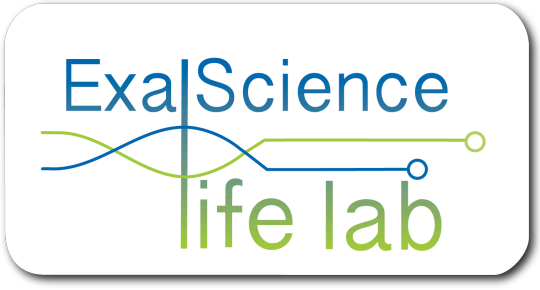


Parallel Distributed BPMF

TOM VANDER AA

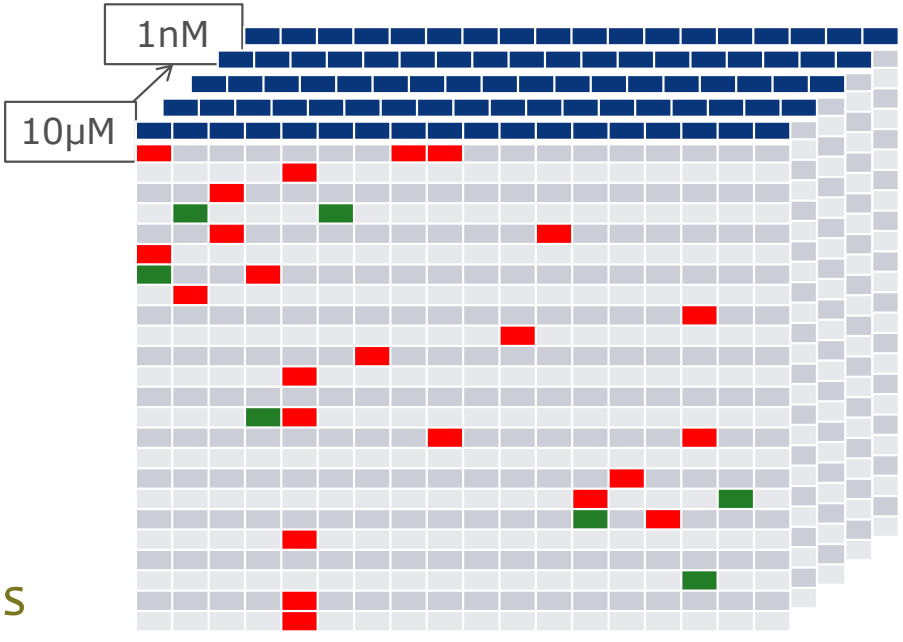
IMEN CHAKROUN, TOM HABER &
THE EXA2CT EU PROJECT

Overview

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

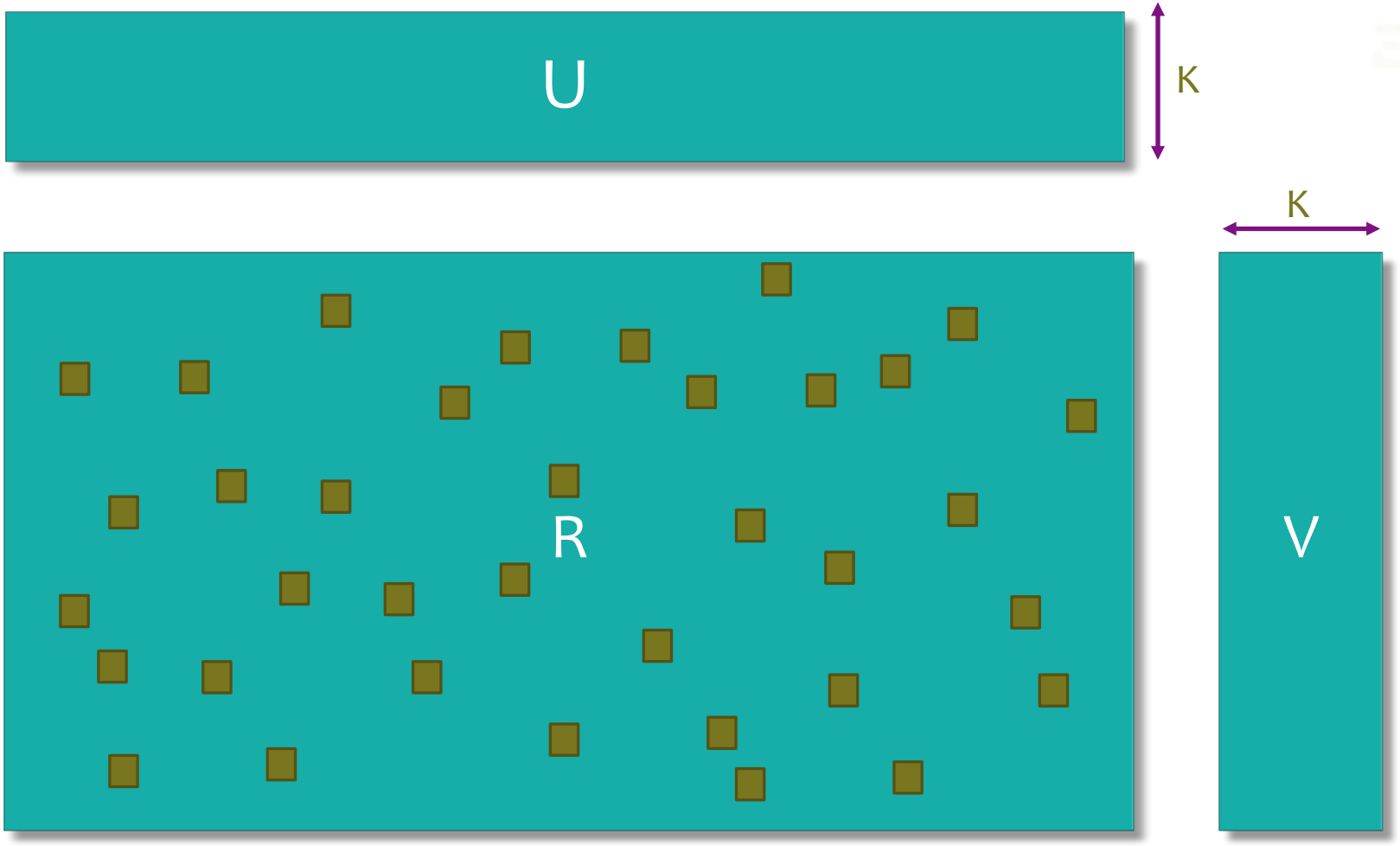
Compound Activity Prediction

- Predict
  - compound activity on
  - protein target
  - aka chemogenomics
- Similar to
  - Netflix: users rating movies
  - Amazon: users rating books



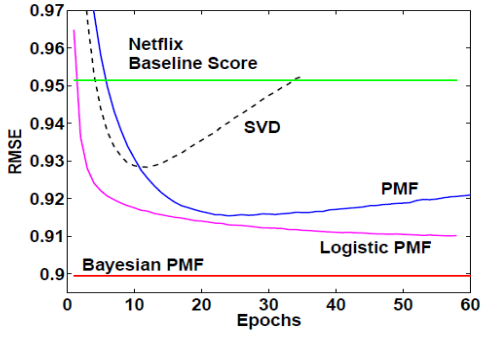
NETFLIX amazon

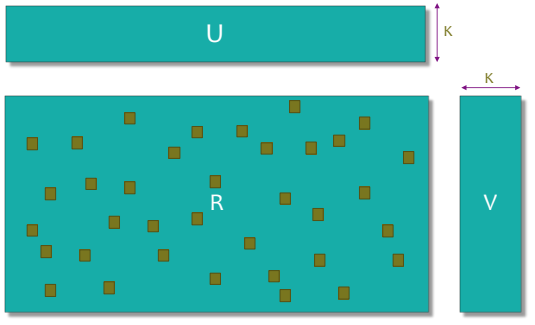
BPMF := Low-rank Matrix Factorization



BPMP: simple, promising but slow

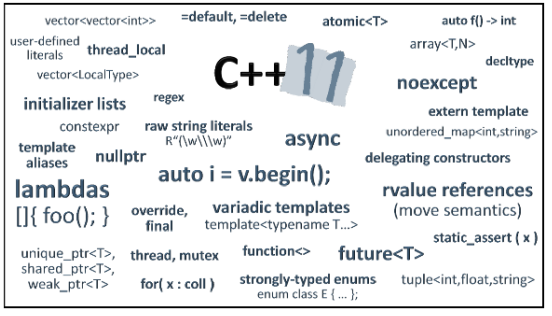
- BPMF is **simple**
  - 25 lines of Julia code
  - 35 lines of Eigen C++ code
- BPMF **predicts well**
  - by using Bayesian approach
  - by using compound fingerprints in Macau
- BPMF is **slow**
  - Sampling based
  - Julia prototype: 15 days / run





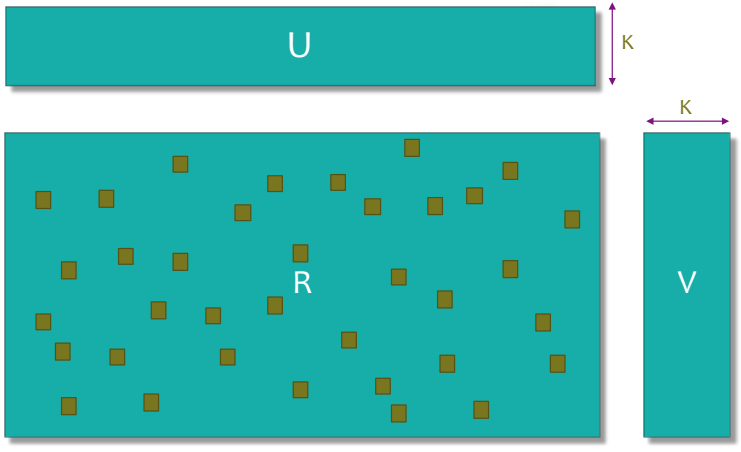
Single Core Optimizations

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



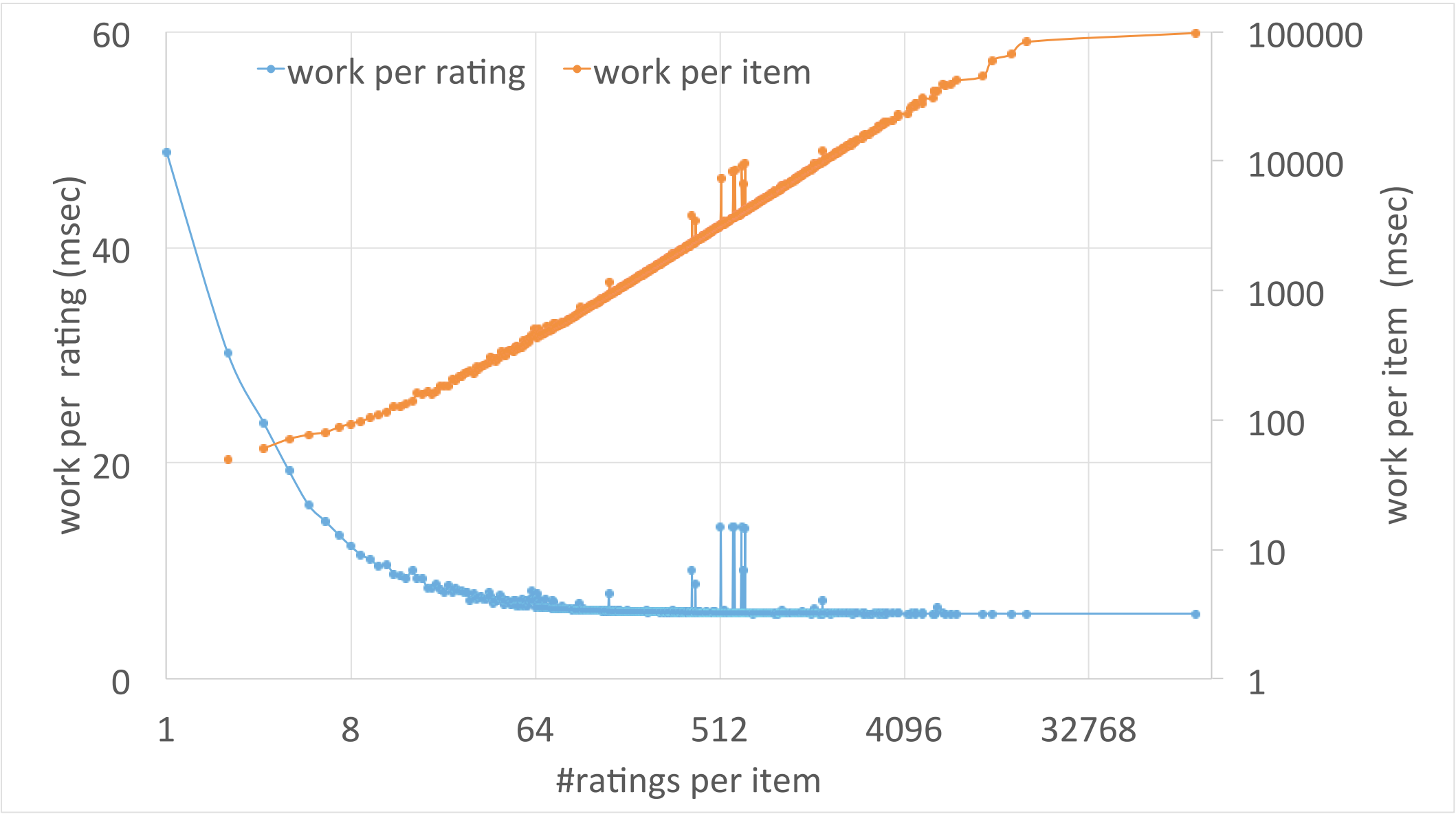
Single Node Parallelization

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

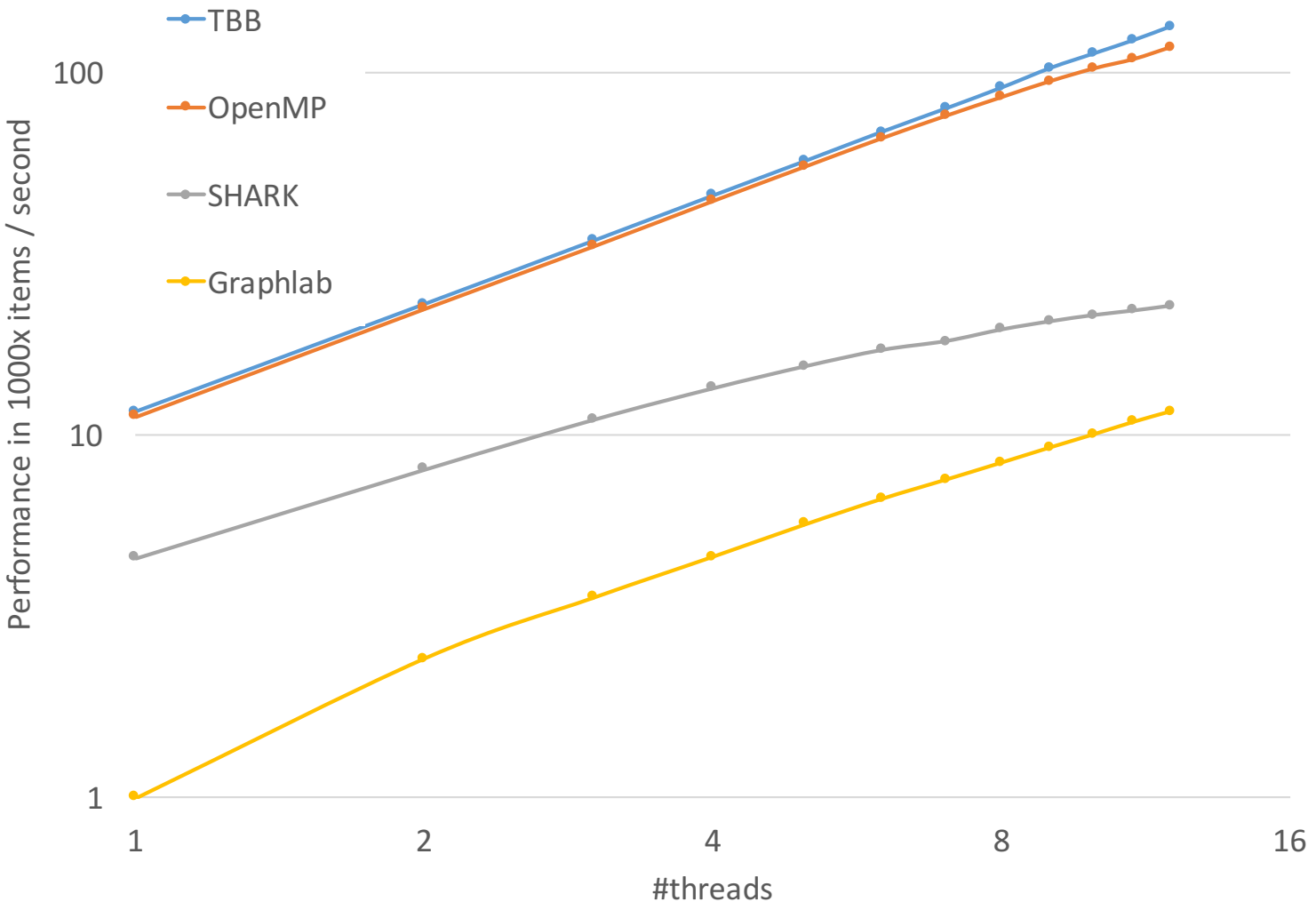


- Shared memory parallelism
  - Memory bandwidth limited
  - Optimize rank R matrix for locality

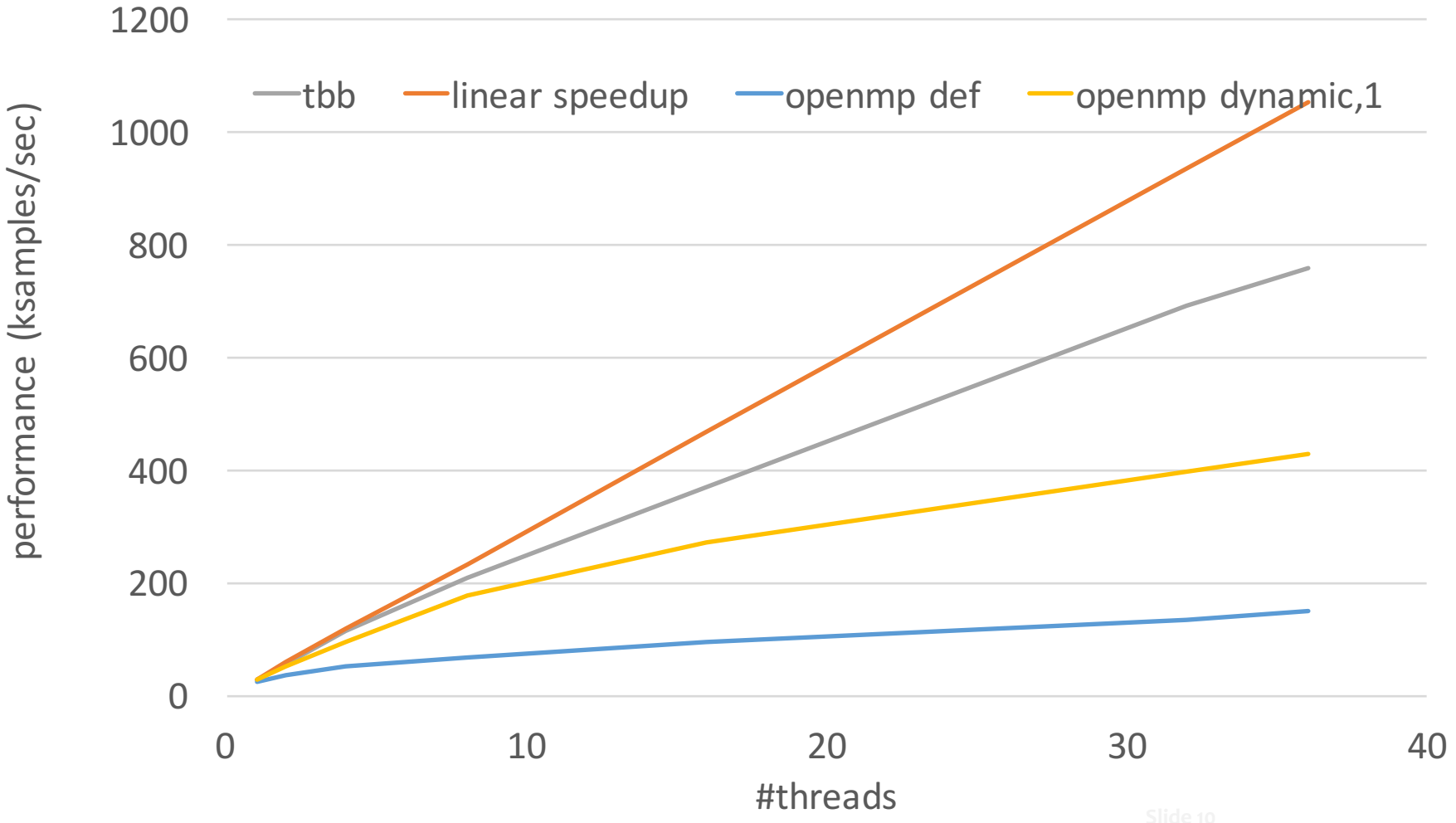
- Load Balancing
  - Load depends on #items, and #nnz
  - Use TBB nested parallelism across + inside items


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

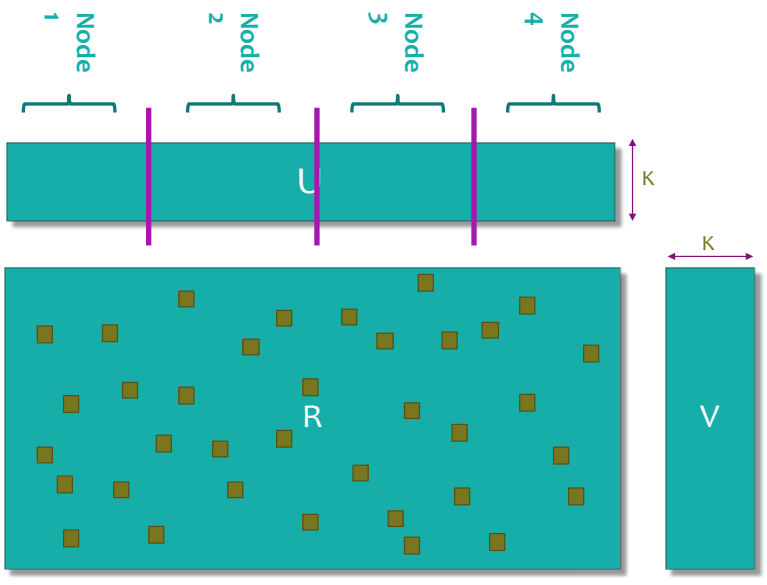
Single Node Performance

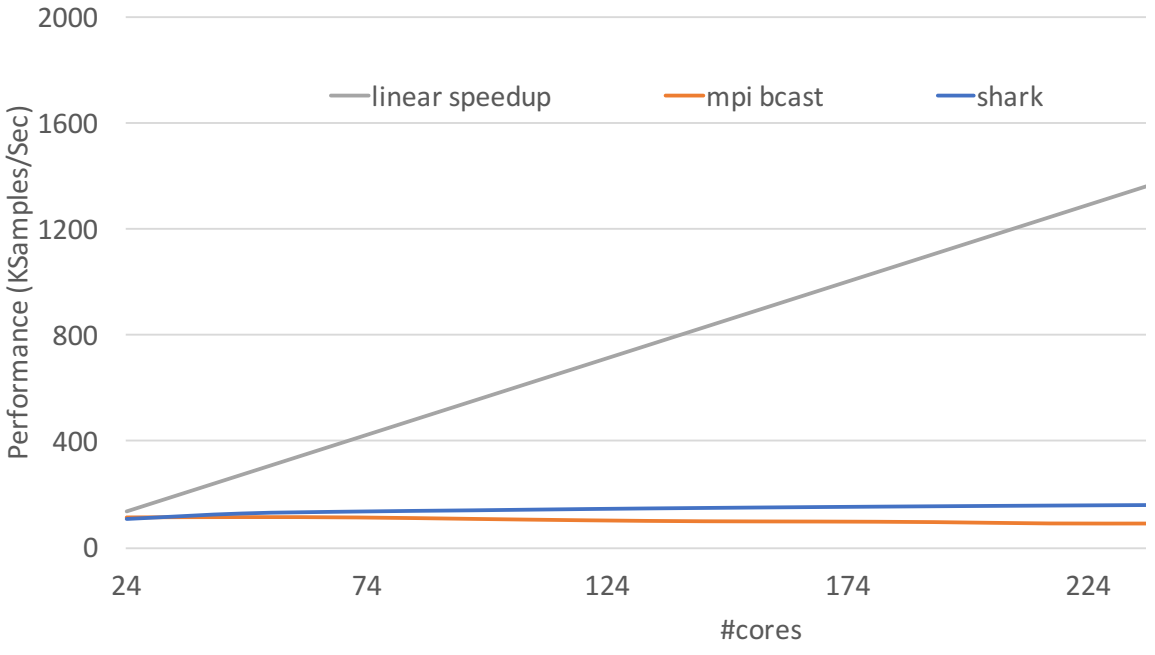


Single Node Performance



Synchronous Distributed BPMF





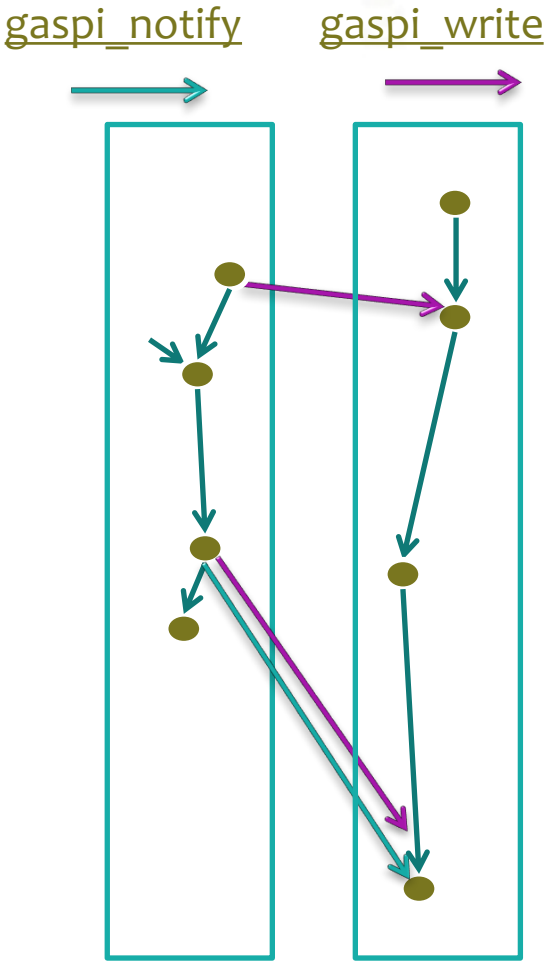
Only 2x improvement with 20x more cores

We need asynchronous communication

GASPI in a nutshell

PGAS API - designed to be

- Multithreaded
- Global asynchronous dataflow
- Interoperability with MPI

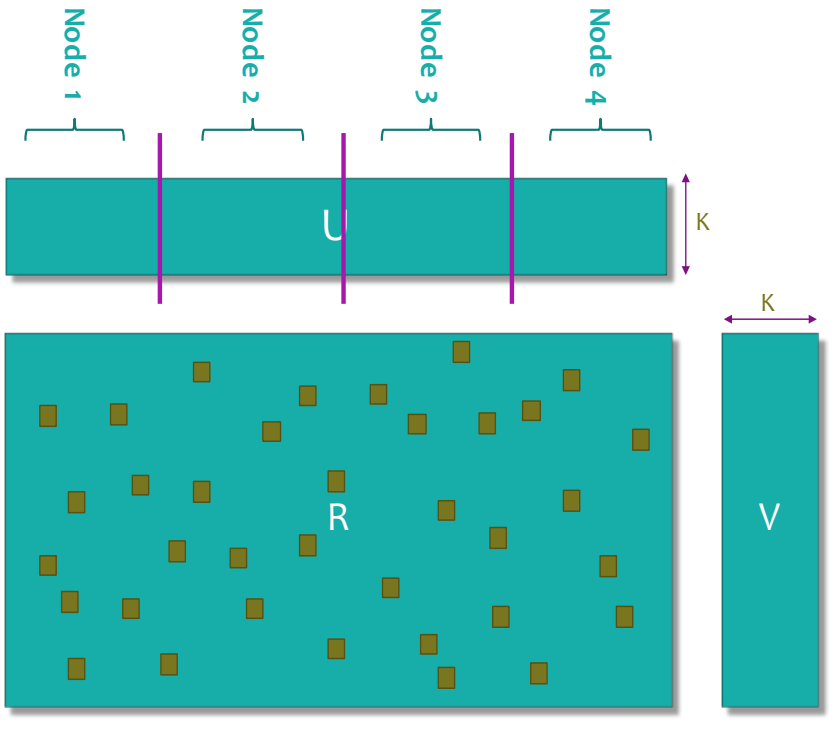


Asynchronous Distributed BPMF

- Split both U/V, optimizing:
  1. Load balance (# rows, #nnz)
  2. Communication

  ➔ Both are determined by R

- Basic Pattern GASPI
  - Compute a column of U/V
  - Send to needed nodes

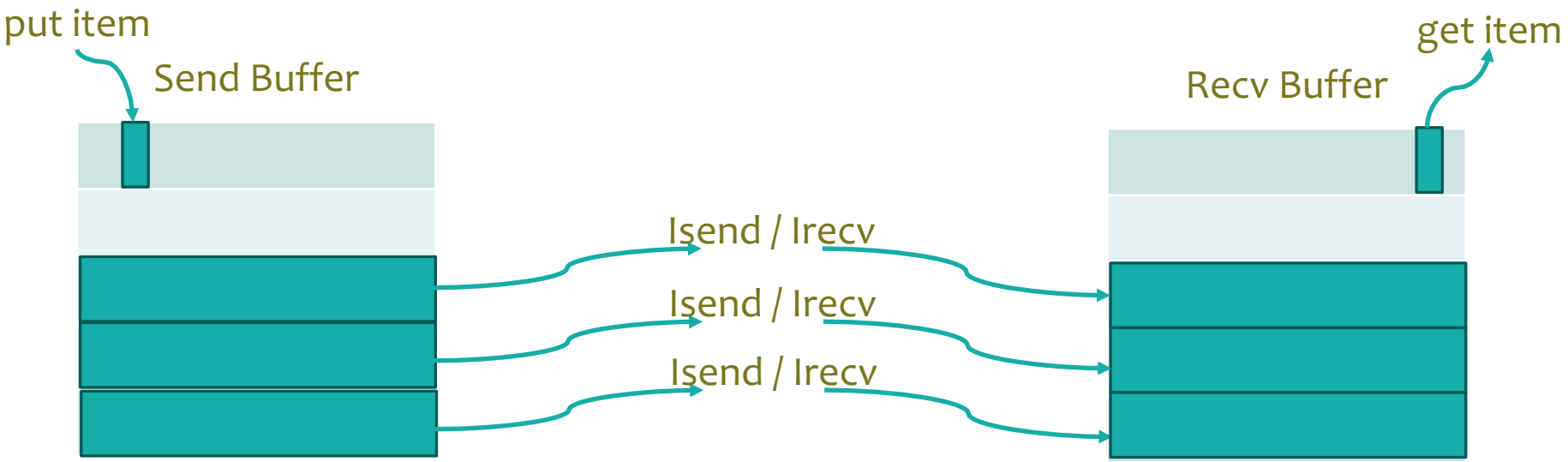


Challenges using MPI

- 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

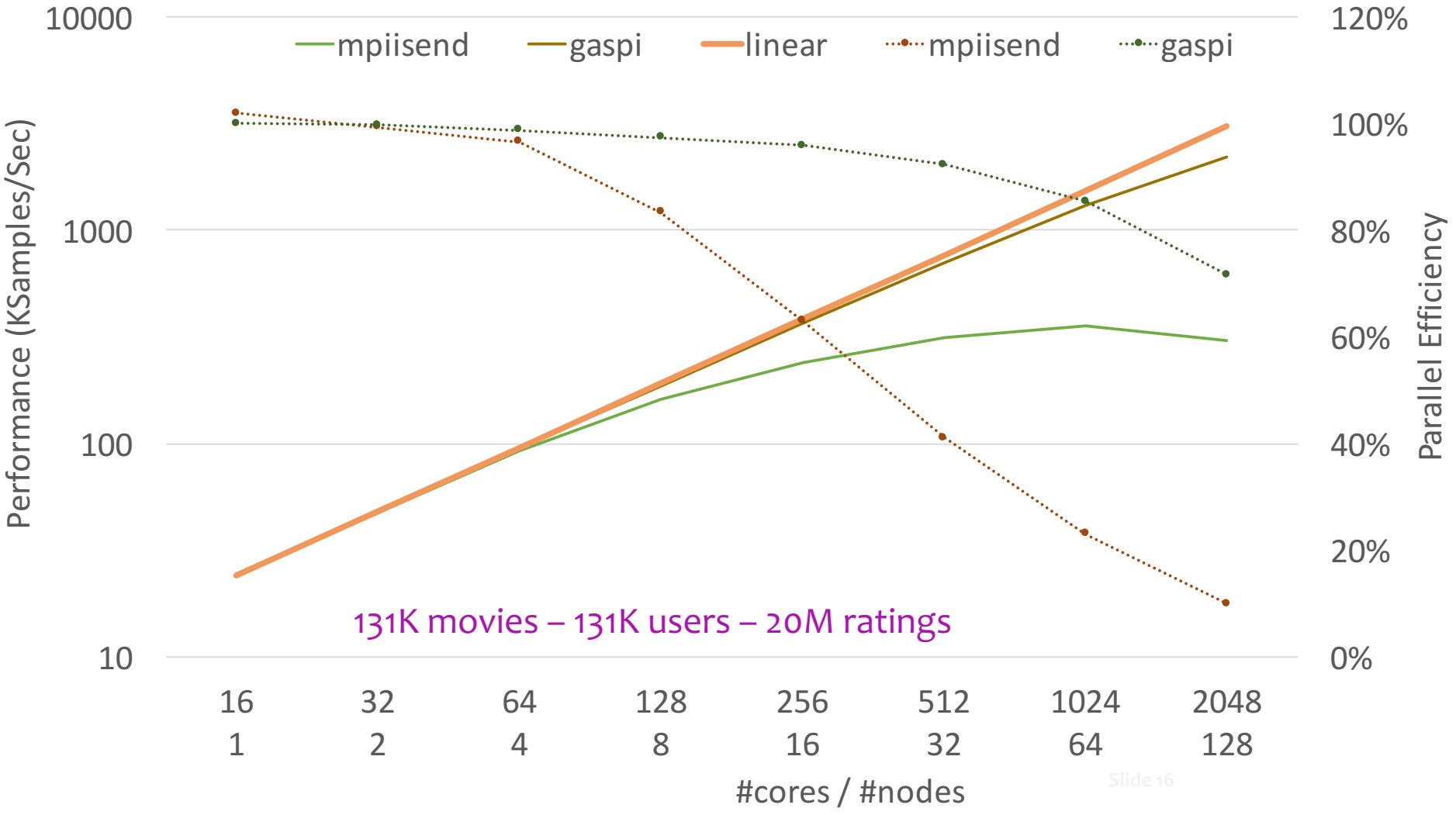
Current best MPI implementation

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

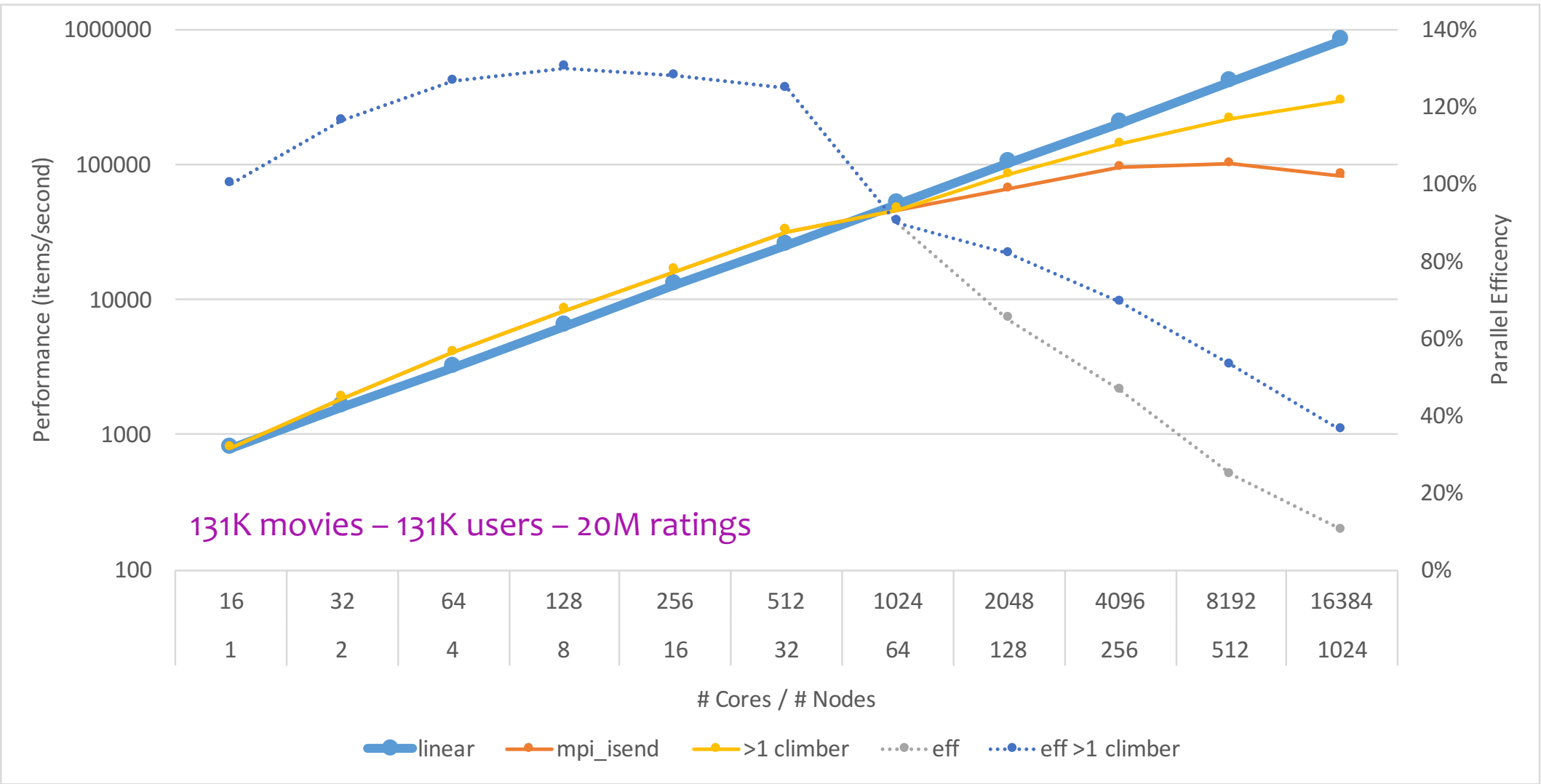


- 5 chunks → maximally 5 Isend/Irecv in flight
- 100 items per chunk

Distributed Performance – 128 nodes



Distributed Performance – 1K nodes



Conclusions optimizing BPMF

- Reduce runtime on JnJ 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