Improving Operational Intensity in Data Bound Markov Chain Monte Carlo

[1]Balazs Nemeth, [1,2]Tom Haber, [2]Thomas J. Ashby, and [1]Wim Lamotte

[1]EDM, Hasselt University, Belgium

[2]Exascience Lab, imec, Belgium

Introduction

Background

- Processor stalls due to discrepancy between memory bandwidth and processor speed
- Efficiency determined by

$$\text{operational intensity} = \frac{\#\text{operations}}{\text{byte}}$$

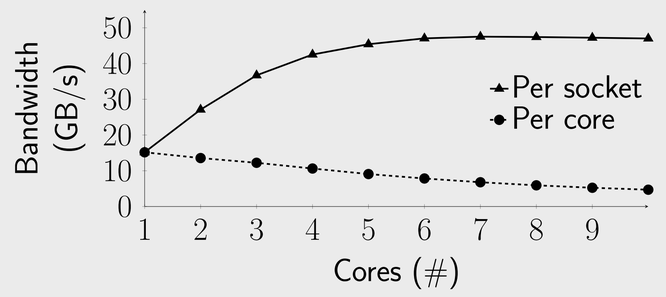
- Perform useful computation during otherwise stalled cycles
- Demonstrated with two Markov chain Monte Carlo samplers on Bayesian Logistic Regression target in big data and machine learning context

$$P(\boldsymbol{X}|\boldsymbol{\theta}) = \prod_i \frac{1}{1 + e^{y_i \boldsymbol{x}_i \cdot \boldsymbol{\theta}}}$$

Memory Hierarchy

- Memory hierarchy places importance on locality
- Multiple cores needed to saturate memory bandwidth
- Per core memory bandwidth diminishes due to shared on-die memory controller
- Operational intensity tends to drop if #cores increases

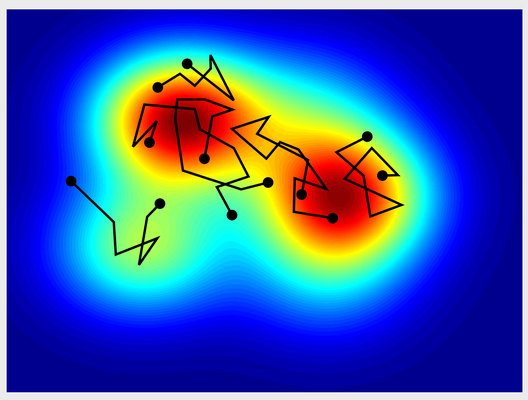
Memory type	Size	Bandwidth
CPU registers	few	1 TB/s
CPU caches	KBs	100 GB/s
Main memory	GBs	10 GB/s
Disk storage	TBs	100 MB/s



Sampler Algorithms

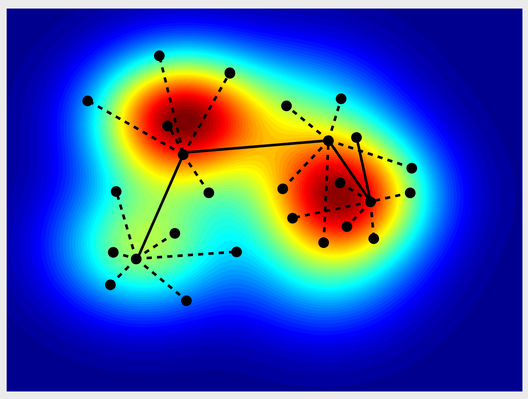
Multiple Chain

- Evolve multiple independent chains in parallel
- Discard some fraction, ρ, of samples from *each* chain as burn-in
- Speedup from parallelism is limited by $1/\rho$



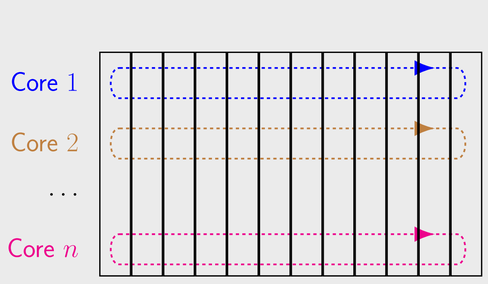
Multiple Proposal

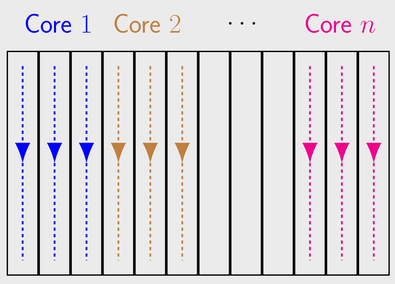
- Evolve a single chain
- Sample from set of proposals around current proposal
- Discard ρ fraction of samples from chain as burn-in



Original vs Restructured Algorithms

Original algorithms	Restructured algorithms
Each core accesses all data	Data is partitioned across cores
Multiple Chain: Each core runs an independent chain (#chains = #cores)	**Multiple Chain**: Each core works on part of a each chain (#chains ≠ #cores)
Multiple Proposal: Each core evaluates the whole Bayesian likelihood at a proposal (#chains = #proposals)	**Multiple Proposal**: Each core evaluates part of the Bayesian likelihood at each proposal (#proposals ≠ #cores)
Stalled cycles while waiting on data to move from memory to CPU caches	Perform useful work during otherwise stalled cycles

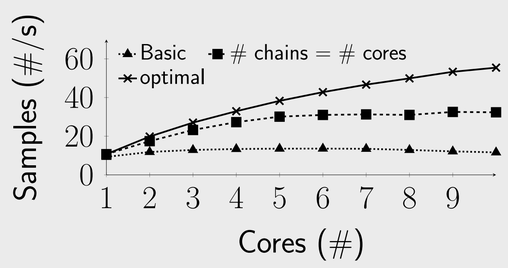


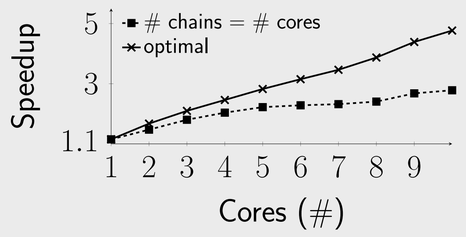


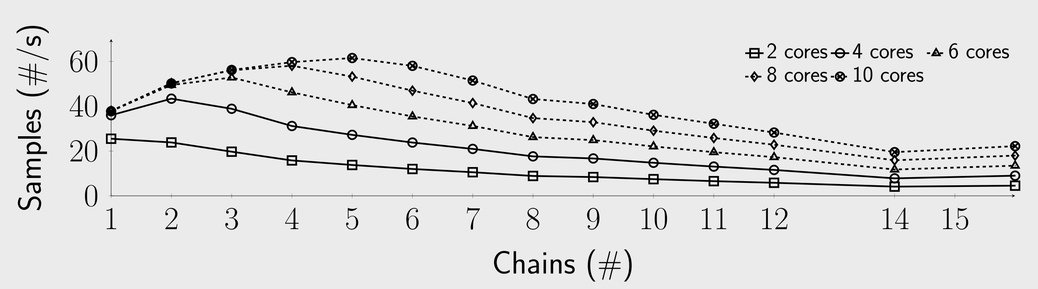
Results

Multiple Chain

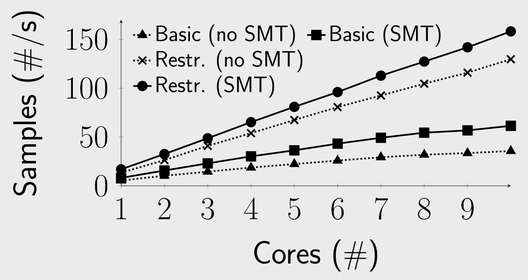
(higher is better)

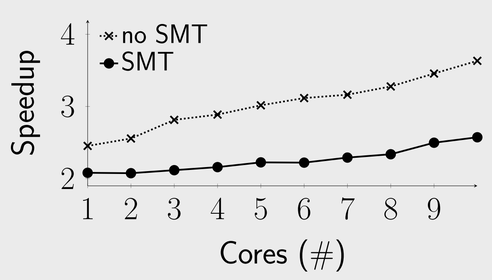


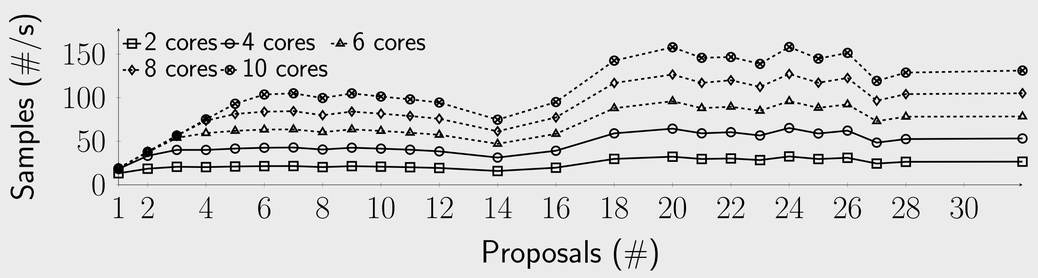




Multiple Proposal







Conclusion and Future Work

- Reducing #chains with the multiple chain sampler and more than twice as many proposal as cores with the multiple proposal sampler gives optimal performance
- Likelihoods that cannot be factored is part of future work
- Automatic tuning of #chains and #proposals will be studied
- Applying methodology to other algorithms will be explored next

Acknowledgments

Part of the work presented in this paper was funded by Johnson & Johnson. This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 671555.