Parallel Matrix Factorization

ExaScience life lab

Tom Vander Aa and Tom Ashby

EuroMPI Tutorial on Machine Learning at Scale

janssen PHARMACEUTICAL COMPANIES OF Johnson & Johnson

intel

iMT

imec

UNIVERSITEIT GENT

universiteit hasselt

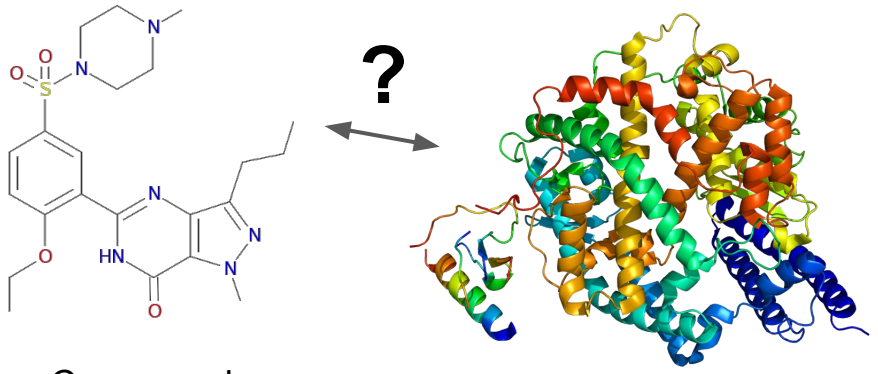
KU LEUVEN

Overview

- Introduction
  - Compound Activity Prediction
  - BPMF
- Single node
  - Shared memory parallelism
  - Load balancing
- Distributed
  - Communication-computation overlap
  - Results

Compound Activity Prediction

- Predict
  - compound activity on
  - protein target
  - aka chemogenomics



Compound

Enzyme

- Similar to
  - Netflix: users rating movies
  - Amazon: users rating books

NETFLIX

amazon

Chemogenomics: Background

~1% can be filled up with experimental dose response data

ABCD

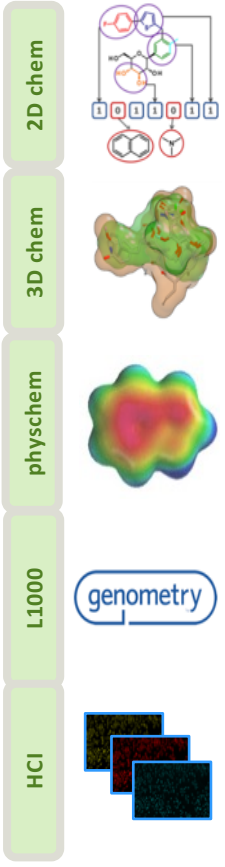
GOSTAR
GVK BIO Online Structure Activity Relationship Database

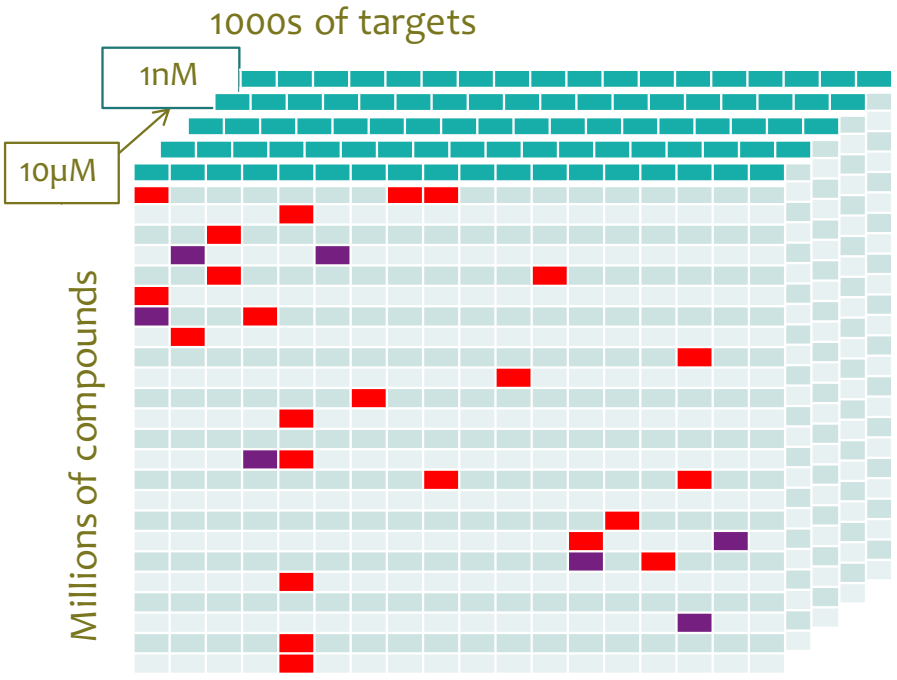
THOMSON REUTERS

ChEMBL

OpenTox

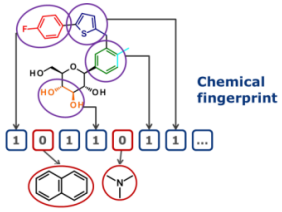
Quarterly updated





Several Methods Developed

Single Task



Activity Prediction

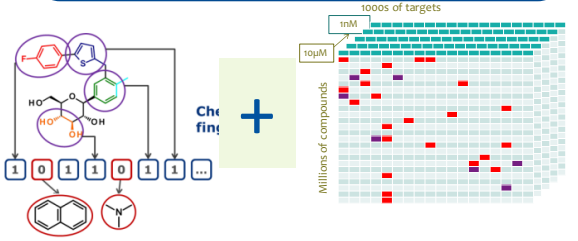
- KNN
- Naïve Bayes
- Logistic Regression
- LogOdd Score

Multi Task

Activity Prediction

- BPMF
- ALS

Multi Task + Side Info



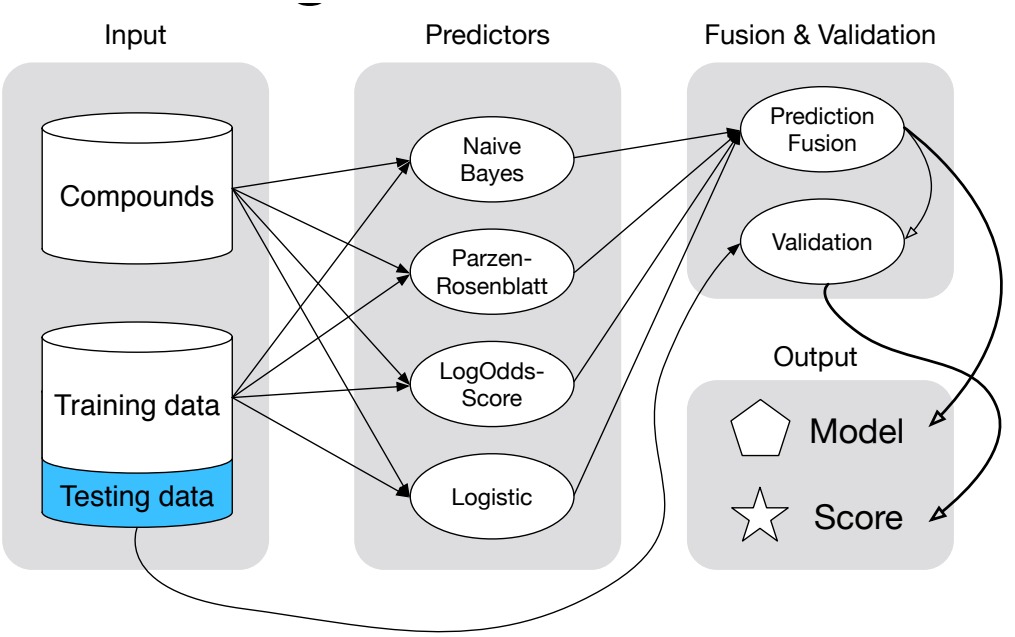
Activity Prediction

- DNN
- GFA
- Macau

Single Task Learning with Spark

Dimensions:

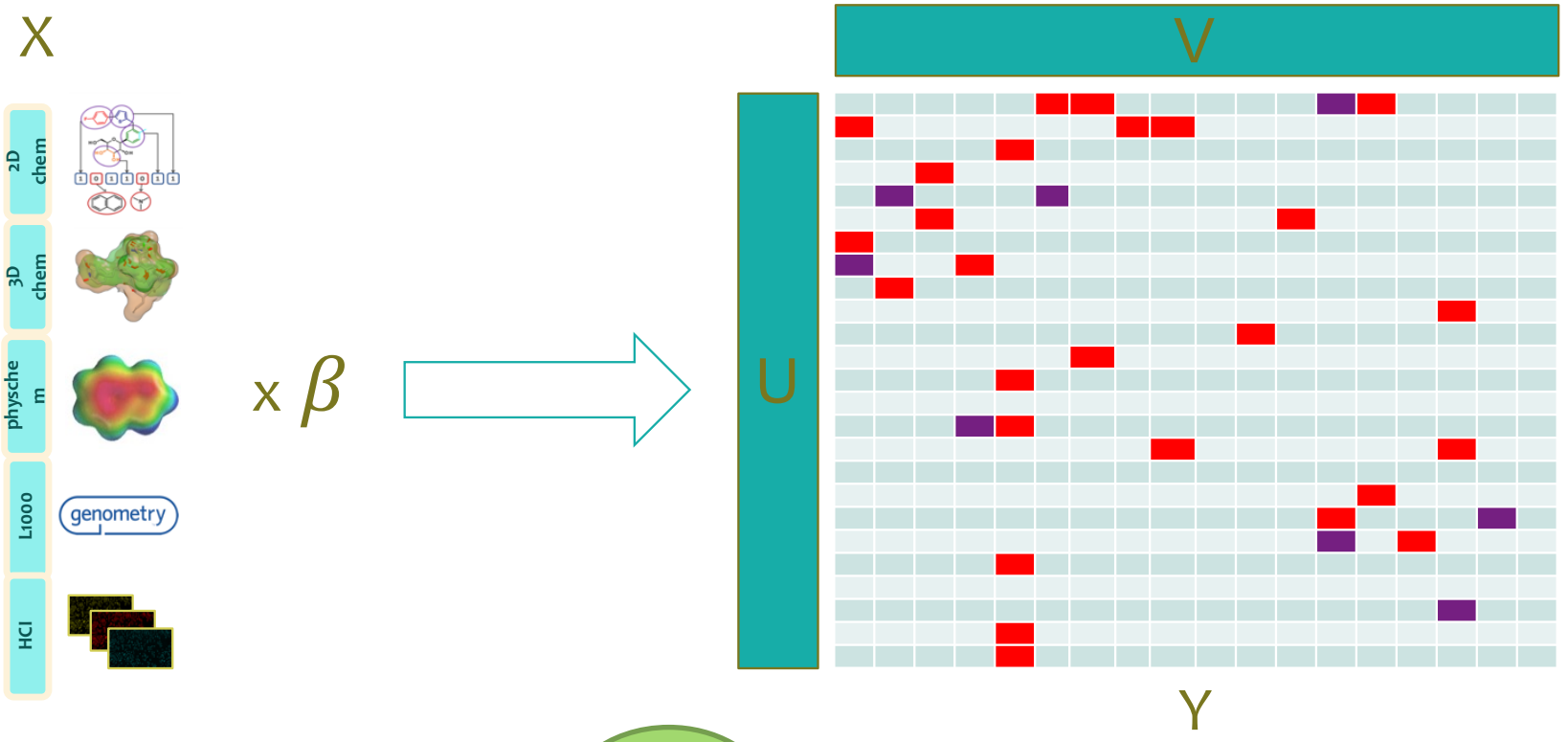
- Cross-Fold
- Parameter Search
- Activity Levels
- Compound



= Embarrassingly Parallel

Spark

MACAU: Multi-Task Learning with Side Info

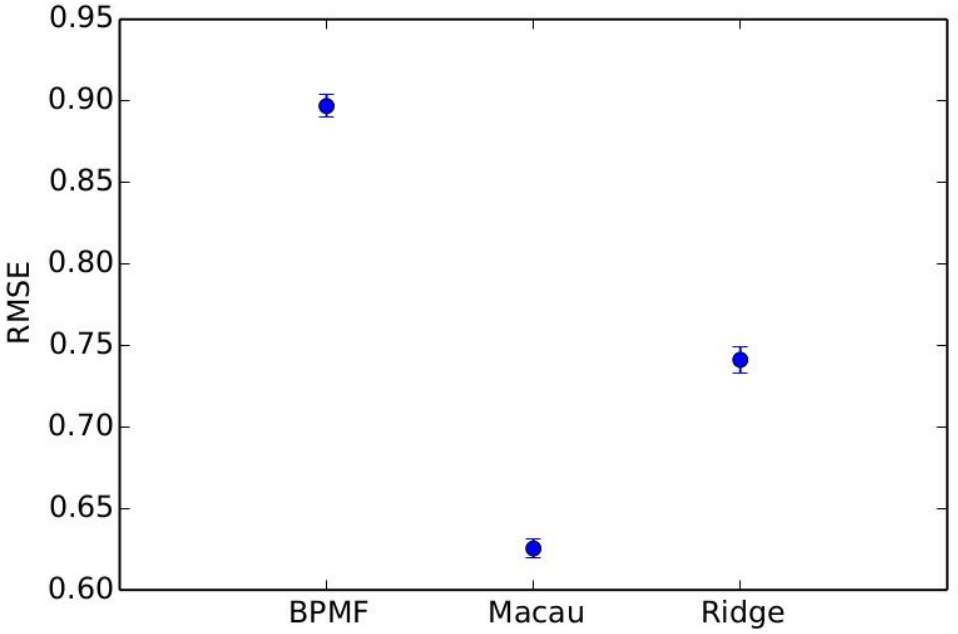


CG Solver

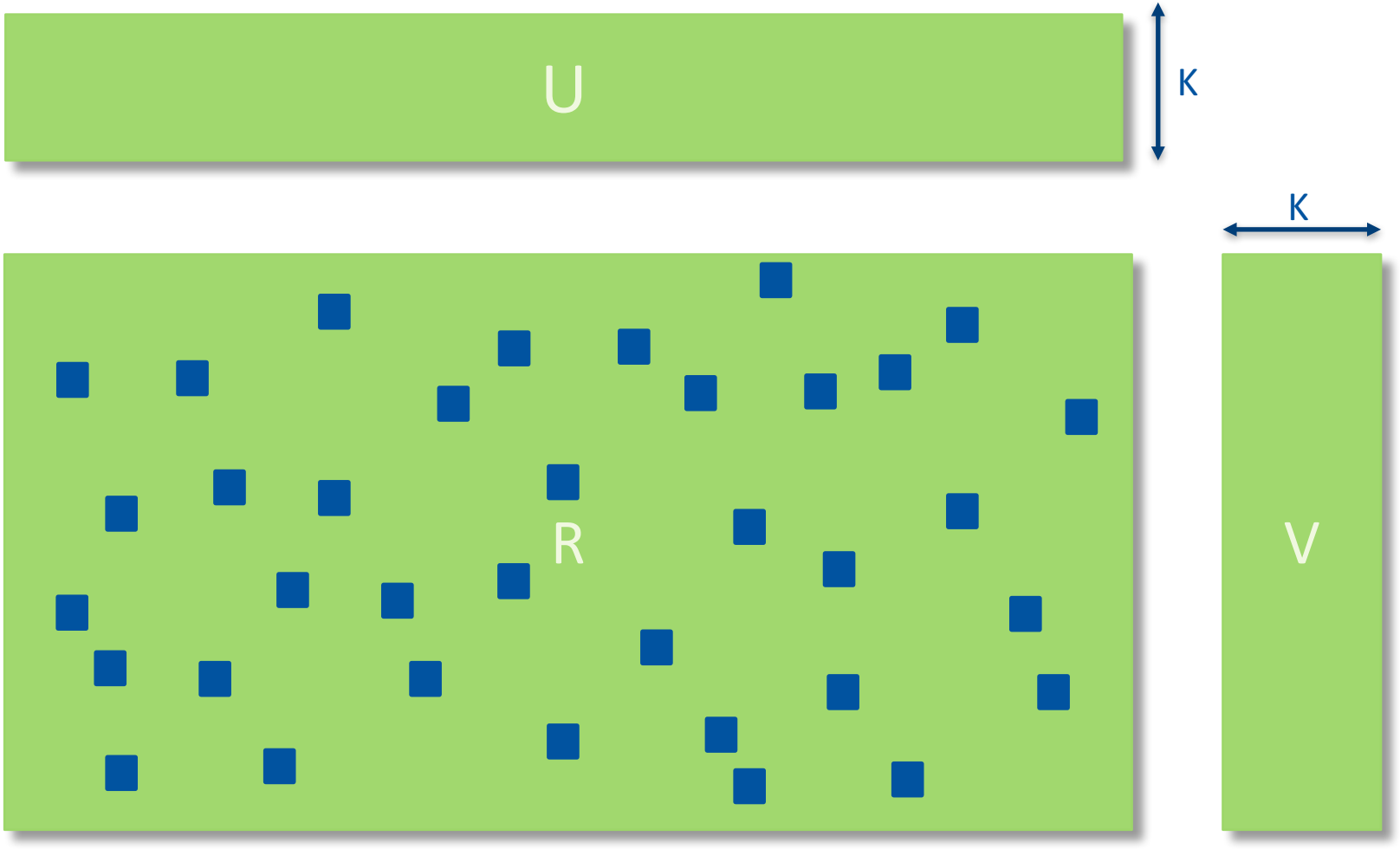
Matrix Factorization

Added Value of Compound Features

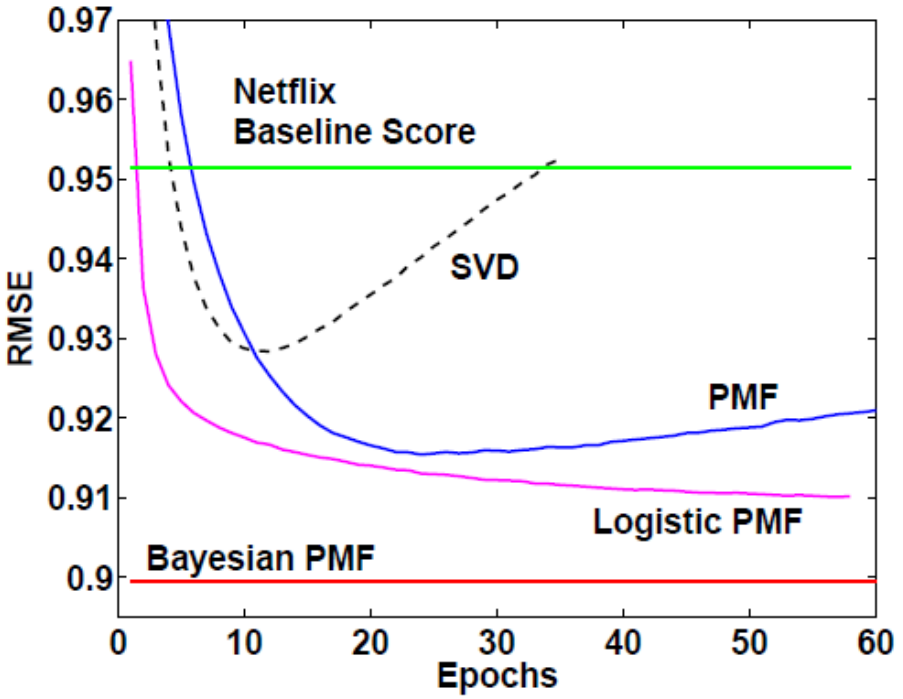
- Version: ChEMBL v19
  - 346 proteins
  - 15,073 compounds
  - 59,280 IC50 measurements
- 105k dimensional ECFP features
- Use 20% as test set
- 10 repeats
- D = 30 (#latent dimensions)

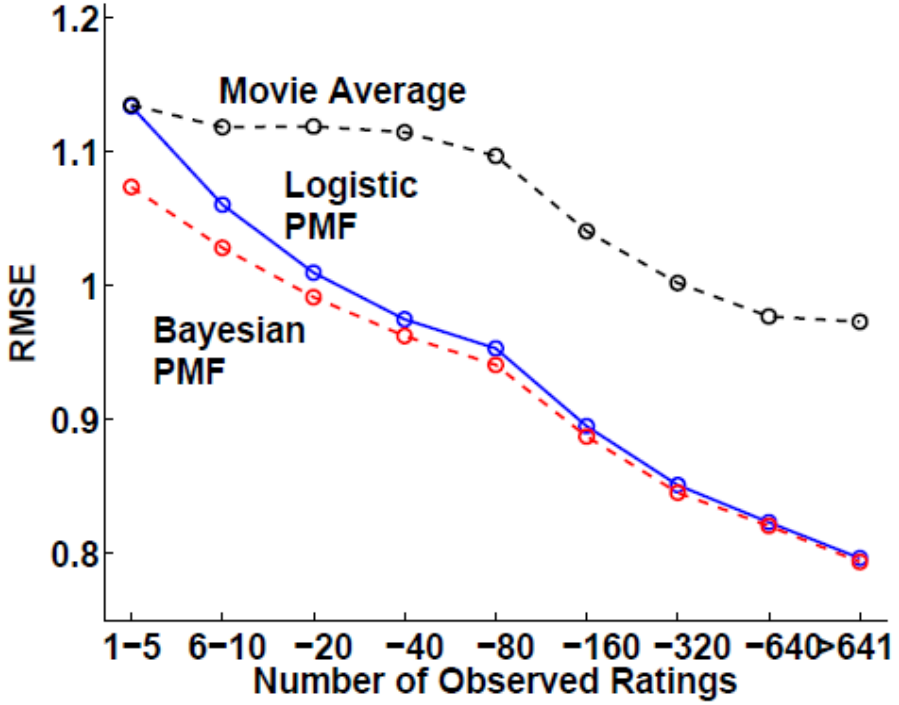


BPMF := Low-rank Matrix Factorization



BPMF seems to predict well





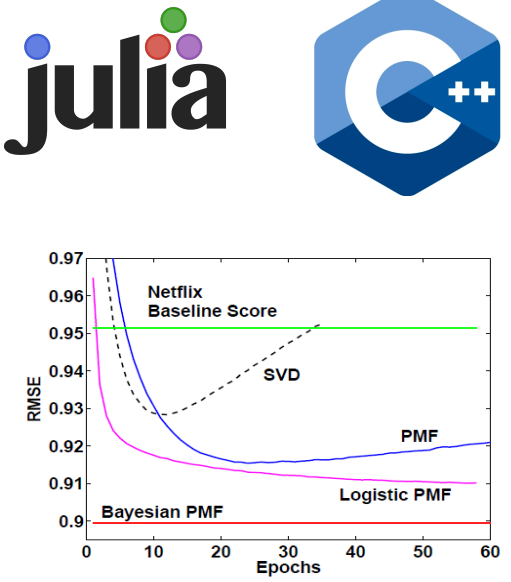
Bayesian: all possible models

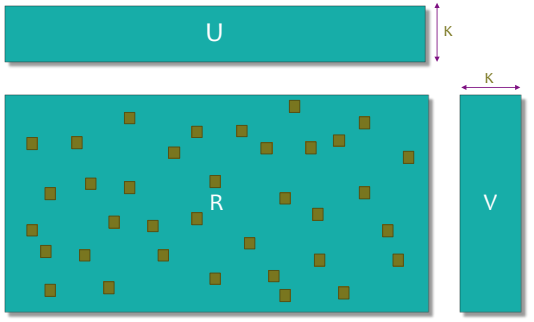
Priors and HyperPriors: Users are Gaussian distributed

Gibbs sampling

BPMP: simple, promising but slow

- BPMF is simple
  - 25 lines of Julia code
  - 35 lines of C++ code
- BPMF predicts well
  - by using Bayesian approach
  - by using compound fingerprints in Macau
- BPMF is slow
  - Sampling based
  - Julia prototype: 15 days / run





Baseline BPMF is simple

For t=1,...,T

- Sample the hyperparameters

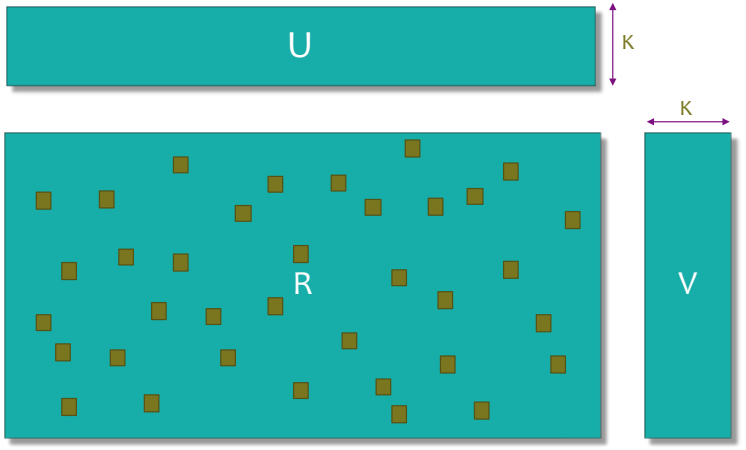
$$\Theta_U^t \sim p(\Theta_U | U^t, \Theta_0)$$
$$\Theta_V^t \sim p(\Theta_V | V^t, \Theta_0)$$

- For each $i = 1, ..., N$ sample user features

$$U_i^{t+1} \sim p(U_i | R, V^t, \Theta_U^t)$$

- For each $i = 1, ..., M$ sample movie features

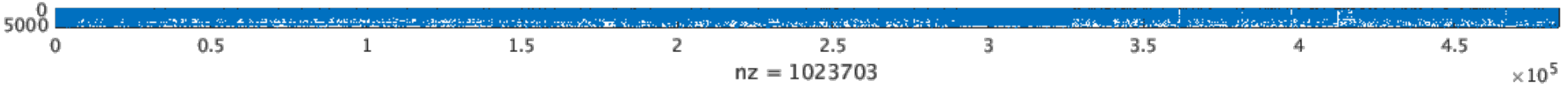
$$V_i^{t+1} \sim p(V_i | R, U^{t+1}, \Theta_V^t)$$



Typical Input Data

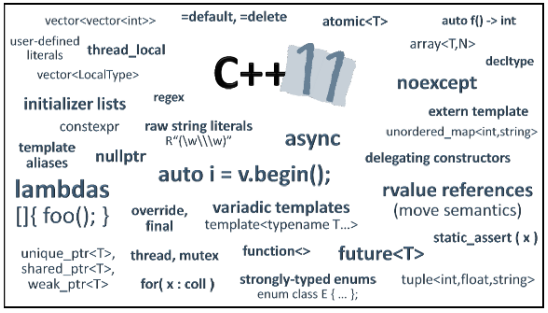
- Parameters
  - Number of latent features: 100
  - Number of sampling iterations: 1000

- Industrial Data
  - 1.7M compounds
  - 2350 targets
  - 36.7M entries (0.8%)



Single Node Implementation

- Operations
  - Matrix/Vector operations
  - Matrix inversion / Cholesky
  - Random numbers
  - Typically on: KxK matrices
    - K: number of latent features
    - Typical K == 100



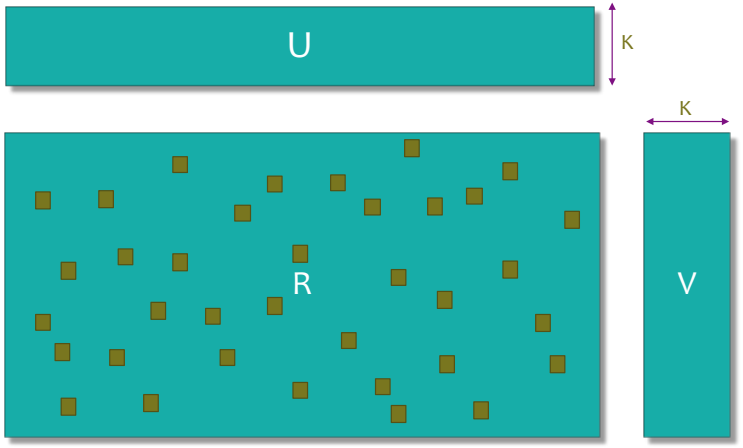
- Choice of tools
  - C++
  - TBB/OpenMP
  - Eigen 3.3

Single Core Optimizations

- Algorithmic
  - Full Inverse vs Cholesky vs Cholesky + rank update
- Eigen
  - Optimize expression templates
  - Annotate aliasing in matrix operations
- Compute in one pass over U
  - Norm, Average, Covariance, Updated U

Single Node Parallelization

- Seems easy to parallelize
  - Across users/movies
  - Inside a user/movie
    - Compute Precision matrix

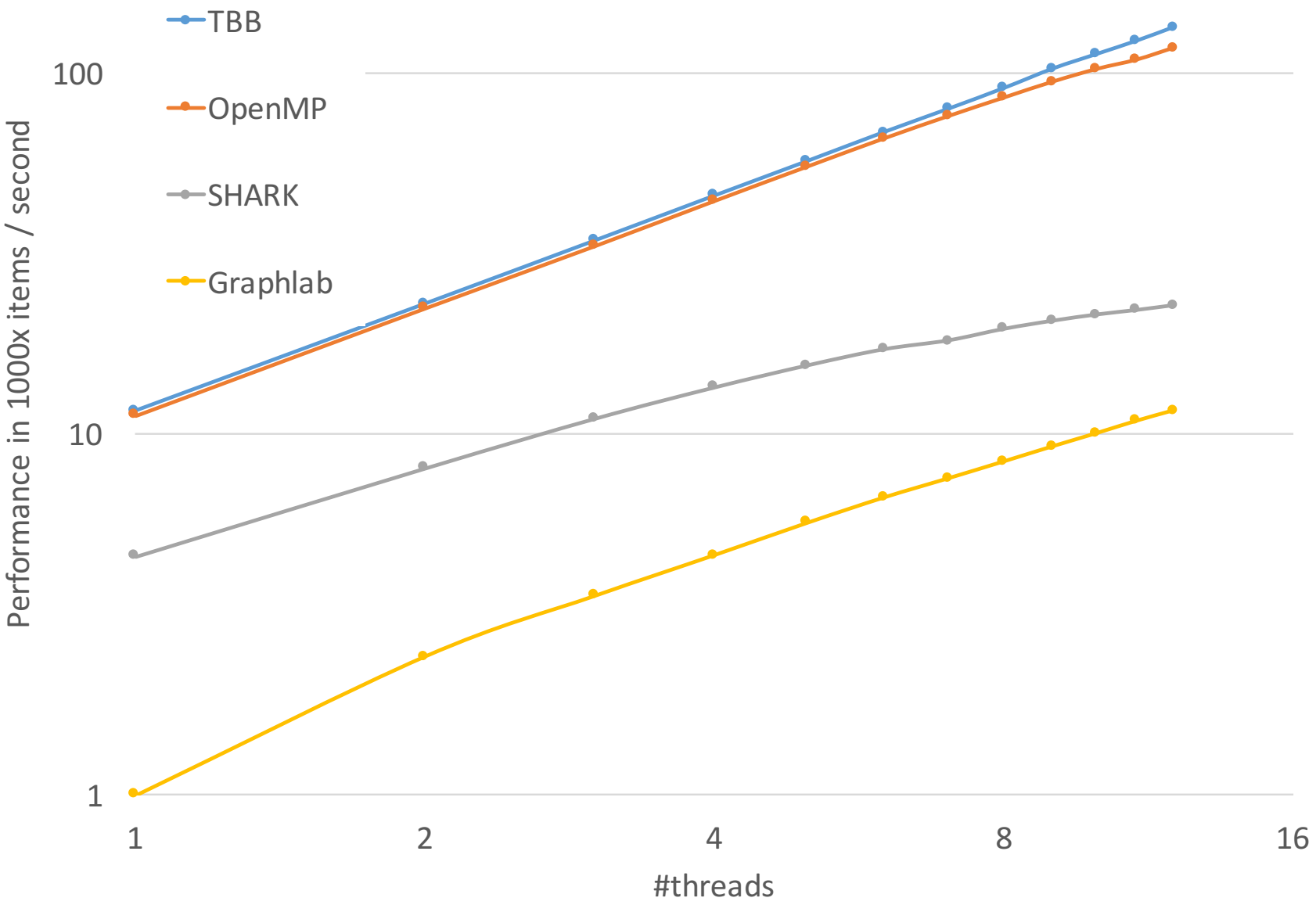


- Shared memory parallelism
  - Memory bandwidth limited
  - Optimize rank R matrix for locality

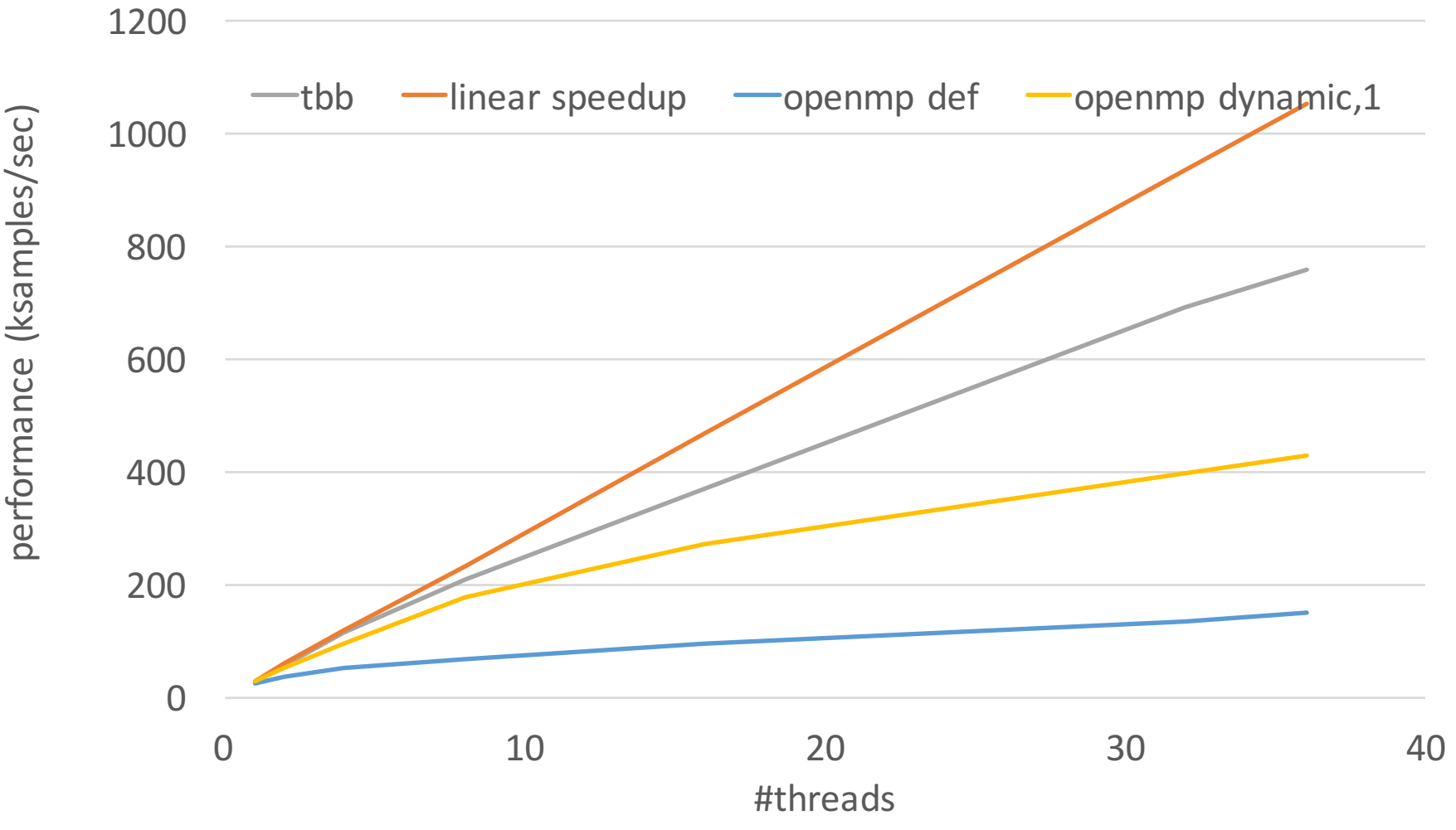
- Load Balancing
  - Load depends on #items, and #nnz
  - Use TBB nested parallelism across + inside items


work per rating
work per item
work per rating (msec)
work per item (msec)
#ratings per item
60
40
20
0
100000
10000
1000
100
10
1
1
8
64
512
4096
32768

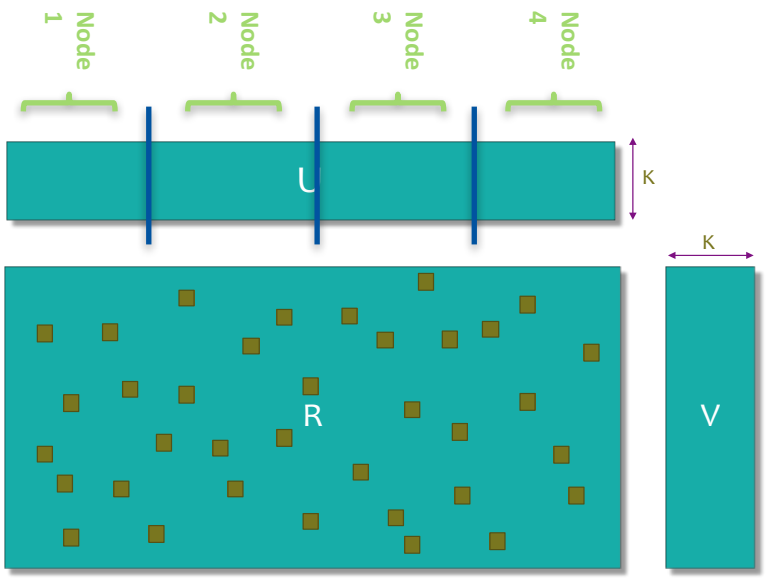
Single Node Performance

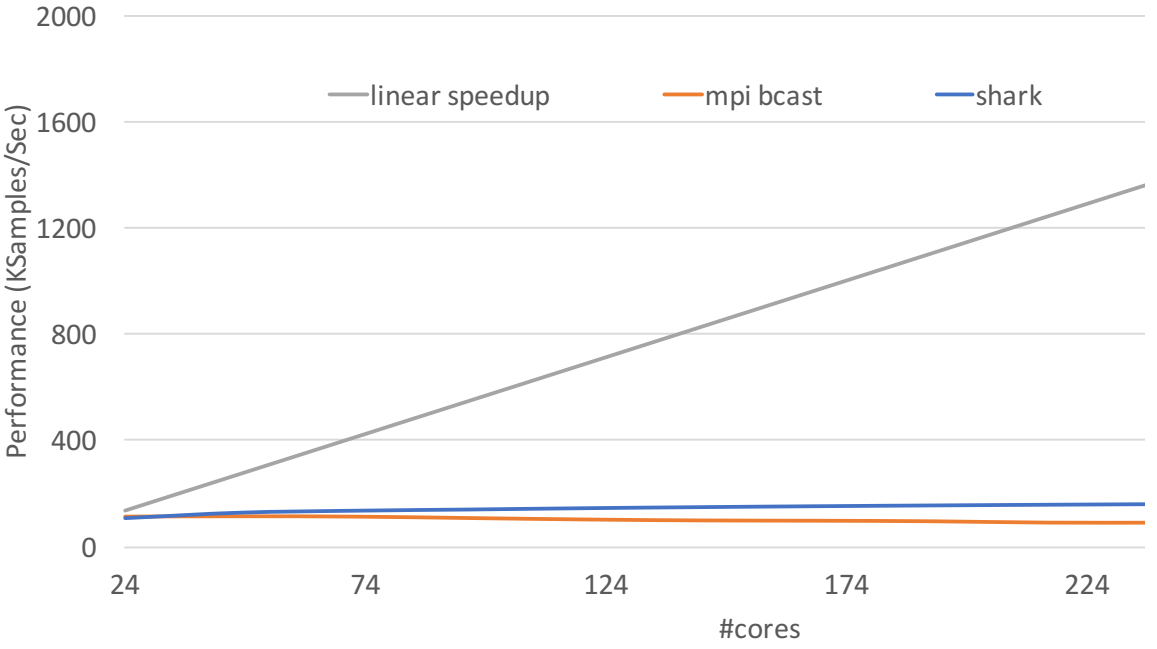


Single Node Performance



Synchronous Distributed BPMF





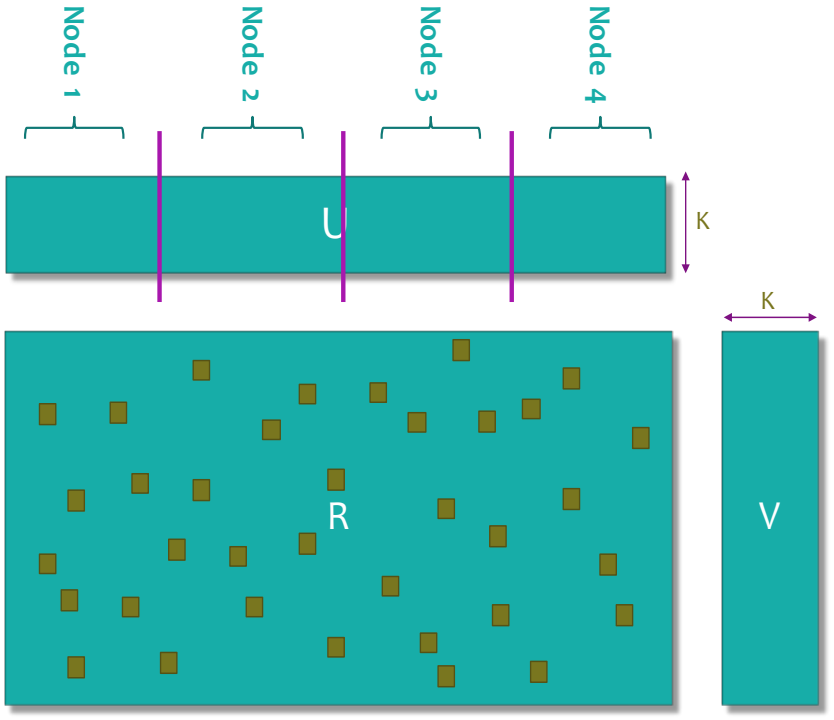
Only 2x improvement with 20x more cores

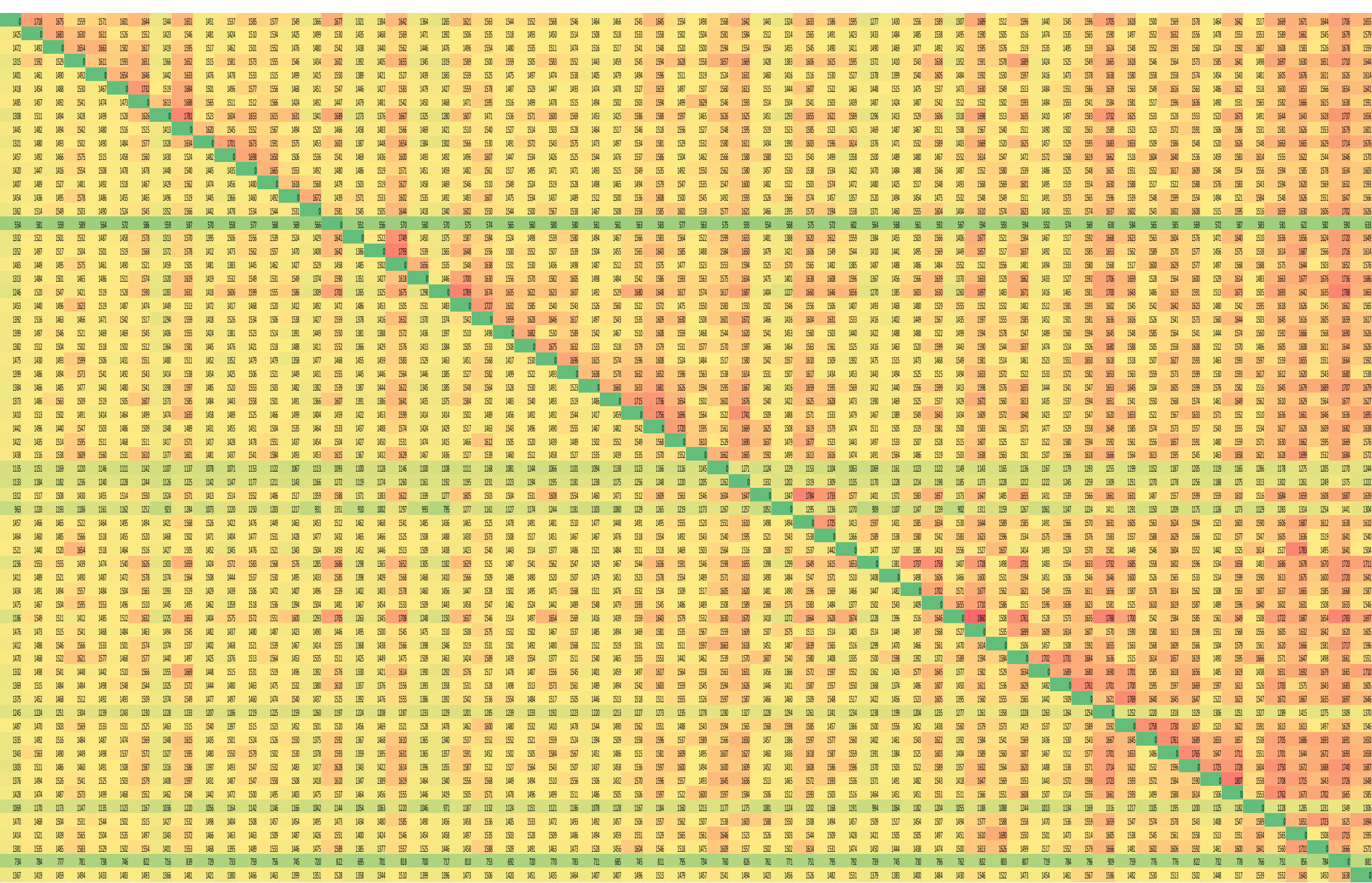
We need asynchronous communication

EXA2CT Slides

Asynchronous Distributed BPMF

- Split both U/V, optimizing:
  - Load balance (# rows, #nnz)
  - Communication
  - ➔ Both are determined by R
- Basic Pattern GASPI
  - Compute a column of U/V
  - Send to needed nodes



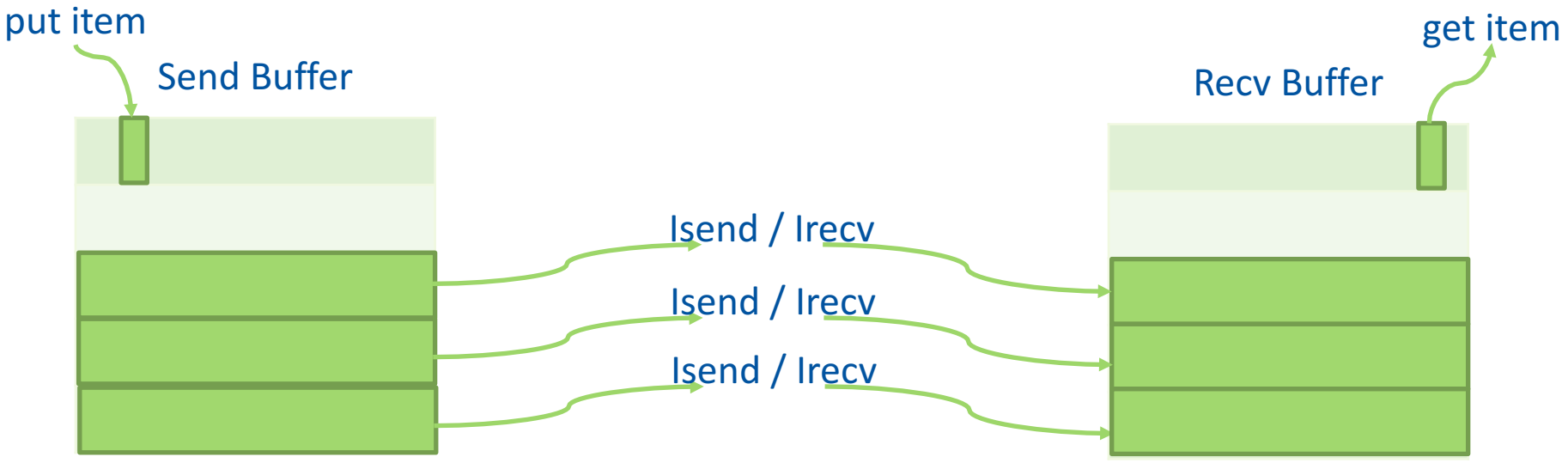


Challenges using MPI

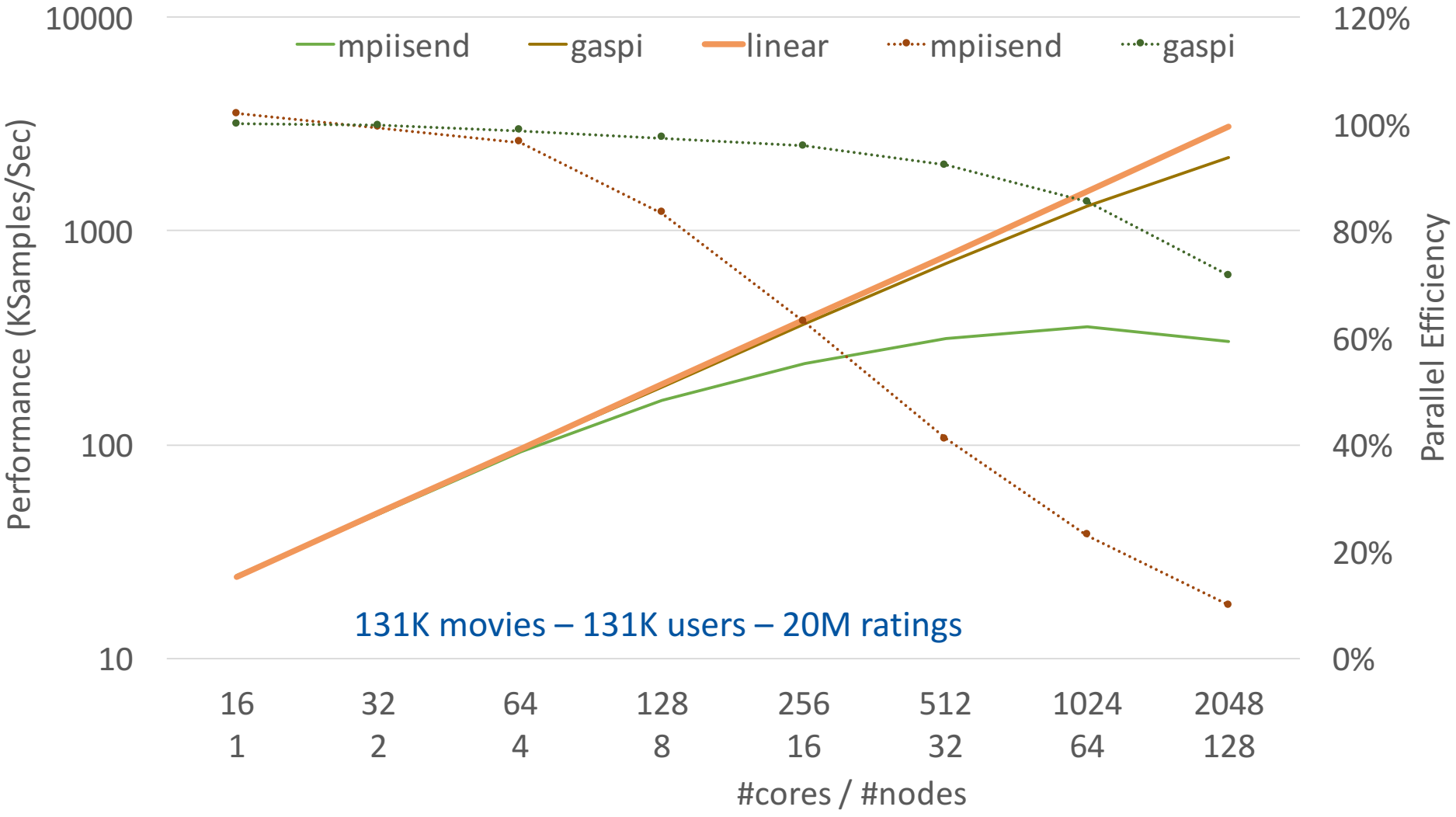
- MPI is not thread-safe by default
  - Multiple work threads, one MPI thread
- MPI calls have high overhead
  - Buffer before send
- Many possible MPI primitives
  - MPI_Bcast
    - simple, synchronous, does not scale well
  - MPI_Put:
    - need to split U in multiple MPI Windows, one window per peer
    - actual MPI work delayed until end of epoch
  - MPI_Isend/Irecv
    - best at doing background work
    - many irecvs/isends in flight

Current best MPI implementation

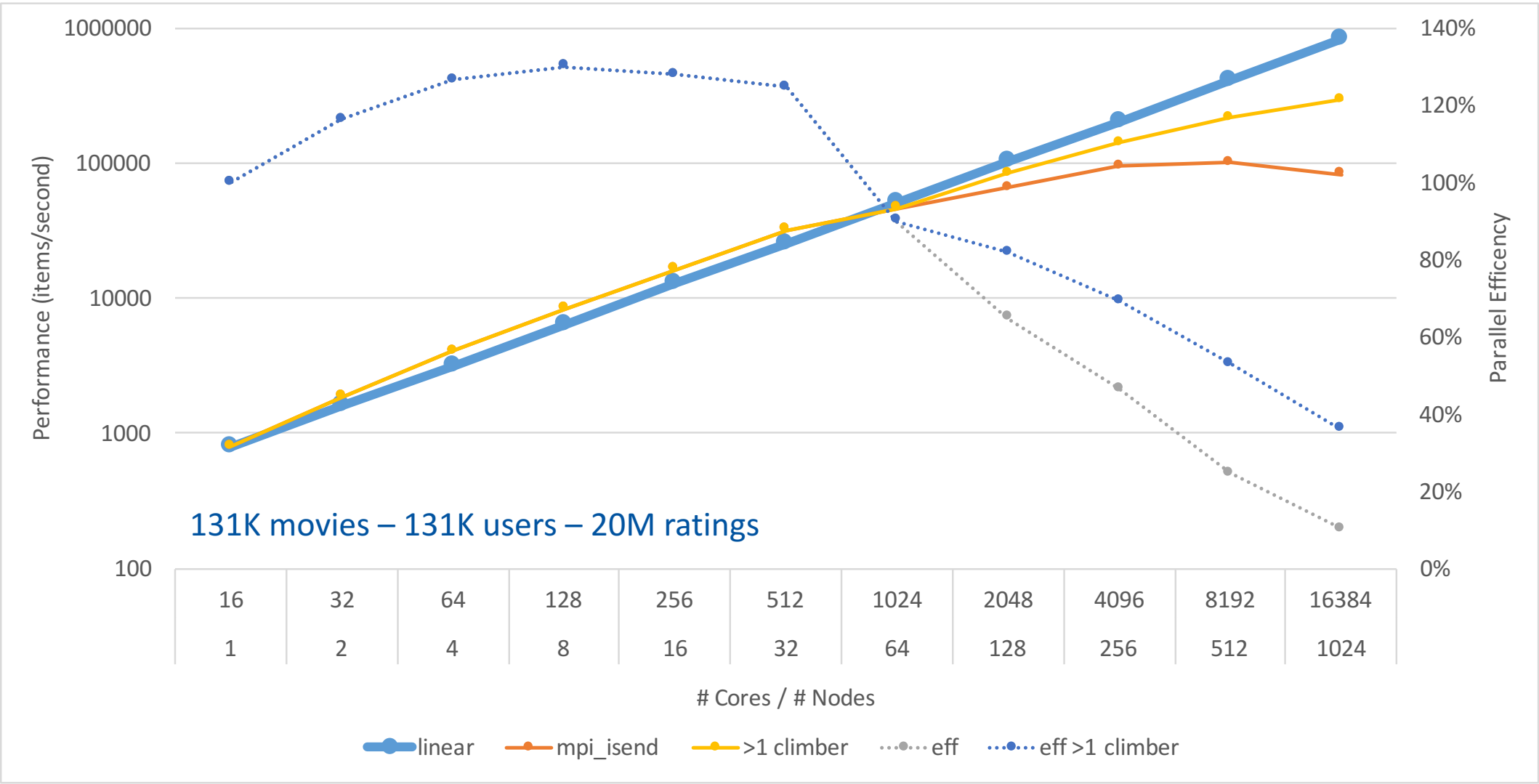
- Buffered ISend/IRecv
  - One buffer-pair per send-receive pair
  - Several chunks per buffer
  - Several items per chunk



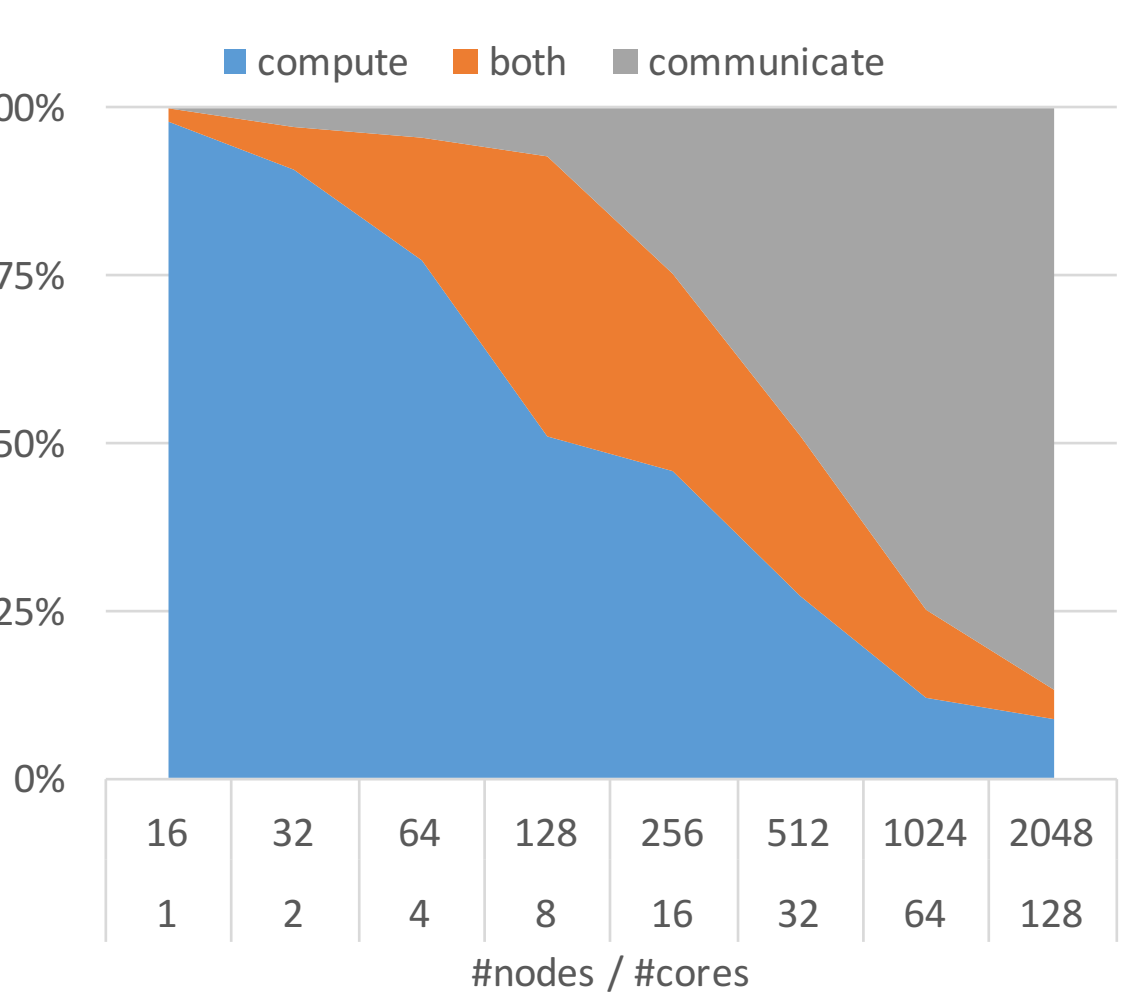
Distributed Performance – 128 nodes



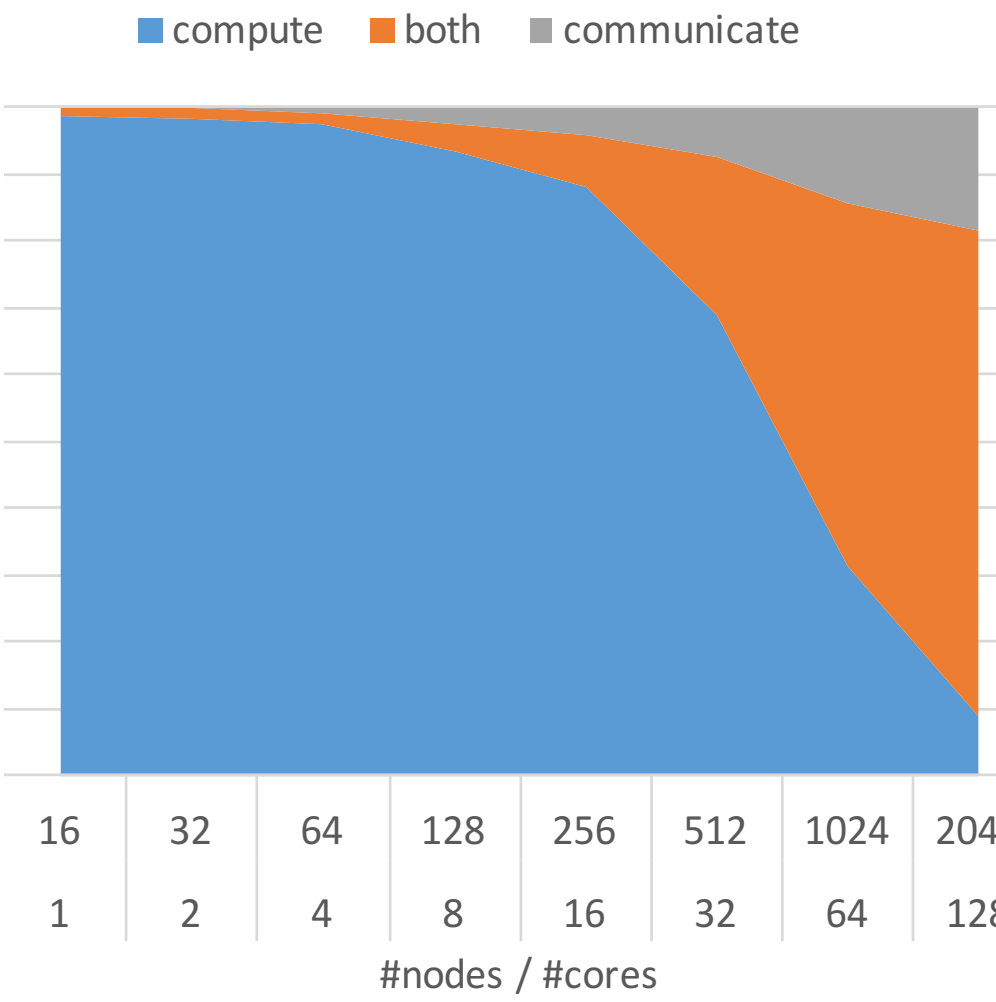
Distributed Performance – 1K nodes



Comm – Comp Overlap



MPI



GASPI

Conclusions optimizing BPMF

- Reduce runtime on industrial data (estimated)

Parallelism	Time 1 Run
Single node - Julia	15 days
Single node - C++ & TBB	1.5 hours
Distributed - C++ & TBB & GASPI	5 minutes

- Parallel efficiency is important
  - Load Balancing and Communication Hiding

- BPMF Released on GitHub
  - https://github.com/ExaScience/bpmf
  - https://github.com/jaak-s/BayesianDataFusion.jl