Fast Parallel Longest Common Subsequence with General Integer Scoring Support

Adnan Ozsoy, Arun Chauhan, Martin Swany

School of Informatics and Computing
Indiana University, Bloomington, USA

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Main problem: Can we do fast string matching?
  - Sequence alignment in bio-informatics
  - Voice and image analysis
  - Improving speech recognition
  - Image retrieval through structural content similarity
  - Social networks for matching event and friend suggestions
  - Computer security virus signature matching
  - Data mining identifying patterns of interest
  - Database query optimization
  - …

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Longest Common Subsequence (LCS)
  - Finding the longest subsequence common to given sequences
- Arbitrary number of input sequences is NP-hard*
- Polynomial time for constant number of sequences
- One-to-many LCS ,Multiple LCS, MLCS
  - A *query* sequence
  - Set of sequences , *subject* sequences

* D. Maier. The complexity of some problems on subsequences and supersequences. Journal of the ACM, 1978

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Main problem: Can we do **fast** string matching?
  - Parallel / Distributed
  - GPU
    - World's fastest supercomputers
      - TITAN, Tianhe-1A, Nebulae, Tsubame 2.0, …etc.

What are GPUs good at

- Scheduling the massively threaded architecture
  - SIMT (Single Instruction Multiple Thread), where each thread in a warp executes the same instruction at a given time
  - Control flow divergence within a single warp is handled by selectively disabling certain threads in the warp, causing performance degradation
- Several memory types: global memory, constant memory, texture memory, shared memory, and registers.
- Asynchronous execution

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Dynamic programing
  - Fill a scoring matrix, H

$$H(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ H(i-1, j-1) + 1 & \text{if } X_i = Y_j, \\ max(H(i, j-1), H(i-1, j)) & \text{otherwise} \end{cases}$$

	init	A	T	C	G	A	G	T
init	0	0	0	0	0	0	0	0
T	0	0	1	1	1	1	1	1
A	0	1	1	1	1	2	2	2
T	0	1	2	2	2	2	2	3
G	0	1	2	2	3	3	3	3
C	0	1	2	3	3	3	3	3
A	0	1	2	3	3	4	4	4
T	0	1	2	3	3	4	4	5

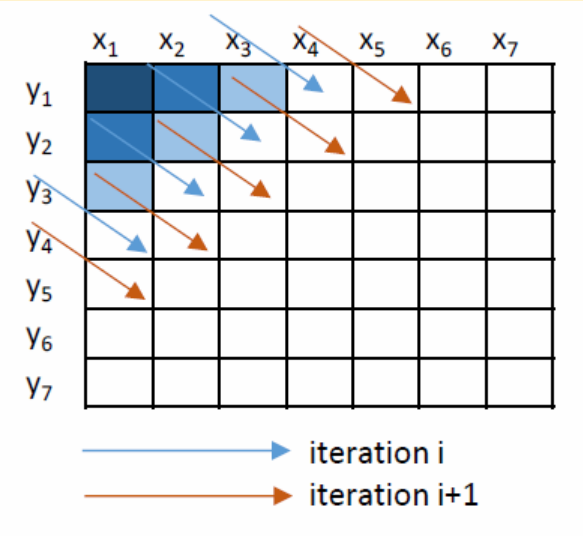
Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Dynamic programing
  - Fill a scoring matrix, H

$$H(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ H(i-1, j-1) + 1 & \text{if } X_i = Y_j, \\ max(H(i, j-1), H(i-1, j)) & \text{otherwise} \end{cases}$$

	init	A	T	C	G	A	G	T
init	0	0	0	0	0	0	0	0
T	0	0	1	1	1	1	1	1
A	0	1	1	1	1	2	2	2
T	0	1	2	2	2	2	2	3
G	0	1	2	2	3	3	3	3
C	0	1	2	3	3	3	3	3
A	0	1	2	3	3	4	4	4
T	0	1	2	3	3	4	4	5

- Dynamic Programming on GPUs



Three problems:

(a) Parallelism is limited in the beginning and the end of computing the matrix

(b) Memory access patterns are not amenable to hardware coalescing.

(c) Space proportional to the product of the sequence lengths

Poor distribution of workload

Sub-optimal utilization of GPU resources

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Proposed Approach
  - Matching information of every single element required
  - Binary matrix
  - Pre-compute matching data for given query string
    - Alphabet-strings
  - Bit parallelism
    - Bits packed into a word
    - Using bit operations on words
  - MLCS

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Allison and Dix
  - Row starts with all zeros and
  - M is the pre-computed alphabet-string
  - Set bits in the last row gives LLCS

$$X = Row_{i-1} \quad OR \quad M_i$$
$$Row_i = X \ AND \ ((X - (Row_{i-1} << 1)) \quad XOR \quad X)$$

	A	A	A	G	T	G	A	C	C	T	A	G	C	C	C	G
init	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
C	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0
G	1	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0
A	1	1	0	0	0	0	1	0	0	0	1	1	0	0	0	0
T	1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0
G	1	1	0	1	0	1	0	0	0	0	0	1	0	0	0	1

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

Ozsoy, Chauhan, Swany (OCS) *

- Achieve Tera CUPS for MLCS with three GPUs, a first for LCS algorithms
- 8.3x better performance than multi-threaded CPU implementation on 12 cores
- Sustainable performance with very large data sets
- Two orders of magnitude better performance compared to previous related work

- *Achieving TeraCUPS on Longest Common Subsequence Problem using GPGPUs - (ICPADS'13)

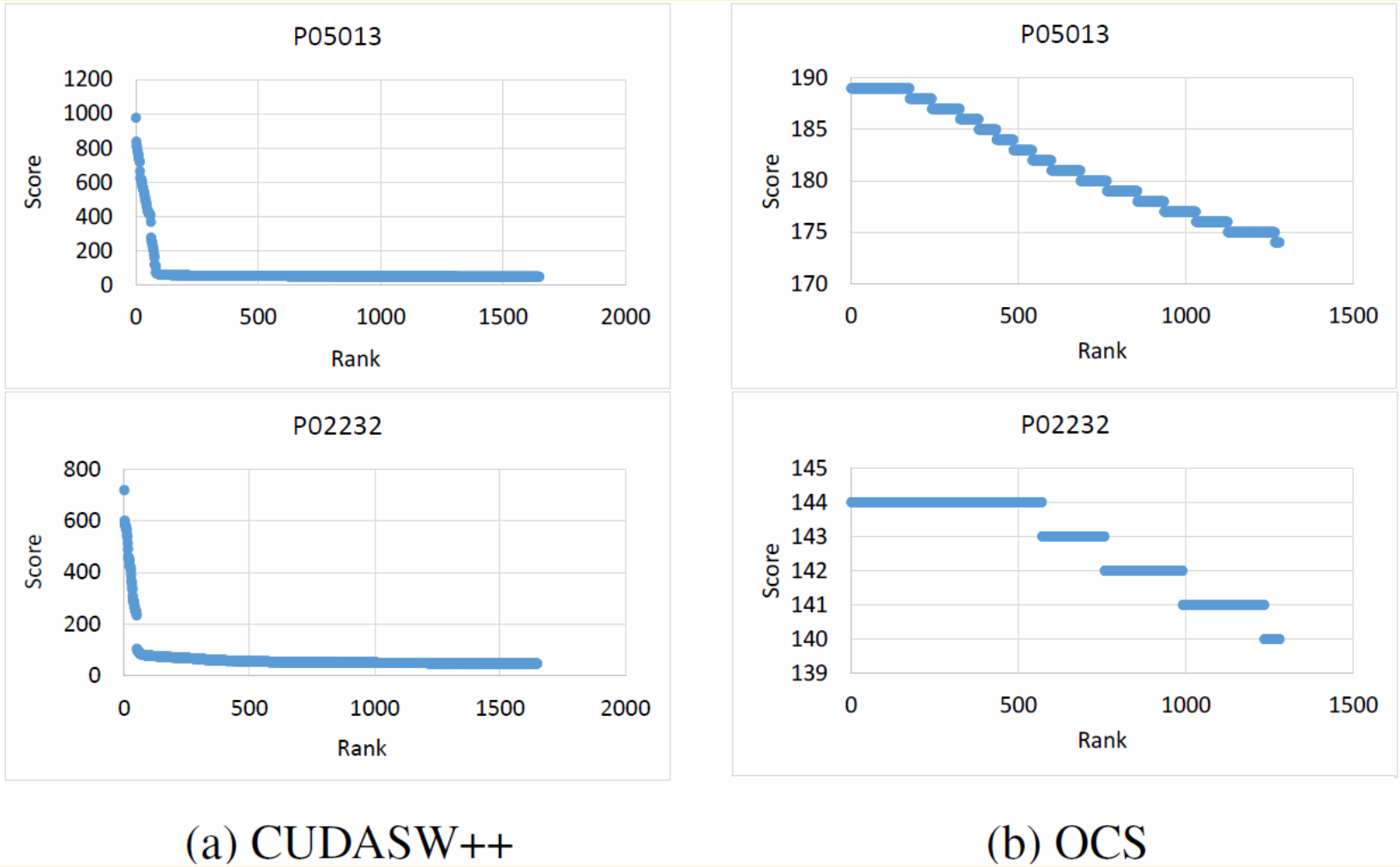
OCS has drawbacks.

- Allison and Dix - no account of weighted scoring
- Similarity score solely depends on the LLCS.
- OCS cannot differentiate the matches with few gaps from those with long gaps,
- May report false negatives and positives.

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Consider an example
  - Querying the sequence “ABC”
  - Database of three subject sequences,
  - ADBDC, ABCD, and ABDDDDC.
  - The LLCS reported by OCS will be three for all
  - The actual LCS will be
  - A-B-C for the first sequence,
  - ABC- for the second,
  - AB---C for the last one
  - Match score is +1 and gap score is -2
  - Scores -1, 1, and -5

Fast Parallel Longest Common Subsequence with General Integer Scoring Support



(a) CUDASW++ (b) OCS

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Applying Scoring
  - The important property is the non-decreasing scores
  - Penalties for non-matching elements diminish score
  - Allison will not be applicable
  - Benson et al. (BHL) - Integer Scoring with Bit-Vector

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Integer Scoring with Bit-Vector

$$H(i,j) = \max \begin{cases} H[i-1,j-1] + match, & if X_i = Y_j, \\ H[i-1,j-1] + mismatch, & if X_i != Y_j, \\ H[i-1,j] + gap, & delete X_i, \\ H[i,j-1] + gap, & delete Y_j \end{cases}$$

- Match, M, Mismatch, I, Gap G
- Instead of keeping a score table
- Keeps track of the score differences between a cell and its above and left neighbors

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Integer Scoring with Bit-Vector

$$\Delta V_{min} \text{ and } \Delta H_{min} = \text{gap}$$
$$\Delta V_{max} \text{ and } \Delta H_{max} = \text{match} - \text{gap}$$

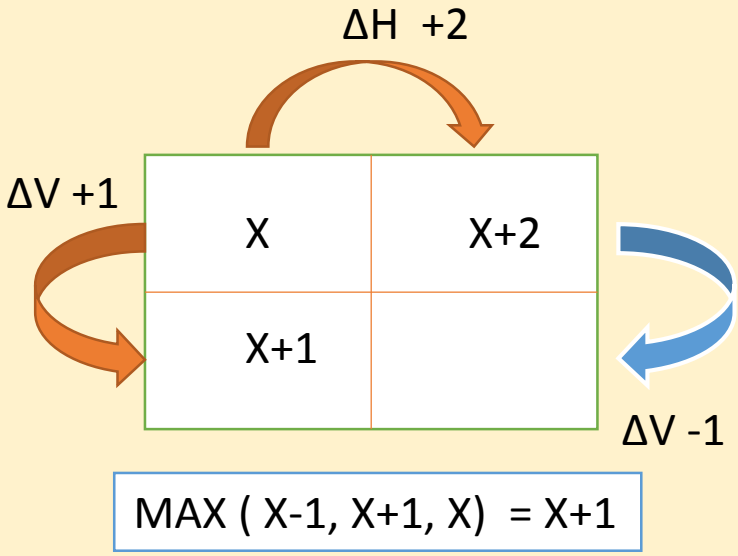
For example, for the weights match = 1, mismatch = -1, and gap = -1, ΔV_{min} and ΔH_{min} will be -1, and ΔV_{max} and ΔH_{max} will be 2.

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Integer Scoring with Bit-Vector

ΔV \ ΔH	-1	0	1	2
-1	-1	-1	-1	-1
0	0	-1	-1	-1
1	1	0	-1	-1
2	2	1	0	-1

The ΔV Function Table for the weights
match = 1, mismatch =-1, gap = -1



Integer Scoring with Bit-Vector

- A variable for each of possible function table values
- Hold the location of corresponding value in a single bit
- Update these values knowing the previous ΔV and ΔH
- Alignment score
  - Another iteration over the last row of the scoring matrix
  - Row wise iteration, 1 bits in the H values are added

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Integer Scoring with Bit-Vector Drawbacks
  - Supports up to single word long sequences
  - Keeping track of all possible values of ΔV and ΔH
  - In sequential calculation variables can be reused
    - 25 million sequence alignments → 30GB memory
  - Time complexity of the BHL algorithm is O(z*m*n/w)
    - z # of bit operations,
    - m and n are sequence sizes,
    - w is the word size
  - 23 bit operations for (0,-1,-1), more than 250 bit operations for (2,-3,-5), more than 1000 for (4,-7,-11)
  - Bit operations > word size (w)

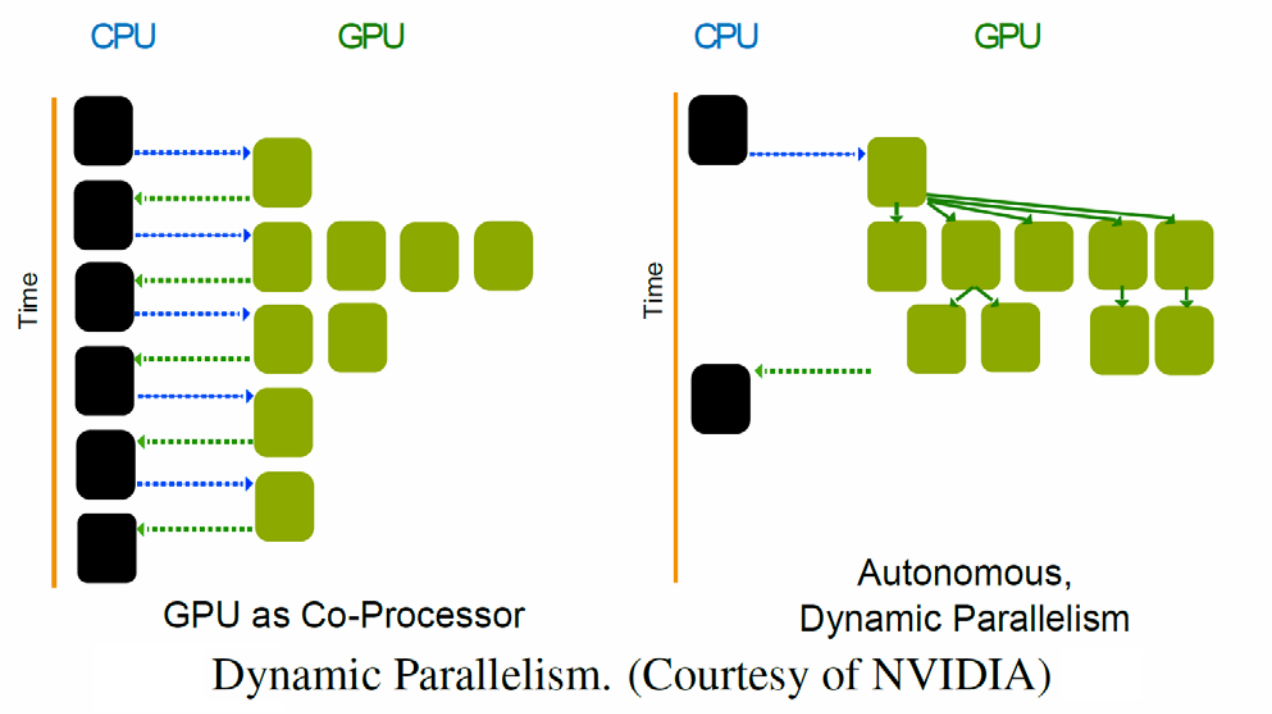
Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Integer Scoring with Bit-Vector
- Pipelined Approach
  - LLCS – on GPU
  - Sort Top N and Sort Final – on CPU
  - Scoring – GPU/CPU?

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- GPU parallelization
  - Multi – word support
    - basic bit operations - AND, OR and XOR
    - Complex operations – carry/borrow bit - SHIFT, BIT-ADD, and BIT-SUBSTRACT
  - Inter task
    - one subject sequence is assigned to each CUDA thread
  - Intra task
    - Dynamic parallelism

- Dynamic parallelism
  - Passed values need to be global memory
  - cudaMalloc or new/delete language constructs



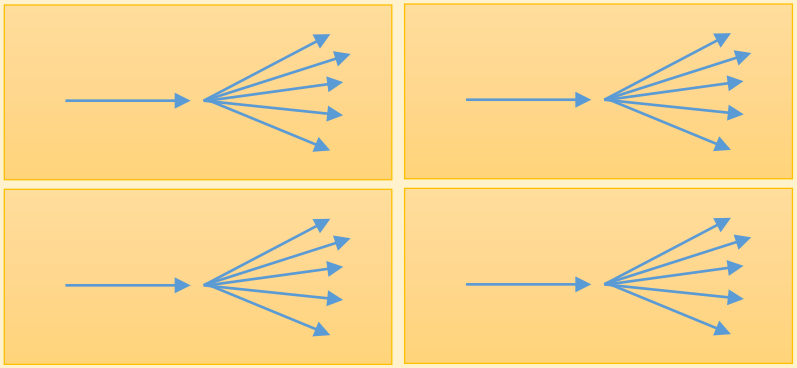
Dynamic Parallelism. (Courtesy of NVIDIA)

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Dynamic parallelism
  - Multi-word update loop

```
for i->0 to num_of_words
        word1[i] = word2[i]  OP word3[i]
```

  - Initial thread allocate global memory – fire multiple threads

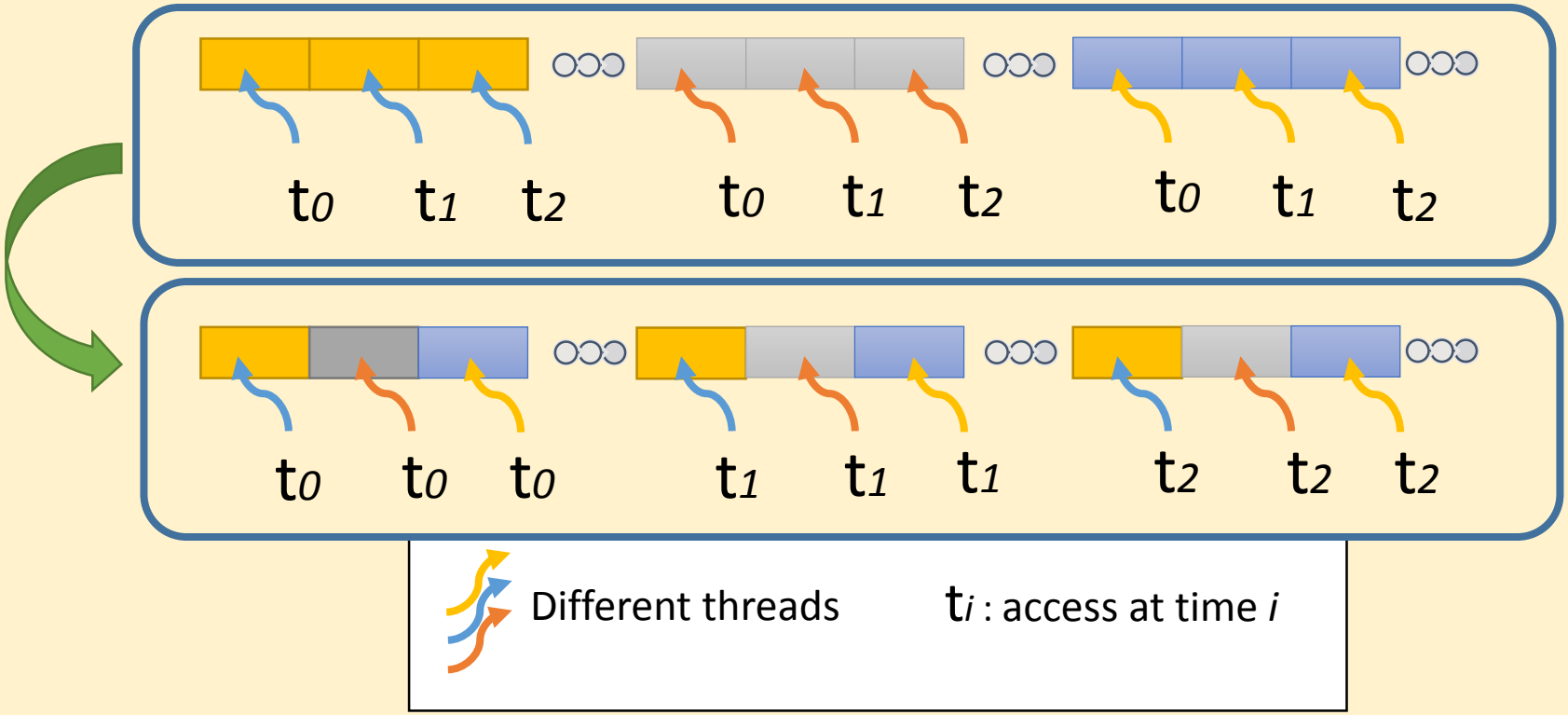


Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Optimizations
  - Memory Spaces
    - Alpha-strings in fast memory space - shared memory
    - Memory bank conflicts
    - Global memory – Coalesced access
    - Data Orientation

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- Optimizations
  - Data Orientation



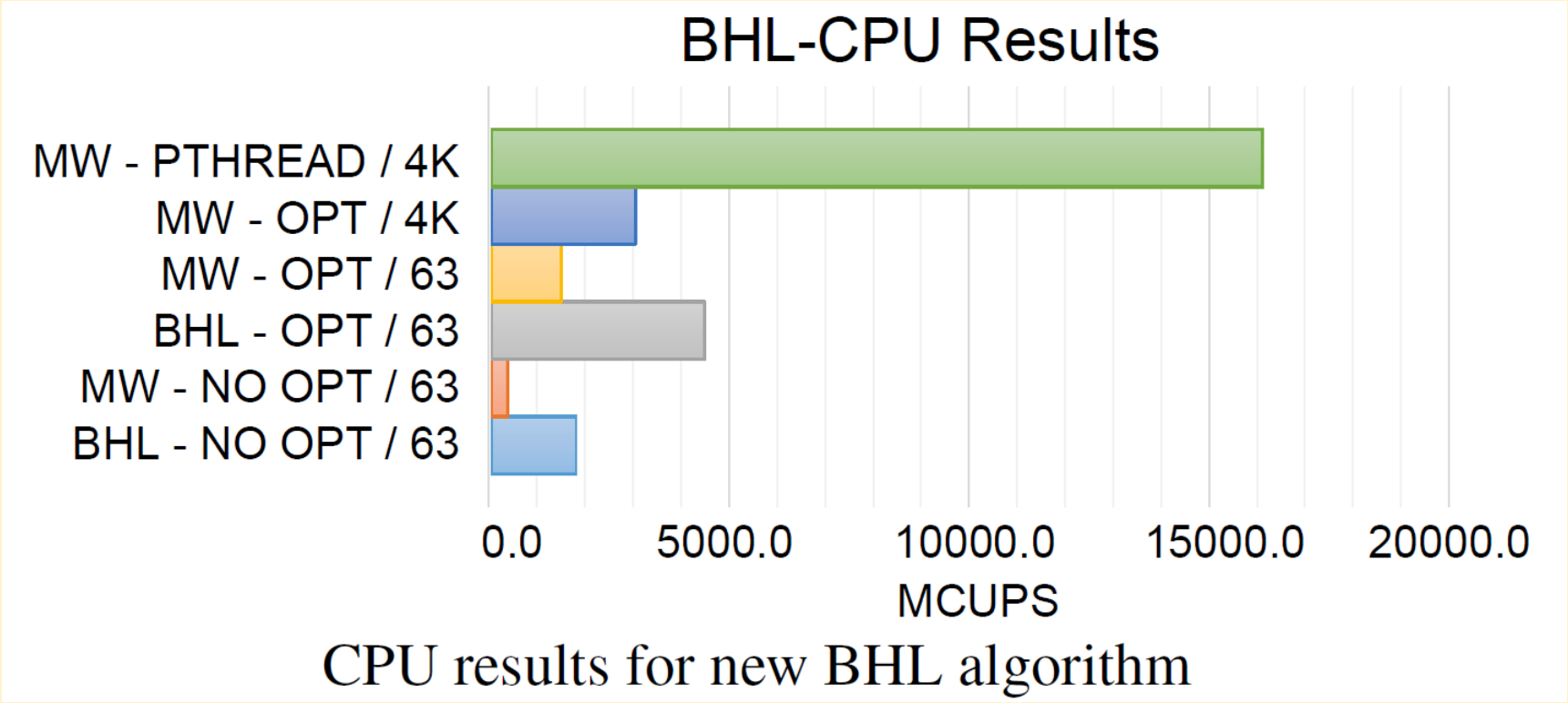
Fast Parallel Longest Common Subsequence with General Integer Scoring Support

- **Testbed configurations**

System	GPU	Number of Cores	Device Memory	CUDA Capab.	CUDA Version	CPU	Host Memory	OS
FutureGrid	C2075 x 2	448	5375 MB	2.0	5.5	X5660	192 GB	Red Hat 4.4
Big Red II	K20	2496	5375 MB	3.5	5.0	Opteron Abu Dhabi	64 GB	Cray Linux
	GTX 680 x 2	1536	2048 MB	3.0	5.0	AMD PII X6	16 GB	Debian 4.4

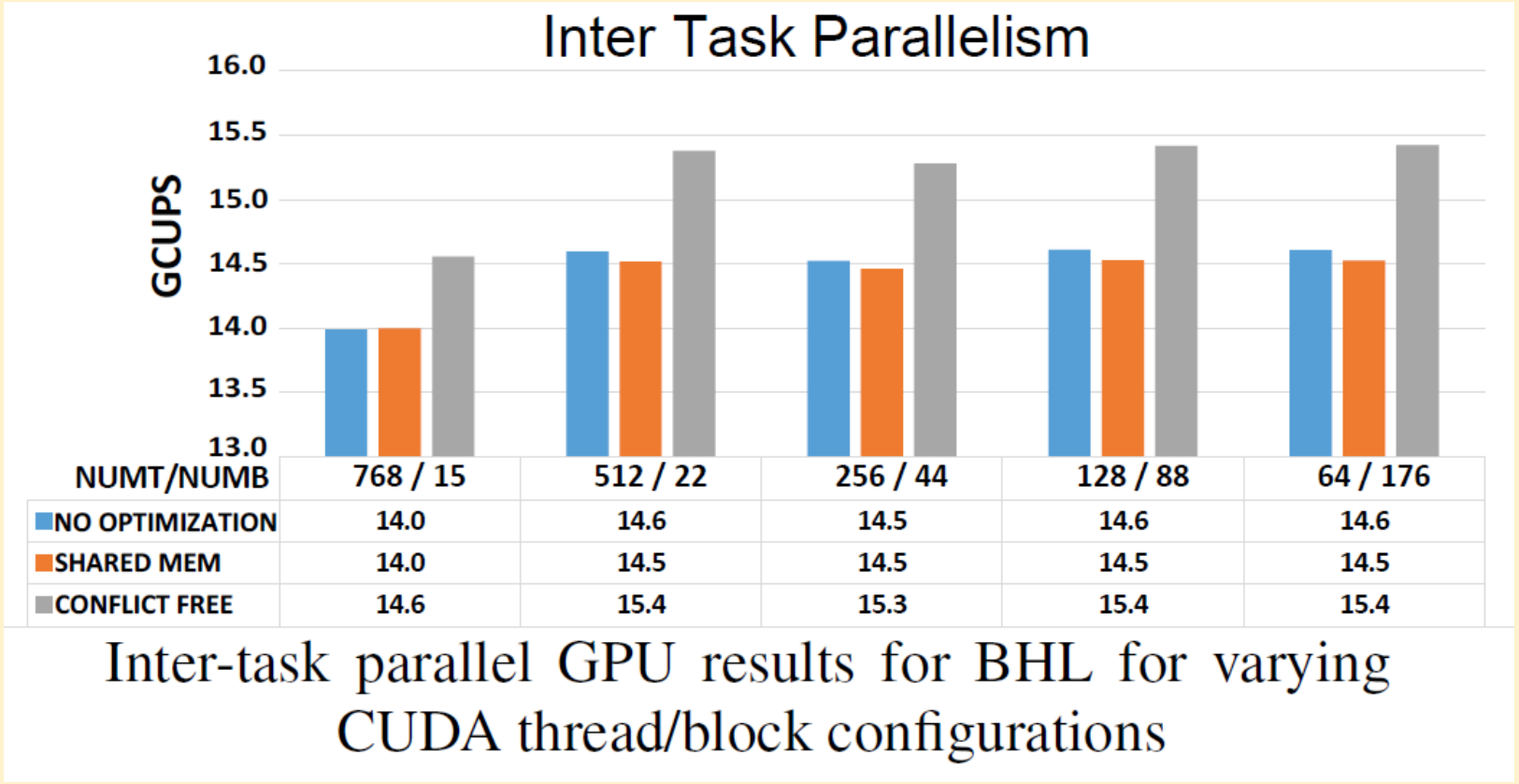
- **Datasets**
  - Each sequence is 4K size
    - Benchmark sequence databases for bioinformatics 99%
  - Different subject sequence sizes; 50000, 188000,720000
    - UniProtDB/Swiss-Prot database includes more than 500000
    - Virus signatures is over hundred thousands

Fast Parallel Longest Common Subsequence with General Integer Scoring Support



CPU results for new BHL algorithm

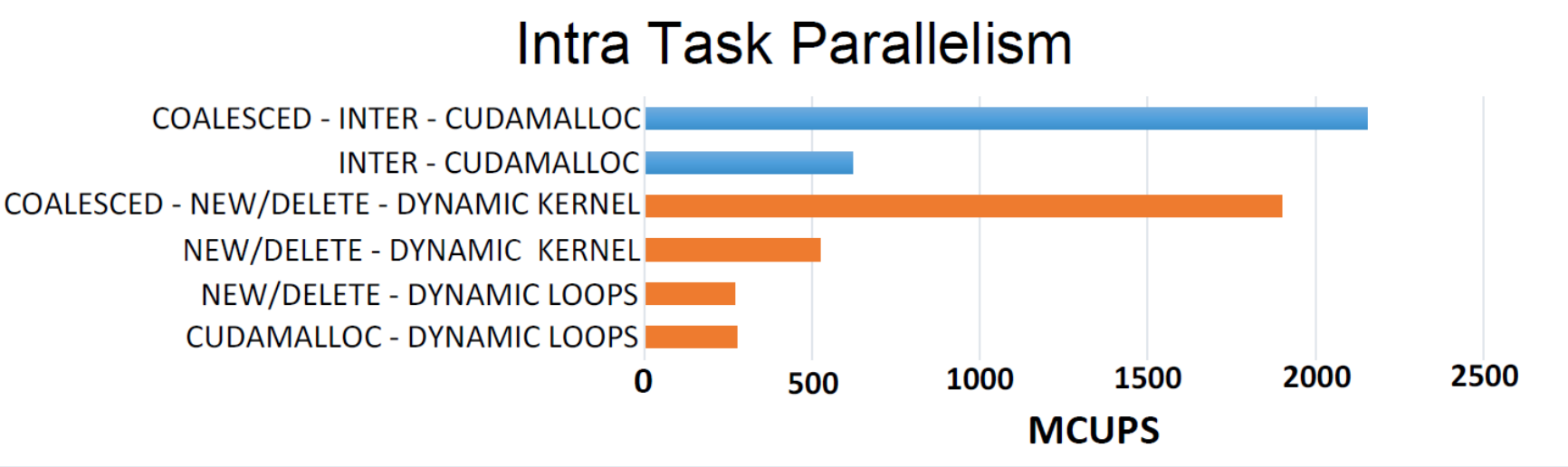
Fast Parallel Longest Common Subsequence with General Integer Scoring Support



NUMT/NUMB	768 / 15	512 / 22	256 / 44	128 / 88	64 / 176
NO OPTIMIZATION	14.0	14.6	14.5	14.6	14.6
SHARED MEM	14.0	14.5	14.5	14.5	14.5
CONFLICT FREE	14.6	15.4	15.3	15.4	15.4

Inter-task parallel GPU results for BHL for varying CUDA thread/block configurations

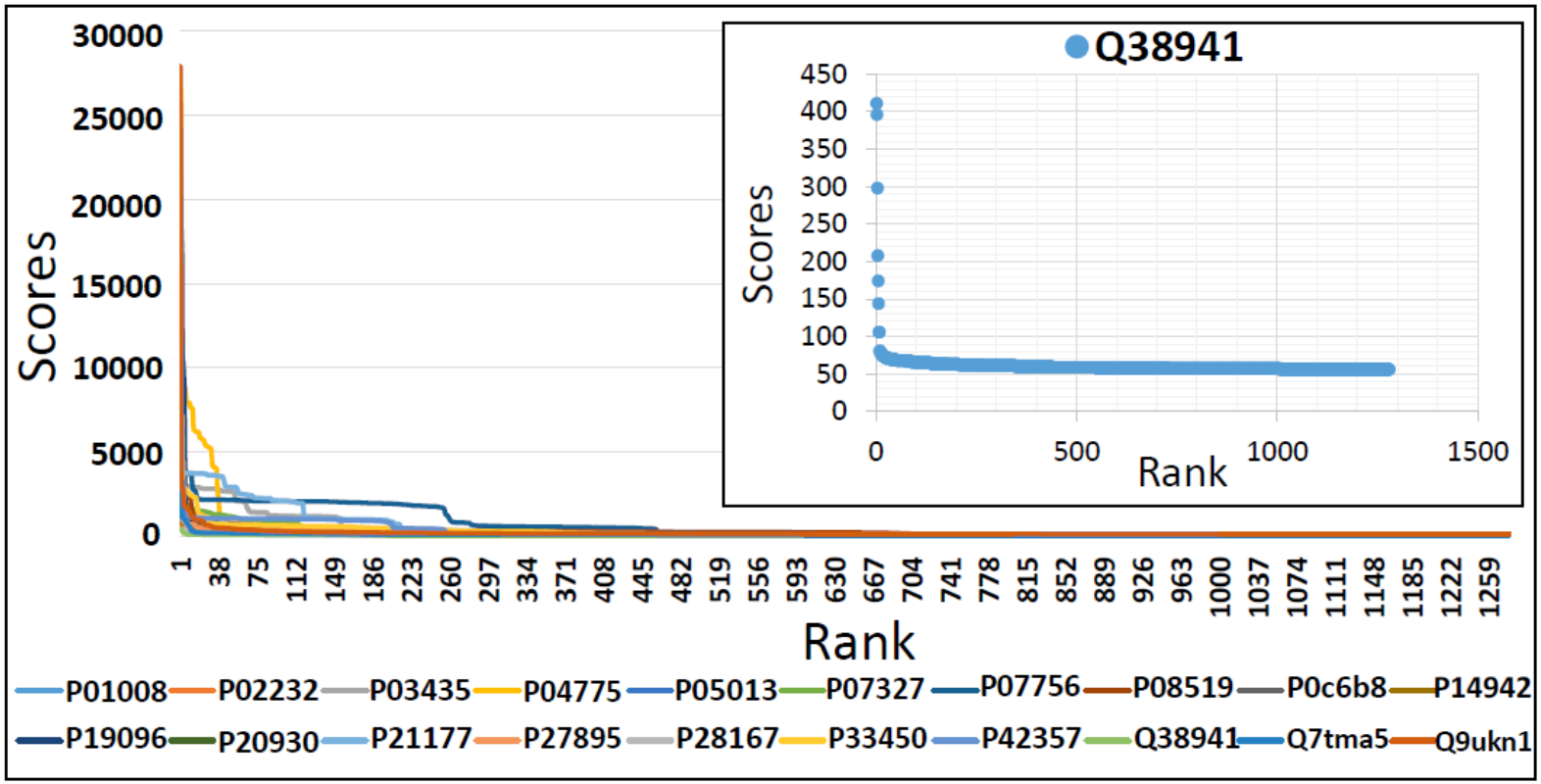
Fast Parallel Longest Common Subsequence with General Integer Scoring Support



Intra-task parallel GPU results for MW-BHL algorithm

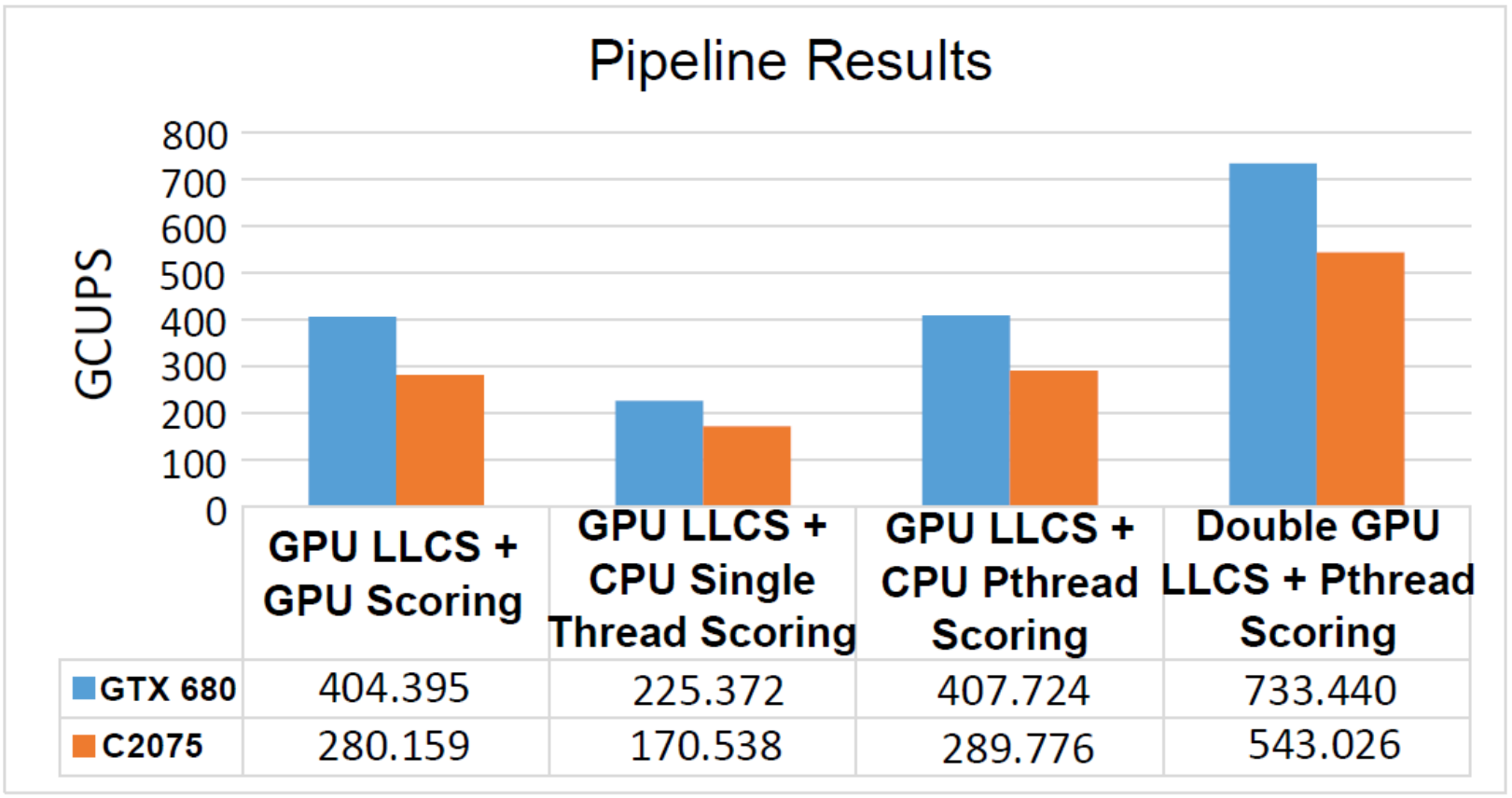
Decision on selecting top N values

Fast Parallel Longest Common Subsequence with General Integer Scoring Support



Top 1280 scores reported by CUDASW++ for 20 different query sequences. Inner graph in the top right corner gives results in detail for one query sequence.

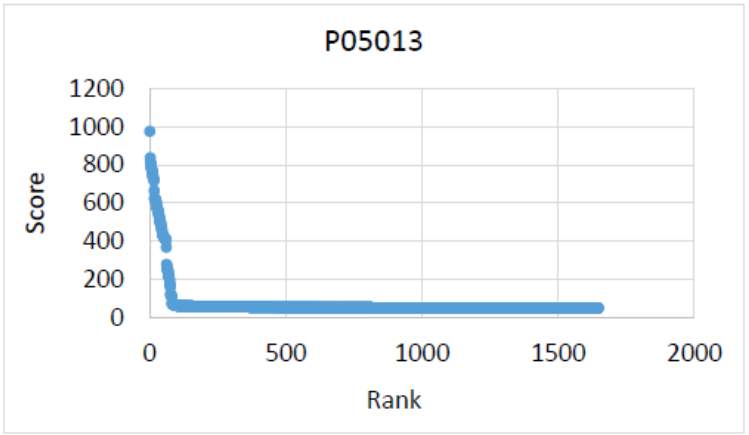
Fast Parallel Longest Common Subsequence with General Integer Scoring Support

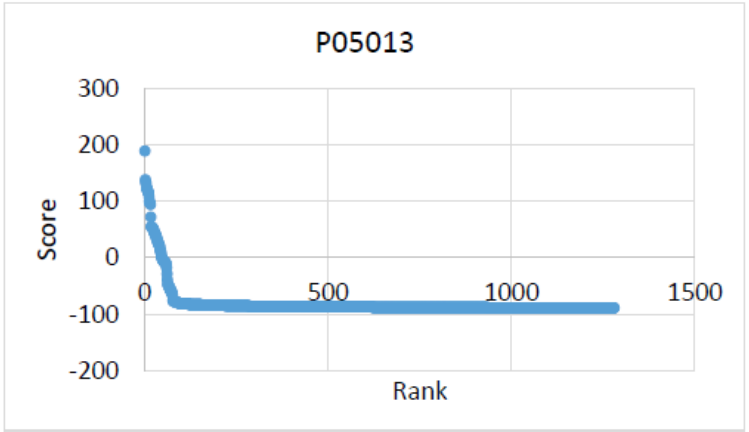


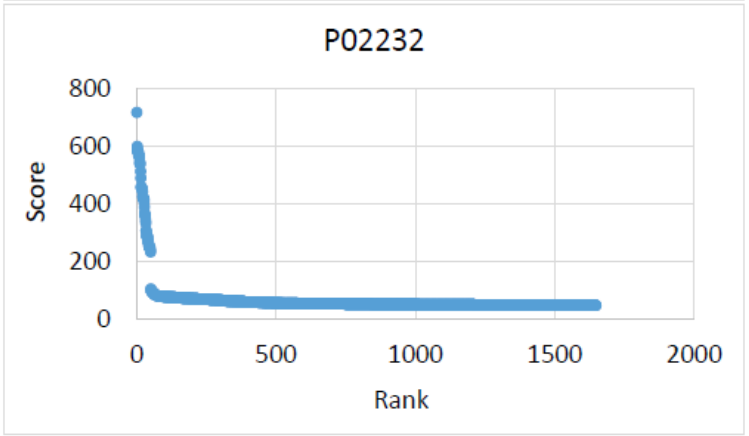
	GPU LLCS + GPU Scoring	GPU LLCS + CPU Single Thread Scoring	GPU LLCS + CPU Pthread Scoring	Double GPU LLCS + Pthread Scoring
GTX 680	404.395	225.372	407.724	733.440
C2075	280.159	170.538	289.776	543.026

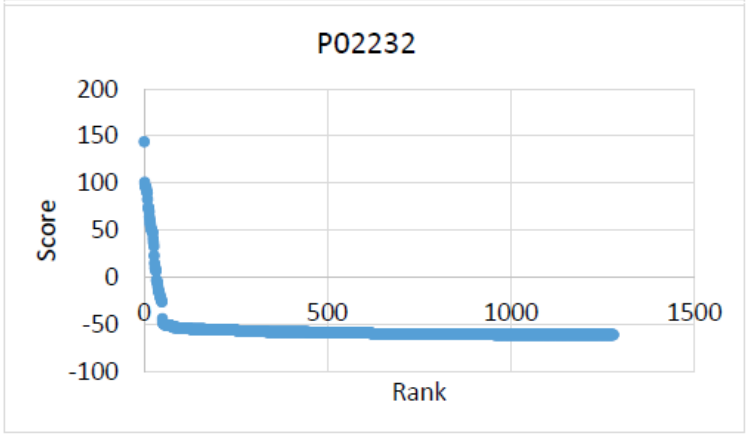
Figure 13: Streaming pipeline for OCL + BHL

Fast Parallel Longest Common Subsequence with General Integer Scoring Support









(a) CUDASW++ (b) OCL + BHL

Figure 14: CUDASW++ vs Streaming pipeline for OCL + BHL

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

Related GPU work

- Manavski and Valle **~3.5 GCUPS** using Smith-Waterman(SW) MLCS.
- CUDASW++2.0 by Liu et al., **~17 to 30 GCUPS** SW
  - anti-diagonal parallelization (less than 1%) and inter-task approach.
- Khajeh-Saeed et al. SW -two large sequences scaling through hundreds of GPUs**. ~1.04 GCUPS** per GPU.
- Kawanami et al. Allison bit-vector, one-to-one LCS **~3 GCUPS**

Bit Parallel

- Improvements made to Allison (Crochemore 2000, Hyyro 2004)
- Benson et al.

We achieve **TeraCUPS (~1000 GCUPS)** on 3 GPUs for LLCS/MLCS

- One to two orders of magnitude better.
- Differences in algorithms and LCS approaches.

Fast Parallel Longest Common Subsequence with General Integer Scoring Support

Conclusion & Future Work

- A pipelined approach to extend MLCS with general scoring on GPUs
- Parallelization and optimization steps bit parallel scoring
- Improved the reported false positives with similar performance

Future Work

- Multi GPU + multi node
- Improve the pipeline stages.
- Adopt varying weights – currently in short range
- As new capabilities arise

- Code will available soon after conference

Q & A

Are you looking for a post-doc?

Thanks to

BIG RED II
INDIANA UNIVERSITY

NVIDIA Hardware
Request Program