BAYNCORE

Pairwise sequence alignment with the Smith-Waterman algorithm

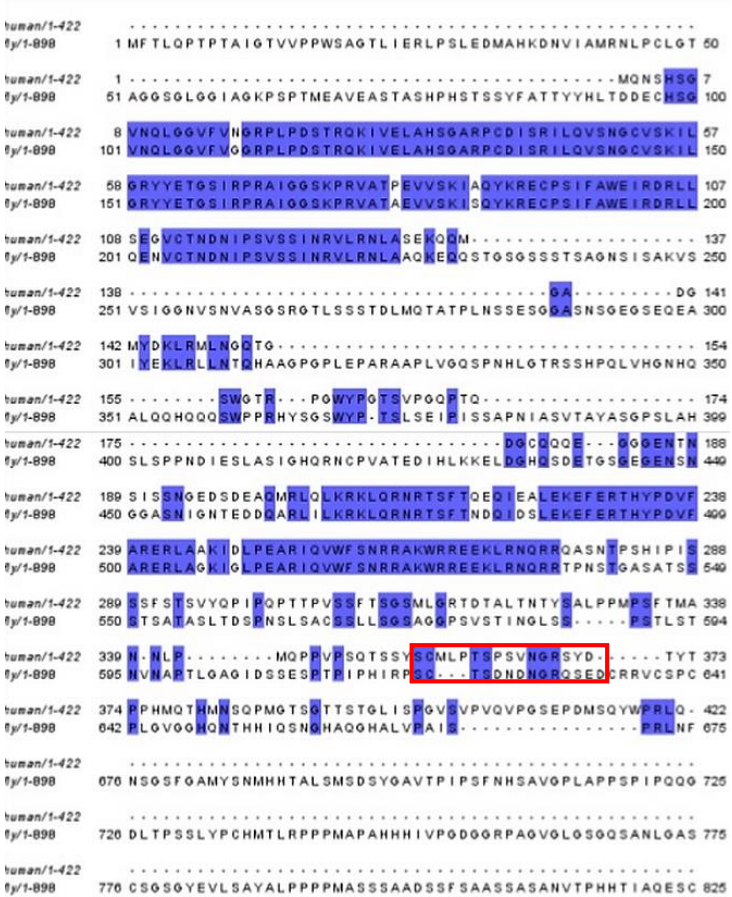
Manel Fernández

Intel HPC Software Workshop Series 2016

HPC Code Modernization for Intel® Xeon and Xeon Phi™

February 18th 2016, Barcelona

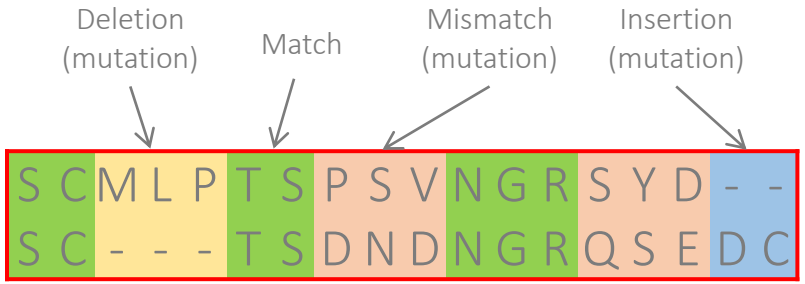
Pairwise sequence alignment



Comparison applied to *pairs* of biomolecular sequences

- RNA, DNA, or amino-acids

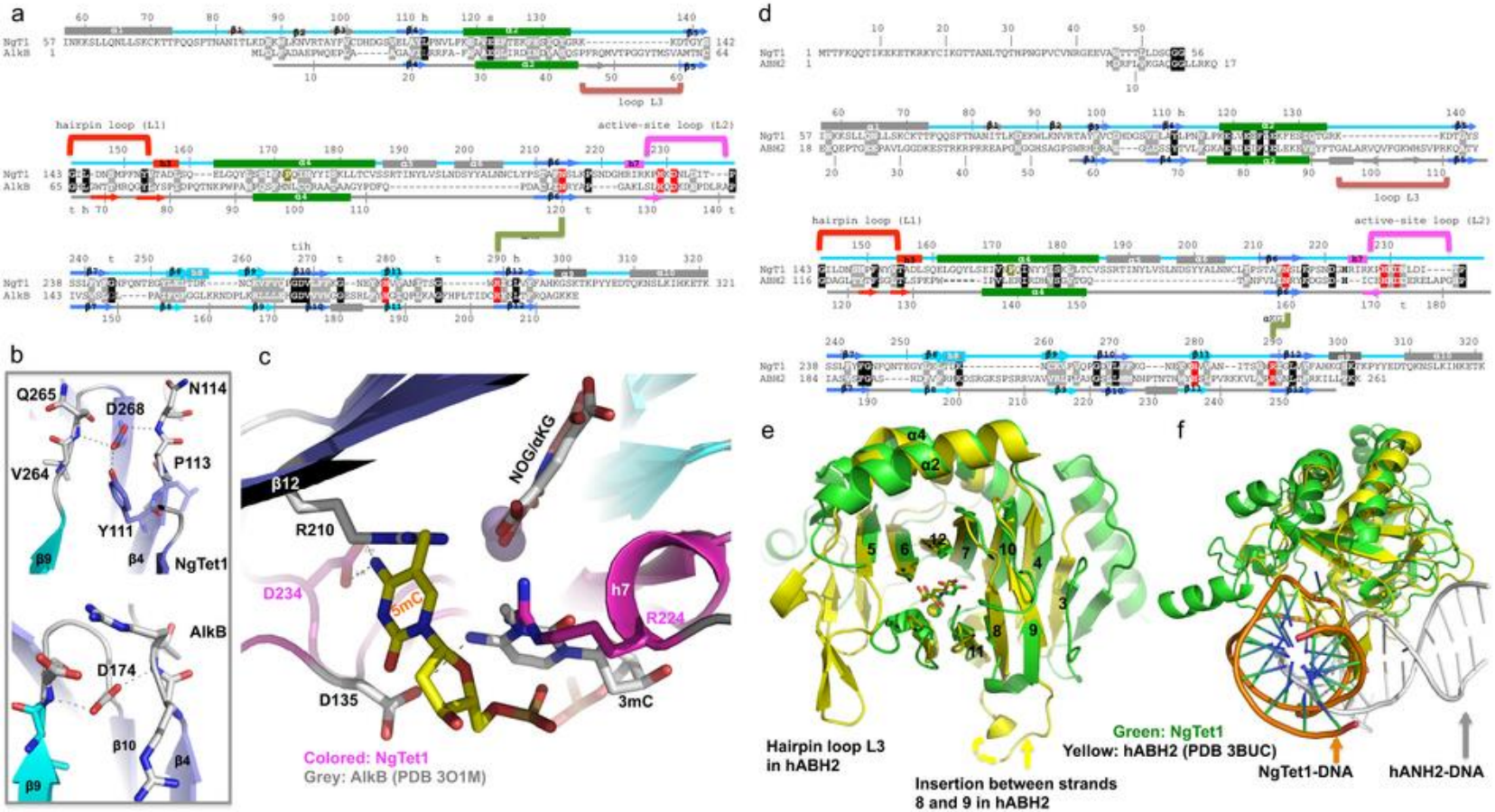
The goal is to identify *how similar* they are (homology)



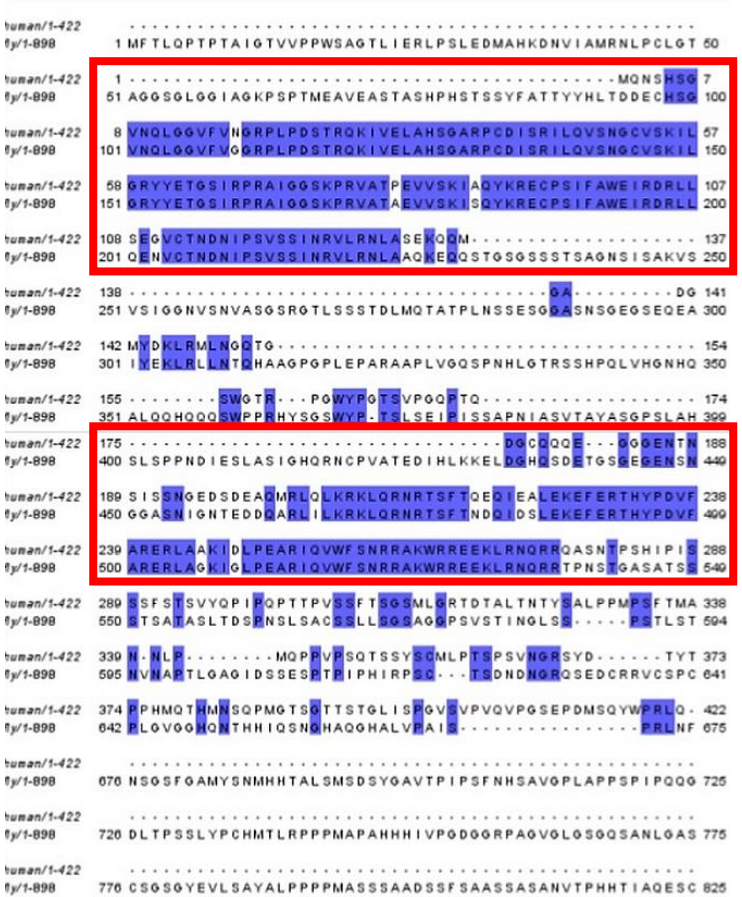
Also used on non-biological sequences, such as natural language and financial data

Pairwise comparison of *Naegleria* NgTet1 vs *E. coli* AlkB (a–c) and human ABH2 (d–f)

From "Structure of a Naegleria Tet-like dioxygenase in complex with 5-methylcytosine DNA", by H. Hashimoto Et Al., Nature 506, 391–395 (20 February 2014)



Pairwise sequence alignment methods



Global alignment

- Covers the entire sequences

```
F T F T A L I L L A V A V
F - - T A L - L L A - A V
```

- *Needleman-Wunsch* algorithm finds the best global alignment

Local alignment

- Only covers parts of the sequences

```
F T F T A L I L L - A V A V
- - F T A L - L L A A V - -
```

- *Smith-Waterman* (S-W) algorithm finds the best local(s) alignment(s)

Smith-Waterman (SW) algorithm

Step 1. Compute the *alignment matrix* or *score matrix* ($\in \mathbb{N}$)

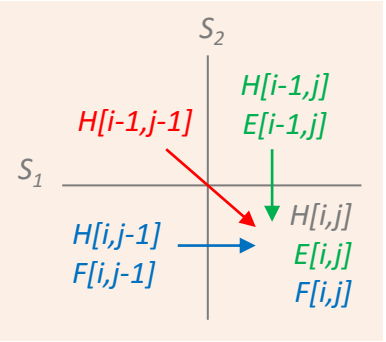
- Best alignment starts on the [i,j] location with the *highest score value* in the matrix

$$H[i,0] = H[0,j] = E[0,j] = F[i,0] = 0 \qquad 0 \le i \le m, 0 \le j \le n$$

$$H[i,j] = \max \begin{cases} 0 & - \\ H[i-1,j-1] + \sigma(S_1[i], S_2[j]) & \textit{Match/Mismatch} \\ E[i,j] = \max \begin{cases} E[i-1,j] - \omega_e \\ H[i-1,j] - \omega_o - \omega_e \end{cases} & \textit{Deletion } (S_1[i] \textit{ aligned to a gap}) \\ F[i,j] = \max \begin{cases} F[i,j-1] - \omega_e \\ H[i,j-1] - \omega_o - \omega_e \end{cases} & \textit{Insertion } (S_2[j] \textit{ aligned to a gap}) \end{cases} \qquad 1 \le i \le m, 1 \le j \le n$$

Where

- $S_1[1..m]$, $S_2[1..n]$ are sequences over alphabet Σ
- $\sigma(a,b)$ is the similarity function on alphabet letters (substitution matrix)
- ω_o and ω_e are gap opening and gap extension penalty scheme



Step 2. Find the *optimum local alignment* in the matrix

- From the highest value, go backwards on the direction of used to construct the matrix
- Alignment sequence ends when a matrix cell with zero value is reached

Smith-Waterman algorithm: example

$S_1 = \texttt{AGCACACA},\ S_2 = \texttt{ACACACTA}$

$\omega_o = -1,\ \omega_e = 0,\ \sigma(a,b) = \begin{cases} +2, a = b \\ -1, a \neq b \end{cases}$

Step 1

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix}$$

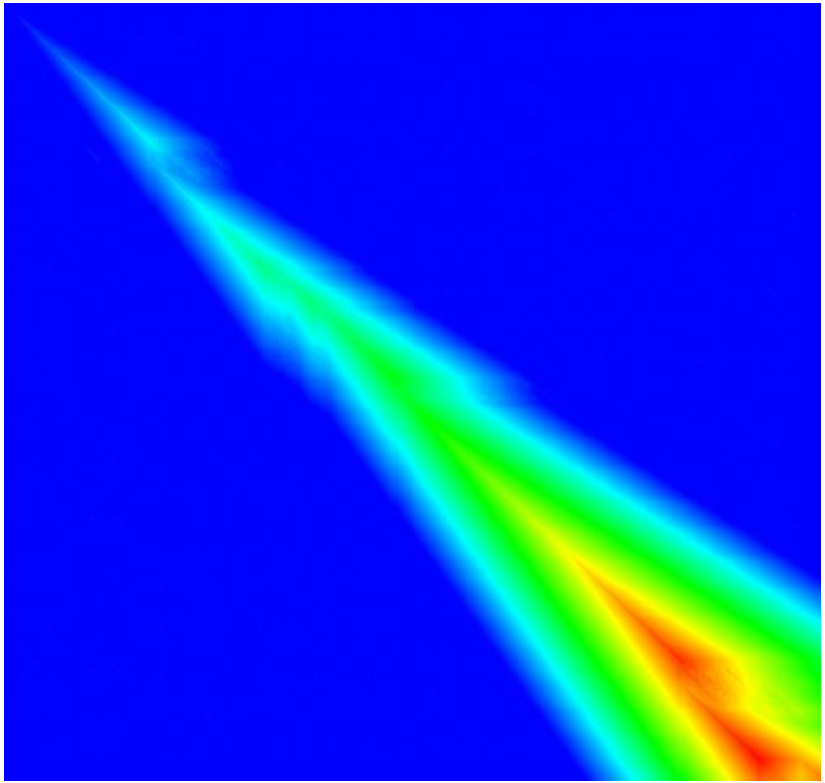
$$T = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ G & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow & \uparrow \\ C & 0 & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow \end{pmatrix}$$

Step 2

$S_1 = \texttt{AGCACAC-A}$

$S_2 = \texttt{A-CACACTA}$

SW score matrix example



Simple example

- Slightly different sequences
- Similar length (~500 symbols)

Colour gradient for matrix values

- Blue for H[i,j] = 0
- Red for high score values

High homology

- About 352 score for this example
- Not necessarily always the case

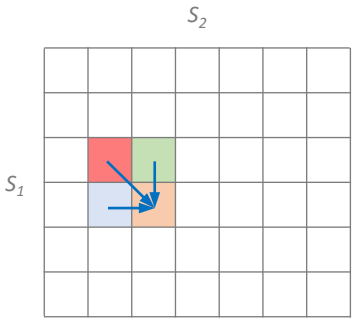
Other algorithm considerations

Only address Step 1 of the algorithm (*)

- Build the score matrix ("*recurrence*") and find the highest score ("*reduce*")
- We will briefly discuss Step 2 later on this presentation

Algorithm complexity

- Computational complexity: ***O(mn)*** *(quadratic)*
  - "Recurrence" and "reduce" steps can be fused
- Space complexity: ~~***O(mn)*** *(quadratic) ??*~~ ***O(m+n)*** *(linear)*
  - No need to store the whole matrix to find the highest score value



(*) Code modernization proposed by "*Pairwise DNA sequence alignment optimization*" by Yongchao Liu and Bertil Schmidt ("*High performance parallelism pearls*", Vol. 2, Ch. 4)

Code modernization of S-W algorithm

How to parallelize the score matrix computation?

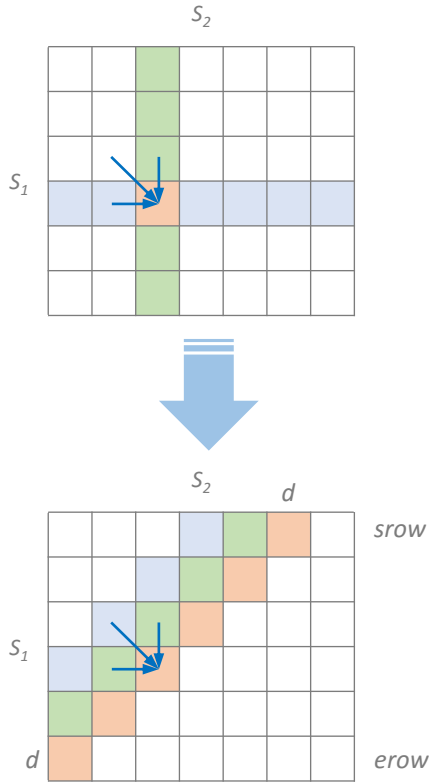
- Dependency when parallelizing by rows or columns
- **Parallelize over NE-SW diagonals!**

Assuming

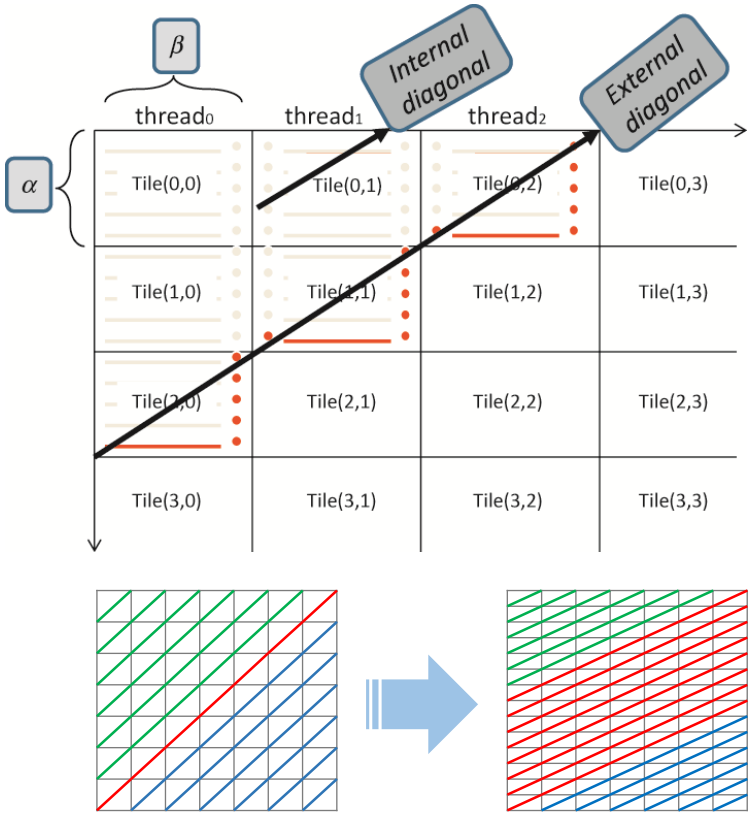
- S_1 is placed vertically, indexed by i, and $\text{length}(S_1) = m$
- S_2 is placed horizontally, indexed by j, and $\text{length}(S_2) = n$
- $m \leq n$

We can see that

- Diagonal $d = i + j$, where $0 \leq d < m + n - 1$
- Starting row index $srow_d = \max(0,\ d - n + 1)$
- Ending row index $erow_d = \min(d,\ m - 1)$



Intra-process (SMP) parallelization



Scale-and-vectorize approach

- Partition the matrix into tiles
  - Tile size is α rows × β columns
  - Matrix size is $R = \left\lceil \frac{m}{\alpha} \right\rceil$ rows × $C = \left\lceil \frac{n}{\beta} \right\rceil$ columns
- TLP on tiles for every external diagonal
- DLP on every internal diagonal

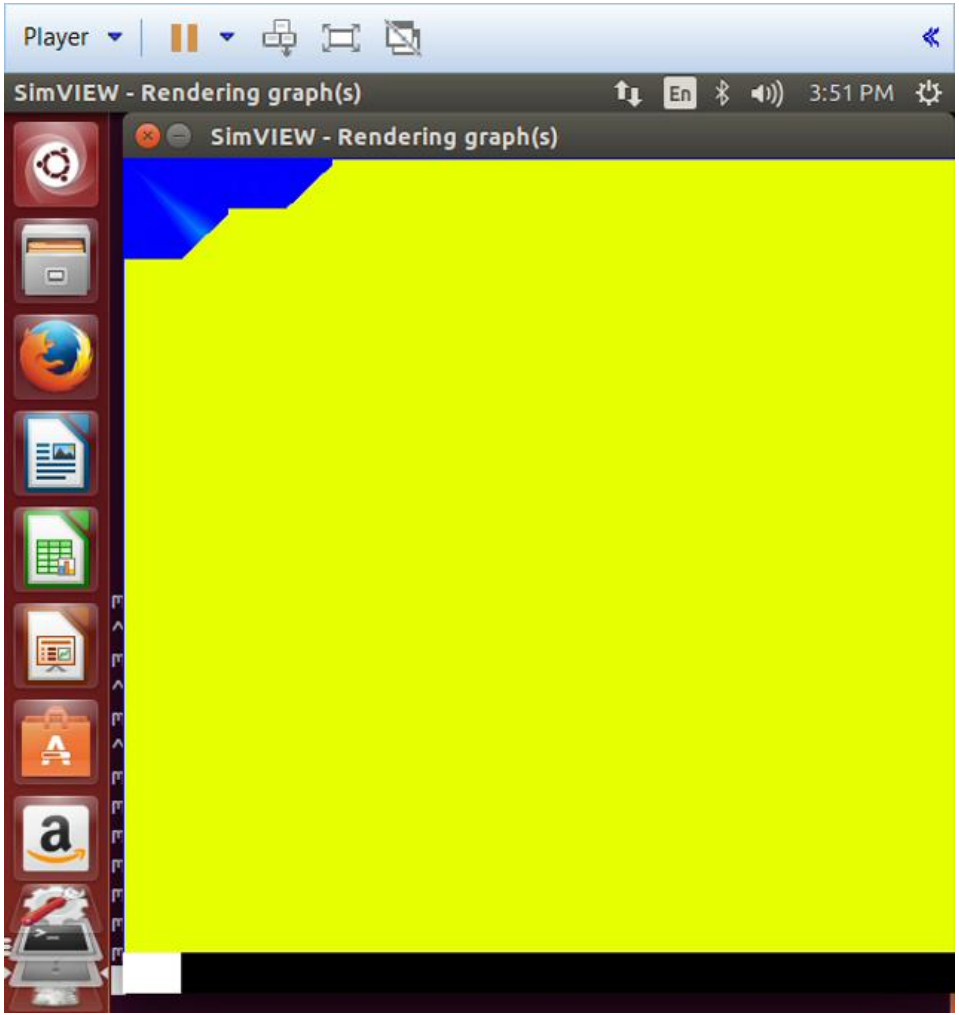
Current implementation parameters

- $C = \#\text{threads},\ \beta = \left\lceil \frac{n}{\#\text{threads}} \right\rceil$
- $\alpha = \text{SIMD}_{\text{length}} \times K$
  - $\text{SIMD}_{\text{length}} = 8$ for AVX2, 16 for AVX-512/IMCI
  - Small K (i.e., $K = 1$) maximize external diagonals with $\#\text{threads}$ tiles, since $\alpha \ll \beta$ and $R \gg C$

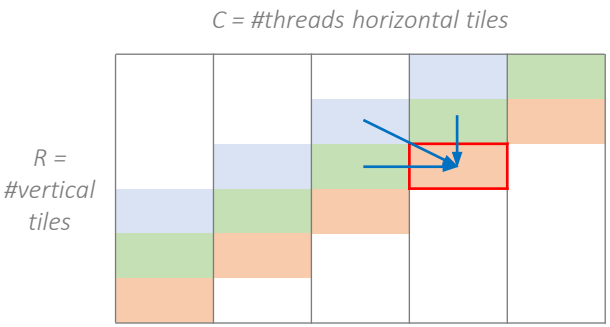
S-W in action

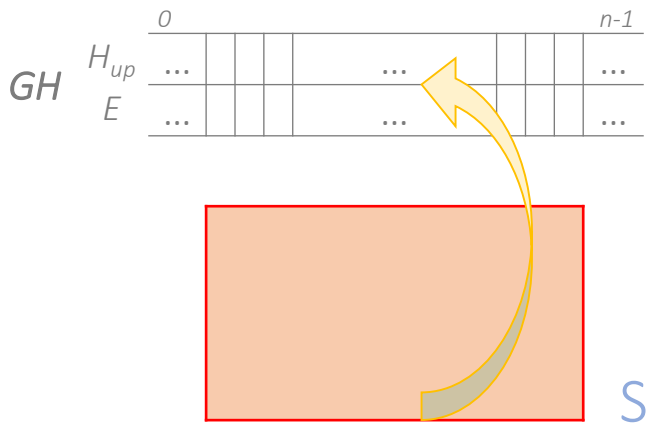
Simple example animation

- 8 threads (column tiles)
- Taller tiles than actual ones



Thread parallelism with OpenMP

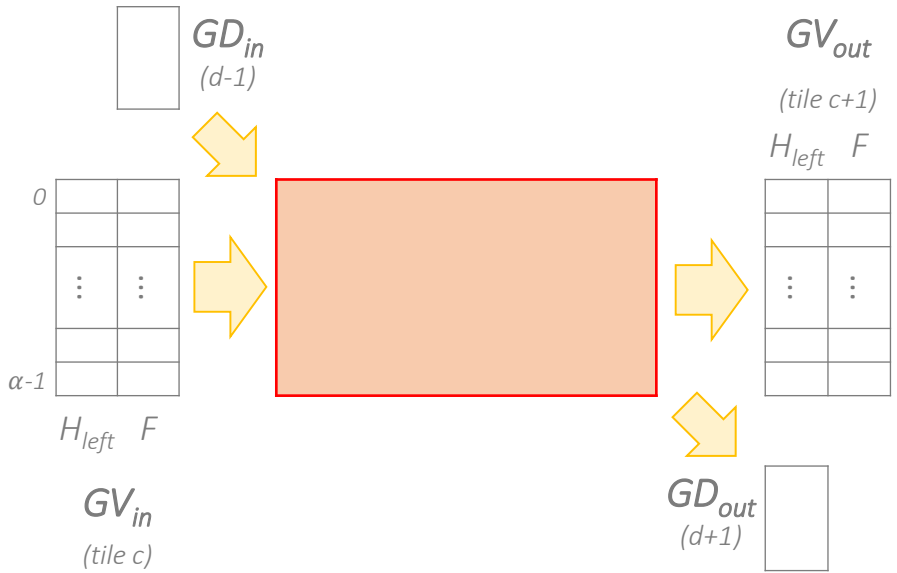




Shared by all threads

Private to every thread

GVs are private pointers to global GVs[C] arrays



OpenMP implementation

```
#pragma omp parallel firstprivate(GV,GD) default(shared)
for (d = 0; d < R+C-1; d++)
{
    sExtRow = max(0,d-C+1);
    eExtRow = min(d,R-1);

    Load the per-thread maximum score lmaxScore;

    #pragma omp barrier

    #pragma omp for schedule(static,1) nowait
    for (r = sExtRow; r <= eExtRow; r++)
        Compute all cells in tile (r,d-r);

    Save lmaxScore to a global variable;
    Swap the input and output of GV and GD;
}

Reduce to get the optimal local alignment score;
```

Threads need to be synchronized before moving to the next external diagonal.

For every diagonal, every thread will process one tile.

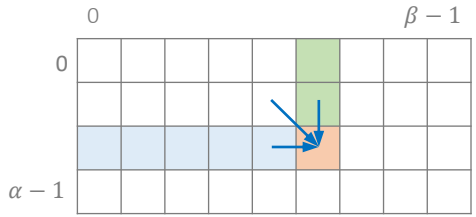
No need to synchronize at the end since updates are not shared.

Place together threads with consecutive IDs to improve locality.

S-W within a tile

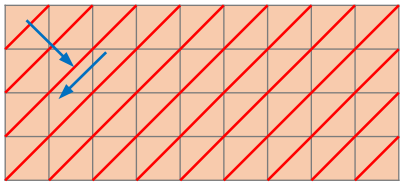
Auto-vectorization with SIMD

```
for (int row = 0; row < alpha; row++)
    for (int col = 0; col < beta; col++)
        H[row][col] = max(0, …);
```



- *loop was not vectorized: vector dependence prevents vectorization*

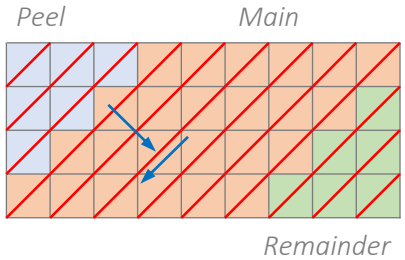
```
for (int d = 0; d < (alpha + beta - 1); d++)
    int row = max(0, d - beta + 1);
    int erow = min(d, alpha - 1);
    int col = min(d, beta - 1);
    int ecol = max(0, d - alpha + 1);
    for (; (row <= erow) && (col >= ecol); row++, col--)
        H[row][col] = max(0, …);
```



- *loop was not vectorized: parallel loop condition operator must be one of <, <=, >, >=, or !=*

Auto-vectorization with SIMD (cont'd)

```
// main (assume beta > alpha)
for (int d = alpha - 1; d < beta; d++)
    for (int row = 0, int col = d; row < alpha; row++, col--);
        H[row][col] = max(0, …);
```



➢ *LOOP WAS VECTORIZED, *but*…*

Resulting code plenty of **gather/scatter** constructs

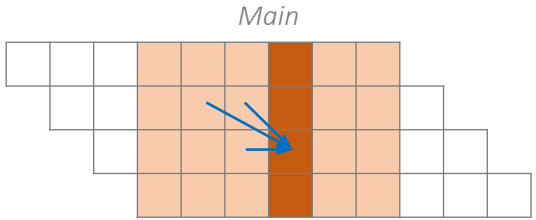
- SIMD elements are not consecutive in memory
- Scatter instruction does not even exists in AVX2 (software scatter)

What other choices?

- ***Guided*** (Cilk Plus)
- ***Low-level vectorization*** (SIMD C++ classes, intrinsics)

Vectorization with SIMD intrinsics

Main loop without peel/remainder (IMCI)

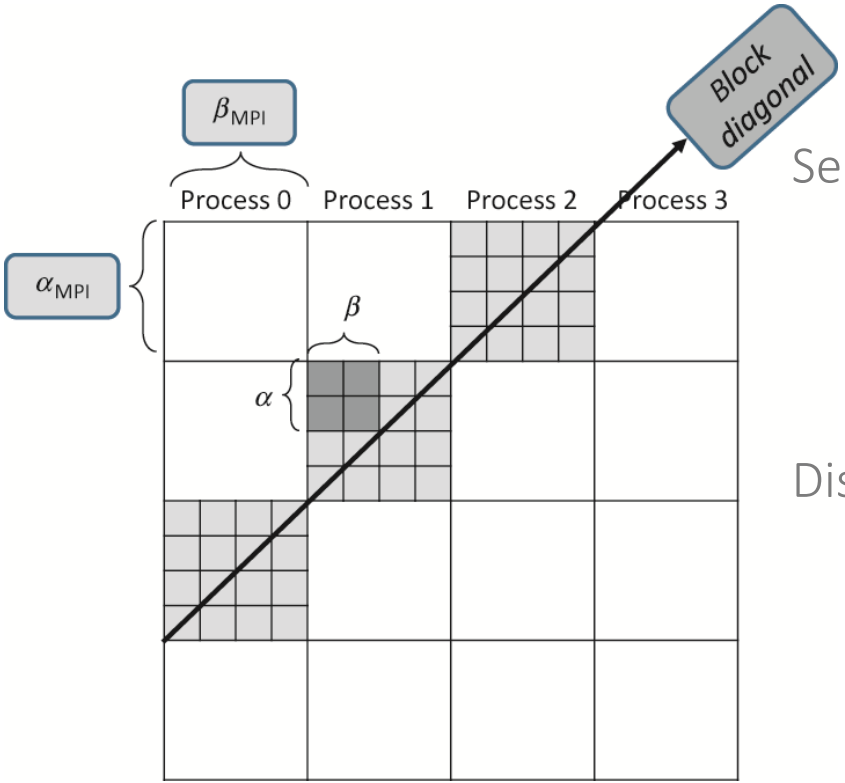


Compute column index `gCol` for current tile;

```
for (d = alpha - 1; d < beta; d++, gCol++)
{
    // vE[:] = max((GH[gcol].e :: vE[:VL-1]) - ge, (GH[gcol].h :: vH[:VL-1]) - goe)
    vHup = _mm512_alignr_epi32(vH, _mm512_set1_epi32(GH[gCol].h), 15);
    vE = _mm512_alignr_epi32(vE, _mm512_set1_epi32(GH[gCol].e), 15);
    vE = _mm512_max_epi32(__mm512_sub_epi32(vE, vGapE), _mm512_sub_epi32(vHup, vGapOE));

    // vF[:] = max(vF[:] - ge, vH[:] - goe)
    vF = _mm512_max_epi32(__mm512_sub_epi32(vF, vGapE), _mm512_sub_epi32(vH, vGapOE));

    Compute vScore[:] = σ(vS1[:], S2[gCol] :: vS2[0:VL-1]);

    // vH[:] = max(vZero, vD[:] + vScore[:], vE, vF)
    vH = _mm512_max_epi32(_mm512_max_epi32(vD, vScore), _mm512_max_epi32(vE, vF));
    vD = vHup;  // Diagonal values for next iteration

    Save vH[VL-1] and vE[VL-1] on GH (new cell on the last row of the stripe);
}
```

Save `vH[0]` and `vF[0]` on `GV.out` (new cell on the last column of the stripe);

Inter-process parallelization with MPI



Second level tiling

- Blocks size is $\alpha_{mpi} \times \beta_{mpi}$
- Matrix size is $R_{mpi} = \left\lceil \frac{m}{\alpha_{mpi}} \right\rceil \times C_{mpi} = \left\lceil \frac{n}{\beta_{mpi}} \right\rceil$
- $\beta_{mpi} = \left\lceil \frac{n}{P} \right\rceil$, being P number of MPI ranks

Distributed approach

- Block-level GH, GV, GD distributed buffers
- MPI communication between block diagonals
  - Non-blocking MPI communication primitives
    - `MPI_Isend()`, `MPI_Irecv()`
- `MPI_Barrier()` for synchronization

Final reduction with `MPI_Reduce()`

Performance evaluation: preliminary questions

Hint: not all answers need to be correct here…

What's the ideal target? Xeon, Xeon Phi, or offload?

- Problem is highly parallel, Xeon Phi should be better target
- There should be no big difference in using native or offload approach
  - Computational complexity bigger than space complexity

Compute bound or memory bandwidth bound?

- Kind of "stencil" which tends to be memory bound
- *But*, proposal does not build a real matrix, so should be compute bound

What's the expected strong/weak scalability?

- Highly parallel and compute bound might indicate good scalability
- Possible issues for MPI/thread communication/synchronization

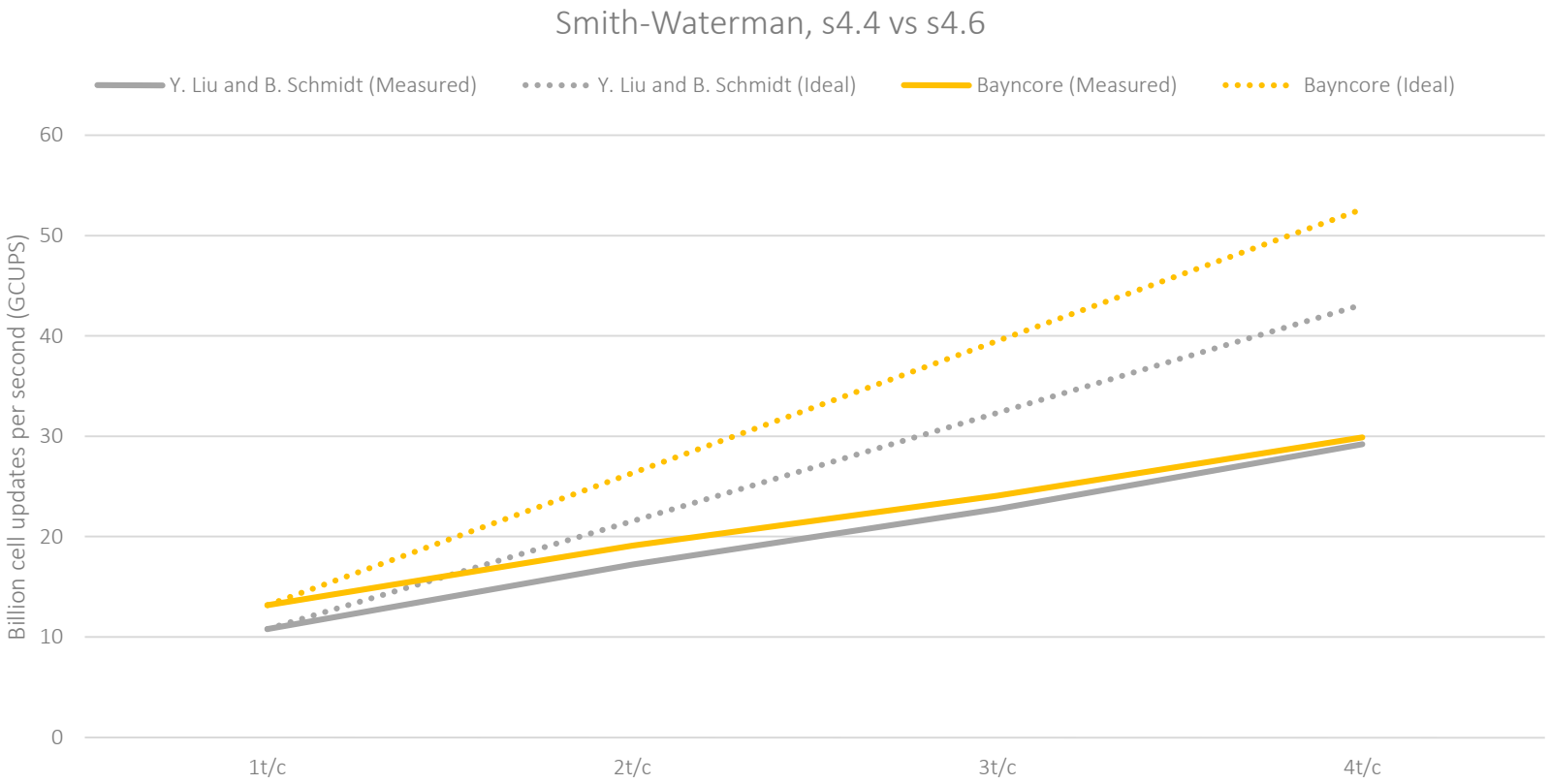
Methodology

	Y. Liu and B. Schmidt			Bayncore
Intel® Xeon® Processor	E5-2670 (2s, 8c/s, 2ht/c)			E5-2650 v2 (2s, 8c/s, 2ht/c)
Intel® Xeon Phi™ Coprocessor	4x 5110P (60c, 4t/c, 1GHz)			2x 7120A (61c, 4t/c, 1.2GHz)
Operating System	Linux			RHEL Server 6.6 (Santiago)
Intel® C++ Compiler Version	15.0 ?			15.0.2.164 (build 20150121)
OpenMP Affinity	Balanced			
MPI Library	OpenMPI			Intel® MPI 5.0
S-W MPI Implementation	Only MPI + Offload + Xeon Phi SIMD source code provided MPI ranks and MPI communication on the processor Every rank offloads to a coprocessor (1-rank per coprocessor) $\alpha = SIMD_{length}$, $\alpha_{mpi} = 128K$			
S-W Input Set (Seq1 vs Seq2) (*)	Seq.	Length	Accession No.	Genome definition
	S4.4	4,411,532	NC_000962.3	Mycobacterium tuberculosis H37Rv
	S4.6	4,641,652	NC_000913.3	Escherichia coli K12 MG1655

(*) More longer sequences in Y. Liu and B. Schmidt's performance evaluation

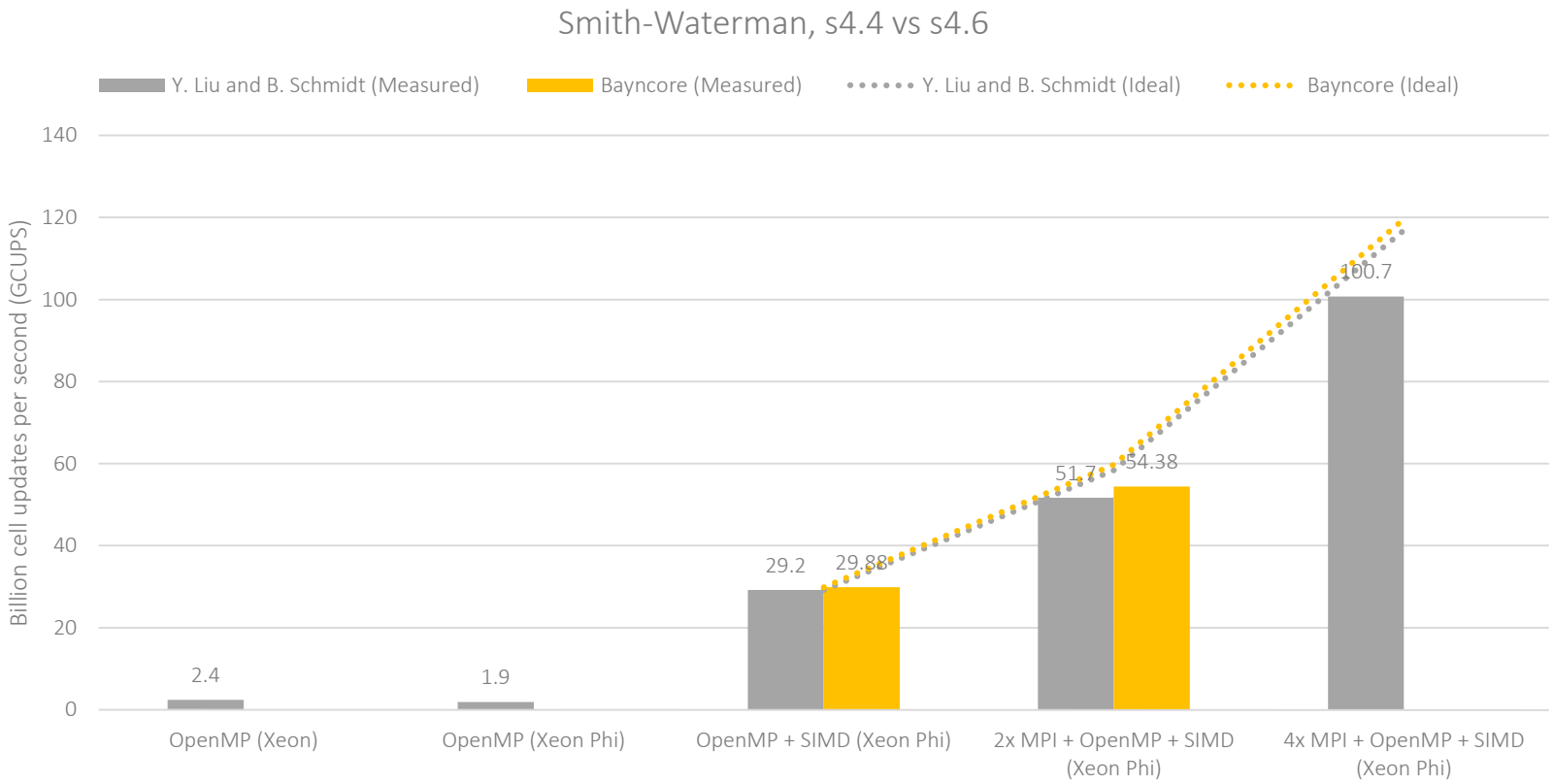
BAYNCORE

Intra-coprocessor scalability

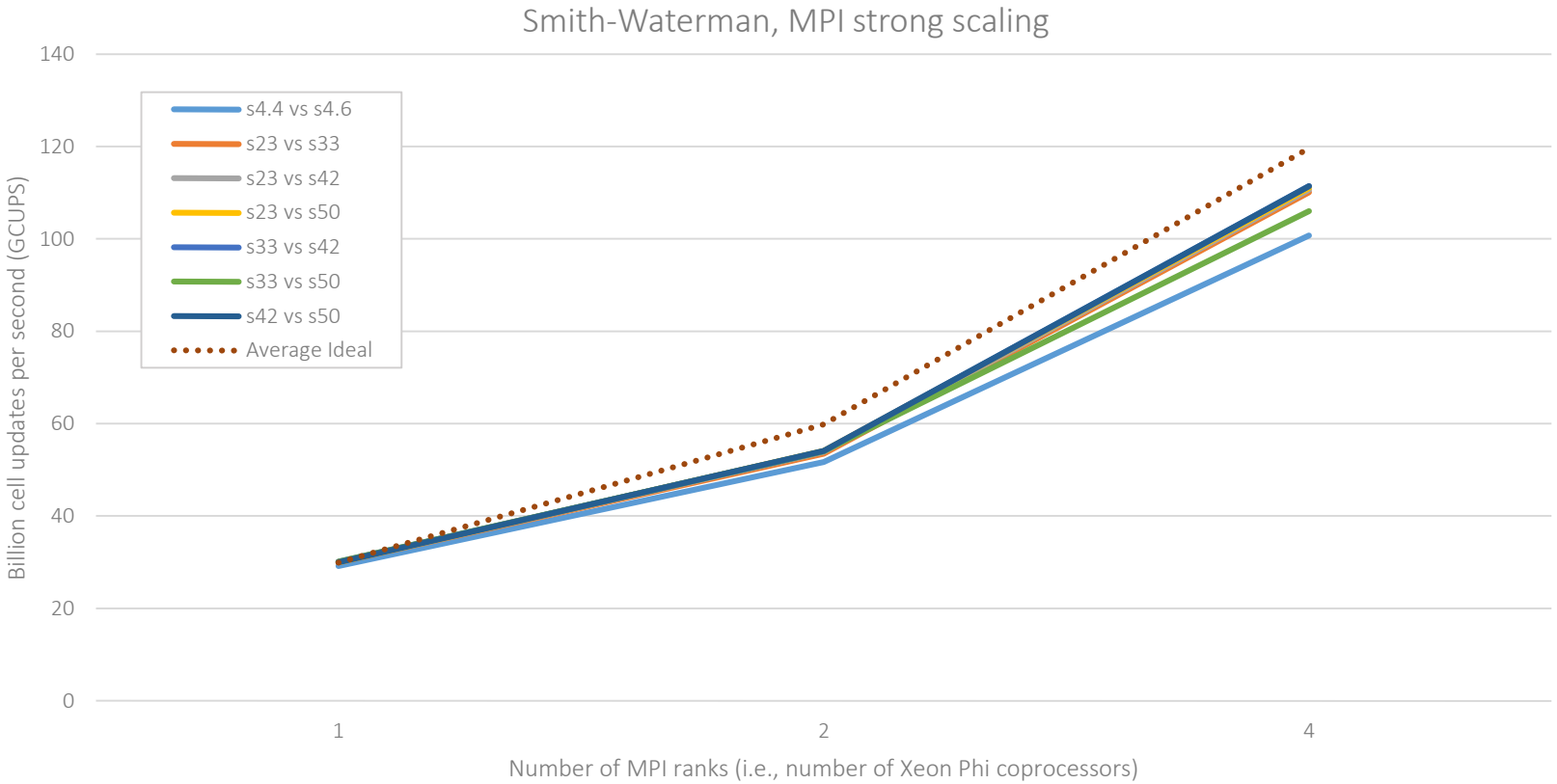


BAYNCORE

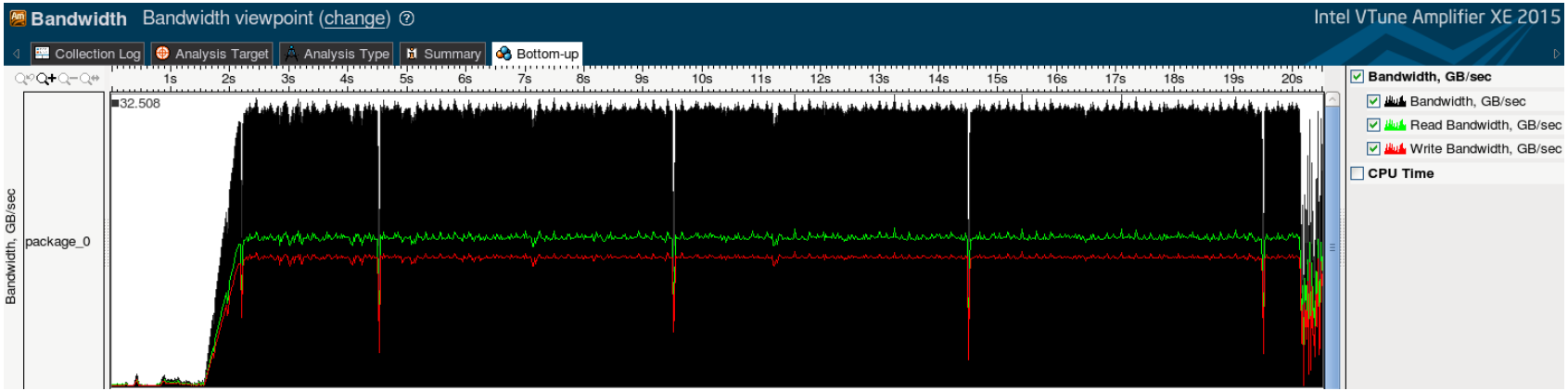
Inter-coprocessor scalability

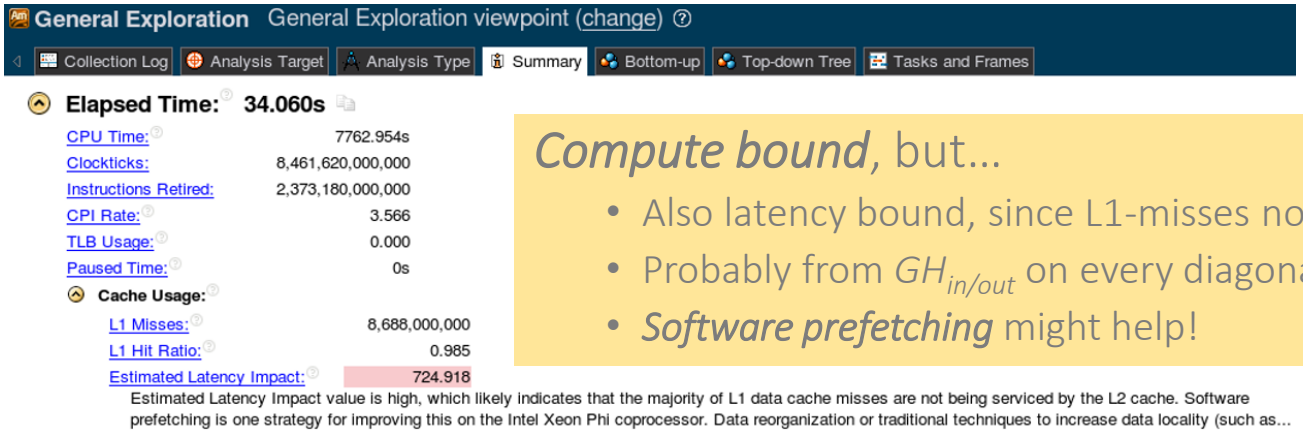


Inter-coprocessor scalability (cont'd)



Compute bound or bandwidth bound?





Compute bound, but…

- Also latency bound, since L1-misses not being served by L2
- Probably from $GH_{in/out}$ on every diagonal
- ***Software prefetching*** might help!

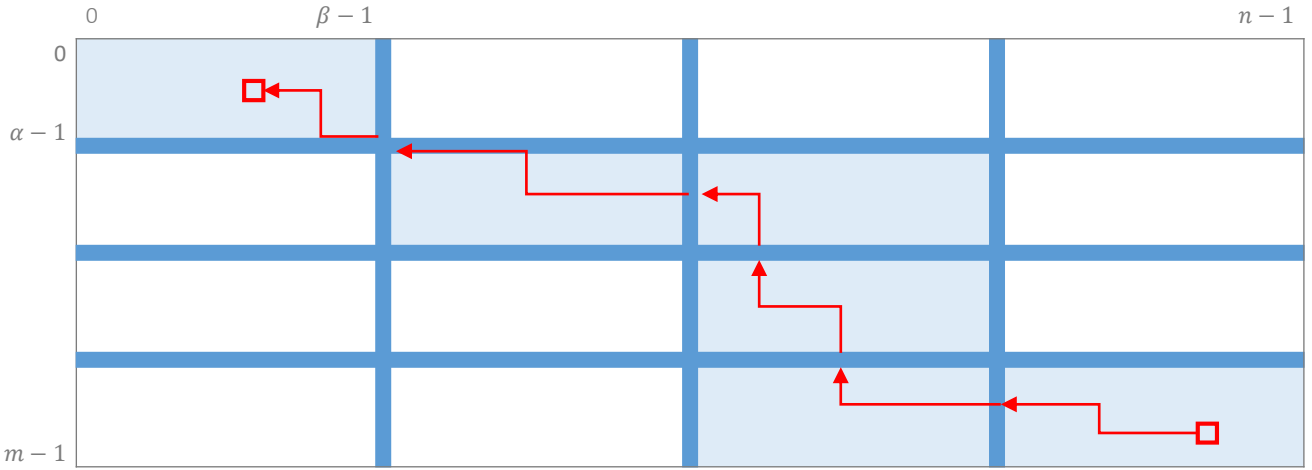
S-W Step 2: Finding optimum local alignment

Traceback procedure leading from highest score value

- It would require to store the complete score matrix, which is prohibitive!

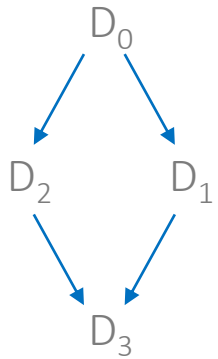
Proposal: store *partial* information of score matrix

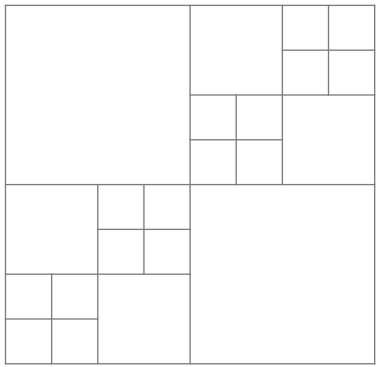
- Then recompute only the tiles following the alignment path
- Space needed from ***m x n*** down to ***C x R***



Code modernization: not a single solution

A whole family of parallel programming models





Cache oblivious algorithm

- Cilk Plus, TBB, OpenMP teams with nested parallelism, etc.

```
void diamond_sw (matrix d)
{
    if (d is small)
        small_sw(d);
    else
    {
        divide d into d0, d1, d2, d3;
        diamond_sw(d0);
        cilk_spawn diamond_sw(d1);
        /*nospawn*/diamond_sw(d2);
        cilk_sync;
        diamond_sw(d3)
    }
}
```

“Future” improvements

Intrinsics are rarely a long-term option

- Compiler ***must*** vectorize the diagonal-based loop in source code
  - Demand the compiler to vectorize the source code loop (w/ or w/o pragmas)
  - Properly generating “alignr” with the expression: `{ c[1:n] = c[0:n-1]; c[0] = x; }`
- File compiler defects at Intel Premier

Observation: values in score matrix are:

- Relatively “small” natural numbers ($\mathrm{n} < 2^k,\ n \in \mathbb{N} \wedge k = \{8,16\}$)
- Stored in 32-bit cells (i.e., 32-bit vector elements)
- Idea: we want longer vectors!

AVX-512 – Byte and Word Instruction (BWI) extension

- 512-bit vectors also supporting 16- and 8-bit datatypes (32 and 64 elements!)
  - DNA algorithms will shine on future AVX-512 BWI extension
- Currently supported by Intel compilers, but not by Intel hardware yet

Summary

Intel® Xeon Phi™ speedup

- vs Intel® Xeon Phi™ w/o SIMD: 15.4x
- vs Intel® Xeon® w/o SIMD: 12.1x
- vs “best” Intel® Xeon® w/ SIMD: ?

Benchmark ends to be compute bound

- Not bad intra-process scalability: 60%-70% efficiency
- Excellent inter-process scalability: 80%-90% efficiency

Still room for additional improvements

- Little extra performance by using MPI ranks with native Xeon Phi binaries
- Software prefetching to mitigate current latency problems
- It would be desirable to get rid of intrinsics
  - Using auto-vectorization or Intel® Cilk Plus™ array notation
- Significant future improvement targeting AVX-512 BWI extension