Scaling Numerical Algorithms and Software via Improved Performance Modeling

Edgar Solomonik

L P N A @CS@Illinois

Department of Computer Science
University of Illinois at Urbana-Champaign

ICL Lunch Talk
University of Tennessee, Knoxville

Laboratory for Parallel Numerical Algorithms

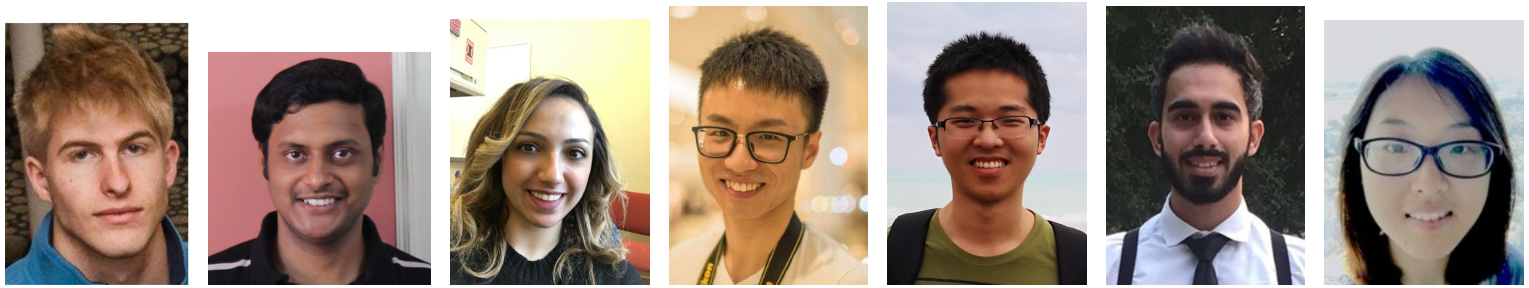
Focus today

- communication avoiding algorithms for dense linear algebra
- Cyclops library for tensor contractions
- inexact autotuning via critical path profiling
- performance modeling via tensor completion

See `http://lpna.cs.illinois.edu` for our group's other work

- quantum simulation
- linear system solvers for interior point methods
- optimization and sketching for tensor decompositions/networks
- parallel sorting

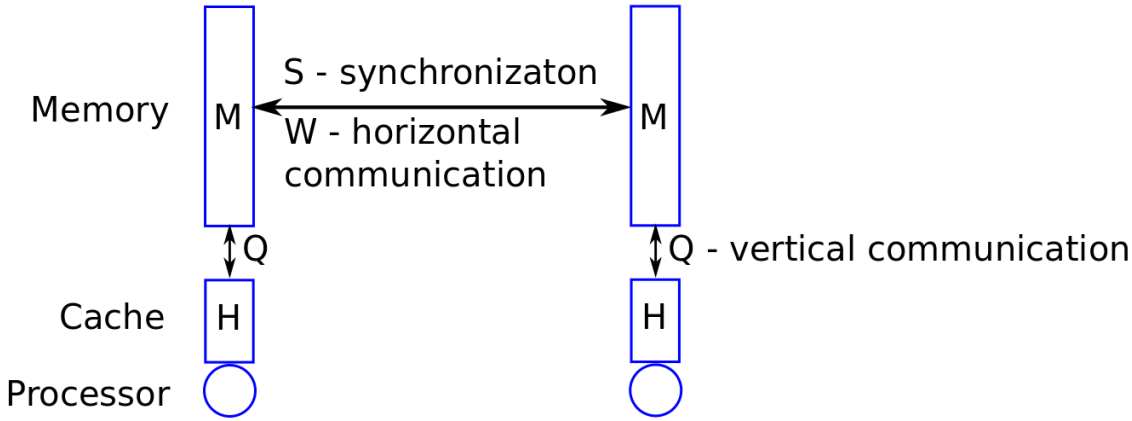
L P N A @CS@Illinois



Beyond computational complexity

Algorithms should minimize communication, not just computation

- communication and synchronization cost more energy than flops
- two types of communication (data movement):



  - vertical (intranode memory–cache)
  - horizontal (internode network transfers)
- parallel algorithm design involves tradeoffs: computation vs communication vs synchronization
- parameterized algorithms provide optimality and flexibility

Cost model for parallel algorithms

We use the Bulk Synchronous Parallel (BSP) model (L.G. Valiant 1990)

- execution is subdivided into S supersteps, each associated with a global synchronization (cost α)
- at the start of each superstep, processors interchange messages, then they perform local computation
- if the maximum amount of data sent or received by any process is w_i (work done is f_i and amount of memory traffic is q_i) at superstep i then the BSP time is

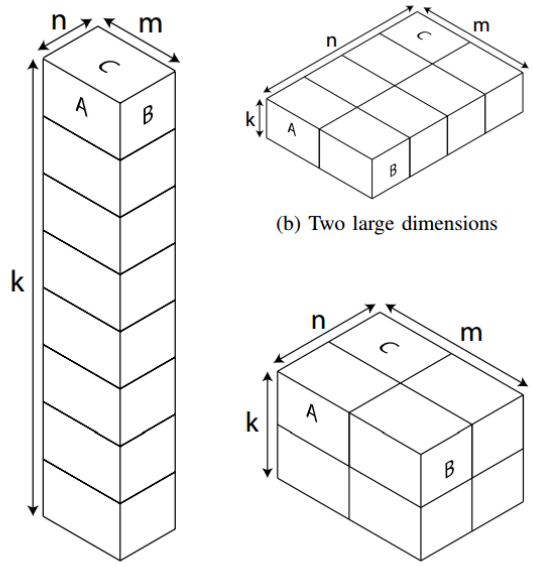
$$T = \sum_{i=1}^{S} \alpha + w_i \cdot \beta + q_i \cdot \nu + f_i \cdot \gamma = O(S \cdot \alpha + W \cdot \beta + Q \cdot \nu + F \cdot \gamma)$$

  where typically $\alpha \gg \beta \gg \nu \gg \gamma$
- we mention vertical communication cost only when it exceeds $Q = O(F/\sqrt{H} + W)$ where H is cache size

Communication complexity of matrix multiplication

Multiplication of $\boldsymbol{A} \in \mathbb{R}^{m \times k}$ and $\boldsymbol{B} \in \mathbb{R}^{k \times n}$ can be done in $O(1)$ supersteps with communication cost $W = O\left(\left(\frac{mnk}{p}\right)^{2/3}\right)$ provided sufficient memory and sufficiently large p

- when $m = n = k$, 3D blocking gets $O(p^{1/6})$ improvement over 2D[1]
- when m, n, k are unequal, need appropriate processor grid[2]



(a) One large dimension

(b) Two large dimensions

(c) Three large dimensions

[1] J. Berntsen, Par. Comp., 1989; A. Aggarwal, A. Chandra, M. Snir, TCS, 1990; R.C. Agarwal, S.M. Balle, F.G. Gustavson, M. Joshi, P. Palkar, IBM, 1995; F.W. McColl, A. Tiskin, Algorithmica, 1999; ...

[2] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, O. Spillinger 2013

3D algorithms for matrix computations

For Cholesky factorization with p processors, BSP (critical path) costs are

$$F = \Theta(n^3/p), \quad W = \Theta(n^2/\sqrt{cp}), \quad S = \Theta(\sqrt{cp})$$

using c matrix copies (processor grid is 2D for $c = 1$, 3D for $c = p^{1/3}$).

Achieving similar costs for LU, QR, and the symmetric eigenvalue problem requires algorithmic changes.

triangular solve	square TRSM ✓[1]	rectangular TRSM ✓[2]
LU with pivoting	pairwise pivoting ✓[3]	tournament pivoting ✓[4]
QR factorization	Givens on square ✓[3]	Householder on rect. ✓[5]
sym. eig.	eigenvalues only ✓[5]	eigenvectors X

✓means costs attained (synchronization within polylog factors).

[1] B. Lipshitz, MS thesis 2013

[2] T. Wicky, E.S., T. Hoefler, IPDPS 2017

[3] A. Tiskin, FGCS 2007

[4] E.S., J. Demmel, EuroPar 2011

[5] E.S., G. Ballard, T. Hoefler, J. Demmel, SPAA 2017

Characteristics of 3D Algorithms

- recursive formulations possible for many problems
  - Tiskin's algorithms for Cholesky, LU with pairwise pivoting, QR
  - some are complicated, latter two recurse on slanted $2{:}1$ panels
- two-level (logical) blocking
  - yields 3D algorithms for LU, QR and reduces vertical communication
  - also used in full-to-band reduction for symmetric eigensolve
- successive band reduction
  - classically used to reduce vertical comm.[1]
  - $\log p$ reduction stages needed to obtain 3D symmetric eigensolve[2]
- alternative numerical formulations
  - different pivoting (pairwise/tournament) for LU, polar decomposition[3]
  - Cholesky-QR2 and variants[4,5], triangular inversion (log depth)[6]

[1] C.H. Bischof, B. Lang, X. Sun, ACM TOMS'00

[2] E.S., G. Ballard, J. Demmel, T. Hoefler, SPAA'17

[3] Y. Nakatsukasa, N.J. Higham, SISC'13. H. Ltaief, D. Sukkari, A. Esposito, Y. Nakatsukasa, D. Keyes, ACM TOPC'19.

[4] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, Y. Yanagisawa, SISC'20

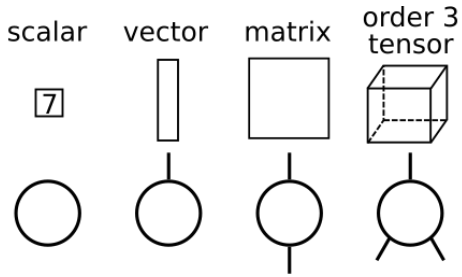
[5] E. Hutter, E.S., IPDPS'19

[6] T. Wicky, T. Hoefler, E.S., IPDPS'17

Tensors

A tensor is a collection of elements

- its dimensions define the size of the collection
- its order is the number of different dimensions
- specifying an index along each tensor mode defines an element of the tensor



A few examples of tensors are

- Order 0 tensors are scalars, e.g., $s \in \mathbb{R}$
- Order 1 tensors are vectors, e.g., $v \in \mathbb{R}^n$
- Order 2 tensors are matrices, e.g., $A \in \mathbb{R}^{m \times n}$
- An order 3 tensor with dimensions $s_1 \times s_2 \times s_3$ is denoted as $\mathcal{T} \in \mathbb{R}^{s_1 \times s_2 \times s_3}$ with elements t_{ijk} for $i \in \{1, \ldots, s_1\}, j \in \{1, \ldots, s_2\}, k \in \{1, \ldots, s_3\}$

Tensor Contractions

A tensor contraction describes a set of products and sums of elements from two tensors

tensor contraction	formula
inner product	$w = \sum_i u_i v_i$
outer product	$w_{ij} = u_i v_{ij}$
pointwise product	$w_i = u_i v_i$
Hadamard product	$w_{ij} = u_{ij} v_{ij}$
matrix multiplication	$w_{ij} = \sum_k u_{ik} v_{kj}$
batched mat.-mul.	$w_{ijl} = \sum_k u_{ikl} v_{kjl}$
tensor times matrix	$w_{ilk} = \sum_j u_{ijk} v_{lj}$

Tensor contractions are prevalent in quantum chemistry methods

Library for Massively-Parallel Tensor Contractions

Cyclops Tensor Framework[1]: sparse/dense generalized tensor algebra

- Cyclops is a C++ library that distributes each tensor over MPI
- Used in chemistry (PySCF, QChem, CC4S)[2], quantum circuit simulation (by IBM/LLNL)[3], and graph analysis (betweenness centrality[4], minimum spanning tree[5])
- Summations and contractions specified via Einstein notation

```
E["aixbjy"] += X["aixbjy"] - U["abu"]*V["iju"]*W["xyu"]
```

- Best distributed contraction algorithm selected at runtime via models
- Support for Python (`numpy.ndarray` backend), OpenMP, and CUDA

[1] https://github.com/cyclops-community/ctf

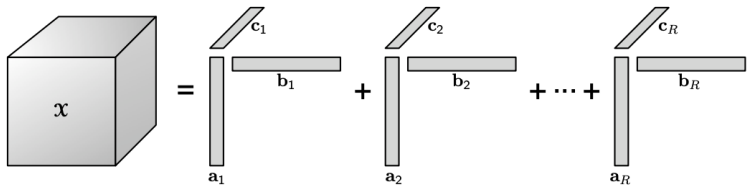
[2] E.S., D. Matthews, J. Hammond, J.F. Stanton, J. Demmel, JPDC 2014

[3] E. Pednault, J.A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E.S., E. Draeger, E. Holland, and R. Wisnieff, 2017

[4] E.S., M. Besta, F. Vella, T. Hoefler, SC 2017

[5] T. Baer, R. Kanakagiri, E.S., SIAM PP 2022

CP Tensor Decomposition



- For a tensor $\mathcal{T} \in \mathbb{R}^{n \times n \times n}$, the CP decomposition[1,2] is defined by matrices A, B, and C such that

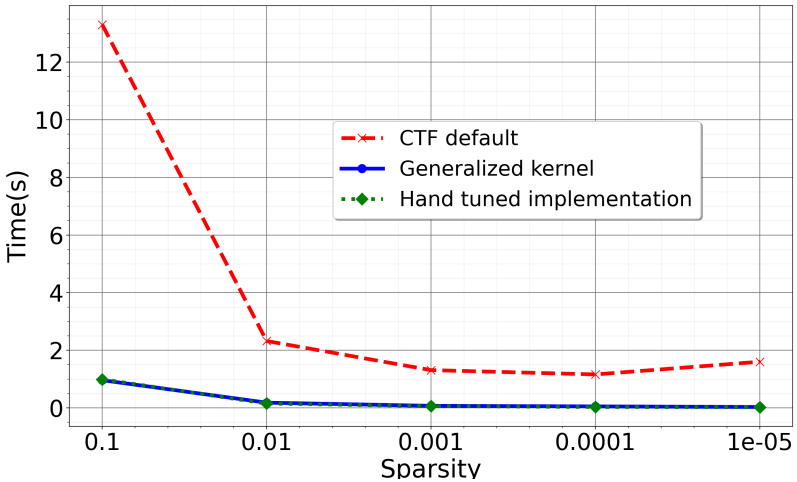
$$t_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$$

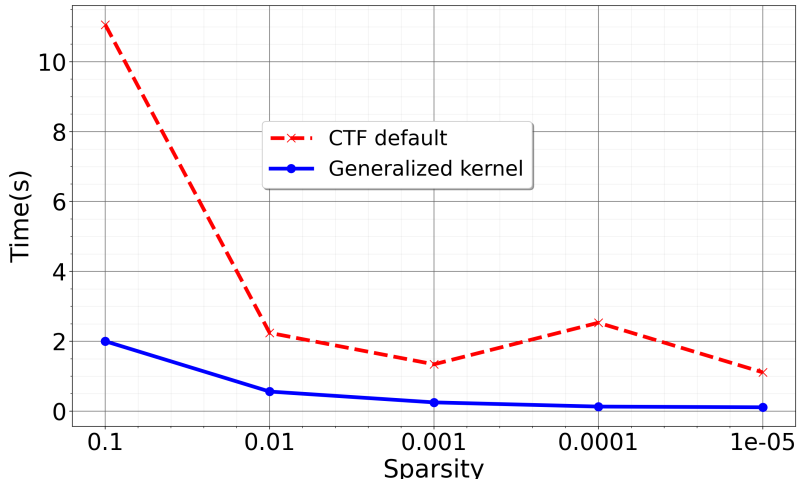
- low-rank CP decomposition is widely used for compression and multi-way data analysis
- high-rank CP decomposition is useful in quantum simulation, search for fast bilinear algorithms

[1] F.L. Hitchcock, Studies in Applied Mathematices 1927

[2] T. Kolda and B. Bader, SIAM Review 2009

Recent and Ongoing Cyclops Developments





- All-at-once contraction for sparse tensor times dense tensor network
  - Driven by tensor completion[1] and quasi-robust density fitting[2]
  - Generalizes MTTKRP (common kernel for CP), TTMc (for Tucker) and other kernels arising in sparse tensor decomposition and completion
  - Working on integration with linear (least-squares) solves
  - Select best loop-nest / integrate BLAS based on performance model

[1] N. Singh, Z. Zhang, X. Wu, N. Zhang, S. Zhang, and E.S., JPDC'22

[2] D.P. Tew, The Journal of Chemical Physics 2018

Accelerating Automatic Tuning

Autotuning – searching for fastest program variant by benchmarking

- widely used for performance tuning today
- computationally expensive, especially if different variants wanted for different inputs

Performance of parallel programs is hard to predict

- communication bottlenecks dependent on architecture
- idle time and load imbalance arise due to dependencies

But many programs largely consist of easy-to-model subkernels

- parallel QR codes repeatedly execute local MMs of similar sizes

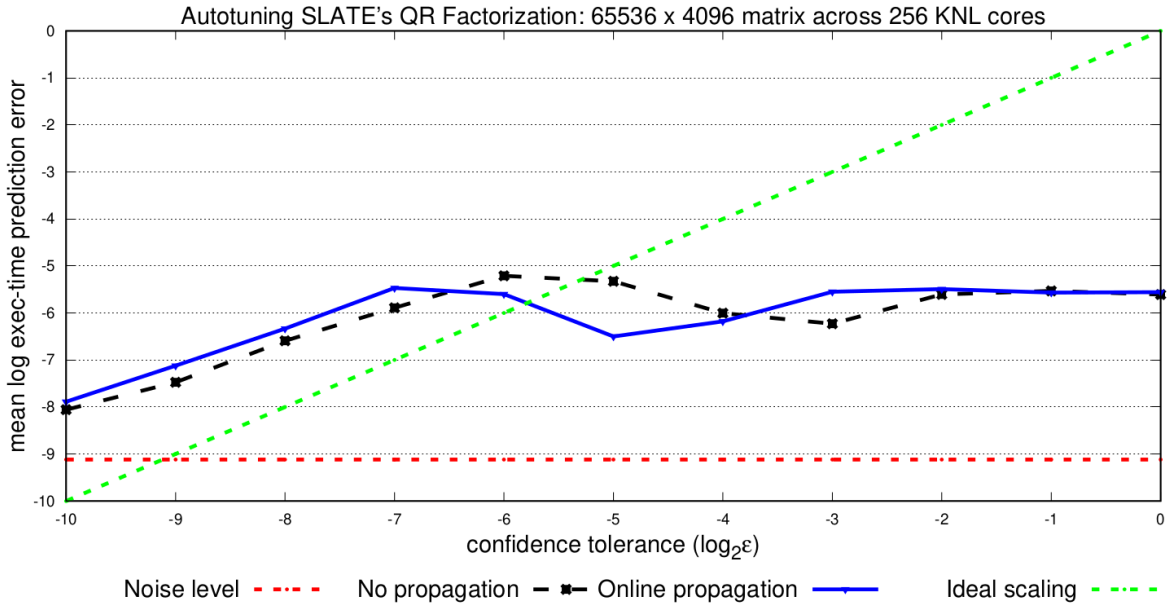
Approximate Automatic Tuning

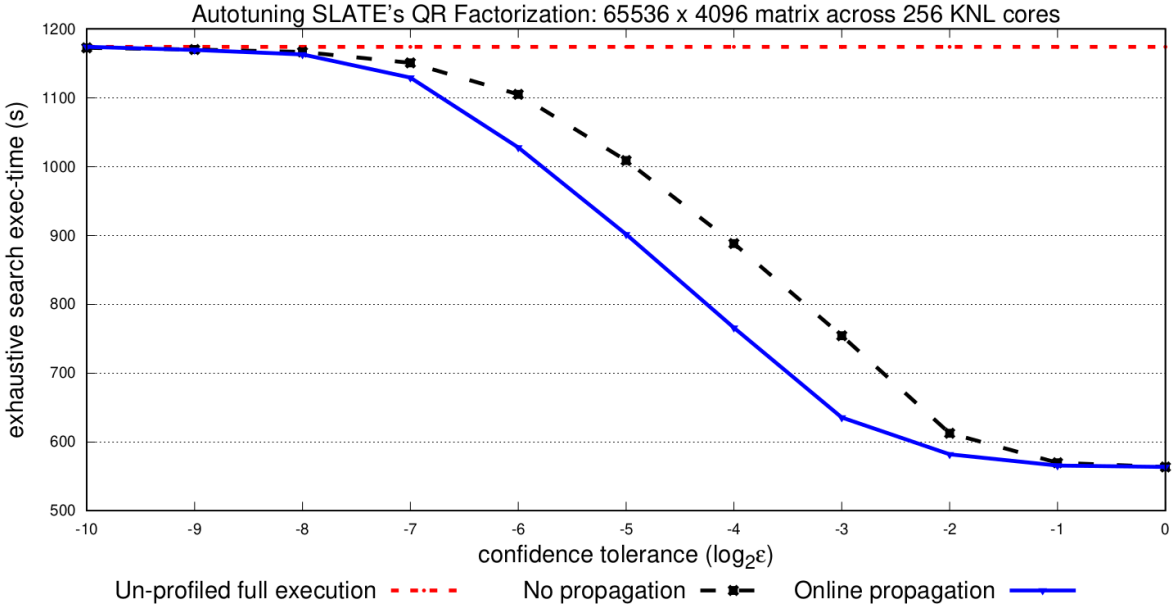
Critter[1] leaverages critical path profiling to efficiently perform approximate autotuning in dense linear algebra

- designed to track critical path communication/synchronization costs
- approximates critical path execution time by predicting subkernel performance
- the more times a subkernel is executed, the more statistical accuracy in overall prediction
- propogates local kernel timings to other processors to reduce cost
- automatically profiles MPI routines and identifies basic BLAS routines

[1] E. Hutter, E.S. IPDPS'21

Approximate Automatic Tuning for QR





Performance Modeling

- selection of best program variants often guided by performance modeling
  - for autotuning, performance models can guide/accelerate
- in practice, performance models may be
  - semi-analytic or derived from program structure
  - learned from sample of benchmark timings
- for complex programs, input and parameter space is high-dimensional
  - multi-task learning problem
  - runtime data is noisy and partially complete

Prediction Model and Accuracy

- Given d input/tuning parameters and executions times $T(p_{i_1}, \ldots, p_{i_d}) \in \mathbb{R}_+$ for $(i_1, \ldots, i_d) \in \Omega$, seek concise model T for all other parameters
- For example, seek to model $m \times k$ by $k \times n$ matrix multiplication
  - analytically, a simple model may be

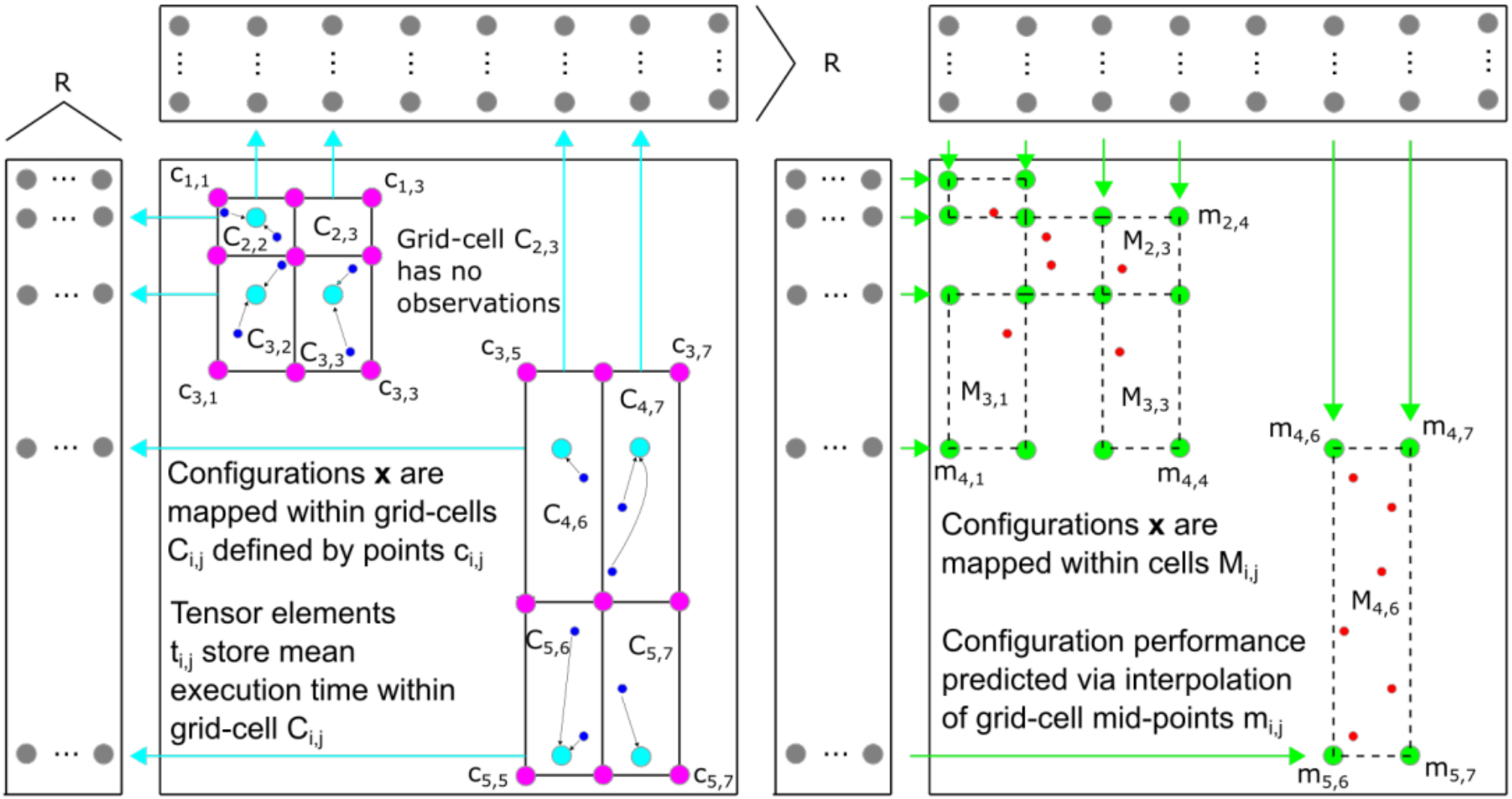
$$T_{\mathsf{MM}}(m, n, k) = \gamma \cdot mnk + \beta \cdot (mn + nk + mk) + \alpha$$

    with α, β, and γ tuned
- Generally, we want the prediction to be accurate in scale, namely we want to minimize error in $\log(T)$
- We apply CP tensor completion for modeling, e.g.,

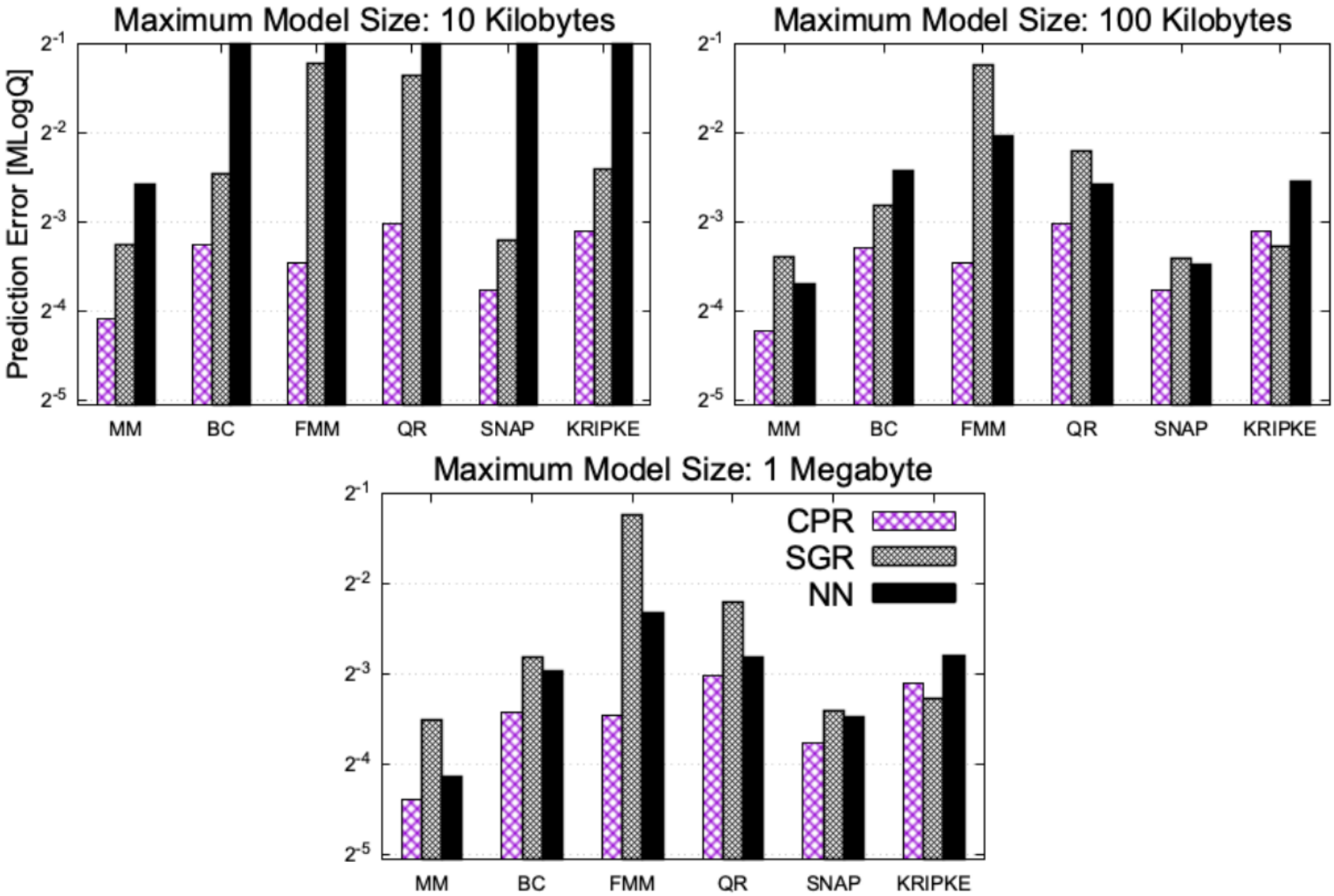
$$\min_{\boldsymbol{\mathcal{X}}=\mathsf{CP}(A,B,C)} \sum_{(u,v,w)\in\Omega} (x_{uvw} - \log(T_{\mathsf{MM}}(m_u, n_v, k_w)))^2 + \lambda \cdots$$

- We find the log significantly improves effective prediction accuracy, and is more efficient than using a log loss function

Discretization and Interpolation



Discretization and Interpolation



Conclusion and Acknowledgements

- For details on tensor completion modeling, see https://solomonik.cs.illinois.edu/preprints/HS22.pdf
- NSF awards #1839204 (RAISE-TAQS), #1931258 (CSSI), #1942995 (CAREER), the DOE CSGF program, and the DOE MMICC program
- Stampede2 resources at TACC via XSEDE
- See https://lpna.cs.illinois.edu for all further info

L P N A @CS@Illinois