SWAPHI: Smith-Waterman Protein Database Search on Xeon Phi Coprocessors

Yongchao Liu and Bertil Schmidt

Institute of Computer Science, University of Mainz, Germany

E-mails: {liuy, bertil.schmidt}@uni-mainz.de

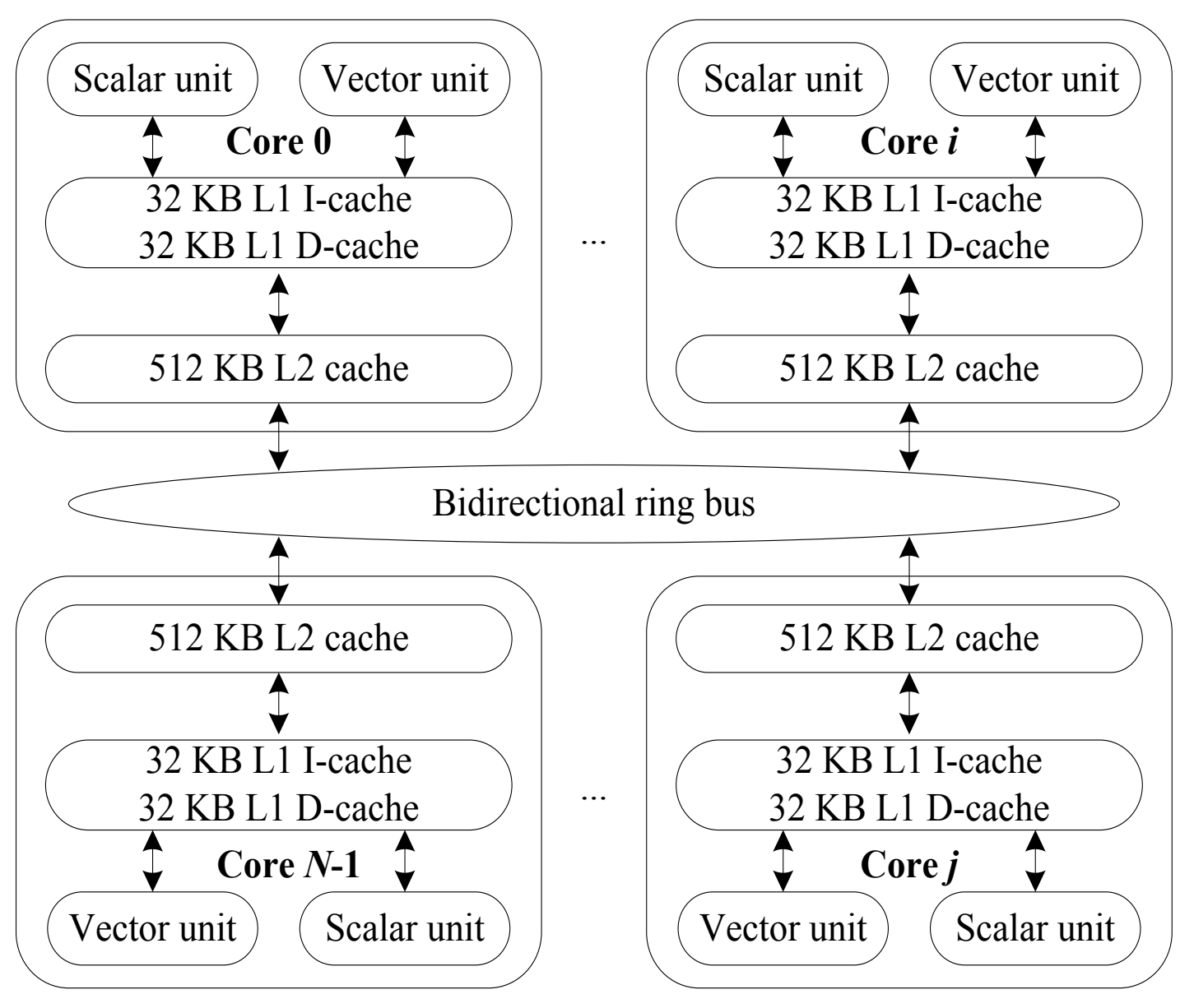
Abstract

SWAPHI (freely available at http://swaphi.sourceforge.net) is the first parallelized algorithm employing the emerging Xeon Phis to accelerate Smith-Waterman protein database search. It is designed based on the scale-and-vectorize approach, i.e. it boosts alignment speed by effectively utilizing both the coarse-grained parallelism from the many coprocessing cores (scale) and the fine-grained parallelism from the 512-bit wide single instruction multiple data (SIMD) vectors per core (vectorize). By searching against the large UniProtKB/TrEMBL protein database (13,208,986,710 amino acids), SWAPHI achieves a performance of up to 58.8 billion cell updates per second (GCUPS) on a single Xeon Phi and up to 228.4 GCUPS on four Xeon Phis.

Xeon Phi Architecture

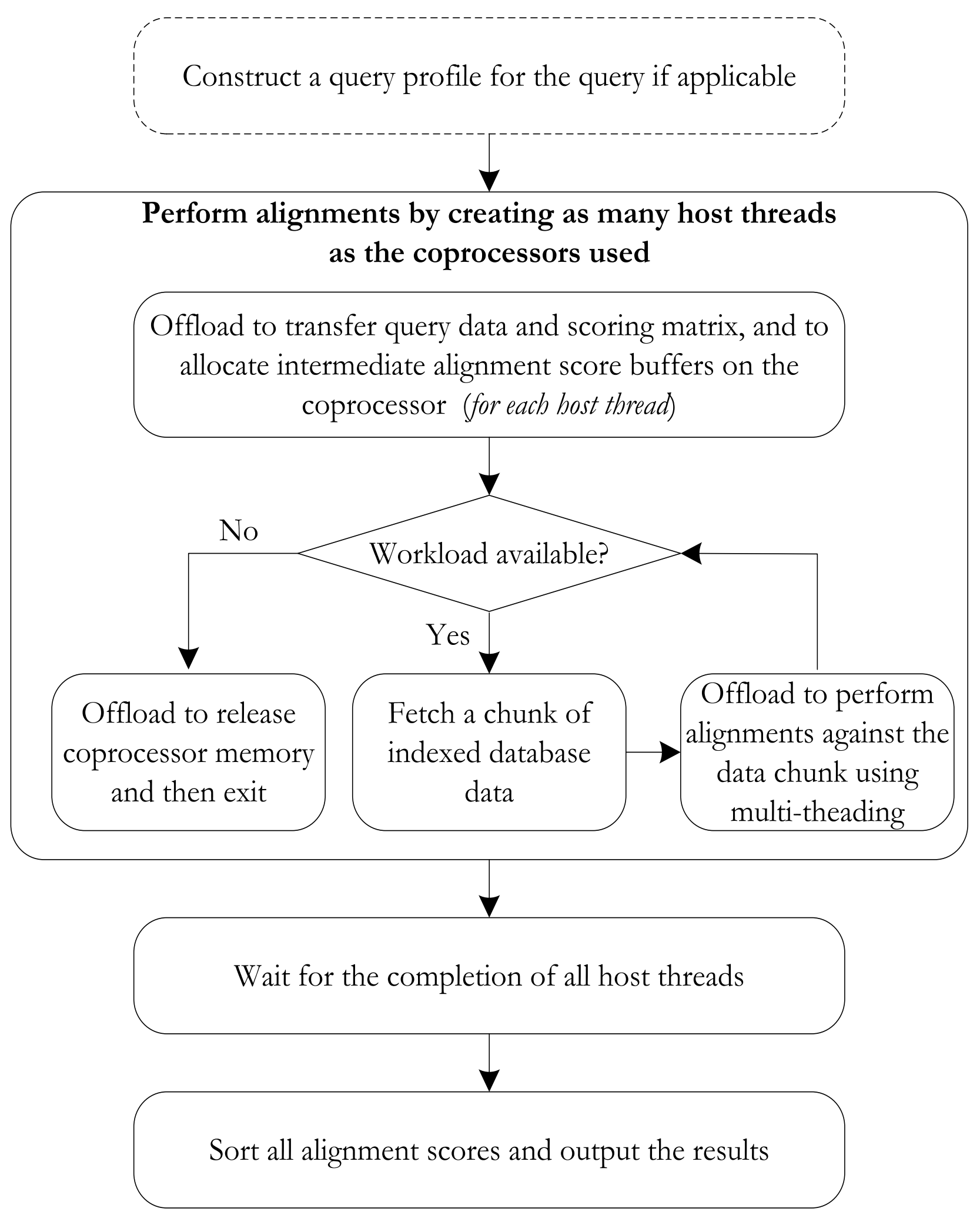
A Xeon Phi is a shared-memory many-core computer running a specialized Linux OS.

- Comprised of a set of processor cores, and each core contains 4 hardware threads
- Each core includes a new vector processing unit (VPU) featuring 512-bit wide SIMD instructions.
- Each vector can be split to either 16 32-bit-wide lanes or 8 64-bit-wide lanes
- Two usage models offload and native (we have adopted the offload model)



Implementation of SWAPHI

- Three variants:
  - Inter-sequence model with score profile (*interSP*)
  - Inter-sequence model with query profile (*InterQP*)
  - Intra-sequence model with query profile (*IntraQP*)
- Adopted a tiled computation
  - To reduce the number of memory accesses to the intermediate buffers
- Database sequence indexing
  - Used sequence profiles proposed in CUDASW++ 3.0 for GPU computing
  - Dynamic sequence data loading at the runtime chunk-by-chunk
  - Memory mapping files to allow for big databases


Construct a query profile for the query if applicable
Perform alignments by creating as many host threads as the coprocessors used
Offload to transfer query data and scoring matrix, and to allocate intermediate alignment score buffers on the coprocessor (for each host thread)
Workload available?
No
Yes
Offload to release coprocessor memory and then exit
Fetch a chunk of indexed database data
Offload to perform alignments against the data chunk using multi-theading
Wait for the completion of all host threads
Sort all alignment scores and output the results

Program workflow of SWAPHI

Category	Intrinsic functions	SIMD parallelization models	
		Inter-sequence	*Intra-sequence*
Vector mask	_mm512_int2mask		✓
Arithmetic	_mm512_add_epi32	✓	✓
	_mm512_mask_sub_epi32	✓	
Compare	_mm512_cmpge_epi32_mask	✓	
	_mm512_cmpgt_epi32_mask		✓
Initialization	_mm512_set_epi32	✓	✓
	_mm512_setzero_epi32	✓	✓
Maximum	_mm512_max_epi32	✓	✓
Load	_mm512_load_epi32	✓	✓
	_mm512_extload_epi32	✓	✓
Shuffle	_mm512_permutevar_epi32	✓	
	_mm512_mask_permutevar_epi32	✓	✓
Store	_mm512_store_epi32	✓	✓
	_mm512_packstorelo_epi32		✓
	_mm512_packstorehi_epi32		✓

Intel C++ compiler intrinsic functions used

(a) Load and pre-process subject sequence residues (outer loop of the SW algorithm)

```
vecInt16 = _mm512_set1_epi32(16); /*offset register*/
/*load one residue vector from  the subject sequence profile (__m128i* __restrict__ sequences)*/
vecDB = _mm512_extload_epi32(sequences, _MM_UPCONV_EPI32_UINT8, _MM_BROADCAST32_NONE, 0);
/*compare each residue index with 16 and returns a vector mask*/
vecMask = _mm512_cmpge_epi32_mask(vecDB, vecInt16);
/*adjust the residue indices that are greater than or equal to 16*/
vecDB = _mm512_mask_sub_epi32(vecDB, vecMask, vecDB, vecInt16);
```

(b) Load substitution scores for each query position (inner loop of the SW algorithm)

```
/*load the low and high 16 elements of the query profile row (__m128i * __restrict__ qrfRow)*/
vecLo = _mm512_extload_epi32(qprfRow, _MM_UPCONV_EPI32_SINT8, _MM_BROADCAST32_NONE, 0);
vecHi= _mm512_extload_epi32(qprfRow + 1, _MM_UPCONV_EPI32_SINT8, _MM_BROADCAST32_NONE, 0);
/*get the substitution scores*/
vecSubScore = _mm512_permutevar_epi32(vecDB, vecLo);
vecSubScore = _mm512_mask_permutevar_epi32(vecSubScore, vecMask, vecDB, vecHi);
```

Code segments used for substitution score loading from a query profile

Smith-Waterman Algorithm

Given two sequences S_1 and S_2, the recurrence of the Smith-Waterman algorithm with affine gap penalty is defined as

$$H_{i,j} = \max\{0, E_{i,j}, F_{i,,j}, H_{i-1,j-1} + sbt(S_1[i], S_2[j])\}$$

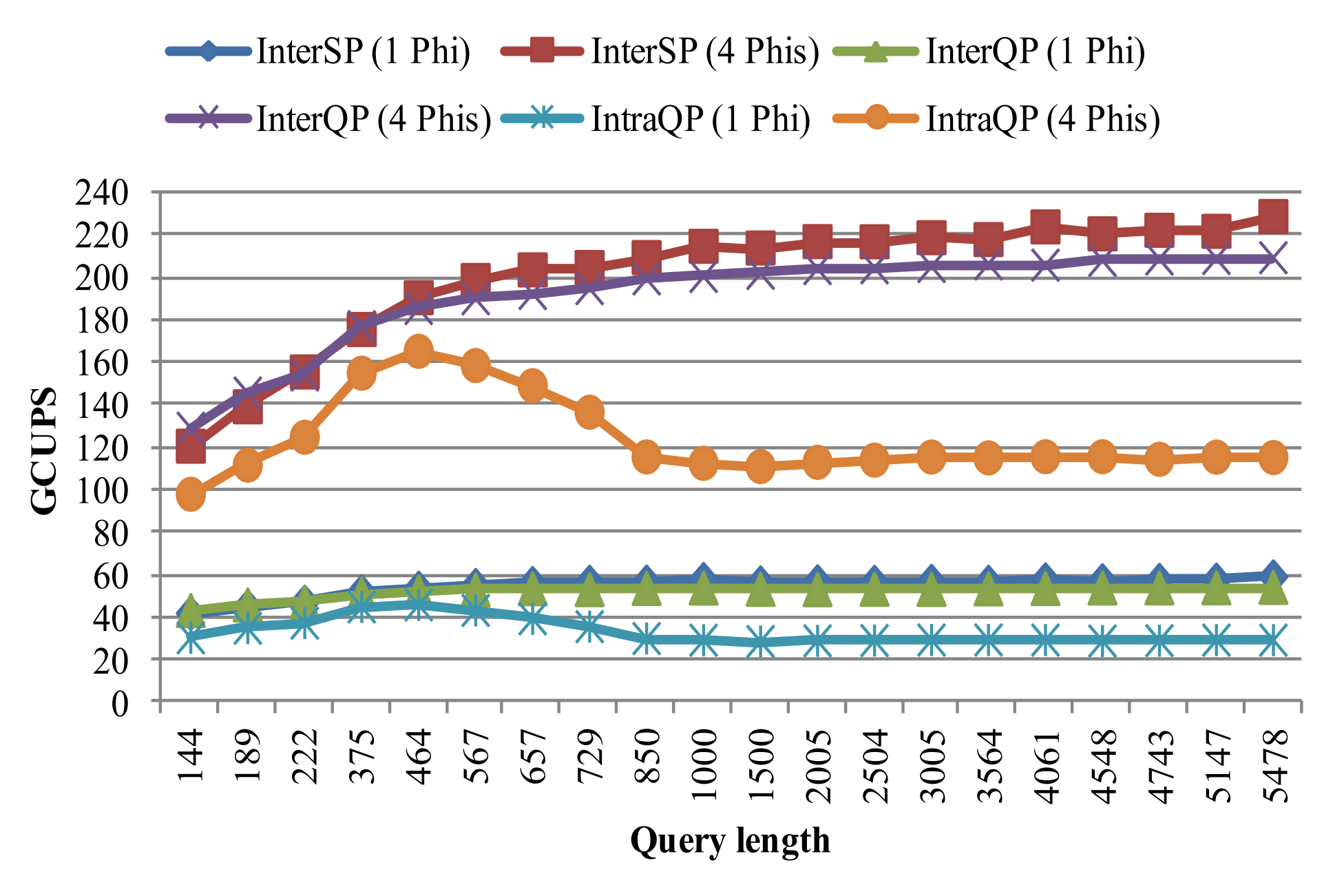
$$E_{i,j} = \max\{E_{i-1,j}, H_{i-1,j} - \alpha\} - \beta$$

$$F_{i,j} = \max\{F_{i,j-1}, H_{i,j-1} - \alpha\} - \beta$$

- α is the gap opening penalty
- β is the gap extension penalty
- *sbt* is a scoring function (usually represented as a scoring matrix) that defines the substitution scores between characters

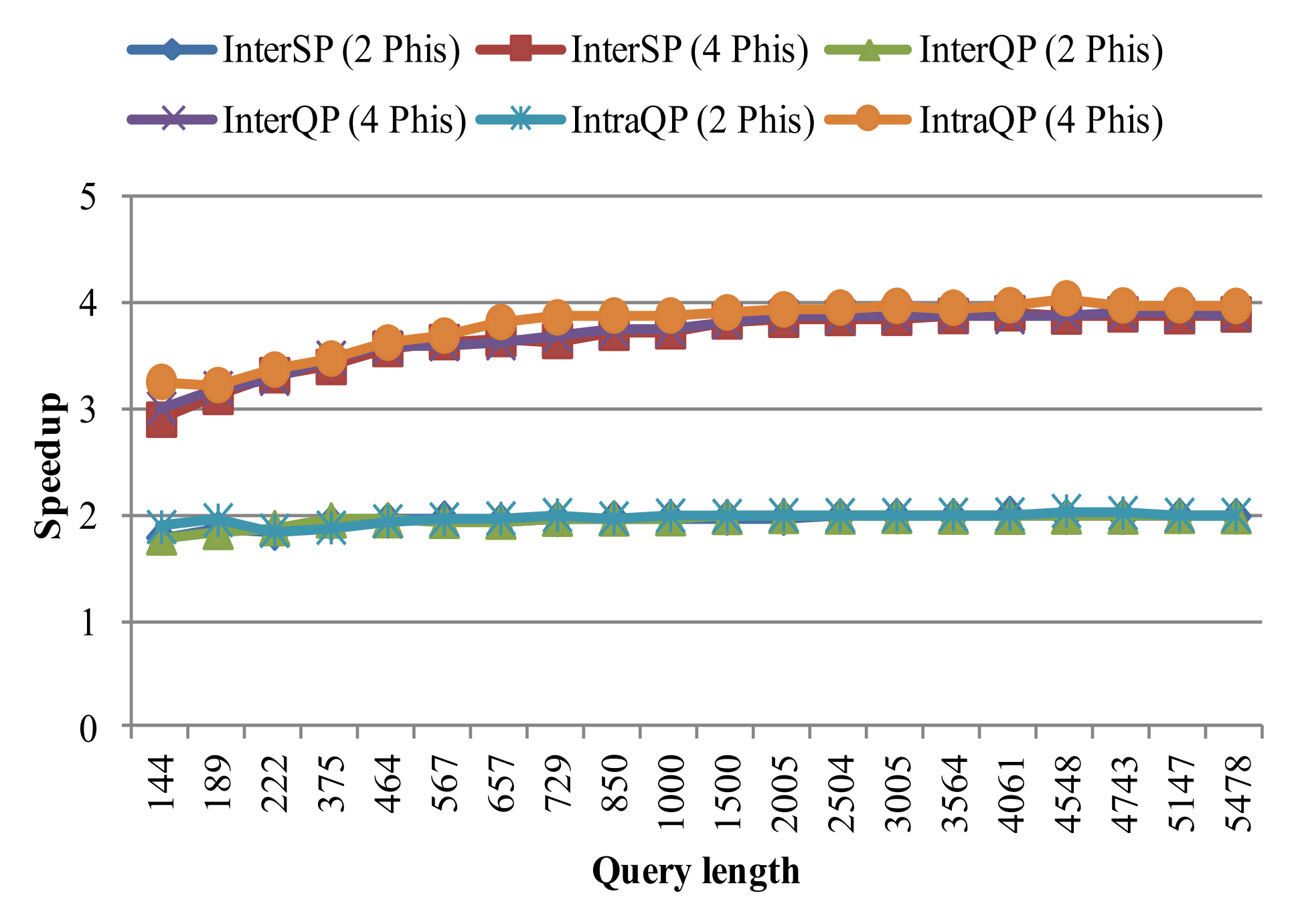
Performance Evaluation

- 20 protein query sequences
  - Lengths range from 144 to 5,478
- UniProtKB/TrEMBL database
  - Contains 41,451,118 sequences
  - Has 13,208,986,710 amino acids
- A compute node with two Intel E5-2670 8-core 2.60 GHz CPUs and 64 GB RAM
- 4 Xeon Phis
  - Product name B1PRQ-5110P/5120D
  - Each Xeon Phi has 60 processor cores and 7.9 GB RAM



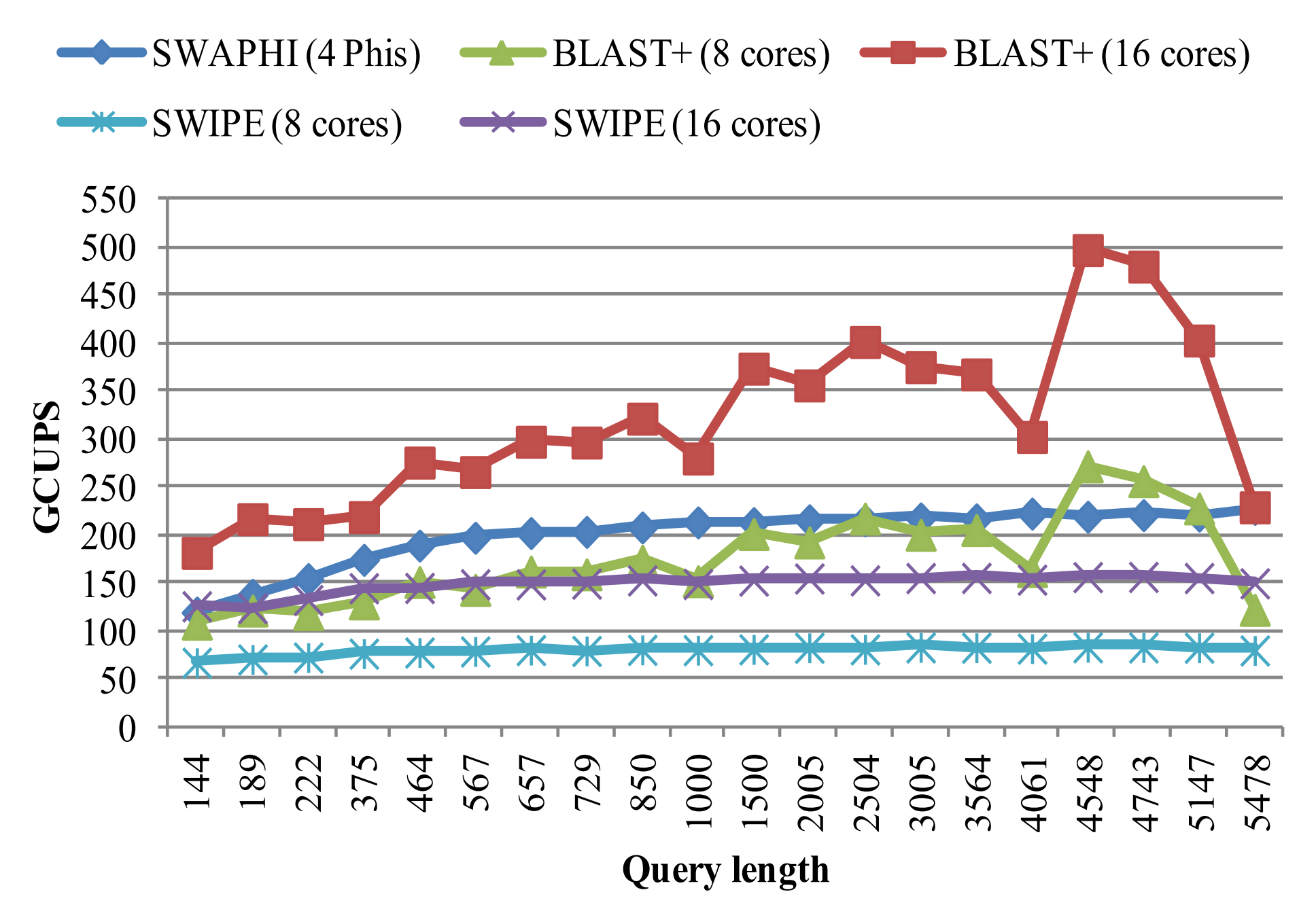
Comparison between our three variants

- Both *InterSP* and *InterQP* are superior to *IntraQP*
- SWAPHI achieves a performance of up to 58.8 GCUPS on a single Xeon Phi and up to 228.4 GCUPS on four Xeon Phis, by using *InterSP*.



Scalability in terms of number of Xeon Phis

- Compared to BLAST+ on 8 cores, SWAPHI performs better for most queries and runs 1.19× faster on average (1.86× maximally)
- Compared to SWIPE on 8 and 16 cores, SWAPHI gives a speedup of 2.49 and 1.34 on average (2.83 and 1.52 maximally), respectively



Comparison to SWIPE and BLAST+

Conclusion and Future Work

- Computational characteristics observed from our programming and evaluations:
  - Device memory accesses on the Xeon Phi are still heavy in some sense, albeit with two-level caching and high memory bandwidth.
  - Data accesses should be aligned as much as possible.
  - Gather intrinsic functions are not as lightweight as expected, even if the data accesses has good locality.
- Future work
  - Employ a hybrid parallelism model to concurrently conduct alignments on both CPUs and Xeon Phis.
  - Trace back optimal alignments on Xeon Phis for short biological sequences, e.g. next-generation sequencing reads.