Viability of HPTN
A domain specific language for Tensor Networks.

Rahul Manavalan
Masters student at the Chair for Scientific Computing
Technical University of Munich
rahul.manavalan@tum.de

***Abstract*—There is increasing need for more performant implementations of Tensor Network algorithms. Recent algorithmic advances in dense tensor contraction can be exploited to further the efficiency of existing implementations of Tensor Networks. In this work,the question of viability of a high performance DSL for tensor networks based on a high performance tensor contraction software package is examined.**

***Index Terms*—high performance, domain specific language, tensor networks, HPTT, TCL**

I. Introduction

Tensor Networks have been successfully applied to study strongly correlated systems. They have also found prevalence in quantum information theory. As a result, efficient implementations of the algorithms in the domain of Tensor Networks is desired.

The landscape of tensor computation software [1] includes new algorithmic gains with respect to performance. The Tensor Contraction Library - TCL [2] which introduces among other things, GETT (GEMM-like Tensor-Tensor multiplication algorithm) is one such example. TCL performs tensor contraction using algorithms that are appropriate for the problem and the resources available at run time.

Utilization of this library to the kernels of Tensor Network computation could result in improved performance. It also poses questions on the efficacy of the asymptotic bounds in the tensor contractions, as prescribed by conventional complexity analysis. In this work, we seek to address these questions, while justifying the need for a new domain specific language for Tensor Networks.

II. Approach

A. Kernels of 1D Tensor Network Computations

Matrix Product States (MPS) or Tensor Train model strongly correlated atoms on a one dimensional lattice. The treatment of the 1D case for the investigation is justified, since an insignificant improvement would disqualify its use for higher dimensional ansatzae.

Subsequently, subroutines(kernels) are identified that recur in MPS algorithms such as left orthonormalization, right orthonormalization and Density Matrix Renormalization Group.

Dr.Edoardo Di Napoli, Dr.Matteo Rizzi

We reserve λ,ρ,ϑ for the left update, right update and the effective Hamiltonian update in the DMRG algorithm. Similarly π,ω denote left isometrization and the right isometrization of a tensor in the MPS train.

B. Design of Experiments

Validation of the performance improvement and verifying asymptotic complexity entails implementation of kernels. TCL offers interfaces in C,C++ and Python. An early choice of prototyping the kernels with the Python interface resulted in underwhelming performances. Consequently, all implementations were ported to C++.

The question of performance is answered by comparing the runtime of kernels implemented using TCL with corresponding implementations using flexmps. [1]

It is also of interest to determine if TCL's algorithms improve the runtime complexity of different tensor contraction orderings.[2] This is achieved by contracting the participant tensors in the kernels, in different permutation sequences and evaluating the relative discrepancy in the run time across the different libraries.

With respect to the parameters for the experiment, n denotes the bond dimensions of the Tensor Train, s the internal dimensions of the wavefunction, and d the number of transition states in the MPO.

Bearing in mind that TCL is a dense tensor contraction library, the sparsity optimizations of flexmps were turned off. Furthermore the benchmarks were realized on randomized entries for the MPS and the MPO train elements.

III. Nomenclature of kernels

A. DMRG kernels

1) UpdateL: The UpdateL kernel (Fig.1) contracts a 3d tensor block from the previous iteration with two isometrized tensors from the MPS trains and the MPO tensor. The kernel

[1]Flexmps is a fortran based library for Tensor Networks.

[2]Contraction of Tensors A,B,C can have different complexities when executed in different orders, i.e. while tensor contraction is associative, its complexity is not.

has its origins in the left pass of the DMRG algorithm.

Fig.1 reveals the enumerations for the participants of tensor contraction in the kernel. The block is symbolized as 0, the Isometry of the top MPS as 1,the MPO as 2 and the Isometry of the bottom MPS as 3.

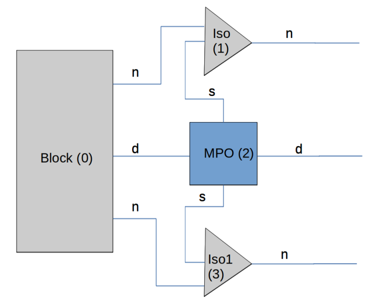

Fig. 1. λ kernel - Update left

2) UpdateR: The UpdateR kernel is symmetric to the UpdateL kernel. The notations used in the previous kernel is analogous. In addition the complexity is also similar. However, it is of interest to study this kernel, since the memory access pattern is different in comparison to an UpdateL kernel.

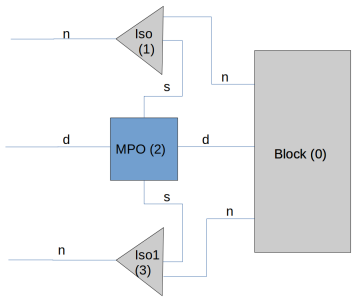

Fig. 2. ρ kernel - Update right

3) UpdateH: Occurrence of this kernel is in the update of the central tensors in the DMRG algorithm. Adhering to convention, we denote block 1 as 0, the MPS tensor 1,the MPO tensor 2 and the second block as 3.

Due to the prevalence of five indices that model bond dimensions, certain permutations in this kernel (0231 for instance), requires exponential amounts of memory to store the participant tensors. As a result, those permutations that violate the low rank approximation are excluded from analysis.

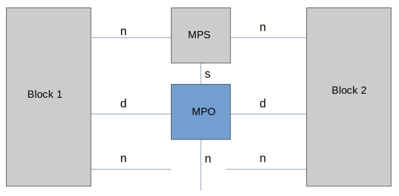

Fig. 3. ϑ kernel - Update Effective Hamiltonian

B. IsoL kernel

An element of an MPS train is isometrized using the IsoL kernel. It follows from Fig(4) that given a matrix R and a tensor A, one can reformulate this expression as an Isometry I and a residual matrix R_{i+1} using Singular Value Decomposition.

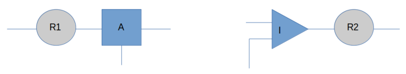

Fig. 4. π kernel - Isometrization Left

C. IsoR kernel

The IsoR kernel is a symmetric counterpart to the IsoL kernel. We include this in our investigation, as this has an alternative access pattern.

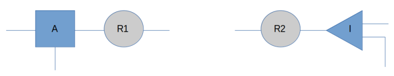

Fig. 5. ω kernel - Isometrization right

IV. Observations

A. UpdateL kernel - Runtime

Fig(6) indicates reduced runtime in the TCL implementation for the λ kernel. The flexmps implementation to the right (Fig(6)) strongly agrees with the characteristics from complexity analysis. On the other hand, the TCL implementation shows weaker agreement.

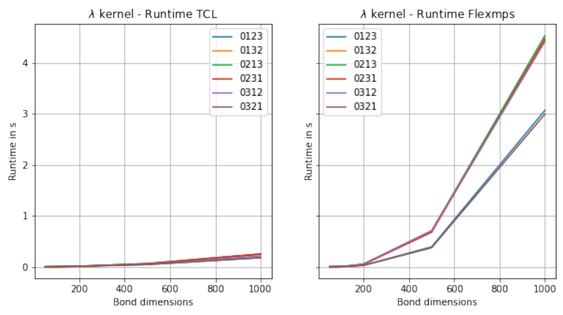

Fig. 6. Runtime comparison - λ kernel

B. UpdateR kernel - Runtime

Being symmetric to λ kernel, the observations are similar. It is interesting to note that for smaller bond dimensions, the runtime for ρ kernel is approximately half the runtime for λ kernel. However, this overhead in transposition diminishes, as the bond dimensions increases and in turn the computational costs begin to dominate.

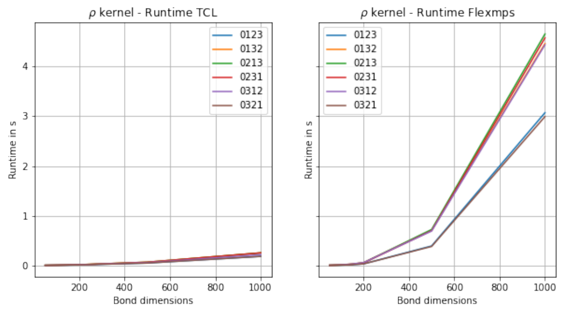

Fig. 7. Runtime comparison - ρ kernel

C. UpdateH kernel - Runtime

Besides the apparent improvement in the run time, a change in the complexity characteristics is observed. For permutation sequence 0213, the tcl implementation is much faster in relative measure with the flexmps counterpart.

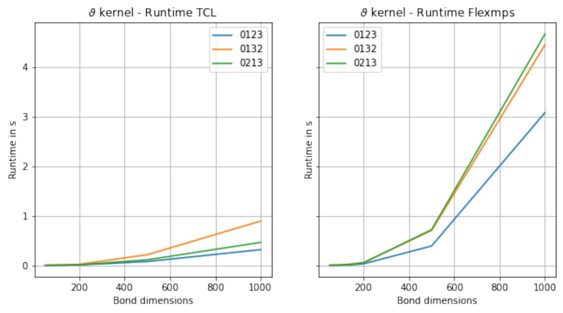

Fig. 8. Runtime comparison - ϑ kernel

D. Speedup

The λ and ρ kernels report a speedup of $\approx$ 20 for the largest bond dimension.(Fig9, Fig10) It is interesting to note that the tcl implementation changes the complexity coefficients of the algorithm as evidenced by the fact that the permutation sequence 0312 exhibits the most speedup. However further analysis reveals that despite the fact, the sequence 0123 exhibits the fastest run-time.

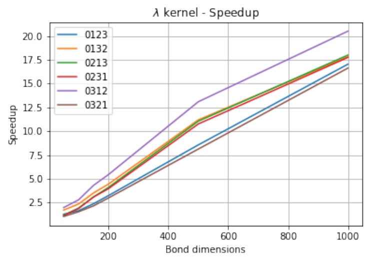

Fig. 9. Speedup - λ kernel

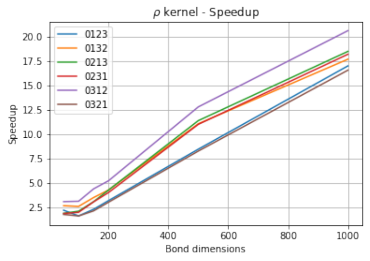

Fig. 10. Speedup - ρ kernel

The ϑ kernel reports an average speedup of $\approx$ 10 for the largest bond dimension.(Fig11)

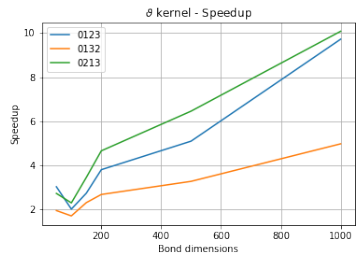

Fig. 11. Speedup - ϑ kernel

V. Conclusion

1) Initial benchmarking suggests improvements in the performance of kernel computations for relatively larger bond dimensions.
2) The use of the algorithms in TCL seem to diminish the bottleneck caused by large complexities in the kernels.

The results indicate that a reimplementation of the TN library features from flexmps using TCL is warranted.[3] However, neither TCL nor the underlying transposition library HPTT [3] is actively maintained. Therefore, it would be logical to adapt these algorithms to a more modern framework in the form of the Julia Programming Language [4].

A domain specific language has the nature of a "black box". Julia's multiple dispatch mechanism would allow exploration

[3]This would mean that flexmps needs to be ported from Fortran to C++.

of several algorithms and efficient code generation, while being oblivious to the interface. Furthermore, efficient integration to heterogeneous architecture can also be realized within the Julia ecosystem.

Lastly, porting to Julia would serve as an ideal comparison benchmark as some of the existing TN libraries such as TensorToolkit [5] and iTensors [6] already have implementations in Julia.

WHY DO WE NEED A NEW DSL FOR TN?

A DSL with symbolic representations of the elements of a Tensor Network[4] computation enjoys the following advantages.

1) Code generation is the prerogative of the compiler, ensuring that the code is always efficient.
2) Tuning the computation for the problem size and the available run-time resources is internally handled, deterring the user from ever worrying about implementations on heterogeneous architecture.
3) More significantly, the competitor libraries [5] [6] do not support this feature. As a consequence, the new DSL's existence (development) is merited.

ACKNOWLEDGMENT

Special thanks is directed towards Dr. Edoardo Napoli and Prof. Dr. Matteo Rizzi for their unwavering faith and endless patience. I would also like to thank Dr.Ivo Kabadshow for his guidance during the JSC-GSP 2021.

REFERENCES

[1] C. Psarras, L. Karlsson, and P. Bientinesi, "The landscape of software for tensor computations," *CoRR*, vol. abs/2103.13756, 2021. [Online]. Available: https://arxiv.org/abs/2103.13756

[2] E. Peise and P. Bientinesi, "The elaps framework: Experimental linear algebra performance studies," *The International Journal of High Performance Computing Applications*, vol. 33, no. 2, pp. 353–365, 2019. [Online]. Available: https://doi.org/10.1177/1094342018763042

[3] P. Springer, T. Su, and P. Bientinesi, "HPTT: A high-performance tensor transposition C++ library," *CoRR*, vol. abs/1704.04374, 2017. [Online]. Available: http://arxiv.org/abs/1704.04374

[4] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *CoRR*, vol. abs/1411.1607, 2014. [Online]. Available: http://arxiv.org/abs/1411.1607

[5] J. Hageman, "Tensortoolkit.jl - a julia package."

[6] M. Fishman, S. R. White, and E. M. Stoudenmire, "The ITensor software library for tensor network calculations," 2020.

[4] An analogue of ModelingToolkit.jl. See https://mtk.sciml.ai/stable/