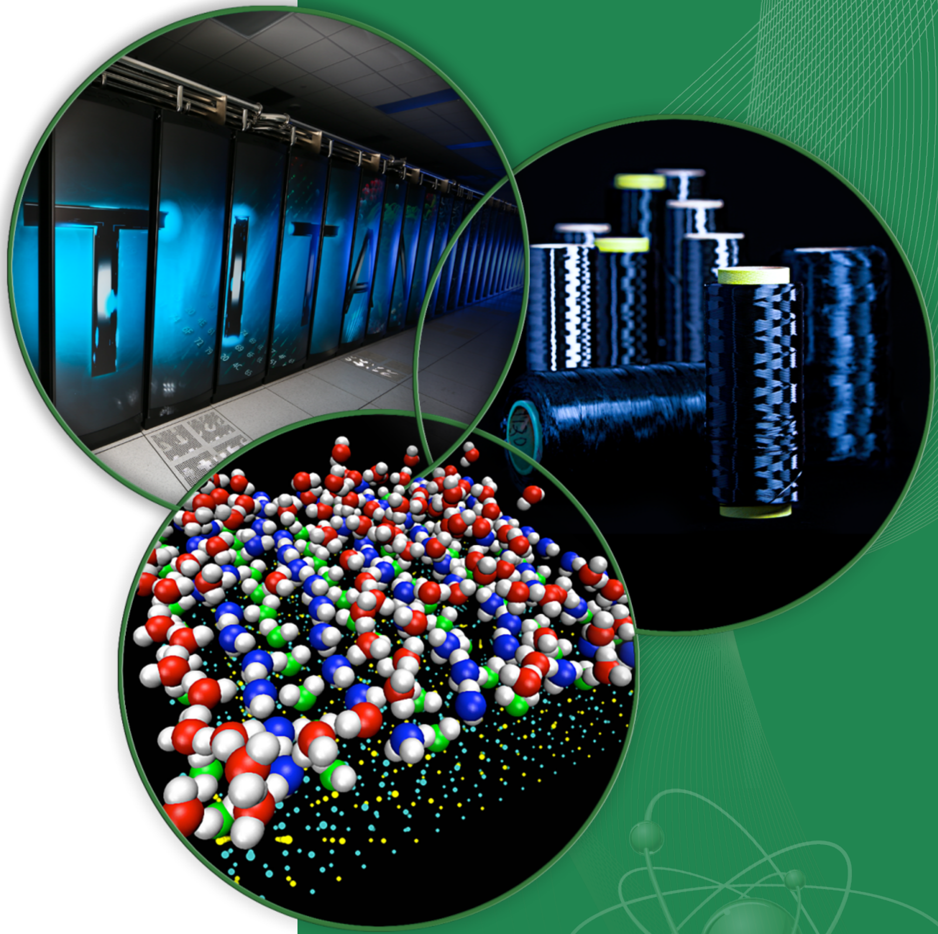
Parallelizing the Smith-Waterman Algorithm using OpenSHMEM and MPI-3 One-Sided Interfaces

Matthew Baker, Aaron Welch and Manjunath Gorentla Venkata



Smith and Waterman Algorithm

- Commonly used pattern matching algorithm (mostly in bioinformatics codes)
- Local alignment algorithm
  - Sub string can be optimal
  - Used for comparing DNA segments
- Dynamic programming algorithm
- 2d score matrix
  - Derived from main sequence length m and match sequence length n
  - Run time is O(mn)

Key optimizations

- Anti-diagonal representation
  - Keeping data as local as possible
  - Reduce memory usage by discarding old anti-diagonals
    - Memory requires reduced from O(mn) to O(m) where m >= n
- Non blocking gets
  - Don't have to wait for data
  - Pre-fetch the next loop data

Key optimizations

- Two loops
  - Outer loop iterates over each antidiagnal
    - Not parallel unfortunately
  - Inner loop iterates over each entry in the antidiagnal
    - Loop independent

Why OpenSHMEM for Smith-Waterman?

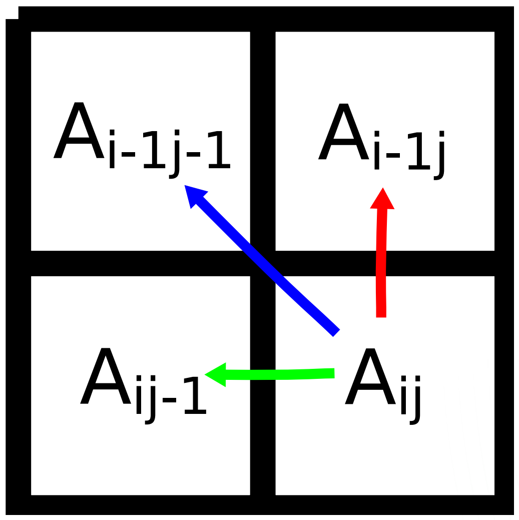
- Simplifies first phase
  - With 2 sided communications each node must compute who wants the local data and what remote data it wants
- Second phase of Smith-Waterman traces backwards in dynamic programming matrix
  - Unstructured and unknown path through matrix
  - Favors short fetches

Computing a score

- Algorithm scores two codon chains looking for matches
- Each matrix element A(i,j) depends on 3 previous matrix entries A(i-1,j) A(i,j-1) and A(i-1,j-1)

Smith and Waterman data dependencies

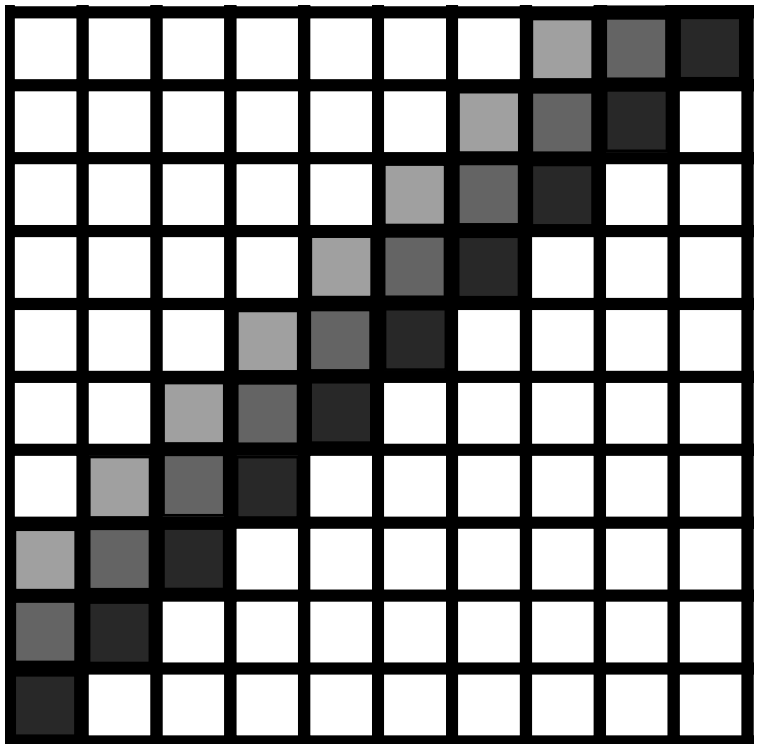
Note how A(i,j) only depends on the pervious two anti-diagnals

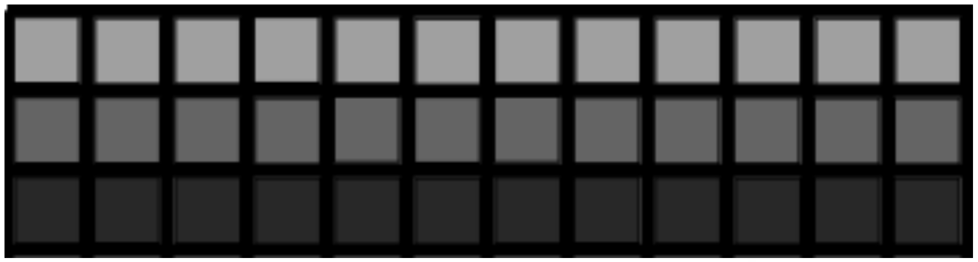


Remapping score matrix

- Naive implementation very bad
  - Allocate whole array
  - Fill in each A(i,j) as data available
  - Wasteful of memory
  - Poor cache performance

Anti-diagonal formatted Array

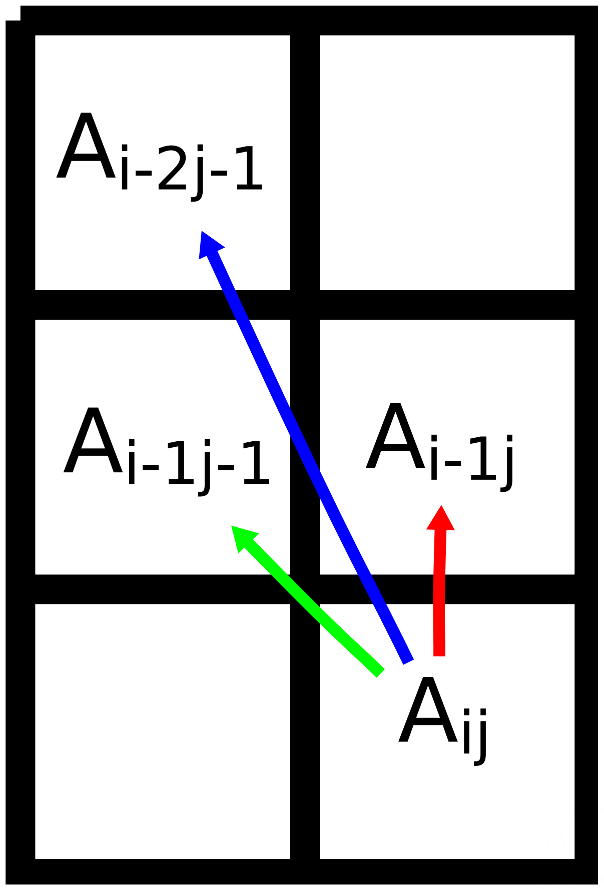




New score matrix shape

- Each row depends on previous two rows
- No row depends on other columns in its

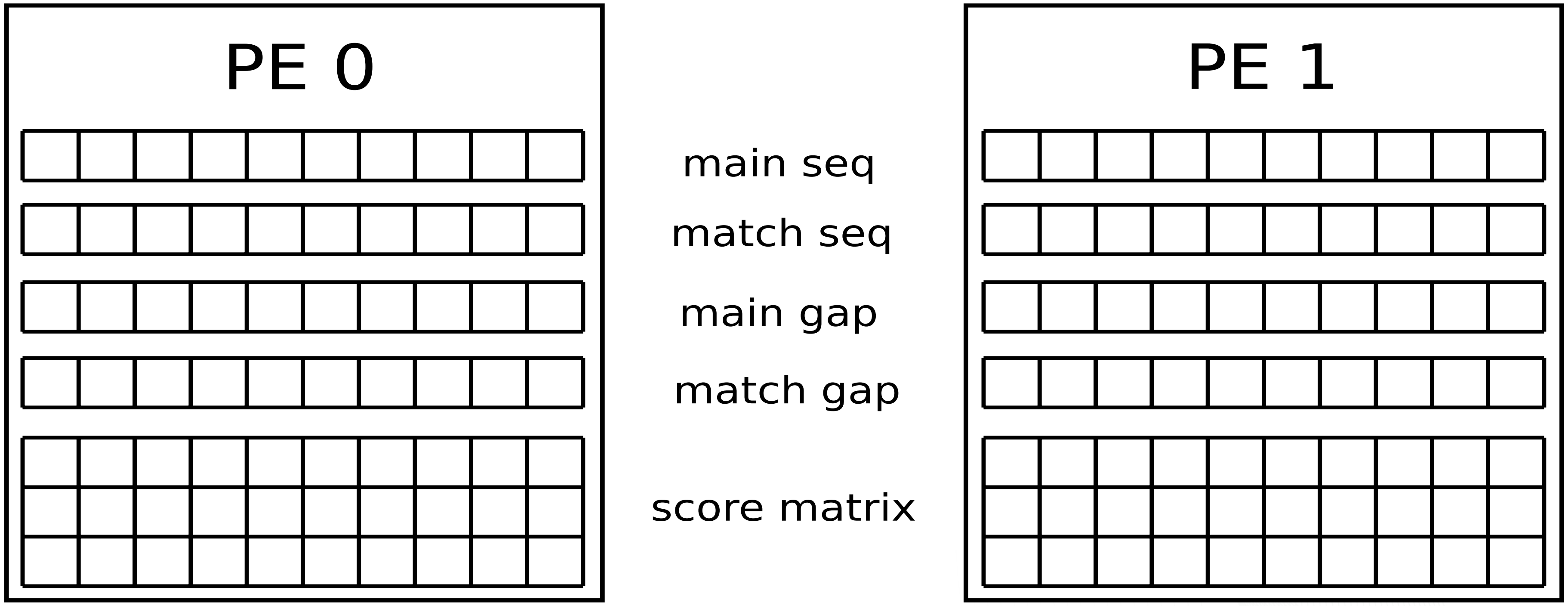
New dependency shape



Distributing Smith-Waterman

- With anti-diagonal format distribution is simple
- All data is split evenly between PEs
  - Main sequence
  - Match sequence
  - Main gap score
  - Match gap score
  - Score matrix

PE data layout



Prefetch

- Computing next indexes trivial
- Needs non-blocking get
  - Uses Cray's non-blocking SHMEM extensions
  - Can also use MPI3 one sided communications

Restructuring loop

- Blocking inner loop in brief
  - Fetch codon from main, match, gaps, previous score
  - Score main and match codons
  - Score gaps
  - Compare new scores and keep best one
  - Update score and gaps with puts

Restructuring loop

- Non-blocking inner loop
  - Wait for previous gets
  - New gets for codon from main, match, gaps, previous score
  - Score main and match codons
  - Score gaps
  - Compare new scores and keep best one
  - Update score and gaps with puts

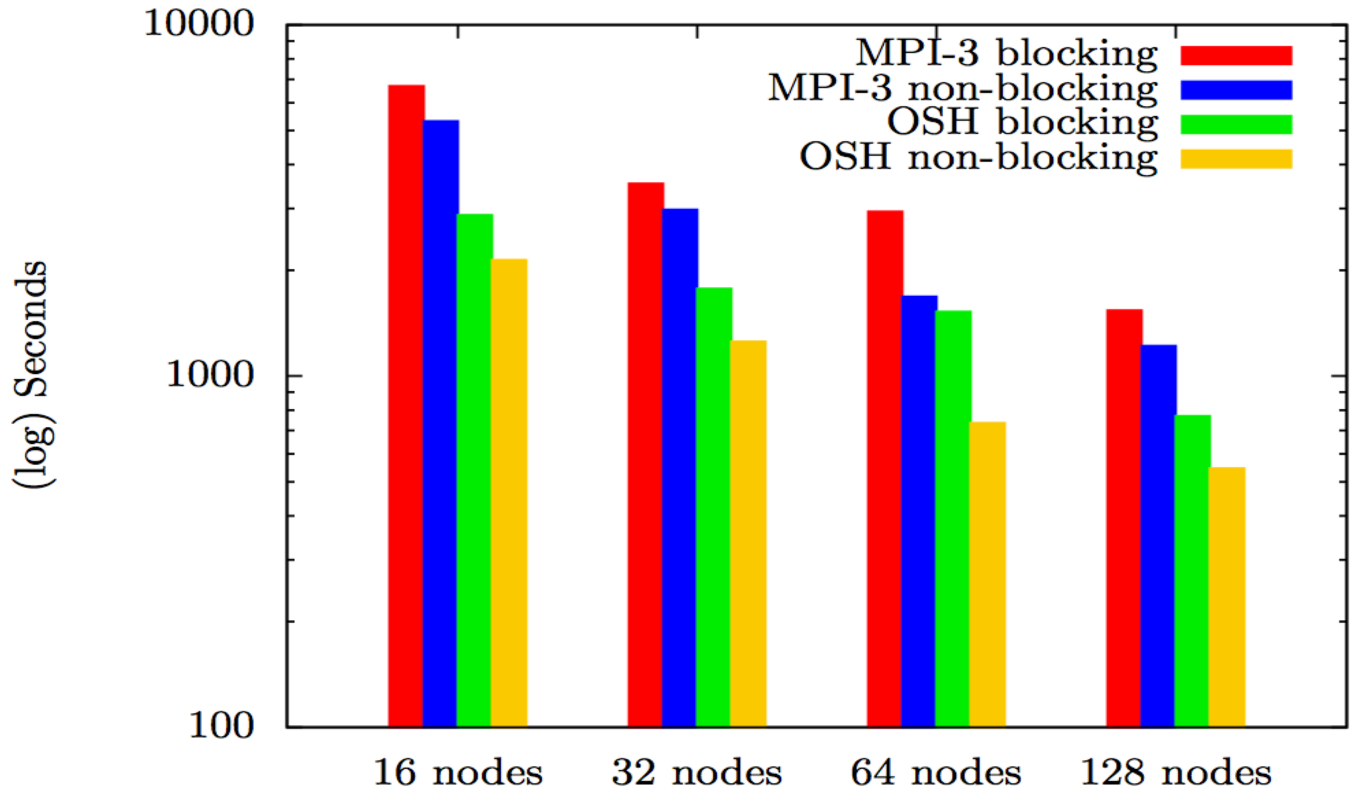
Restructuring loop

- Life is not so simple
  - No separate function for completing non-blocking operations
  - Must use shmem_quiet()
  - Bottom of inner loop updates score and gap arrays
  - Insert shmem_quiet() before puts to update scores

Comparing MPI-3.0 and OpenSHMEM

- Run on ORNL's Titan
  - Used 16, 32, 64, and 128 cores
  - SCALE=32
    - Main and match sequences are 65536 codons long
    - Score matrix is 4,294,967,296 entries.
  - Run with MPI3 blocking and non-bocking gets
  - Run with OpenSHMEM blocking and non-blocking gets
  - Used 1 PE per node (maximize internode communication)

OpenSHMEM Outperforms MPI-3 Implementation



Performance overview

- OpenSHMEM
  - 16 nodes blocking: 2877.5
  - 16 nodes non-blocking: 2142
  - 128 nodes blocking: 770
  - 128 nodes non-blocking: 546.5
- MPI3 one sided
  - 16 nodes blocking: 6693
  - 16 nodes non-blocking: 5318
  - 128 nodes blocking: 1539
  - 128 nodes non-blocking: 1220

Highlights

- MPI3 one sided saw more performance gains from blocking versus non-bocking
- OpenSHMEM was usually 2x as fast as MPI3 one sided.
- OpenSHMEM non-blocking was, at worst, 34% faster, at best 41%

Acknowledgements

This work was supported by the United States Department of Defense & used resources at Oak Ridge National Laboratory.

Questions?

OAK RIDGE National Laboratory | MRF HPC RESEARCH PROGRAM