This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2020-6214C

Parameter Sensitivity Analysis of the SparTen High Performance Sparse Tensor Decomposition Software

Jeremy M. Myers
College of William and Mary
Sandia National Laboratories
jermyer@sandia.gov

Daniel M. Dunlavy
Sandia National Laboratories
dmdunla@sandia.gov

Keita Teranishi
Sandia National Laboratories
knteran@sandia.gov

D.S. Hollman
Sandia National Laboratories
dshollm@sandia.gov

***Abstract*—Tensor decomposition models play an increasingly important role in modern data science applications. One problem of particular interest is fitting a low-rank Canonical Polyadic (CP) tensor decomposition model when the tensor has sparse structure and the tensor elements are nonnegative count data. SparTen is a high-performance C++ library which computes a low-rank decomposition using different solvers: a first-order quasi-Newton or a second-order damped Newton method, along with the appropriate choice of runtime parameters. Since default parameters in SparTen are tuned to experimental results in prior published work on a single real-world dataset conducted using MATLAB implementations of these methods, it remains unclear if the parameter defaults in SparTen are appropriate for general tensor data. Furthermore, it is unknown how sensitive algorithm convergence is to changes in the input parameter values. This report addresses these unresolved issues with large-scale experimentation on three benchmark tensor data sets. Experiments were conducted on several different CPU architectures and replicated with many initial states to establish generalized profiles of algorithm convergence behavior.**

***Index Terms*—tensor decomposition, Poisson factorization, Kokkos, Newton optimization**

I. Introduction

The Canonical Polyadic (CP) tensor decomposition model has garnered attention as a tool for extracting useful information from high dimensional data across a wide range of applications [1]–[5]. Recently, Hansen *et al.* developed two highly-parallelizable Newton-based methods for low-rank tensor factorizations on Poisson count data in [6], one a first-order quasi-Newton method (PQNR) and another a second-order damped Newton method (PDNR). They were first implemented in MATLAB Tensor Toolbox [7] as the function `cp_apr`, referring to this approach as computing a CP decomposition using Alternating Poisson Regression (i.e., CP-APR). These methods fit a reduced-rank CP model to count data, assuming a Poisson error distribution. PDNR and PQNR are implemented in SparTen,[1] a high-performance C++ library of CP-APR solvers for sparse tensors. SparTen improves on the MATLAB implementation to provide efficient execution for large, sparse tensor decompositions, exploiting the Kokkos hardware abstraction library [8] to harness parallelism on diverse HPC platforms, including x86-multicore, ARM, and GPU computer architectures.

[1]SparTen is a portmanteau word derived from *Sparse* and *Tensor*. The SparTen code is available at `http://gitlab.com/tensors/sparten`.

SparTen contains many algorithmic parameters for controlling the optimization subroutines comprising PDNR and PQNR. To date, only anecdotal evidence exists for how best to tune the algorithms. Parameter defaults in SparTen were chosen according to previous results using the MATLAB implementations described by Hansen *et al.* [6]. However, their analysis was limited to a single real-world dataset, and thus may not be optimal for computing decompositions of more general tensor data. Furthermore, it is unknown how the initial guess to a solution affects convergence, since SparTen methods may converge slowly—or worse, stagnate—on real data if the initial state is far from a solution. And, lastly, the average impact of input parameters on algorithm convergence is unclear.

To address these unknowns, we present the results of numerical experiments to assess the sensitivity of software parameters on algorithm convergence for a range of values with benchmark tensor problems. Every experiment was replicated with 30 randomly chosen initial guesses on three diverse computer architectures to aid statistical interpretation. With our results, we (1) provide new results that offer a realistic picture of algorithm convergence under reasonable resource constraints, (2) establish practical bounds on parameters such that, if set at or beyond these values, convergence is unlikely, and (3) identify areas of performance degradation and convergence toward qualitatively different results owing to parameter sensitivities.

We limited our study to multicore CPU architectures only, using OpenMP [9] to manage the parallel computations across threads/cores. Although SparTen, through Kokkos, can leverage other execution backends—e.g., NVIDIA's CUDA framework for GPU computation—we focus solely on diversity in CPU architectures in this work.

This paper is structured as follows. Section II summarizes basic tensor notation and details. Section III describes the hardware environment, test data, and experimental design of the sensitivity analysis. Section IV provides detailed results of the sensitivity analyses. Section V offers concluding remarks and lays out future work.

II. Background

We briefly describe below the problem we are addressing in this report; for a detailed description of CP decomposition

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

algorithms implemented in SparTen, refer to the descriptions in Hansen *et al.* [6].

An N-way data tensor $\mathcal{X}$ has dimension sizes $I_1 \times I_2 \times \cdots \times I_N$. We wish to fit a reduced-dimension tensor model, $\mathcal{M}$, to $\mathcal{X}$. The R-component Canonical Polyadic (CP) decomposition is given as follows:

$$\mathcal{X} \approx \mathcal{M} = [\![\boldsymbol{\lambda}; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \ldots \circ \mathbf{a}_r^{(N)}, \quad (1)$$

where $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_R]$ is a scaling vector, $\mathbf{a}_r^{(n)}$ represents the r-th column of the factor matrix $\mathbf{A}^{(n)}$ of size $I_n \times R$, and $\circ$ is the vector outer product. We refer to the operator $[\![\cdot]\!]$ as a Kruskal operator, and the tensor $\mathcal{M}$, with its specific multilinear model form, as a Kruskal tensor in (1). See [1] for more details regarding these definitions.

SparTen addresses the special case when the elements of $\mathcal{X}$ are nonnegative counts. Assuming the entries in $\mathcal{X}$ follow a Poisson distribution with multilinear parameters, the low-rank CP decomposition in (1) can be computed using the CP-APR methods, PDNR and PQNR, introduced by Hansen *et al.* [6].

III. Methods

In this section, we describe the hardware platforms, data, and SparTen algorithm parameters used in our experiments.

A. Hardware Platforms

We used diverse computer architectures to perform our experiments, with hardware and compiler specifications detailed in Table I. Intel 1–4 are production clusters with hundreds to thousands of nodes, whereas ARM and IBM clusters are advanced architecture research testbeds with tens of nodes each. We employed the maximum number of OpenMP threads available per node from each platform to maximize throughput and configured the maximum wall-clock limit as 12:00 hours for all experiments. All parallelism was solely across threads on a single node. The GNU compiler, `gcc`, was used, with `-O3` optimization and Kokkos architecture-specific flags enabled.

TABLE I
Hardware characteristics and software environment of the clusters in this paper. *Threads* and *RAM (GB)* are per node.

Arch	Processor	Threads	RAM (GB)	GCC
ARM	ThunderX2	256	255	7.2.0
IBM	Newell Power9	80	319	7.2.0
Intel 1	Sandy Bridge	16	64	8.2.1
Intel 2	Broadwell	72	128	8.2.1
Intel 3	Sandy Bridge	16	64	8.2.1
Intel 4	Sandy Bridge	32	64	8.2.1

B. Data

We conducted experiments using sparse tensors of count data from the FROSTT collection [10]. Specifically, we chose the three datasets listed in Table II to account for size, dimensionality, and density (i.e., the ratio of nonzero entries to the total number of elements in the tensor). Throughout the discussion below, we refer to the data using the short names listed in the table.

TABLE II
Sparse tensor datasets from the FROSTT collection.

FROSTT Name (short name)	Dimensions	Density
chicago-crime-comm (*chicago*)	$(6186, 24, 77, 32)$	1.5×10^{-02}
lbnl-network (*lbnl*)	$(1605, 4198, 1631, 4209, 868131)$	4.2×10^{-14}
nell-2 (*nell*)	$(12092, 9184, 28818)$	2.4×10^{-05}

C. Software Parameter Definitions & Experimental Ranges

PQNR and PDNR are composed of standard techniques in the numerical optimization literature. Specifically, for each tensor mode, the Newton optimization computes the gradient and Hessian matrix. Then, the inverse Hessian is approximated to compute a search direction and an Armijo backtracking line search is used to compute the Newton step. How the inverse Hessian is approximated differentiates PDNR and PQNR. PDNR shifts the eigenvalues by a damping factor μ to guarantee the Hessian matrix is semi-positive definite, and solves the resulting linear system exactly. PQNR approximates the inverse Hessian directly with a limited-memory BFGS (L-BFGS) approach, computed with a small number of update pairs. Since the algorithm parameters analyzed here are those presented in several equations and algorithms in [6], we defer to that paper for specific details.

A brief description of each software parameter, the default value in SparTen, and the experimental ranges tested in these experiments are given in Table III. We note that the stability parameters used to safeguard against numerical errors—e.g., offset tolerances to avoid divide-by-zero floating point errors—do not appear in the corresponding Matlab Tensor Toolbox method `cp_apr`.

D. Experiments

An individual experiment is a job j on platform m solving a PDNR/PQNR row subproblem for dataset d with SparTen solver s, parameter p, parameter value v, and random initialization r; all remaining software parameters are fixed at the default values listed in Table III. Certain experiments denoted with a dagger† were run only on Intel hardware due to limited resources associated with the other architectures; this accounts for the larger number of experiments reported for these platforms. We conducted tests on these values to provide better resolution of the impact of the parameter where nearby values—i.e., on the bounds of the test range—contained uncertainty in the results. Furthermore, we split up the experiments across the Intel platforms by parameter, running the full set of experiments across all parameter values and all random initializations on a single platform. The superscripts denoted for each parameter in the table denote the Intel platform number specified in Table I. Since we report only the number of function evaluations and outer iterations in our results, we expect that running our experiments in this way has produced valid results.

TABLE III
SparTen software parameter descriptions and values used in our experiments.

Parameter	Description	Default	Values Used in Experiments
`max_outer_iterations`^{1}	Maximum number of outer iterations	10^5	1, 2, 4, 8, 16, 32, 64, 128, 256, 512
`max_inner_iterations`^{1}	Maximum number of inner iterations	20	20, 40, 80, 160
`max_backtrack_steps`^{2}	Maximum number of backtracking steps in line search	10	$1^\dagger$, 2, 4, 8, 10, $12^\dagger$, 16
`min_step_size`^{2}	Tolerance for nonzero line search step length	10^{-7}	$10^{-1\dagger}$, 10^{-3}, 10^{-7}, $10^{-15\dagger}$
`step_reduction_factor`^{2}	Factor to reduce line search step size	0.5	0.1, $0.3^\dagger$, 0.5, $0.7^\dagger$, 0.9
`suff_decrease_tol`^{2}	Tolerance to ensure decreases in the objective function	10^{-4}	10^{-2}, 10^{-4}, $10^{-8\dagger}$, $10^{-12\dagger}$
`mu_initial`^{3}	Initial value of damping parameter	10^{-5}	10^{-2}, 10^{-5}, 10^{-8}
`damping_increase_factor`^{3}	Scalar value to increase damping parameter in next iterate	3.5	1.5, $2.5^\dagger$, 3.5, $4.5^\dagger$, 5.5
`damping_decrease_factor`^{3}	Scalar value to decrease damping parameter in next iterate	2/7	0.1, 2/7, $0.3^\dagger$, 0.5, $0.7^\dagger$, 0.9
`damping_increase_tol`^{3}	Tolerance to increase the damping parameter	0.25	0.1, 0.25, 0.495
`damping_decrease_tol`^{3}	Tolerance to decrease the damping parameter	0.75	0.505, 0.75, 0.9
`size_LBFGS`^{3}	Number of limited-BFGS update pairs	3	1, 2, 3, 4, 5, 10, 15, 20
`eps_div_zero_grad`^{4}	Guard against divide-by-zero computing gradient and Hessian	10^{-10}	10^{-5}, $10^{-8\dagger}$, 10^{-10}, $10^{-12\dagger}$, 10^{-15}
`log_zero_safeguard`^{4}	Tolerance to avoid computing log(0)	10^{-16}	$10^{-4\dagger}$, 10^{-8}, $10^{-12\dagger}$, 10^{-16}, $10^{-24\dagger}$, 10^{-32}
`eps_active_set`^{4}	Tolerance defining active (nonzero) variables	PDNR: 10^{-3}	10^{-1}, 10^{-3}, $10^{-5\dagger}$,$10^{-8\dagger}$
		PQNR: 10^{-8}	10^{-1}, 10^{-3}, $10^{-5\dagger}$,$10^{-8\dagger}$

$^{1-4}$Intel platform used for experiments; †values evaluated on Intel platform only

In all experiments, we fit a 5-component CP decomposition using a tolerance of 10^{-4} (i.e., the value of τ in Equation (20) in [6], the violation of the Karush-Kuhn-Tucker (KKT) conditions, used as the stopping criterion for the methods we explore here). Computation of a CP decomposition using PDNR or PQNR in SparTen requires an initial guess of the model parameters—i.e., initial values for $\mathcal{M}$ in (1)—drawn from a uniform distribution in the range $[0,1]$. As such, all experiments were replicated using 30 random initializations. We report results on the amount of computation required for convergence (i.e., the number of evaluations of the the negative log likelihood objective function, $f(\mathcal{M})$, defined in Equation (4) of [6]) and the quality of the solution (i.e., the value of the negative log likelihood objective function). As each of our experiments consists of 30 replicates (i.e., 30 random initializations) across three CPU architectures, we report sample means and 95% confidence intervals (as defined in [11]) when presenting statistical trends in the results.

IV. Results

In this section we analyze the results of the parameter sensitivity experiments and describe the statistical relationships between the convergence properties of the PDNR and PQNR methods and their input parameters.

In total, 21,960 unique experiments were planned, accounting for running PDNR and PQNR with random initializations across all parameter value ranges on the various hardware architectures described in Sections III-A and III-C. An experiment *converged* if the final KKT violation is less than the value of $\tau = 10^{-4}$; an experiment reached *maximum iterations* if the number of outer iterations exceeded the maximum limit (i.e., `max_outer_iterations`) and did not converge; an experiment was *canceled* if it exceeded the wall-clock limit (i.e., SparTen neither converged to a solution nor reached maximum number of outer iterations within 12 hours); and an experiment was *missing* if it did not run due to a failure of the system to launch the experiment or other system issue. Of the planned experiments, we collected data from 16,139 experiments.

Table IV presents the number of experiments planned (*plan.*) as defined above and the number of planned experiments where data was collected (i.e., planned minus missing). For those collected (*coll.*), the table shows the percentage of experiments that were canceled (*canc.*), converged (*conv.*), or exceeded the maximum iterations (*max. iter.*). We note that the most complete set of experiment results were obtained on the Intel platforms. Although there are many missing experiment results (*miss.*) for the IBM and ARM platforms,

TABLE IV
Experiments run on the different datasets and hardware platforms.

CPU	Solver	Data	Plan.	Collect.	Canc.	Conv.	Max. Iter.	Miss.
ARM	PDNR	*chicago*	1110	1110	4.8%	82.2%	13.0%	0.0%
		lbnl	1110	1110	10.5%	76.5%	13.0%	0.0%
		nell	1110	390	5.4%	39.2%	55.4%	64.9%
	PQNR	*chicago*	990	281	0.0%	55.5%	44.5	71.6%
		lbnl	990	237	0.0%	0.0%	100.0%	76.1%
		nell	990	390	23.3%	0.3%	76.4	60.6%
IBM	PDNR	*chicago*	1110	855	5.4%	77.8%	16.8%	23.0%
		lbnl	1110	692	11.3%	73.3%	15.4%	37.7%
		nell	1110	424	51.2%	12.0%	36.8%	61.8%
	PQNR	*chicago*	990	676	10.2%	76.3%	13.5%	31.7%
		lbnl	990	293	61.8%	0.0%	38.2%	70.4%
		nell	990	481	31.0%	6.6%	62.4%	51.4%
Intel	PDNR	*chicago*	1680	1673	5.0%	86.4%	8.6%	0.4%
		lbnl	1680	1663	11.0%	80.6%	8.4%	1.0%
		nell	1680	1643	44.7%	42.2%	13.1%	2.2%
	PQNR	*chicago*	1440	1434	12.1%	78.6%	9.3%	0.4%
		lbnl	1440	1363	78.0%	0.0%	22.0%	5.3%
		nell	1440	1424	68.8%	10.1%	21.1%	1.1%

we attempt to identify patterns in the data we collected if there is strong evidence to support our claims. We note that a few parameters (`eps_active_set`, `min_step_size`, `suff_decrease_tol`, `damping_increase_tol`, `damping_decrease_tol`) showed no statistically significant differences across the range of input values used in the experiments. We conjecture that we did not find values where the parameters display sensitivities in the chosen tensor problems, thus it remains unclear if this behavior holds in general.

A. General Convergence Results on Real-World Data

As discussed in Section I, applying PDNR and PQNR to real-world data has been explored previously in the literature only for a single problem. From Table IV, we observed that PQNR is canceled more than PDNR in the allotted time across datasets and CPU platforms. This confirms our intuition, since it is a classical result in iterative methods that damped Newton methods converge quadratically, in comparison to quasi-Newton methods, which converge superlinearly. Specifically, PQNR calls the objective function 2.7 times more than PDNR on average on the *chicago* data and fails to converge for any experiment on *lbnl* data across all hardware platforms. By contrast, PDNR converges in 86% of *lbnl* experiments across platforms when only 32 outer iterations are allowed.

B. Sensitivity of Convergence and Solution Behavior

There are certain ranges of parameter values that lead to good or bad convergence behaviors in general. This is illustrated in Figures 1–3, where parameter values and random initializations are depicted across the horizontal and vertical axes, respectively. These heatmaps display total objective function evaluations, where solid columns of a single shade indicate the same convergence behavior across all 30 random initializations. Green shades are consistent with *converged* experiments. Vertical bands not shaded green identify values that may impact algorithm performance, due either to iteration constraints (blue hues) or excessive computations corresponding to slow convergence or stagnation (red hues). Hatches denote non-converging exit status. Grey represents *missing* data, i.e., experiments that were planned but never conducted due to resource limitations—e.g., dequeued by the cluster administrator—or a system failure. Nearly solid column lines of the same shade indicate similar behavior, but also that there is some sensitivity of those parameter values to the initial starting point of the iterative methods.

In several cases, there is a tendency to time-out at one or both bounds of the test ranges for both solvers. The behavior of numerical stability parameters `eps_div_zero_grad` and `log_zero_safeguard` was consistent across combinations of solver, data, and CPU hardware. When `eps_div_zero_grad` is large, gradient directions that do not lead to objective function improvements may be scaled the same as gradient directions that do lead to such improvements. Furthermore, the corresponding eigenvalues of the Hessian matrix are amplified and Hessian information may be lost when determining the next iterate. For example, PDNR loses Hessian information as `eps_div_zero_grad` increases on *chicago* data; PDNR rarely converges and PQNR never converges when this parameter is relatively large—i.e. 10^{-5}. Moreover, both algorithms are sensitive to the parameter's lower bound, as small values may be insufficient to avoid an ill-conditioned Hessian matrix. In either case, additional iterations follow to correct errors incurred by `eps_div_zero_grad` values, large and small.

PDNR typically does not converge for large `log_zero_safeguard` values on large tensor problems. This parameter sets a nonzero offset in logarithm calculations to avoid explicitly computing $\log(0)$. High precision in logarithm computations tends to ensure the objective function is minimized accurately. When the value is too large, the calculated logarithm may be too small, and more backtracking steps are required to sufficiently decrease the objective function in the line search routine, making time-outs more likely. On the other hand, the effect of the parameter on convergence is indistinguishable for values smaller than 10^{-8} across all experiments.

This effect—when convergence behavior is similar for values set within sensitivity constraints—is common among several algorithm parameters corresponding to the different numerical optimization subroutines that comprise PDNR and PQNR. Two examples are `damping_increase_factor` and `damping_decrease_factor`, which control updates to the PDNR Hessian matrix damping parameter μ. SparTen rarely converges when the former is set too low (1.5); the likely effect is that the updated damping factor is insufficient to guarantee a well-conditioned Hessian and too many unimportant directions are considered when computing the search direction. Above the 1.5 bound, the cost in objective function calls does not change significantly. The PQNR-specific parameter, `size_LBFGS`, behaves similarly; the only observable difference occurs when the update size is 1, using only the current iterate in the BFGS update.

Other parameters show meaningful differences in cost, defined in terms of the number of function evaluations required before convergence is achieved, when varied. Hansen *et al.* predict in [6] that when the damping parameter μ is set too large, a loss of Hessian information follows, which impacts convergence. For example, when `mu_initial` is large, the computational cost grows dramatically and time-outs become more likely, since the initial step length will at first be very small in every outer iteration and useful Hessian information is discarded in early stages of the inner loop solves. Convergence is most likely for a large, but not too-large, value, i.e., `mu_initial` $= 10^{-5}$. Cost grows 177.2% on *lbnl* and nearly doubles (+92.2%) on *chicago* as `mu_initial` grows from 10^{-5} to 10^{-2}. It is important to note in the former case that this cost is skewed by one experiment that converged after nearly 42,000 outer iterations, in comparison to 1,300 for the other parameter values on average, illustrated in Figure 4, where the *x*-axis is truncated to highlight the differences in total cost. Smaller values (i.e, 10^{-8}) seem to perform better for *chicago*, the smallest, densest problem and larger values (i.e., 10^{-5}) tend to perform better for large, sparse problems.

Allowing many backtracking steps during the line search

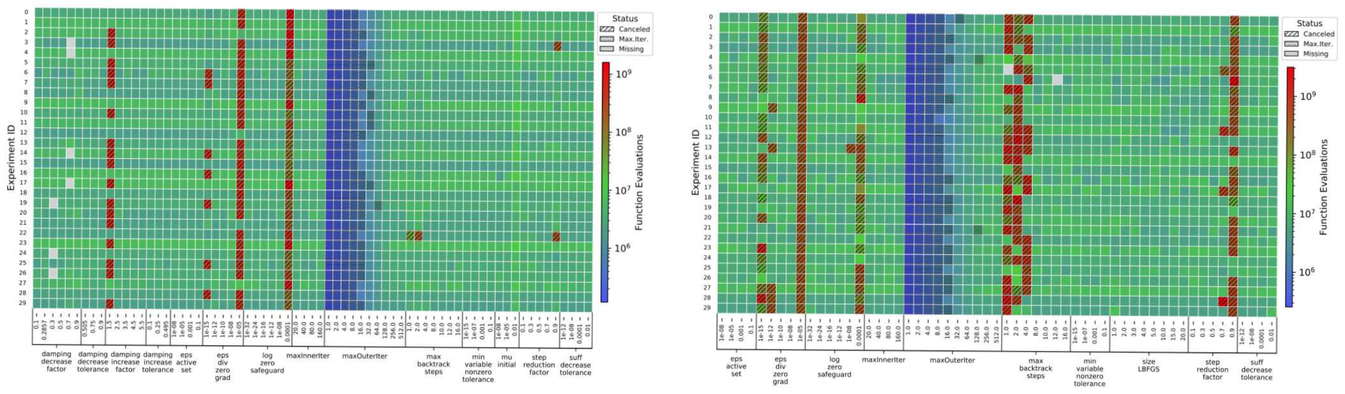

Fig. 1. Total function evaluations computed solving *chicago* on Intel architecture. Left: PDNR, Right: PQNR.

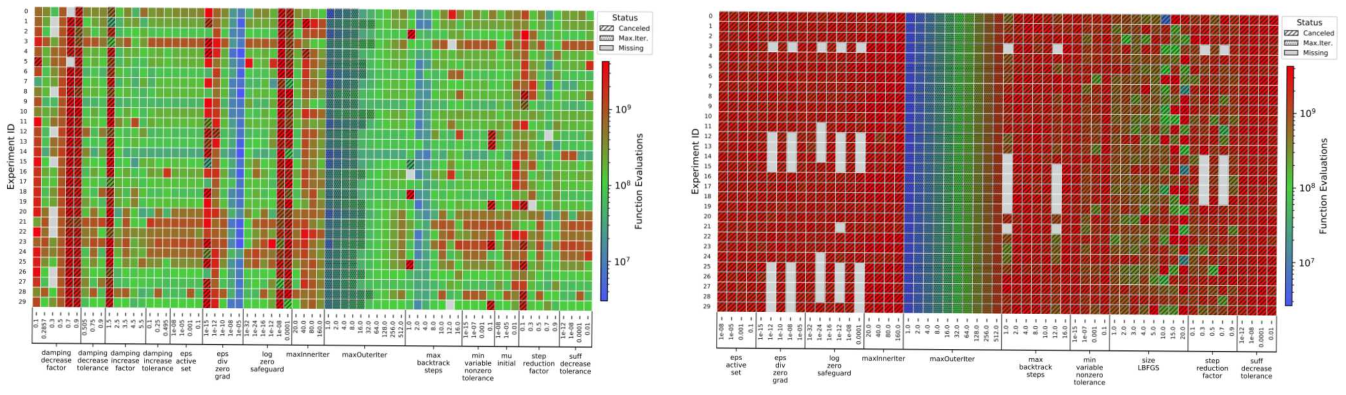

Fig. 2. Total function evaluations solving *lbnl* on Intel architecture. Left: PDNR, Right: PQNR.

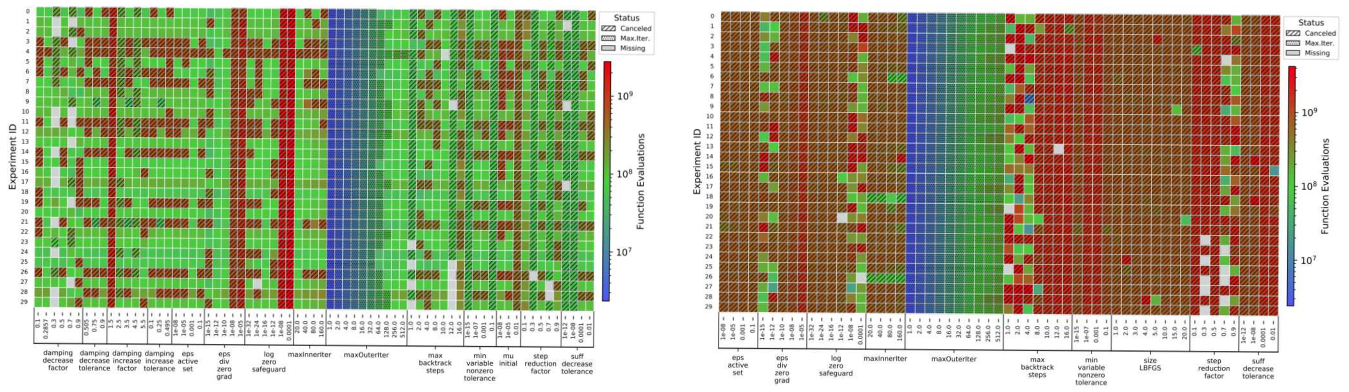

Fig. 3. Total function evaluations solving *nell* on Intel architecture. Left: PDNR, Right: PQNR.

may cause PDNR to waste effort; however, PQNR appears to perform better, in general, with more steps. PDNR is sensitive to the number of backtracking steps on *chicago*: average work performed is less when the maximum number of allowed steps is large and more work is performed when the number of steps is small. On *lbnl*—the sparsest tensor problem considered—PDNR performs better with fewer backtracking steps (see Figure 5). The average cost incurred by PQNR decreases as `max_backtrack_steps` increases.

The line search parameter `step_reduction_factor` is used to reduce the line search step size between iterations. On a large, sparse tensor problem, increasing this parameter may accelerate convergence. On the other hand, a small value makes convergence less certain. Figure 6 illustrates this behavior on the *lbnl* data: the average total cost decreased by 77% as `step_reduction_factor` increased from 0.1 to 0.5 (SparTen default) and decreased another 28% from 0.5 to 0.9. On *nell* data, PQNR *only* converged for large values (0.7, 0.9).

Parameter sensitivities affect not only convergence behavior, but may also produce qualitatively different results. Figure 7 illustrates the effect where large `eps_div_zero_grad`—and consequently, small step length—minimizes calls to the ob-

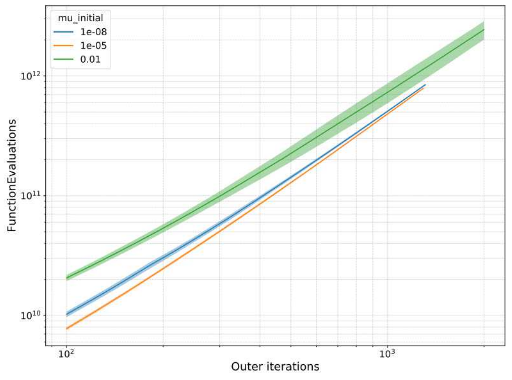

Fig. 4. Varying `mu_initial` solving *lbnl*, with mean function evaluations and 95% CI. PDNR damps out Hessian information and is more prone to time-outs when `mu_initial` is large.

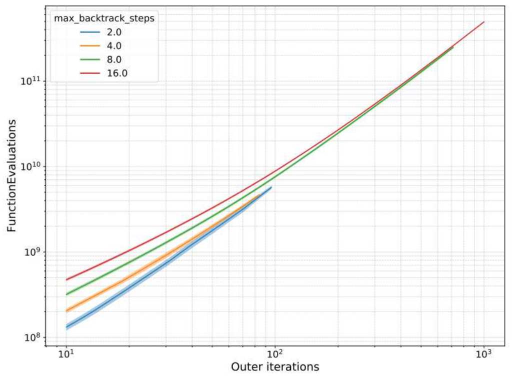

Fig. 5. `max_backtrack_steps` becomes more expensive as the number of steps grows on *lbnl*. The *x*-axis is truncated at 1,000 outer iterations, but 6 experiments with 16 maximum backtracking steps required more iterations.

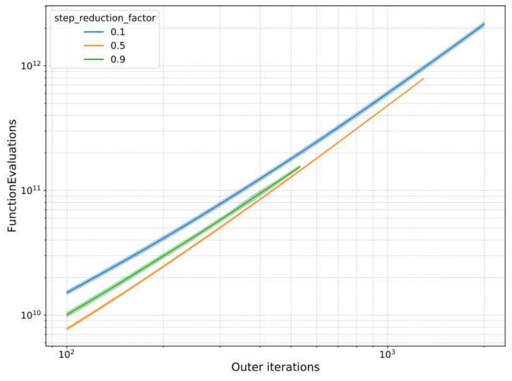

Fig. 6. `step_reduction_factor` may accelerate convergence on large, sparse tensor data (*lbnl*). The *x*-axis is truncated at 2,000 outer iterations; 3 experiments with reduction factor 0.1 continued for over 4,200 outer iterations.

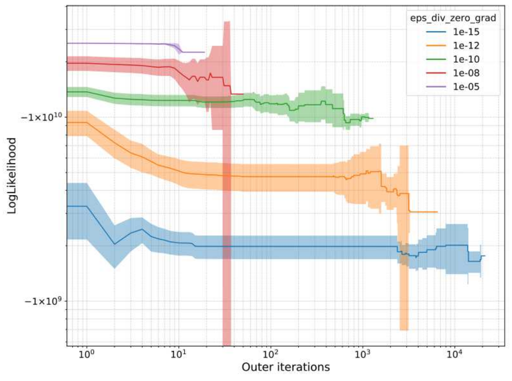

Fig. 7. Mean objective function values with 95% confidence interval, varying `eps_div_zero_grad`, PDNR on *lbnl*.

jective function *and* results in minimal objective function value (higher values on the *y*-axis correspond to a smaller negative LogLikelihood). Most striking is that larger `eps_div_zero_grad` decreases the objective function more than an order of magnitude. This result was collected from 79 of 90 planned PDNR experiments on *lbnl*, and thus we consider this interesting effect worthy of further investigation.

V. Conclusions

Using results from more than 16,000 numerical experiments on several hardware platforms, we presented experimental results that expand our understanding of average PDNR and PQNR convergence on real-world tensor problems. We have shown that when using PQNR to compute large tensor decompositions convergence is less-likely under reasonable resource constraints. We have shown that some software parameters are sensitive to bounds on values. Further, we showed that varying several parameters can dramatically impact algorithm performance, and in some cases, may produce qualitatively different results.

Future work may address the issue of stagnation in Newton optimization methods for CP decompositions. We showed examples where the solver converged to a solution slowly but within the allotted time of 12 hours. For those experiments that timed out, it is unknown whether SparTen would eventually converge to a solution or stagnate without making progress. We anticipate that stagnation could be determined if the objective function values converge to a statistical steady state without satisfying the convergence criterion. Future development of SparTen may include dynamic updates to algorithm parameters based on local convergence information. Lastly, future experiments could explore coupled sensitivities among algorithm parameters, as this work was limited to single parameter, univariate analyses.

Acknowledgement

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

[1] T. Kolda and B. Bader, “Tensor decompositions and applications,” *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[2] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. PHAN, “Tensor decompositions for signal processing applications: From two-way to multiway component analysis,” *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[3] F. L. Hitchcock, “The expression of a tensor or a polyadic as a sum of products,” *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, 1927.

[4] J. Carroll and J. Chang, “Analysis of individual differences in multidimensional scaling via an n-way generalization of “eckart-young” decomposition,” *Psychometrika*, vol. 35, pp. 283–319, 1970.

[5] R. A. Harshman, “Foundations of the PARAFAC procedure: models and conditions for an “explanatory” multi-modal factor analysis,” *UCLA Working Papers in Phonetics*, vol. 16, no. 1, pp. 84–84, 1970.

[6] S. Hansen, T. Plantenga, and T. G. Kolda, “Newton-based optimization for Kullback-Leibler nonnegative tensor factorizations,” *Optimization Methods and Software*, vol. 30, no. 5, pp. 1002–1029, April 2015.

[7] B. W. Bader, T. G. Kolda *et al.*, “Matlab tensor toolbox version 3.0-dev,” Available online, August 2017. [Online]. Available: https://gitlab.com/tensors/tensor_toolbox/

[8] H. C. Edwards, C. R. Trott, and D. Sunderland, “Kokkos: Enabling manycore performance portability through polymorphic memory access patterns,” *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202 – 3216, 2014, domain-Specific Languages and High-Level Frameworks for High-Performance Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001257

[9] L. Dagum and R. Menon, “Openmp: an industry standard api for shared-memory programming,” *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[10] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis, “FROSTT: The formidable repository of open sparse tensors and tools,” Available online, 2017. [Online]. Available: http://frostt.io/

[11] L. M. Leemis and S. K. Park, *Discrete-Event Simulation: A First Course*. USA: Prentice-Hall, Inc., 2005.