Modeling computation kernels with Stan

Hoël Jalmin
Tutored by Arnaud Legrand and Tom Cornebize

The 21th of June, 2019

A faire: refaire les images

Context

With the current need for high performance computing, and the hardware complexity:

- How to predict the duration of calculations?
- How to detect performance anomaly?

For this talk:

1. Brief presentation of the context
2. Introduction to Bayesian sampling
3. Examples of application

Background on HPC and Polaris research

Modern context

- HPC systems use thousands of nodes, cache, hyperthreading, etc
  $\rightarrow$ makes it difficult to predict performance
- Some functions (like DGEMM in the BLAS library) are used everywhere, and called thousands of times in a program.

Previous work

- Simulating high performance programs to optimize them at a lesser cost
- Elaborated complex models but needed to evaluate and confirm the models

Model Let's say $y \sim \mathcal{N}(\alpha * x + \beta, \sigma)$

- α, β, σ: Model parameters
- y: Dependent data (posterior)
- x: Independent data

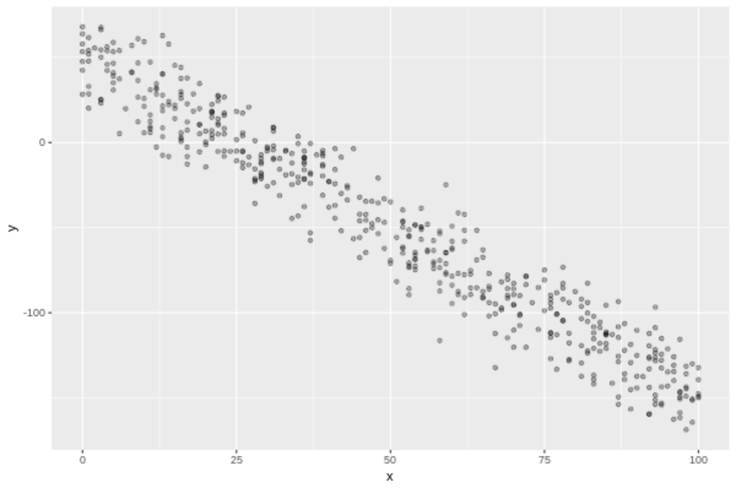
We observe some data and need to find model parameters

The vocabulary

- Posterior: The distribution of the parameters
- Likelihood: A function of the parameters, the model
- Prior: Existing knowledge of the system, guesses on the parameters values (σ>0 per example)

A Bayesian Sampler, Stan

With a simple example

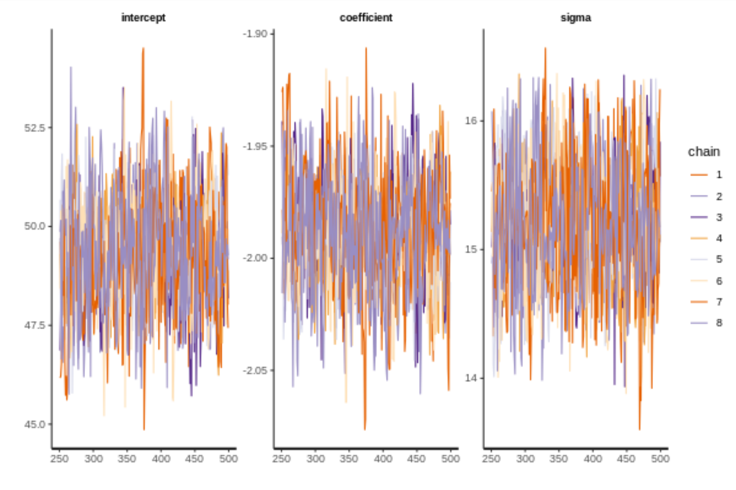


Using this data, we'll try to find the parameters that were used to generate it.

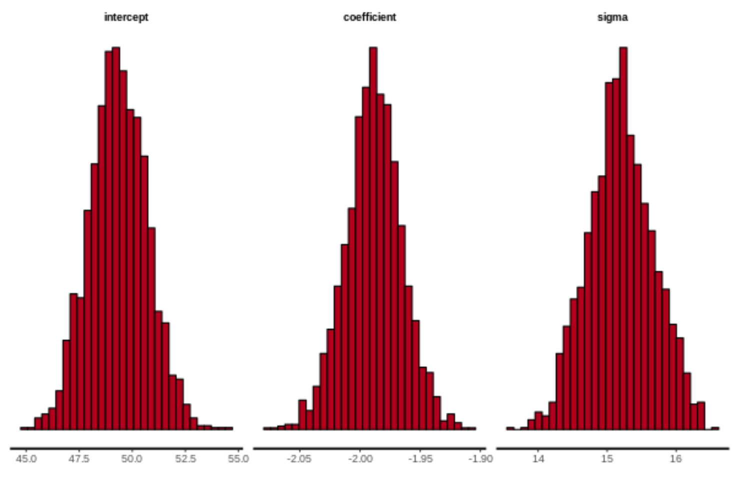
The Stan model

```
library(rstan)

modelString = "data { // the observations
    int<lower=1> N; // number of points
    vector[N] x;
    vector[N] y;
}
parameters { // what we want to find
    real beta;
    real alpha;
    real<lower=0> sigma; // indication: sigma cannot be negative
}
model {
    // We define our priors
    beta   ~ normal(0, 10); // We know that all the parameters follow a normal d
    alpha ~ normal(0, 10);
    sigma       ~ normal(0, 10);

    // Then, our likelihood function
    y ~ normal(alpha*x + beta, sigma);
}
"
sm = stan_model(model_code = modelString)
```

Checking the convergence



Looking at the histogram of the parameters

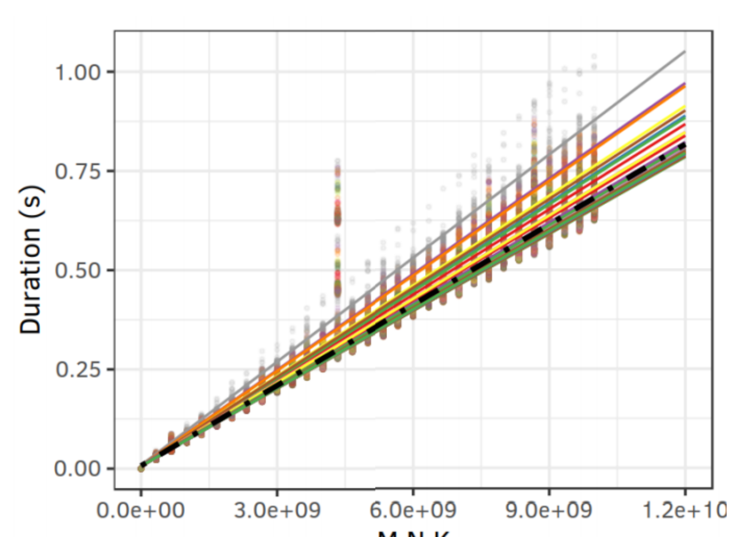


The importance of the priors

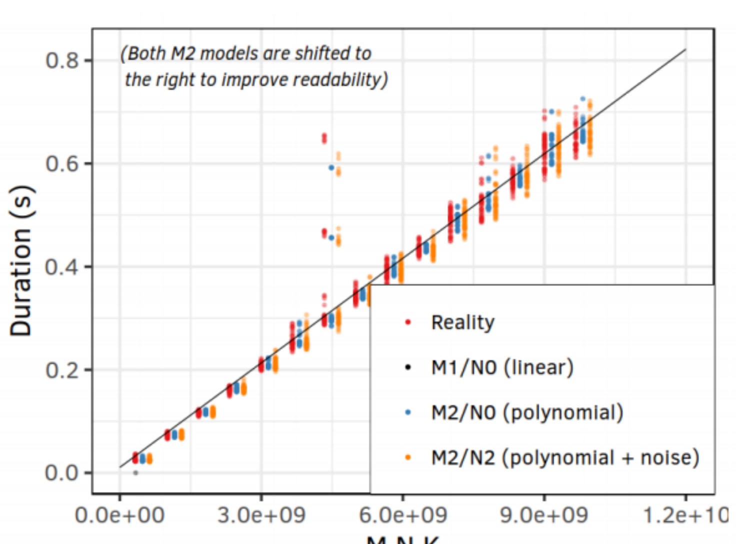
- The priors are necessary to have convergence in the fit
- Non-informative prior vs informative (careful not to have a falsely informative one and introduce bias)
- A little bit of precision is better, but initialisation values can do the trick

The different models for dgemm

- DGEMM's duration depends on the matrix size, but also on the CPU used to run it



Different possible models to account for the variabilities, some more accurate than others:



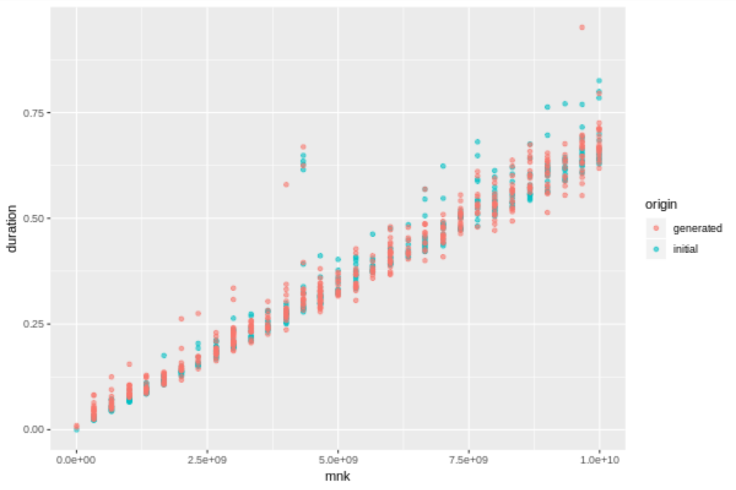
A polynomial model with noise depending on x

Like a linear model but with more parameters (in this case 10).

The model follows this:

$duration \sim \mathcal{N}(\mu[1]*mnk+\mu[2]*mn+\mu[3]*mk+\mu[4]*nk+\mu[5], \sigma[1]*mnk+\sigma[2]*mn+\sigma[3]*mk+\sigma[4]*nk+\sigma[5])$

The generated data



The same model with parameters depending on the host

- Much like the previous model, but with different observations for each host
- Added a variable for the number of hosts, and used matrices instead of vectors for all the parameters.

For this model we have:

$$duration[i] \sim \mathcal{N}(\mu[i,1]*mnk+\mu[i,2]*mn+\mu[i,3]*mk+\mu[i,4]*nk+\mu[i,5], \sigma[i,1]*mnk+\sigma[i,2]*mn+\sigma[i,3]*mk+\sigma[i,4]*nk+\sigma[i,5])$$

A hierarchical linear model

- Useful to find the value of hyperparameters from which we get the parameters
- From this we could calculate new parameters for new CPUs
- Here μ_α and σ_α are the hyperparameters for α, and the same goes for the other parameters

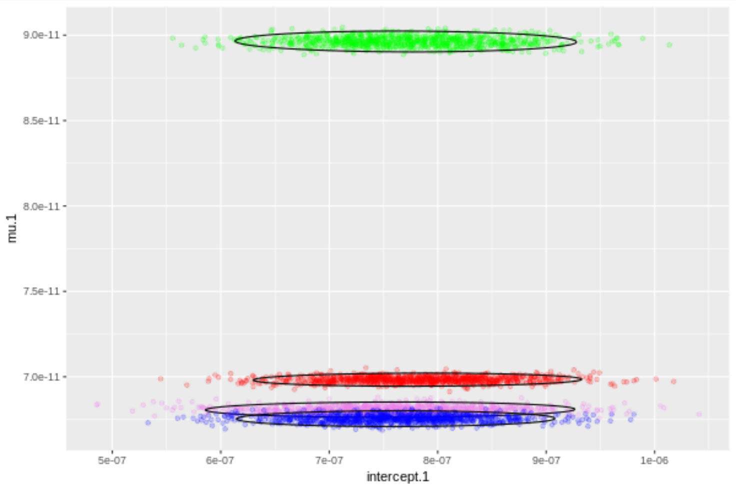
$\mu_\alpha \sim \mathcal{N}(\alpha\_\text{moy},\alpha\_\text{sd})$ with α_moy and α_sd the priors

$\sigma_\alpha \sim \mathcal{N}(0, 1)$

$\alpha[i] \sim \mathcal{N}(\mu_\alpha, \sigma_\alpha)$

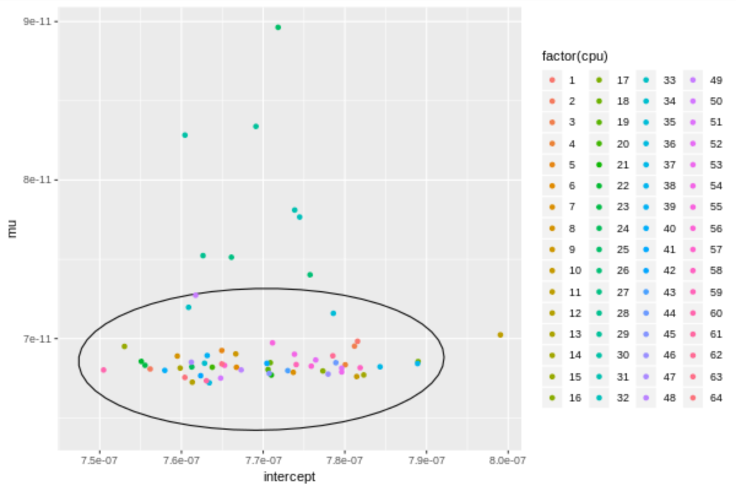
$duration[i] \sim \mathcal{N}(\alpha[i] * mnk + \beta[i], \theta[i] * mnk + \gamma[i])$

The posterior with models depending on the host shows a lot of difference between hosts (here we have 3 ”average” CPU and a slow one):



Posterior visualisation

If we look at the means of the parameters' values for each host, we get a range of values in which most hosts are.



Conclusion

Following up work

- Modeling other calculation kernels
- Modeling the network communications
- Parsing and converting Stan code to C, to generate new data more efficiently
- Anomaly detection