Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 2348–2352

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Improving Operational Intensity in Data Bound Markov Chain Monte Carlo

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

[1] Expertise Centre for Digital Media, Wetenschapspark 2, 3590 Diepenbeek, Belgium
[2] Exascience Lab, Imec, Kapeldreef 75, B-3001 Leuven, Belgium

Abstract
Typically, parallel algorithms are developed to leverage the processing power of multiple processors simultaneously speeding up overall execution. At the same time, discrepancy between DRAM bandwidth and microprocessor speed hinders reaching peak performance. This paper explores how operational intensity improves by performing useful computation during otherwise stalled cycles. While the proposed methodology is applicable to a wide variety of parallel algorithms, and at different scales, the concepts are demonstrated in the machine learning context. Performance improvements are shown for Bayesian logistic regression with a Markov chain Monte Carlo sampler, either with multiple chains or with multiple proposals, on a dense data set two orders of magnitude larger than the last level cache on contemporary systems.

Keywords: Operational intensity, MCMC, Bayesian logistic regression, HPC, Big Data

1 Introduction

It has been a long established fact that the memory subsystem in contemporary systems plays a key role in determining performance. It was predicted that performance of all algorithms would be dominated by the memory subsystem [11], but this is only so for algorithms that neglect these system specifics. Various techniques in modern super-scalar processors, like prefetching, simultaneous multithreading (SMT) and instruction level parallelism (ILP), try to automatically alleviate this. Increased core counts provide more compute capacity but since memory contention increases, per core memory bandwidth diminishes impairing performance of parallel algorithms.

As long as the number of operations per unit of data, the *operational intensity* [10], is high, cores are less likely to stall from data starvation. This paper proposes to perform useful computation during otherwise stalled cycles shifting the computation kernel in the roofline model [10] towards maximum compute performance and way from memory bandwidth limits.

For many machine learning applications such as regression, support vector machines and neural networks, the training data set needs to stream through the memory hierarchy to compute the cost associated with the current parameter estimate. With Big Data, the situation worsens as hierarchy depth increases since data is loaded from disk or networked storage.

The remainder of this paper is structured as follows. Section 2 provides background on operational intensity, Bayesian inference, Markov chain Monte Carlo (MCMC) algorithms and references related work. Section 3 describes the methodology applied to those algorithms. Section 4 provides and discusses results and Section 5 concludes the paper.

2 Background

2.1 Bayesian Inference

High-dimensional feature data sets in today's Big Data era are not uncommon and of interest for machine learning since more complex structures can be revealed. It has been a subject of research resulting in methodologies to solve problems under certain constraints [7].

In Bayesian Inference, the posterior, $P(\boldsymbol{\theta}|\boldsymbol{X})$, is formed by combining prior knowledge with evidence. Logistic regression [6] is typically used to model categorical data. Assume independent data entries $(\boldsymbol{x}_i, y_i)$, with binary labels $y_i \in \{-1, 1\}$, the log likelihood for this model is given by $\sum_i -\log(1 + e^{y_i \boldsymbol{x}_i \cdot \boldsymbol{\theta}})$. Evaluation requires all entries to stream through the processor caches.

This paper focuses on Bayesian inference using logistic regression on a data set larger than the last level cache since it is both generally applicable, and has a low operational intensity by default. However, the idea is generally applicable to any parallel algorithm. The ratio of bandwidth to latency of successive caching layers will mostly determine the speedup.

2.2 Markov Chain Monte Carlo

MCMC methods provide a tool to sample from arbitrary distributions [2]. These are popularized by the fact that they suffer much less from the curse of dimensionality. One such method is the Metropolis Hastings (MH) algorithm [8]. For what follows, π is the target distribution, $q(.|\boldsymbol{\theta}^t)$ is the proposal distribution conditioned on the current position, $\boldsymbol{\theta}^t$, $0 \leq \rho < 1$ denotes the portion of samples discarded from a chain as *burn in*. For Bayesian regression, execution becomes dominated by posterior evaluations as the amount of data grows since $\pi(\boldsymbol{\theta}) = P(\boldsymbol{\theta}|\boldsymbol{X})$.

2.3 Parallel Markov Chain Monte Carlo Algorithms

A straightforward extension to the MH algorithm is the multiple chains (MC) algorithm. The left of Figure 1 illustrates one iteration with n chains. This algorithm suffers from Amdahl's law [1]. The speedup with n chains, denoted by S_n, is limited by $1/\rho$. Nevertheless, this approach is common in practice due to its simplicity.

Recently, a generalization for the MH algorithm that allows parallelization has been introduced [4]. It can be shown that a finite-state Markov chain constructed over a set of proposals, $\tilde{\boldsymbol{\theta}}^{1:n}$ maintains π. Evaluation of $\pi(\tilde{\boldsymbol{\theta}}^i)$ can proceed in parallel. The left of Figure 2 details an iteration of the multiple proposal (MP) algorithm with n proposals.

2.4 Related Work

Scott et al. [9] explores consensus Monte Carlo algorithms in the Big Data context by proposing to shard data, to break up prior information among workers and to combine partial posterior distributions. However, performance is not evaluated from the operational intensity perspective. Korattikara et al. [5] reformulate the accept/reject ratio as a statistical hypothesis test. More biased samples can be gathered per unit time reducing the variance of the estimator. The

$$\begin{tikzcd}
\tilde{\boldsymbol\theta}_1 \sim q(.|\boldsymbol\theta_1^t) \arrow[r] & \pi(\tilde{\boldsymbol\theta}_1) \arrow[r] & \boldsymbol\theta_1^{t+1} \\
\tilde{\boldsymbol\theta}_2 \sim q(.|\boldsymbol\theta_2^t) \arrow[r] & \pi(\tilde{\boldsymbol\theta}_2) \arrow[r] & \boldsymbol\theta_2^{t+1} \\
\vdots & \vdots & \vdots \\
\tilde{\boldsymbol\theta}_n \sim q(.|\boldsymbol\theta_n^t) \arrow[r] & \pi(\tilde{\boldsymbol\theta}_n) \arrow[r] & \boldsymbol\theta_n^{t+1}
\end{tikzcd}$$

$$\begin{tikzcd}
\tilde{\boldsymbol\theta}_1 \sim q(.|\boldsymbol\theta_1^t) \arrow[r] \arrow[dr] \arrow[dddr] & \pi_1(\tilde{\boldsymbol\theta}^1), \ldots, \pi_1(\tilde{\boldsymbol\theta}^n) \arrow[r] \arrow[dr] \arrow[dddr] & \boldsymbol\theta_1^{t+1} \\
\tilde{\boldsymbol\theta}_2 \sim q(.|\boldsymbol\theta_2^t) \arrow[ur] \arrow[r] \arrow[ddr] & \pi_2(\tilde{\boldsymbol\theta}^1), \ldots, \pi_2(\tilde{\boldsymbol\theta}^n) \arrow[ur] \arrow[r] \arrow[ddr] & \boldsymbol\theta_2^{t+1} \\
\vdots & \vdots & \vdots \\
\tilde{\boldsymbol\theta}_n \sim q(.|\boldsymbol\theta_n^t) \arrow[uuur] \arrow[uur] \arrow[r] & \pi_l(\tilde{\boldsymbol\theta}^1), \ldots, \pi_l(\tilde{\boldsymbol\theta}^n) \arrow[uuur] \arrow[uur] \arrow[r] & \boldsymbol\theta_n^{t+1}
\end{tikzcd}$$

Figure 1: An iteration of the basic (left) and restructured (right) MC algorithm with n chains. The target distribution on the right is factored into $\pi_1, \ldots, \pi_l$ each assigned to one of l cores. Core i computes factor π_i for all n chains. A reduction phase follows before producing n samples.

$$\begin{tikzcd}
 & \pi(\tilde{\boldsymbol\theta}^1) \arrow[dr] & \\
\tilde{\boldsymbol\theta}^{1:n} \sim q(.|\boldsymbol\theta^t) \arrow[ur] \arrow[r] \arrow[ddr] & \pi(\tilde{\boldsymbol\theta}^2) \arrow[r] & \boldsymbol\theta^{t+1:t+k} \sim I \\
 & \vdots & \\
 & \pi(\tilde{\boldsymbol\theta}^n) \arrow[uur] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \pi_1(\tilde{\boldsymbol\theta}^1), \ldots, \pi_1(\tilde{\boldsymbol\theta}^n) \arrow[dr] & \\
\tilde{\boldsymbol\theta}^{1:n} \sim q(.|\boldsymbol\theta^t) \arrow[ur] \arrow[r] \arrow[ddr] & \pi_2(\tilde{\boldsymbol\theta}^1), \ldots, \pi_2(\tilde{\boldsymbol\theta}^n) \arrow[r] & \boldsymbol\theta^{t+1:t+k} \sim I \\
 & \vdots & \\
 & \pi_l(\tilde{\boldsymbol\theta}^1), \ldots, \pi_l(\tilde{\boldsymbol\theta}^n) \arrow[uur] &
\end{tikzcd}$$

Figure 2: An iteration of the basic (left) and restructured (right) MP algorithm with proposals, $\tilde{\boldsymbol\theta}^{1:n}$, $\pi(\tilde{\boldsymbol\theta}^i)$ calculated in parallel forming a stationary distribution, I, from which k samples are drawn. The target distribution π is factored into $\pi_1, \ldots, \pi_l$ for the restructured version.

idea of improving results while fixing computation time is analogous, but all data entries are considered in this work making it orthogonal. Delayed Acceptance introduced by Banterle et al. [3] is similar in that rejection is possible after partial computation at the expense of increased variance. Zhao et al. [12] propose middleware with a heuristic and a design that allows explicit caching control for distributed storage. Since this paper focuses on processor level performance, the opposite approach of algorithmic modification is needed to leverage caching hardware.

3 Hiding Latency with Useful Computation

As long as the target density evaluation dominates execution, it seems reasonable to assume that the MC algorithm with $\rho = 0$, and the MP algorithm, with the latter being preferred since $\rho > 0$ in practice, should scale. However, cores compete for memory bandwidth, assuming a sufficiently large working set. Data fetched by one of the cores will *potentially* be reused by others. This happens by chance since there are no synchronization points, and as the span of data entries accessed at a given time exceeds cache capacity, performance degrades.

The posterior from Section 2.1 can be factored and the algorithms can be restructured as shown by Figures 1 and 2. This assures data is loaded exactly once from memory since each core operates on a partition. The main advantage is that the choice of n is independent from l allowing to introduce work during otherwise stalled cycles and to increase ILP.

4 Results

To compare scalability, performance, expressed as average number of samples per second, in function of the number of cores, l, was measured. The test system had an Intel E5-2690v2 processor with 10 cores and 32 GB of memory. For reproducibility and reduced variance across runs, operating frequency of the processor was fixed, operating system features to migrate memory were disabled, and software threads were affinitized one-to-one with hardware threads.

The results highlight improvements for each algorithm separately, but a comparison shows that the MP algorithm outperforms the MC algorithm since less samples are discarded.

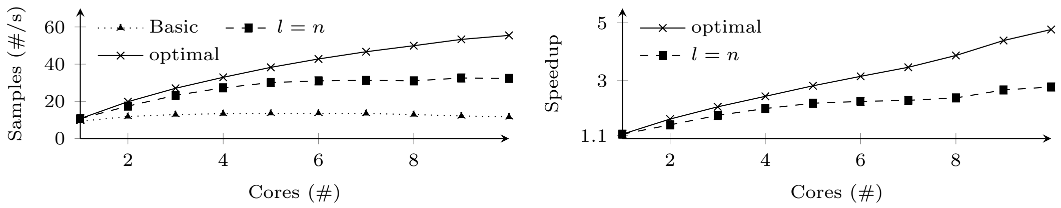

Figure 3: Performance of the MC algorithms. The basic algorithm shows no scaling, while $S_n \leq 2$ for the restructured algorithm with $l = n$ and the best configuration scales beyond that (left). Speedup (right) increases with l as the basic algorithm becomes more memory bound.

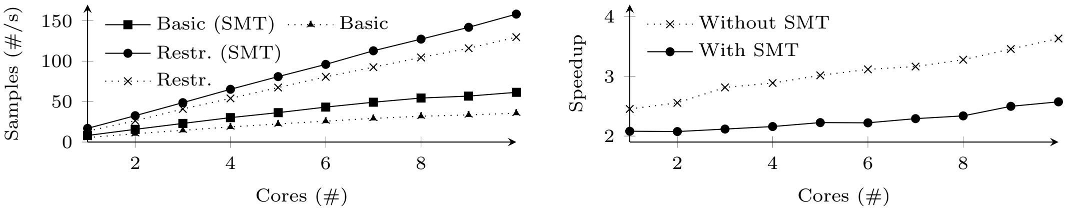

Figure 4: Performance (left) and speedup (right) of the MP algorithms with and without SMT. The restructured algorithm scales better than the basic algorithm. SMT is relatively less beneficial for the restructured algorithm. Speedup is higher without SMT as the restructured algorithm hides relatively more memory latency.

Figure 3 shows performance of the MC algorithms in function of l. Since $\rho = 0.5$, $S_n \leq 2$ for the basic algorithm but scaling is far worse due to memory bandwidth contention. The restructured algorithm performs better. If $l = n$ then S_n reaches 2. The optimal configuration scales beyond that since n is reduced. Figure 3 also relates the performance of the restructured algorithm with both configurations. Speedup here is different from S_n in that the baseline is the basic algorithm and each data point reflects performance improvements while fixing l. The optimal for n is derived from the top of Figure 5.

Figure 4 compares performance of the MP algorithms. The basic algorithm is more memory bound since, in relative terms, it benefits more from SMT than the restructured algorithm. The number of proposals, n, for the basic algorithm was determined by the number of hardware threads. Again, performance does not scale linearly. With the restructured algorithm, n need not be equal to l and is derived from the bottom of Figure 5. Note that the optimal n is *higher* than the number of hardware threads, as this hides latency caused by data being loaded.

5 Conclusion and Future Work

It is well known that the memory subsystem in contemporary systems plays a key role in performance. Using the presented method, introducing useful computations, stalls are hidden resulting in a significant speedup. This has been demonstrated in the case of Bayesian inference with two MCMC algorithms. The optimal for the MC algorithm and the MP algorithm was a reduction in the number of chains and more than twice as many proposals as cores respectively.

Future work includes exploring effectiveness of restructuring with likelihoods that cannot be factored (requiring to at least double the amount of work per core) and automatically finding the optimal configuration during the initial phase making the solution more practical. The benefits for other algorithms, where computation is interleaved fetching subsequent data elements will

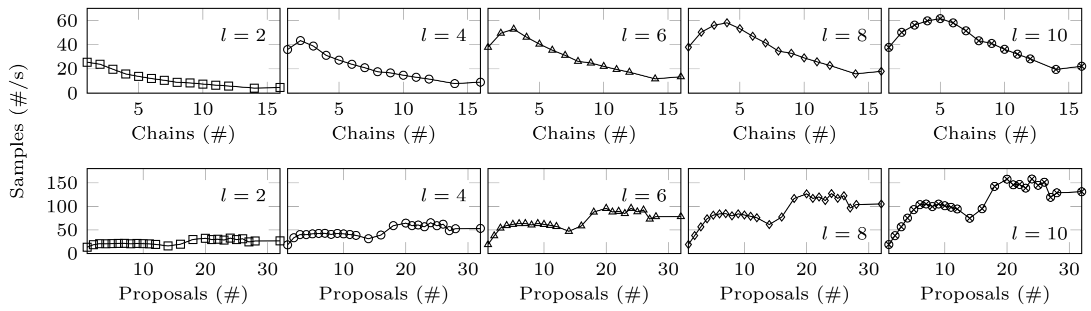

Figure 5: Performance of the restructured MC (top) and MP (bottom) algorithm in function of n. The optimal n for the MC and MP algorithm is between 2 and 5 and around 24 respect.

also be studied. Examples include model selection and ensemble models.

6 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.

References

[1] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. of the April 18-20, 1967, Spring Joint Computer Conf.*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[2] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

[3] Marco Banterle, Clara Grazian, and Christian P Robert. Accelerating Metropolis-Hastings algorithms: Delayed acceptance with prefetching. *arXiv*, page 20, 2014.

[4] Ben Calderhead. A general construction for parallelizing Metropolis-Hastings algorithms. *Proc. of the National Academy of Sciences of the United States of America*, 111(49):17408–13, 2014.

[5] Anoop Korattikara, Yutian Chen, and Max Welling. Austerity in MCMC Land: Cutting the Metropolis-Hastings Budget. *Proc. of the 30th Int. Conf. on Machine Learning*, 32:1–23, 2013.

[6] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* 2012.

[7] Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67, 2016.

[8] Christian P. Robert. The Metropolis-Hastings algorithm. (Mcmc):1–15, 2015.

[9] Steven L. Scott, Alexander W. Blocker, Fernando V Bonassi, Hugh A Chipman, Edward I George, and Robert E. McCulloch. Bayes and big data: the consensus Monte Carlo algorithm. *Int. Journal of Management Science and Engineering Management*, 11(2):78–88, 2016.

[10] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):6576, 2009.

[11] Wm. a. Wulf and Sally a. McKee. Hitting the memory wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News*, 23:20–24, 1995.

[12] Dongfang Zhao, Kan Qiao, and Ioan Raicu. Towards cost-effective and high-performance caching middleware for distributed systems. *Int. Journal of Big Data Intell.*, 3(2):92, 2016.