Parallel Sparse Tensor Decomposition in Chapel

Thomas B. Rolinger, Tyler A. Simon,
Christopher D. Krieger

IPDPSW 2018
CHIUW

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

Outline

1. Motivation and Background
2. Porting SPLATT to Chapel
3. Performance Evaluation: Experiments, modifications and optimizations
4. Conclusions

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

Motivation and Background

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

1.) Motivation: Tensors + Chapel

- Why focus on Chapel for this work?
  - Tensor decompositions algorithms are complex and immature
    - Expressiveness and simplicity of Chapel would promote maintainable and extensible code
    - High performance is crucial as well

1.) Motivation: Tensors + Chapel

- Why focus on Chapel for this work?
  - Tensor decompositions algorithms are complex and immature
    - Expressiveness and simplicity of Chapel would promote maintainable and extensible code
    - High performance is crucial as well
  - Existing tensor tools are based on C/C++ and OpenMP+MPI
    - No implementations within Chapel (or similar framework)

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

1.) Background: **Tensors**

- Tensors: Multidimensional arrays
  - Typically very large and sparse
    - Can have billions of non-zeros and densities on the order of 10^{-10}

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

LPS
The Laboratory for Physical Sciences

1.) Background: Tensors

- Tensors: Multidimensional arrays
  - Typically very large and sparse
    - Can have billions of non-zeros and densities on the order of 10^{-10}
- Tensor Decomposition:
  - Higher-order extension of matrix singular value decomposition (SVD)
  - CP-ALS: Alternating Least Squares
    - Critical routine: Matricized tensor times Khatri-Rao product (MTTKRP)

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

1.) Background: SPLATT

- SPLATT: The Surprisingly ParalleL spArse Tensor Toolkit
  - Developed by University of Minnesota (Smith, Karypis)
  - Written in C with OpenMP+MPI hybrid parallelism
- Current state of the art in tensor decomp.
- We focus on SPLATT's the shared-memory (single locale) implementation of CP-ALS for this work
- Porting SPLATT to Chapel serves as a "stress test" for Chapel
  - File I/O, BLAS/LAPACK interface, custom data structures and non-trivial parallelized routines

Porting SPLATT to Chapel

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

2.) Porting SPLATT to Chapel: Overview

- **Goal**: simplify SPLATT code when applicable but preserve original implementation and design
- Single-locale port
  - Multi-locale port left for future work
- Mostly a straightforward port
  - However, there were some cases that required extra effort to port: **mutex/locks**, work sharing constructs, jagged arrays

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

LPS
The Laboratory for Physical Sciences

2.) Porting SPLATT to Chapel:

Mutex Pool

- SPLATT uses a mutex pool for some of the parallel MTTKRP routines to synchronize access to matrix rows
- Chapel currently does not have a native lock/mutex module
  - Can recreate behavior with **sync** or **atomic** variables
  - We originally used **sync** variables, but later switched to **atomic** (see Performance Evaluation section).

```
proc set(pool : [] atomic bool, lockID : int) {
  while pool[lockID].testAndSet() {
    chpl_task_yield();
  }
}
proc unset(pool : [] atomic bool, lockID : int) {
  pool[lockID].clear();
}
```

Performance Evaluation

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

4.) Performance Evaluation: **Set Up**

- Compare performance of Chapel port of original C/OpenMP code
- Default Chapel 1.16 build (Qthreads, jemalloc)
- OpenBLAS for BLAS/LAPACK
- Ensured both C and Chapel code utilize same # of threads for each trial
  - OMP_NUM_THREADS
  - CHPL_RT_NUM_THREADS_PER_LOCALE

4.) Performance Evaluation : Datasets

Name	Dimensions	Non-Zeros	Density	Size on Disk
YELP	**41k x 11k x 75k**	**8 million**	**1.97E-7**	**240 MB**
RATE-BEER	27k x 105k x 262k	62 million	8.3E-8	1.85 GB
BEER-ADVOCATE	31k x 61k x 182k	63 million	1.84E-7	1.88 GB
NELL-2	**12k x 9k x 29k**	**77 million**	**2.4E-5**	**2.3 GB**
NETFLIX	480k x 18k x 2k	100 million	5.4E-6	3 GB

See paper for more details on data sets

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

4.) Performance Evaluation: Summary

- Profiled and analyzed Chapel code
  - Initial code exhibited very poor performance
- Identified 3 major bottlenecks
  - MTTKRP: up to **163x slower** than C code
  - Matrix inverse: up to **20x slower** than C code
  - Sorting (refer to paper for details)
- After modifications to initial code
  - Achieved competitive performance to C code

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

LPS
The Laboratory for Physical Sciences

4.) Performance Evaluation :

MTTKRP Optimizations: Matrix Row Accessing

```
double *mat = ...; // row-major 1D array
double *row = mat + (i*cols);
for(int j = 0; j < cols; j++) {
   row[j]...
}
```

Original C: number of cols is small (35) but number of rows is large (tensor dims)

4.) Performance Evaluation :

MTTKRP Optimizations: Matrix Row Accessing

```
double *mat = ...; // row-major 1D array
double *row = mat + (i*cols);
for(int j = 0; j < cols; j++) {
   row[j]...
}
```

Original C: number of cols is small (35) but number of rows is large (tensor dims)

```
var mat [0..rows-1,0..cols-1] = ...;
ref row = mat[i,0..cols-1];
for j in 0..cols-1 {
   row[j]...
}
```

Initial Chapel: use slicing to get row reference → very slow since cost of slicing is not amortized by computation on each slice

4.) Performance Evaluation :

MTTKRP Optimizations: Matrix Row Accessing

```
double *mat = ...; // row-major 1D array
double *row = mat + (i*cols);
for(int j = 0; j < cols; j++) {
   row[j]...
}
```

Original C: number of cols is small (35) but number of rows is large (tensor dims)

```
var mat [0..rows-1,0..cols-1] = ...;
ref row = mat[i,0..cols-1];
for j in 0..cols-1 {
   row[j]...
}
```

Initial Chapel: use slicing to get row reference → very slow since cost of slicing is not amortized by computation on each slice

```
var mat [0..rows-1,0..cols-1] = ...;
for j in 0..cols-1 {
   row[i,j]...
}
```

2D Index: use (i,j) index into original matrix instead of getting row reference → 17x speed up over initial MTTKRP code

4.) Performance Evaluation :

MTTKRP Optimizations: Matrix Row Accessing

```
double *mat = ...; // row-major 1D array
double *row = mat + (i*cols);
for(int j = 0; j < cols; j++) {
   row[j]...
}
```

Original C: number of cols is small (35) but number of rows is large (tensor dims)

```
var mat [0..rows-1,0..cols-1] = ...;
ref row = mat[i,0..cols-1];
for j in 0..cols-1 {
   row[j]...
}
```

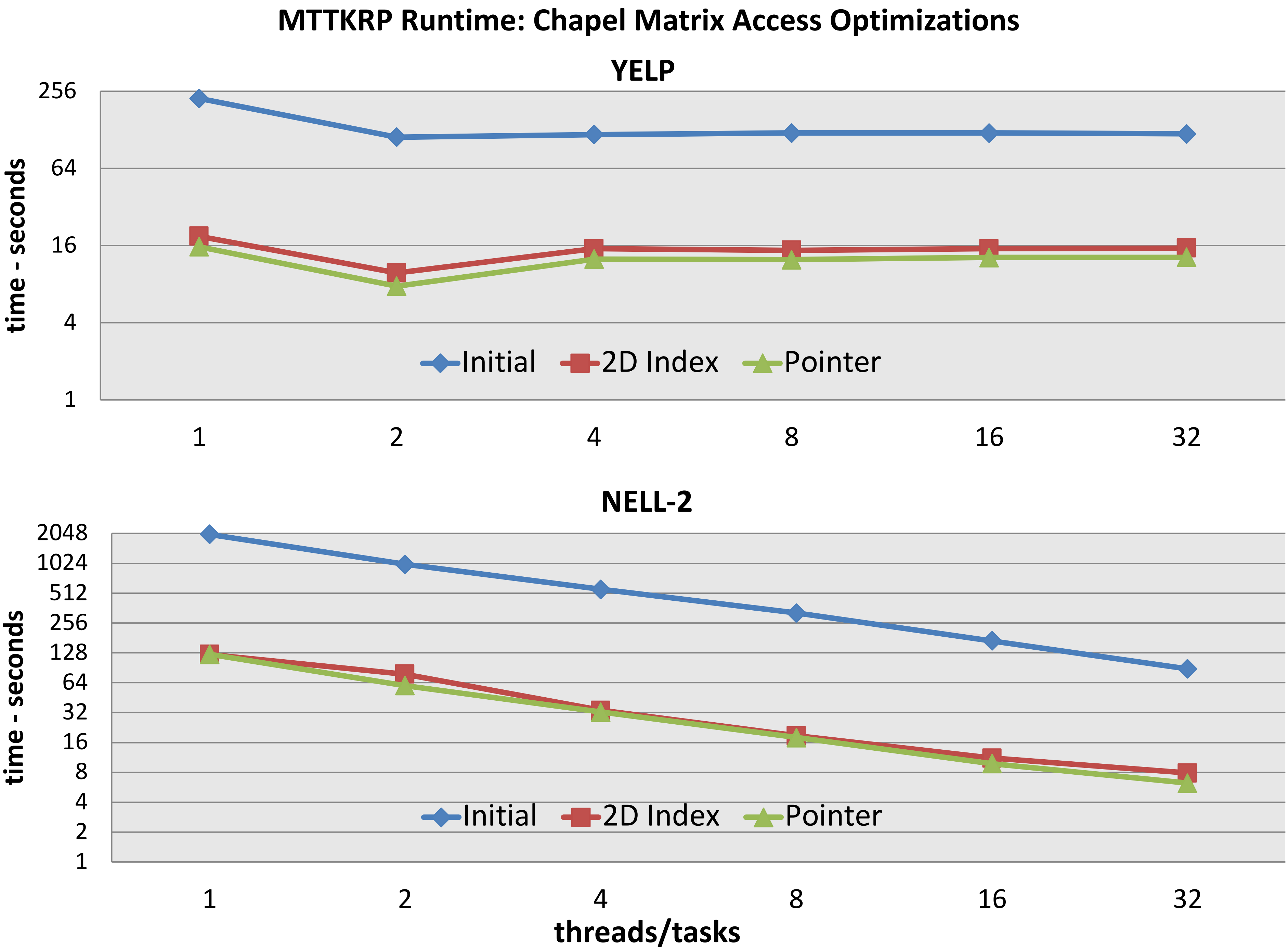
Initial Chapel: use slicing to get row reference → very slow since cost of slicing is not amortized by computation on each slice

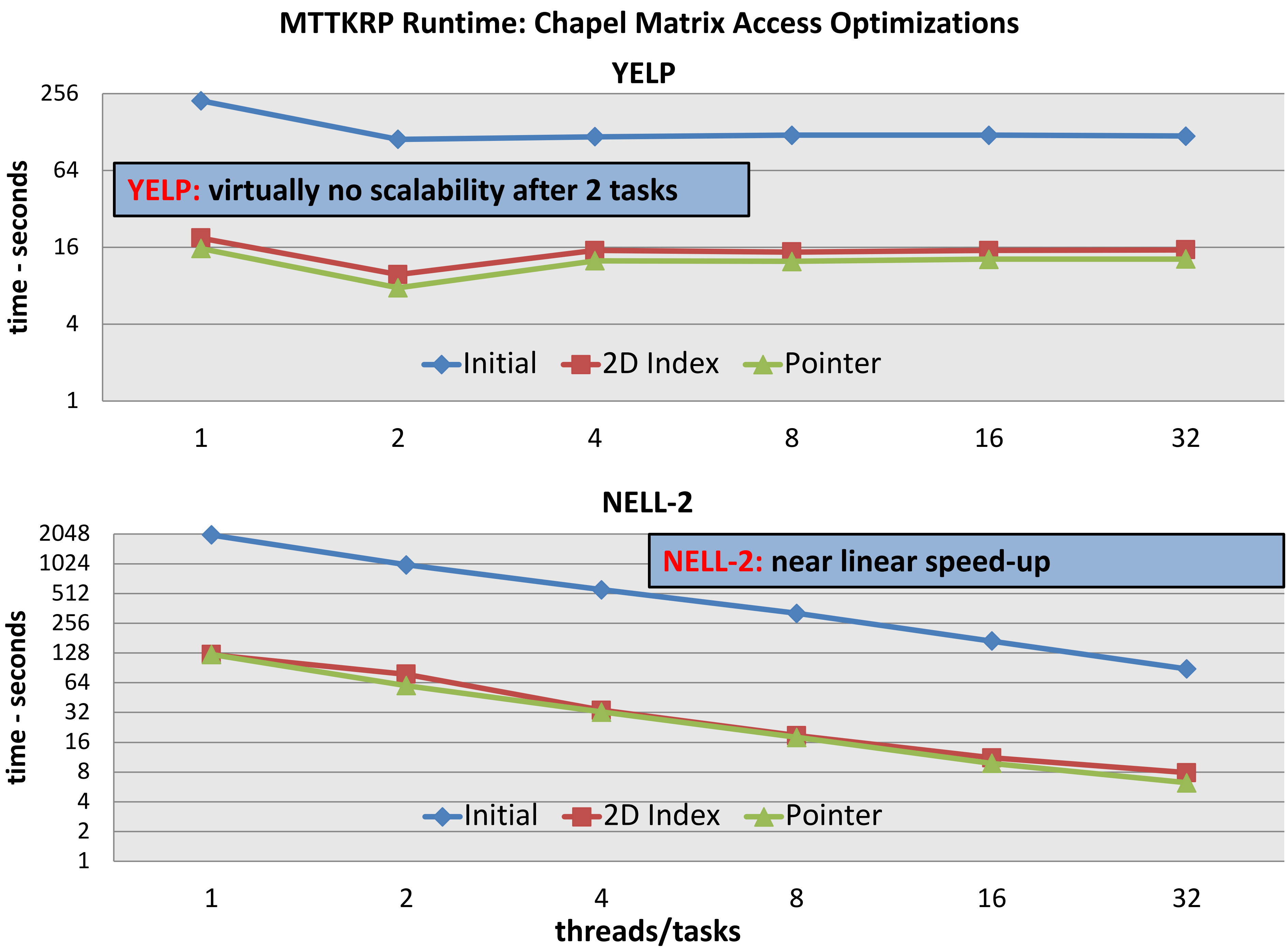
```
var mat [0..rows-1,0..cols-1] = ...;
for j in 0..cols-1 {
   row[i,j]...
}
```

2D Index: use (i,j) index into original matrix instead of getting row reference → 17x speed up over initial MTTKRP code

```
var mat [0..rows-1,0..cols-1] = ...;
var matPtr = c_ptrTo(mat);
var row = matPtr + (i*cols);
for j in 0..cols-1 {
   row[j]...
}
```

Pointer: more direct C translation → 1.26x speed up over 2D indexing


MTTKRP Runtime: Chapel Matrix Access Optimizations
YELP
256
64
16
4
1
time - seconds
Initial
2D Index
Pointer
1
2
4
8
16
32
NELL-2
2048
1024
512
256
128
64
32
16
8
4
2
1
time - seconds
Initial
2D Index
Pointer
1
2
4
8
16
32
threads/tasks



MTTKRP Runtime: Chapel Matrix Access Optimizations
YELP
time - seconds
256
64
16
4
1
YELP: virtually no scalability after 2 tasks
Initial
2D Index
Pointer
1
2
4
8
16
32
NELL-2
NELL-2: near linear speed-up
2048
1024
512
256
128
64
32
16
8
4
2
1
Initial
2D Index
Pointer
threads/tasks

4.) Performance Evaluation :

MTTKRP Optimizations: **Mutex/Locks**

- YELP requires the use of locks during the MTTKRP and NELL-2 does not
  - Decision whether to use locks is highly dependent on tensor properties and number of threads used

4.) Performance Evaluation : MTTKRP Optimizations: Mutex/Locks

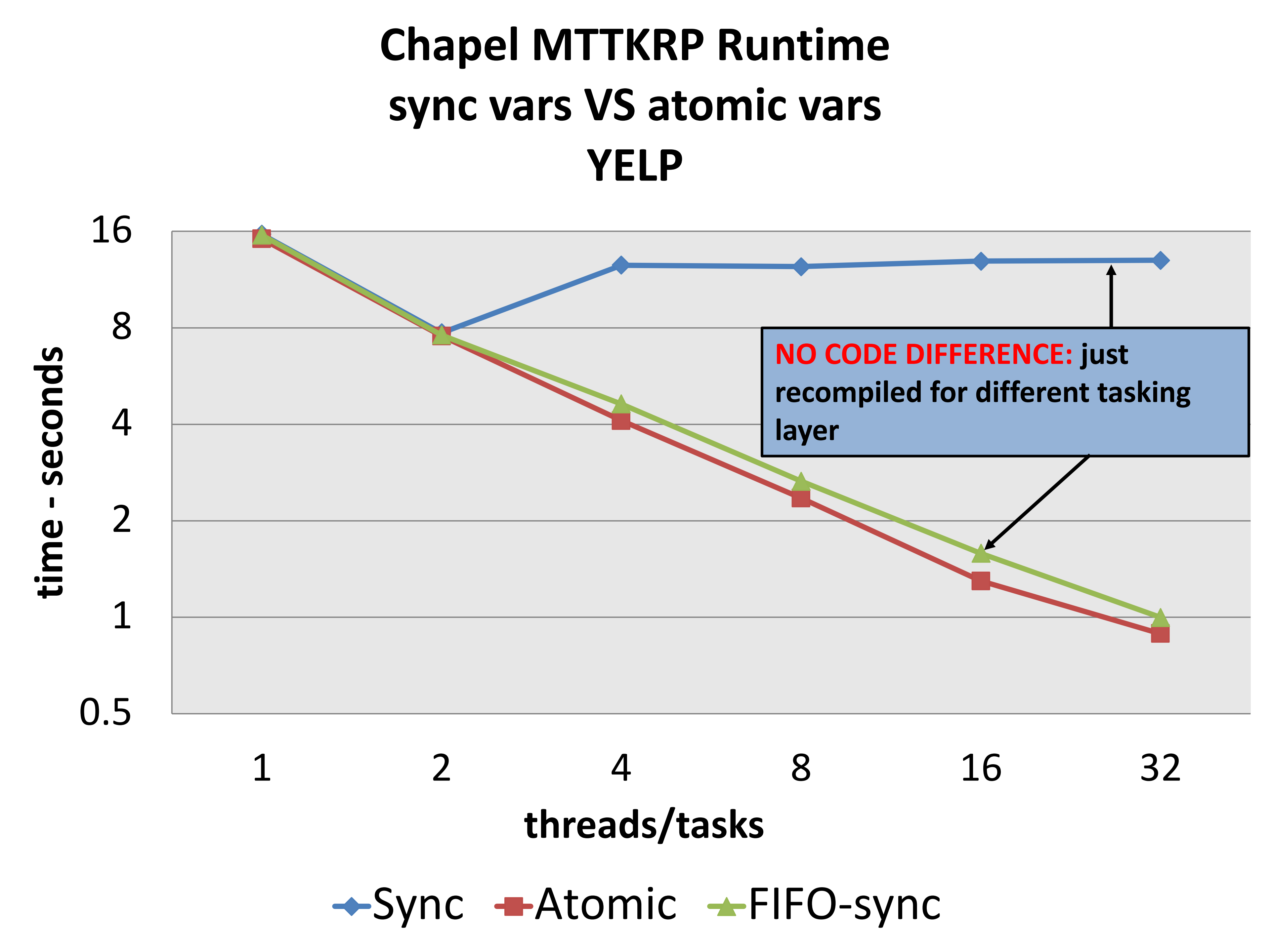
- YELP requires the use of locks during the MTTKRP and NELL-2 does not
  - Decision whether to use locks is highly dependent on tensor properties and number of threads used
- Initially used **sync** vars
  - MTTKRP critical regions are short and fast
    - Not well suited for how **sync** vars are implemented in Qthreads
  - Switched to **atomic** vars
    - Up to **14x** improvement on YELP

4.) Performance Evaluation : MTTKRP Optimizations: Mutex/Locks

- YELP requires the use of locks during the MTTKRP and NELL-2 does not
  - Decision whether to use locks is highly dependent on tensor properties and number of threads used
- Initially used **sync** vars
  - MTTKRP critical regions are short and fast
    - Not well suited for how **sync** vars are implemented in Qthreads
  - Switched to **atomic** vars
    - Up to **14x** improvement on YELP
- FIFO w/ **sync** vars competitive with Qthreads w/ **atomic** vars
  - Troubling: just recompiling the code can drastically alter performance

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences



Chapel MTTKRP Runtime
sync vars VS atomic vars
YELP
16
8
4
2
1
0.5
time - seconds
1
2
4
8
16
32
threads/tasks
NO CODE DIFFERENCE: just recompiled for different tasking layer
Sync
Atomic
FIFO-sync

4.) Performance Evaluation :
Matrix Inverse (OpenBLAS/OpenMP)

- SPLATT uses LAPACK routines to compute matrix inverse
  - Experiments used OpenBLAS, parallelized via OpenMP

4.) Performance Evaluation : Matrix Inverse (OpenBLAS/OpenMP)

- SPLATT uses LAPACK routines to compute matrix inverse
  - Experiments used OpenBLAS, parallelized via OpenMP
- Observed **15x slow down** in matrix inverse runtime for Chapel when using 32 threads (OpenMP and Qthreads)
- **Issue:** interaction of Qthreads and OpenMP is messy

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Problem:** OpenMP and Qthreads stomp over each other
- **Reason:** Default → Qthreads pinned to cores
  - OpenMP threads all end up on 1 core due to how Qthreads uses sched_setaffinity
- **Result**: Huge performance loss for OpenMP routine

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

LPS
The Laboratory for Physical Sciences

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Try:** Explicitly bind OpenMP threads to cores
- **Result:** Chapel will fall back to only using 1 thread

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Try:** Explicitly bind OpenMP threads to cores
- **Result:** Chapel will fall back to only using 1 thread
- **Reason:** Same as OpenMP in previous slide
  - Difference: Chapel detects this over subscription and will prevent it by only using 1 thread

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Try:** Explicitly bind OpenMP threads to cores
- **Result:** Chapel will fall back to only using 1 thread
- **Reason:** Same as OpenMP in previous slide
  - Difference: Chapel detects this over subscription and will prevent it by only using 1 thread
- **Problem:** Not always clear to users
  - If CHPL_RT_NUM_THREADS_PER_LOCALE is set, then a warning is displayed about falling back to 1 thread
  - If not, users expect default (# threads == # cores) but only a single thread is used and **no warning given**

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Attempted solutions:**
  - 1.) QT_AFFINITY=no, QT_SPINCOUNT=300
  - 2.) Remove Chapel over subscription warning/check and allow both Qthreads and OpenMP threads to bind to cores

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Attempted solutions:**
  - 1.) QT_AFFINITY=no, QT_SPINCOUNT=300
  - 2.) Remove Chapel over subscription warning/check and allow both Qthreads and OpenMP threads to bind to cores
- **Overall Results:**
  - (1) and (2) provided roughly equal improvement of OpenMP runtime but still **4x slower** than the C code

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Another issue**:
  - Improving OpenMP runtime caused a **7 to 13x slow down** in a Chapel routine that followed
  - Still resource contention on cores

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

LPS
The Laboratory for Physical Sciences

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Another issue**:
  - Improving OpenMP runtime caused a **7 to 13x slow down** in a Chapel routine that followed
  - Still resource contention on cores
- No clear solution to overcome issues
  - We set OMP_NUM_THREADS=1 for Chapel runs since OpenMP runtime is generally negligible

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

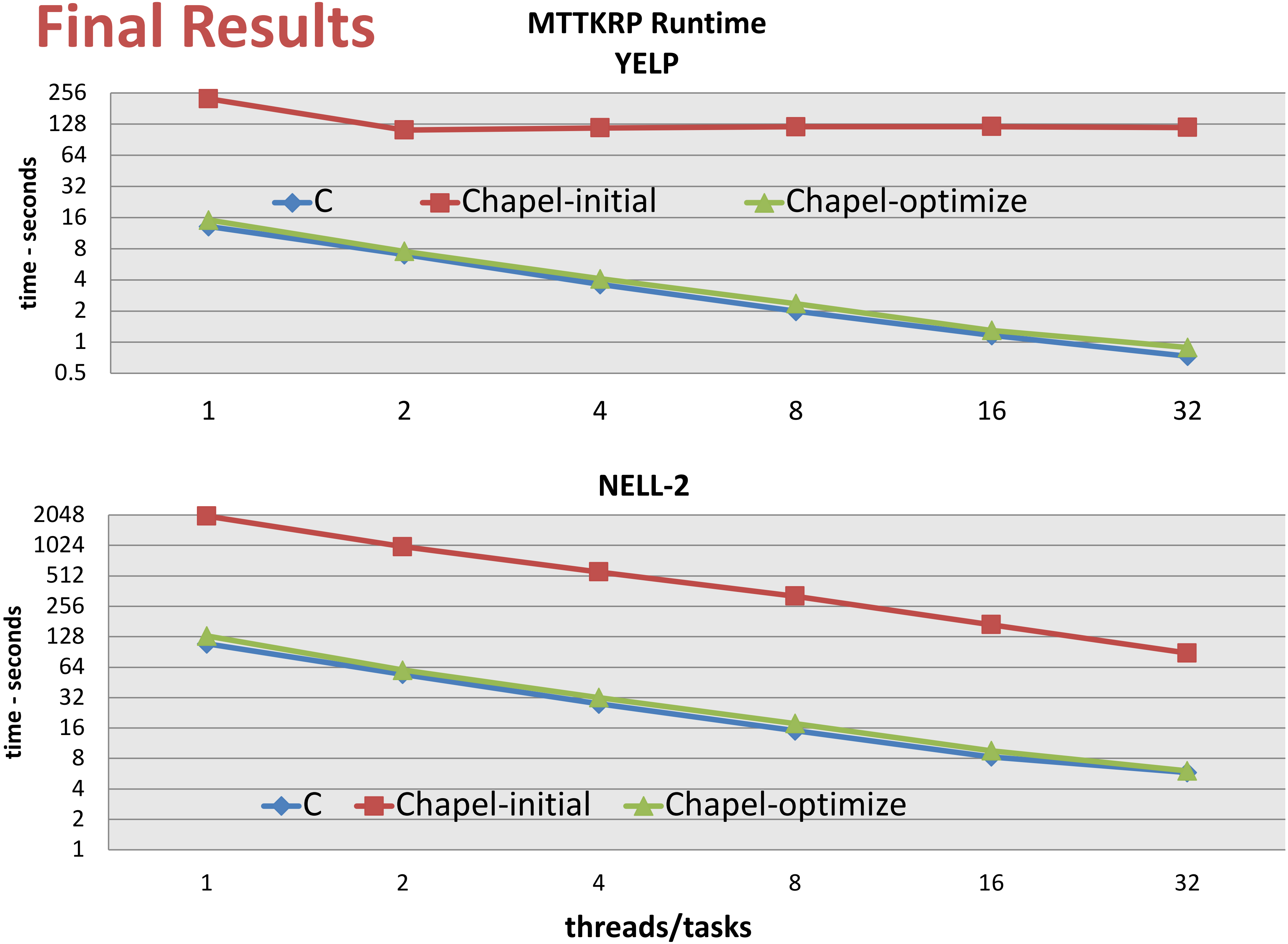
The Laboratory for Physical Sciences

4.) Performance Evaluation :

Matrix Inverse (OpenBLAS/OpenMP) cont.

- **Another issue**:
  - Improving OpenMP runtime caused a **7 to 13x slow down** in a Chapel routine that followed
  - Still resource contention on cores
- No clear solution to overcome issues
  - We set OMP_NUM_THREADS=1 for Chapel runs since OpenMP runtime is generally negligible
- Brings up crucial question regarding library integration:
  - **When does it make sense to provide native Chapel implementations rather than integrate with existing libraries?**

Final Results



5.) Conclusions

- Implemented parallel sparse tensor decomposition in Chapel
- Identified bottlenecks in code
  - Array slicing
  - **sync** vs **atomic** variables for locks
  - Conflicts between OpenMP and Qthreads
- Achieved 83-96% of the original C/OpenMP performance after modifications to initial port
- Suggestions for Chapel:
  - Create a mutex/lock library
  - More documentation/experiments with integrating 3rd party code that utilize different threading libraries
- Future work:
  - Multi-locale version
  - Closer inspection of code to make it more Chapel-like
    - Will the performance suffer or improve?

Questions

Contact: tbrolin@cs.umd.edu

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

Back up Slides

Matricizing a Tensor

$$\mathbf{X}_1 = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}, \quad \mathbf{X}_2 = \begin{bmatrix} 13 & 16 & 19 & 22 \\ 14 & 17 & 20 & 23 \\ 15 & 18 & 21 & 24 \end{bmatrix}$$

$$\mathbf{X}_{(1)} = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 \\ 2 & 5 & 8 & 11 & 14 & 17 & 20 & 23 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \end{bmatrix},$$

$$\mathbf{X}_{(2)} = \begin{bmatrix} 1 & 2 & 3 & 13 & 14 & 15 \\ 4 & 5 & 6 & 16 & 17 & 18 \\ 7 & 8 & 9 & 19 & 20 & 21 \\ 10 & 11 & 12 & 22 & 23 & 24 \end{bmatrix},$$

$$\mathbf{X}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & \cdots & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & \cdots & 21 & 22 & 23 & 24 \end{bmatrix}$$

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

LPS
The Laboratory for Physical Sciences

Kronecker and Khatri-Rao Prodcuts

Kronecker Product

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_1 \otimes \mathbf{b}_2 & \mathbf{a}_1 \otimes \mathbf{b}_3 & \cdots & \mathbf{a}_J \otimes \mathbf{b}_{L-1} & \mathbf{a}_J \otimes \mathbf{b}_L \end{bmatrix}$$

Khatri-Rao Product

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_K \otimes \mathbf{b}_K \end{bmatrix}$$

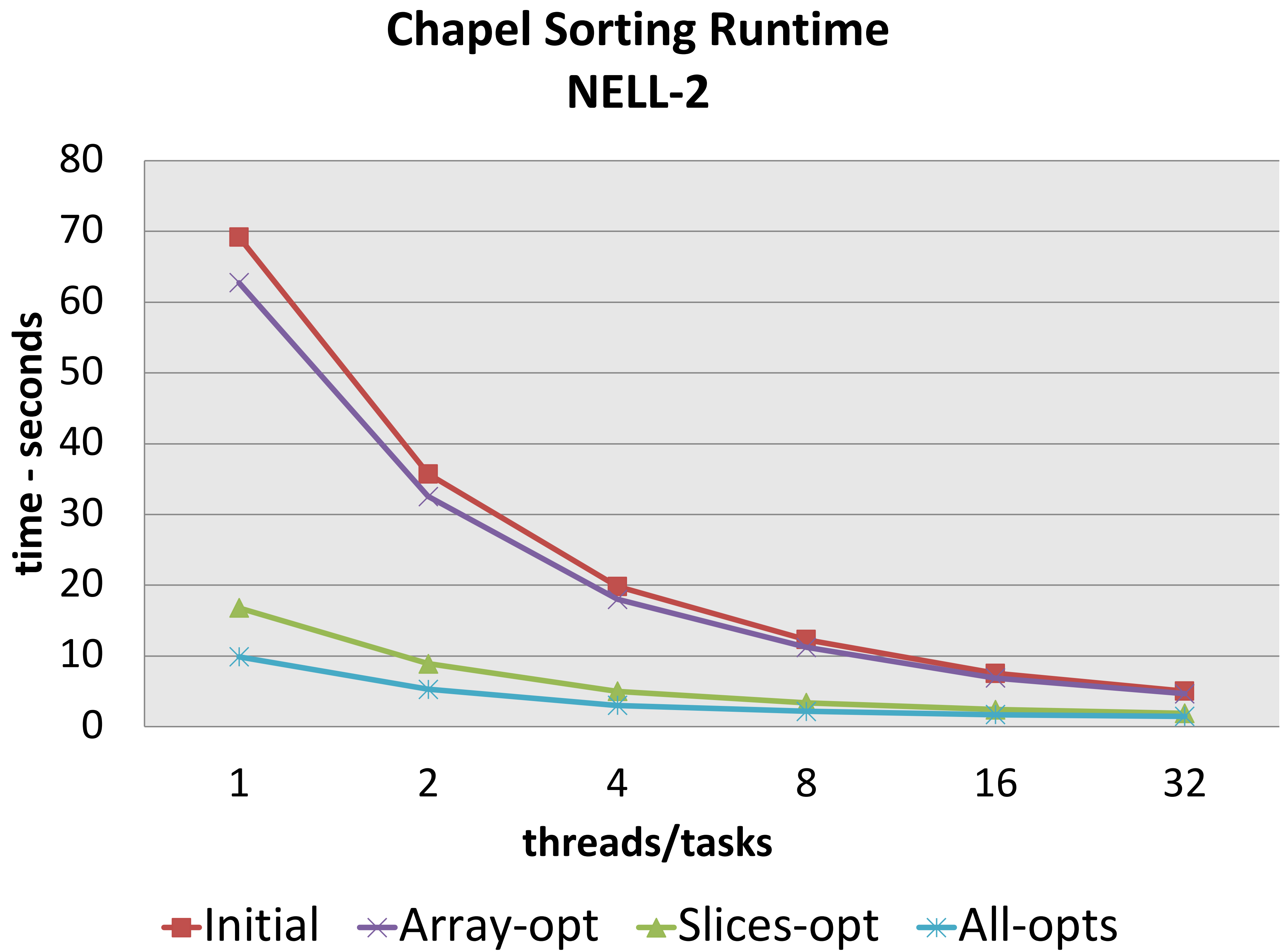
4.) Performance Evaluation :

Sorting Optimizations

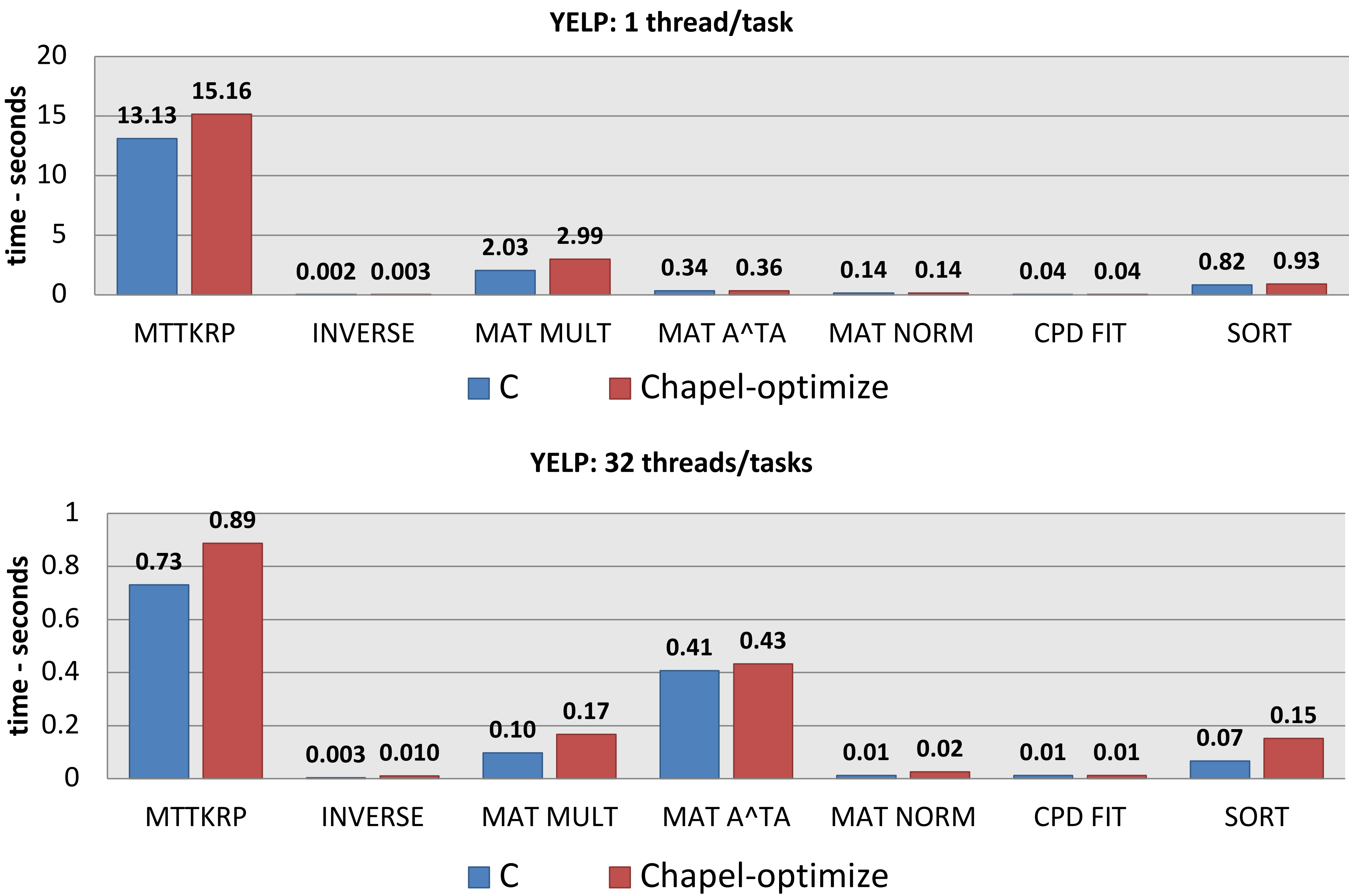
- Profiled customized sorting routine in Chapel code and found two bottlenecks:
  - Creation of small array in recursive routine
    - Created millions of times due to recursion and large tensors: consumed up to 10% of the sorting runtime
    - **Solution**: just declare local ints rather than an array (possible since this array was only of length 2)
  - Reassignment of array of arrays
    - C code: array of pointers → simple pointer assignment
    - Chapel code:
      - Initially 2D matrix → used slicing for reassignment (slow due to large size of slices)
      - Changed to array of arrays → whole array assignment (slow due to copying the arrays)
      - **Final**: get pointer to arrays and use pointer reassignment (similar to C code)
- Modifications resulted in roughly **4x** improvement

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

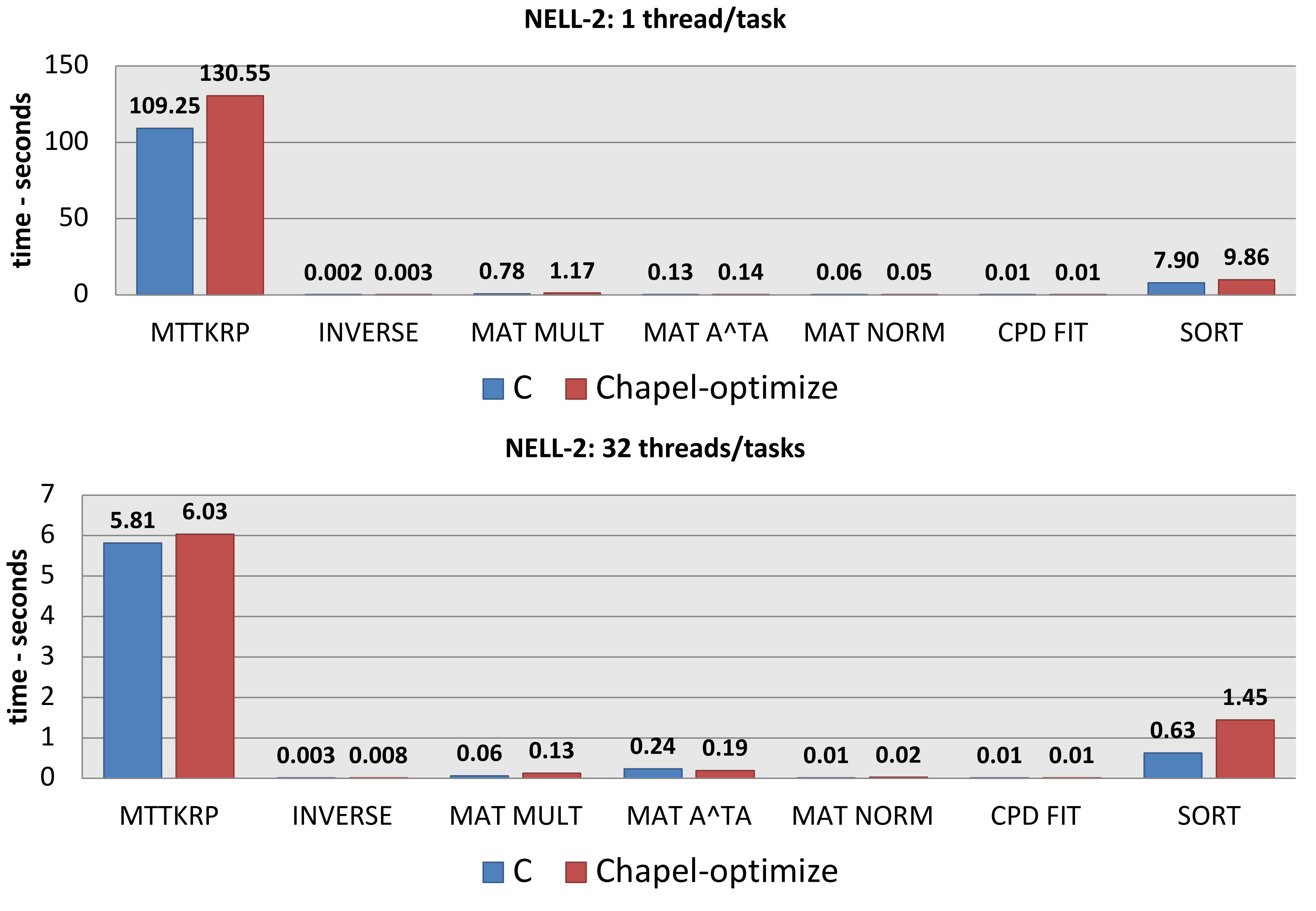
LPS
The Laboratory for Physical Sciences


Chapel Sorting Runtime
NELL-2
time - seconds
80
70
60
50
40
30
20
10
0
1
2
4
8
16
32
threads/tasks
Initial
Array-opt
Slices-opt
All-opts

Runtimes for CP-ALS Routines



Runtimes for CP-ALS Routines



4.) Performance Evaluation :

Initial Results: CP-ALS Routines Runtimes

Data set	Threads/tasks	Code	MTTKRP	Sort	Mat A^TA	Mat Norm	CPD Fit	Inverse
YELP	1	C	13.31	0.82	0.34	0.14	0.04	0.94
		Chapel-Initial	225.11	7.21	0.36	0.14	0.04	0.98
	32	C	0.73	0.07	0.41	0.01	0.01	0.05
		Chapel-Initial	118.93	0.47	0.56	0.06	0.01	0.98
NELL-2	1	C	109.25	7.9	0.13	0.06	0.01	0.37
		Chapel-Initial	1999	69.04	0.14	0.06	0.01	0.39
	32	C	5.81	0.63	0.24	0.01	0.01	0.04
		Chapel-Initial	88.3	5.01	0.19	0.02	0.01	0.39

Times shown in seconds

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

4.) Performance Evaluation :

MTTKRP Optimizations: Mutex/Locks

- YELP requires the use of locks during the MTTKRP and NELL-2 does not
  - Decision whether to use locks is highly dependent on tensor properties and number of threads used

Sync vars (Qthreads)	Atomic vars (Qthreads)	Sync vars (FIFO)
- Tasks put to sleep - Suitable for long-held heavily contended locks	- Tasks spin-wait - Suitable for short, non-intensive critical reigions	- Tasks spin-wait, similar to atomic vars in Qthreads

- Initially used **sync** vars
  - MTTKRP critical regions are short and fast
  - Switching to **atomic** vars gave huge improvement for YELP
- FIFO w/ **sync** vars competitive with Qthreads w/ **atomic** vars
  - troubling: simple recompilation of code can drastically alter performance

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

The Laboratory for Physical Sciences

4.) Performance Evaluation :

Initial Results: CP-ALS Routines Runtimes

Data set	Threads/tasks	Code	MTTKRP	Inverse
YELP	1	C	13.31	0.94
		Chapel	225.11→15.15	0.98
	32	C	0.73	0.05
		Chapel	118.93→0.88	0.98
NELL-2	1	C	109.25	0.37
		Chapel	1999→130.54	0.39
	32	C	5.81	0.04
		Chapel	88.3→6.03	0.39

Times shown in seconds

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

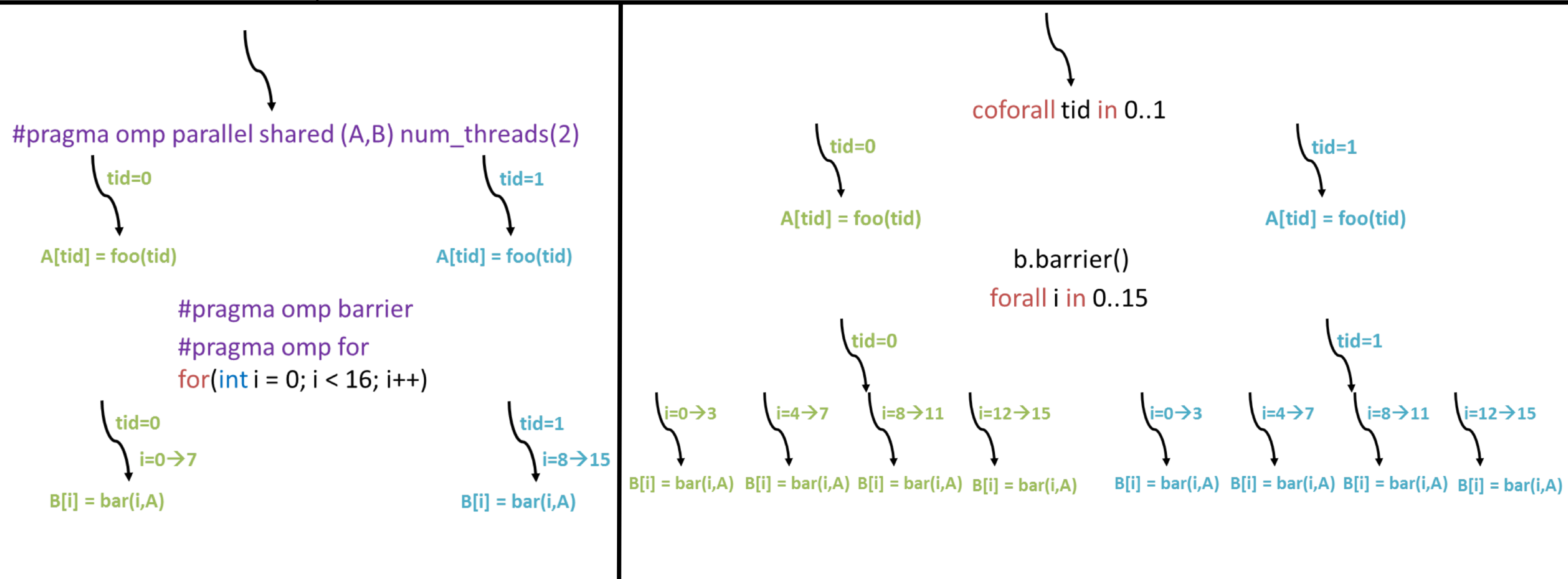
The Laboratory for Physical Sciences

3.) Porting SPLATT to Chapel: Work Sharing Constructs

```
#pragma omp parallel shared(A,B) num_threads(2)
   int tid = omp_get_thread_num();
   A[tid] = foo(tid);
   #pragma omp barrier
   #pragma omp for
   for(int i = 0; i < 16; i++) {
      B[i] = bar(i, A);
   }
}
```

≠

```
coforall tid in 0..1 {
   A[tid] = foo(tid);
   b.barrier();
   forall i in 0..15 {
      B[i] = bar(i, A);
   }
}
```



3.) Porting SPLATT to Chapel:

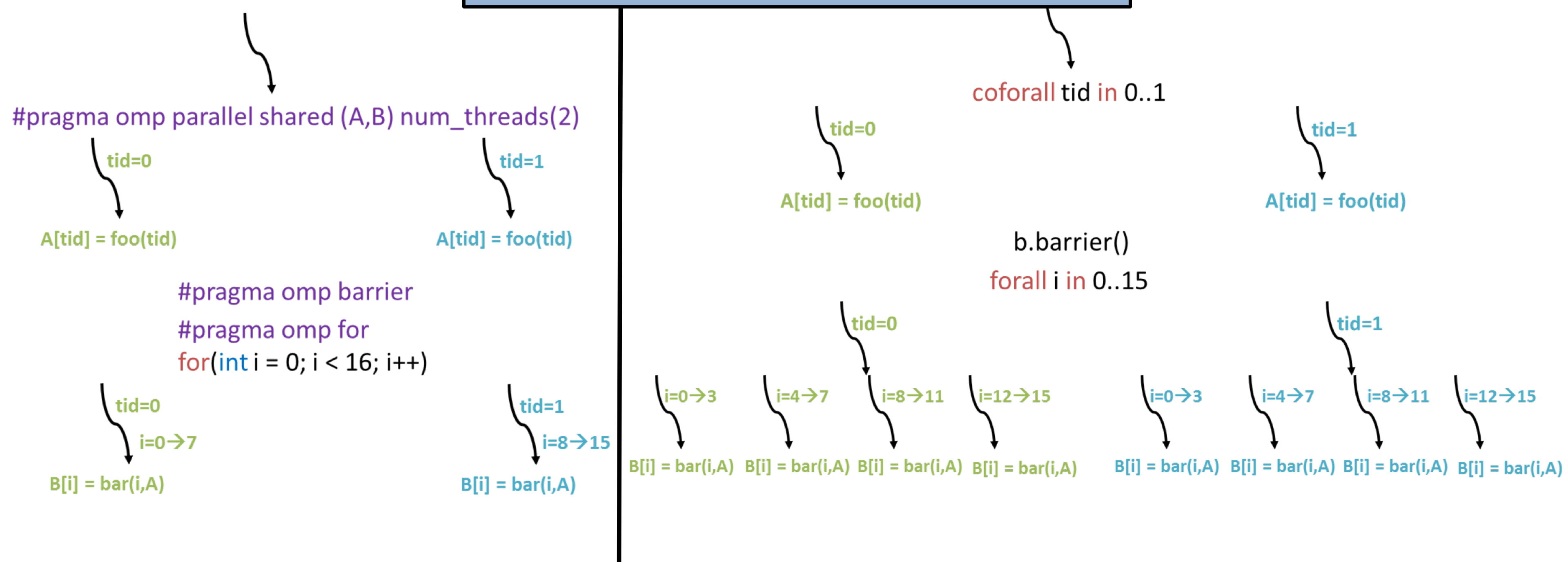
Work Sharing Constructs

```
#pragma omp parallel shared(A,B) num_threads(2)
   int tid = omp_get_thread_num();
   A[tid] = foo(tid);
   #pragma omp barrier
   #pragma omp for
   for(int i = 0; i < 16; i++) {
      B[i] = bar(i, A);
   }
}
```

≠

```
coforall tid in 0..1 {
   A[tid] = foo(tid);
   b.barrier();
   forall i in 0..15 {
      B[i] = bar(i, A);
   }
```

Solution: Manually compute loop bounds for each task



3.) Porting SPLATT to Chapel:
Work Sharing Constructs (cont.)

```
#pragma omp parallel
{
  int tid = omp_get_thread_num();
  double *myVals = thdData[tid];
  #pragma omp for
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      myVals[j] += vals[i][j] * 2;
    }
  }
  // do reduction on myVals
}
```

↓

```
var myVals: [cols] real;
forall r in rows with (+ reduce myVals) do
  for c in cols do
    myVals[c] += vals[r,c] * 2
```

Specific case of perfectly nested loops and partial reduction → clean and concise Chapel translation