates X_{1i} and X_{2i} (21 observations)

$$
\begin{aligned}
r_i &\sim \text{Bin}(p_i, n_i) \\
\text{logit}(p_i) &= \alpha_0 + \alpha_1 X_{1i} + \alpha_2 X_{2i} + \alpha_{12} X_{1i} X_{2i} + \beta_i & \alpha_0, \alpha_1, \alpha_2, \alpha_{12} &\sim \text{N}(\mu_\alpha, \sigma_\alpha^2) \\
\beta_i &\sim \text{N}(\mu_\beta, \sigma_\beta^2) & \sigma_\beta &\sim \text{Unif}(\sigma_{\min}, \sigma_{\max})
\end{aligned}
$$



Generic algorithm used by BUGS

At each MCMC iteration, BUGS does the following:

```
for $v$ in $S$ do
    Do something involving $p(v \mid V_{-v})$
end for
```

Generic algorithm used by BUGS

At each MCMC iteration, BUGS does the following:

```
for $v$ in $S$ do
    Do something involving $p(v \mid V_{-v})$
end for
```

The conditional distribution $p(v \mid V_{-v})$ of a node $v \in S$, given the other nodes V_{-v}, is

$$\begin{aligned} p(v \mid V_{-v}) &\propto p(v \mid \mathrm{pa}(v)) \times \prod_{u \in \mathrm{ch}(v)} p(u \mid \mathrm{pa}(u)) \\ &= p(v \mid \mathrm{pa}(v)) \times L(v) \\ &= \text{“prior” term} \times \text{“likelihood” term} \end{aligned}$$

Generic algorithm used by BUGS

At each MCMC iteration, BUGS does the following:

```
for $v$ in $S$ do
    Evaluate the "prior" $p(v \mid \mathrm{pa}(v))$
    for $u \in \mathrm{ch}(v)$ do
        Evaluate "likelihood" component $p(u \mid \mathrm{pa}(u))$
    end for
    etc ...
end for
```

The conditional distribution $p(v \mid V_{-v})$ of a node $v \in S$, given the other nodes V_{-v}, is

$$\begin{aligned} p(v \mid V_{-v}) &\propto p(v \mid \mathrm{pa}(v)) \times \prod_{u \in \mathrm{ch}(v)} p(u \mid \mathrm{pa}(u)) \\ &= p(v \mid \mathrm{pa}(v)) \times L(v) \\ &= \text{“prior” term} \times \text{“likelihood” term} \end{aligned}$$

Type 1 – Splitting likelihood computation

When a parameter has many children, the likelihood is the product of many terms.

$$L(v) = \prod_{u \in \mathrm{ch}(v)} p(u \mid \mathrm{pa}(u))$$

But, with a partition of the children $\mathrm{ch}(v) = \left\{ \mathrm{ch}^{(1)}(v), \ldots, \mathrm{ch}^{(C)}(v) \right\}$,

$$L(v) = \underbrace{\left[\prod_{u \in \mathrm{ch}^{(1)}(v)} p(u \mid \mathrm{pa}(u)) \right]}_{\text{Core 1}} \times \underbrace{\left[\prod_{u \in \mathrm{ch}^{(2)}(v)} p(u \mid \mathrm{pa}(u)) \right]}_{\text{Core 2}} \times \ldots \times \underbrace{\left[\prod_{u \in \mathrm{ch}^{(C)}(v)} p(u \mid \mathrm{pa}(u)) \right]}_{\text{Core C}}$$

Type 2 – Parallelising sampling of parameters

When a model includes a large number of parameters then computation may be slow in aggregate, even if sampling of each individual parameter is fast.

But parameters that do not directly depend on each other can be updated simultaneously

More precisely, parameters in a mutually conditionally-independent set $W \subseteq S$ can be updated simultaneously. That is, W satisfying

$$\text{all } w_1, w_2 \in W \ (w_1 \neq w_2) \text{ satisfy } w_1 \perp\!\!\!\perp w_2 \mid V \setminus W$$

If not all parameters can be collated into a single W, form a series of Ws and sample in turn.

Identifying sets W

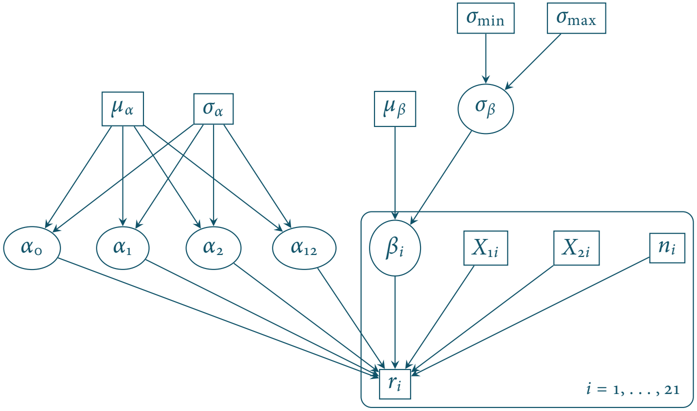
Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \emptyset \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$

Identifying sets W

Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

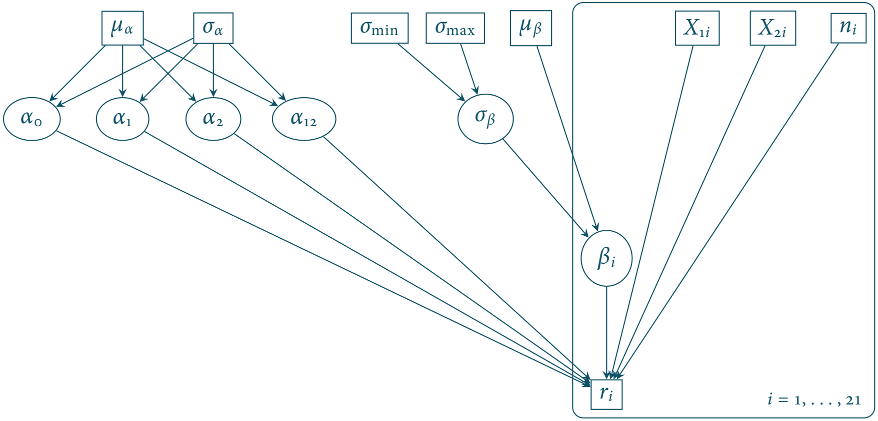
$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \varnothing \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$



Identifying sets W

Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

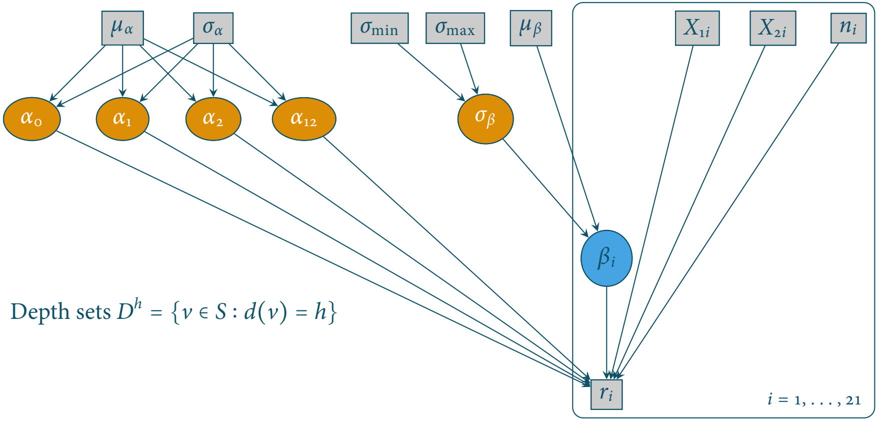
$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \varnothing \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$



Identifying sets W

Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

$$d(v) = \begin{cases} 0 & \text{if } \operatorname{pa}(v) = \varnothing \\ 1 + \max_{u \in \operatorname{pa}(v)} d(u) & \text{otherwise} \end{cases}$$

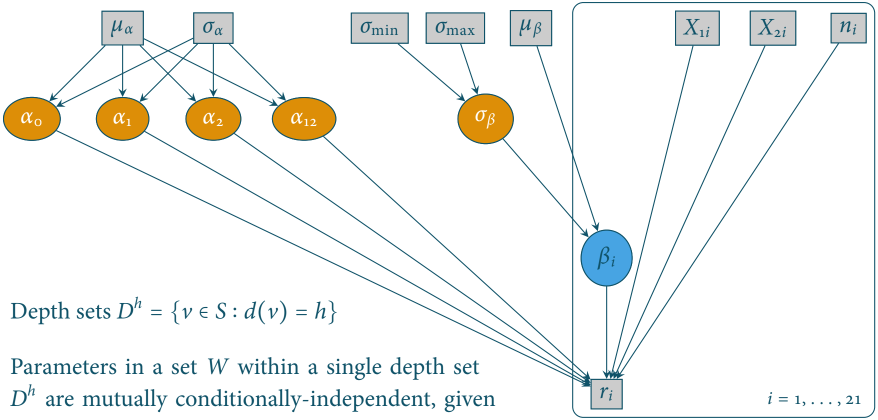


Depth sets $D^h = \{v \in S : d(v) = h\}$

Identifying sets W

Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \varnothing \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$



Depth sets $D^h = \{v \in S : d(v) = h\}$

Parameters in a set W within a single depth set D^h are mutually conditionally-independent, given the other nodes $V \setminus W$, if the parameters in W have no child node in common.

Which type of parallelism to exploit for each parameter in the model?

MultiBUGS aims to

- Parallelise the evaluation of the “likelihood” of ‘fixed effect’-like parameters (Type 1)
- Parallelise sampling of ‘random effect’-like parameters (Type 2)

Which type of parallelism to exploit for each parameter in the model?

MultiBUGS aims to

- Parallelise the evaluation of the "likelihood" of 'fixed effect'-like parameters (Type 1)
- Parallelise sampling of 'random effect'-like parameters (Type 2)

Let $\overline{\text{ch}} = \text{mean}_{v \in S} |\text{ch}(v)|$ be the mean number of children across parameters

Heuristic algorithm:
Consider each depth set D^h in turn, starting with the 'deepest' set

```
if a parameter has more than $2 \times \overline{\text{ch}}$ children then
    Parallelise evaluation of this parameter's "likelihood" (Type 1)
else
    Sample this parameter in parallel, if possible (Type 2)
end if
```

Computation schedule for illustrative example, with 4 cores

There are 26 stochastic parameters.

Computation schedule for illustrative example, with 4 cores

There are 26 stochastic parameters.

Topological depths:

- $d(\beta_1) = \cdots = d(\beta_{21}) = 2$
- $d(\alpha_0) = \cdots = d(\alpha_{12}) = d(\sigma_\beta) = 1$

Computation schedule for illustrative example, with 4 cores

There are 26 stochastic parameters.

Topological depths:

- $d(\beta_1) = \cdots = d(\beta_{21}) = 2$
- $d(\alpha_0) = \cdots = d(\alpha_{12}) = d(\sigma_\beta) = 1$

1. Parameters $\beta_1, \ldots, \beta_{21}$
    - Likelihood evaluation not parallelised these parameter have only 1 child and $\overline{\text{ch}} \approx 4.8$.
    - But, $\beta_1, \ldots, \beta_{21}$ are mutually conditionally-independent and so sampling can be parallelised
    - $21 \bmod 4 \neq 0$ so we will have idle cores

	Core			
Row	1	2	3	4
1	β_1	β_2	β_3	β_4
2	β_5	β_6	β_7	β_8
3	β_9	β_{10}	β_{11}	β_{12}
4	β_{13}	β_{14}	β_{15}	β_{16}
5	β_{17}	β_{18}	β_{19}	β_{20}
6	β_{21}			

Computation schedule for illustrative example, with 4 cores

There are 26 stochastic parameters.

Topological depths:

- $d(\beta_1) = \cdots = d(\beta_{21}) = 2$
- $d(\alpha_0) = \cdots = d(\alpha_{12}) = d(\sigma_\beta) = 1$

1. Parameters $\beta_1, \ldots, \beta_{21}$
    - Likelihood evaluation not parallelised these parameter have only 1 child and $\overline{\text{ch}} \approx 4.8$.
    - But, $\beta_1, \ldots, \beta_{21}$ are mutually conditionally-independent and so sampling can be parallelised
    - $21 \bmod 4 \neq 0$ so we will have idle cores
2. Parameters $\alpha_0, \alpha_1, \alpha_2, \alpha_{12}$ and σ_β
    - All of these parameters have 21 children $\overline{\text{ch}} \approx 4.8$, so likelihood computation is parallelised

	Core			
Row	1	2	3	4
1	β_1	β_2	β_3	β_4
2	β_5	β_6	β_7	β_8
3	β_9	β_{10}	β_{11}	β_{12}
4	β_{13}	β_{14}	β_{15}	β_{16}
5	β_{17}	β_{18}	β_{19}	β_{20}
6	β_{21}			
7	α_{12}	α_{12}	α_{12}	α_{12}
8	α_1	α_1	α_1	α_1
9	α_2	α_2	α_2	α_2
10	α_0	α_0	α_0	α_0
11	σ_β	σ_β	σ_β	σ_β

Implementation notes

Each core keeps

- The complete DAG, the computation schedule, and associated sampling algorithms
- A copy of the current state of the MCMC
- Two pseudo-random number generation (PRNG) streams
    1. "Core-specific" stream, initialised with a different seed for each core
        - Used when we wish to sample independently across cores
    2. "Common" stream, initialised using the same seed on all cores.
        - Used when we wish to obtain the same samples across cores

Implementation notes

Each core keeps

- The complete DAG, the computation schedule, and associated sampling algorithms
- A copy of the current state of the MCMC
- Two pseudo-random number generation (PRNG) streams
    1. "Core-specific" stream, initialised with a different seed for each core
        - Used when we wish to sample independently across cores
    2. "Common" stream, initialised using the same seed on all cores.
        - Used when we wish to obtain the same samples across cores

Most existing code can be reused!

Implementation notes

Each core keeps

- The complete DAG, the computation schedule, and associated sampling algorithms
- A copy of the current state of the MCMC
- Two pseudo-random number generation (PRNG) streams
    1. “Core-specific” stream, initialised with a different seed for each core
        - Used when we wish to sample independently across cores
    2. “Common” stream, initialised using the same seed on all cores.
        - Used when we wish to obtain the same samples across cores

Most existing code can be reused!

Likelihood parallelisation: Just delete the children whose likelihood contribution is calculated elsewhere. MPI function `Allreduce` used to aggregate.

For Metropolis-Hastings: the prior, the sampling of new value, and Metropolis test (redundantly) replicated on every core, using “common” PRNG stream

Implementation notes

Each core keeps

- The complete DAG, the computation schedule, and associated sampling algorithms
- A copy of the current state of the MCMC
- Two pseudo-random number generation (PRNG) streams
  1. "Core-specific" stream, initialised with a different seed for each core
     - Used when we wish to sample independently across cores
  2. "Common" stream, initialised using the same seed on all cores.
     - Used when we wish to obtain the same samples across cores

Most existing code can be reused!

Likelihood parallelisation: Just delete the children whose likelihood contribution is calculated elsewhere. MPI function `Allreduce` used to aggregate.

For Metropolis-Hastings: the prior, the sampling of new value, and Metropolis test (redundantly) replicated on every core, using "common" PRNG stream

Sampling parallelisation: Just delete the nodes that are sampled elswhere from the list of nodes to be updated. Sample using "core-specific" PRNG stream, and propagate new values across cores using `Allgather`.

Hierarchical regression example

Based on an analysis linked database of methadone prescriptions given to opioid dependent patients in Scotland (Gao *et al.*, 2016)

425,112 observations, with the following structure:

- Geographic regions ($i = 1, \ldots, 8$)
  - Containing patients (20410 in total)
    - Each of whom may have multiple prescriptions

For some measurements patient-level identifiers are available:

$$y_{ijk} = \sum_{m=1}^{4} \beta_m \times \underbrace{x_{mij}}_{\text{covariates}} + \underbrace{u_i}_{\substack{\text{region-}\\\text{specific}\\\text{intercept}}} + \underbrace{v_i}_{\substack{\text{region-}\\\text{specific}\\\text{slope}}} \times \underbrace{r_{ijk}}_{\text{covariate}} + \underbrace{w_{ij}}_{\substack{\text{patient-}\\\text{level}\\\text{intercept}}} + \varepsilon_{ijk}$$

For other measurements no patient-level identifier is available:

$$z_{il} = \lambda + \underbrace{u_i}_{\substack{\text{region-}\\\text{specific}\\\text{intercept}}} + \underbrace{v_i}_{\substack{\text{region-}\\\text{specific}\\\text{slope}}} \times \underbrace{s_{il}}_{\text{covariate}} + \eta_{il}$$

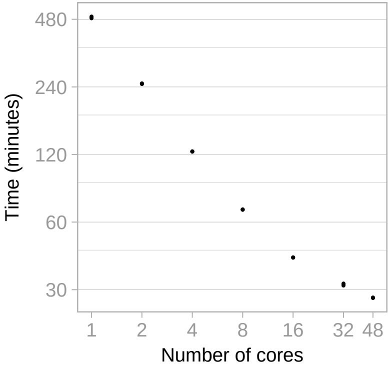
Gao, L. *et al.* (2016). "Risk-Factors for Methadone-Specific Deaths in Scotland's Methadone-Prescription Clients between 2009 and 2013". *Drug and Alcohol Dependence* **167**, 214–223.

Timings

In OpenBUGS, running chains 2 for 15,000 iterations takes about 32 hours.

In MultiBUGS:

- Sampling of pairs of random-effect means and variances parallelised;
- Sampling of person-level random effects w_{ij} parallelised, except for the component corresponding to the person with the most observations (176 observations)
- The likelihood computation of all the other parameters in the model is parallelised



When is MultiBUGS quicker than OpenBUGS?

Independent-chain parallelisation is almost always be advantageous whenever sufficient cores are available, since no communication across cores is needed.

Within-chain parallelisation will be most useful for models involving parameters with a large number of likelihood terms and/or a large number of conditionally independent parameters.

- e.g. many standard regression-type models involving both fixed and random effects

For models without these features, the overheads of within-chain parallelisation may outweigh the gains on some computing hardware.

Note the mixing properties are the same as in OpenBUGS (but the exact samples will differ due to different PRNG stream use)

Outlook

MultiBUGS 1.0 is finished – `https://www.multibugs.org`

MultiBUGS 2.0[1] uses a more efficient system for communicating partial models to cores

Released version requires Windows, but we did port MultiBUGS 1.0 to Ubuntu

A compendium of 'big models' would be useful

Martyn Plummer is adopting a similar idea in JAGS (using OpenMP)

[1] `https://github.com/MultiBUGS/MultiBUGS`