"Case Study: optimization of Profrager, a protein structure and function prediction tool developed at LNCC”

Silvio Stanzani, Rogério Iope, Raphael Cóbe, Igor Freitas

Agenda

- NCC Presentation 5 min
- Hybrid Parallel Architectures 10 min
- Protein Structure Prediction / Profrager 10 min
- Advisor 10 min
- Evaluating Profrager with Advisor 15 min
- MultiCore optimization 15 min
- MultiCore/ManyCore optimization 15 min
- Evaluation and Results 10 min

UNESP Center for Scientific Computing

- Consolidates scientific computing resources for São Paulo State University (UNESP) researchers
  - It mainly uses Grid computing paradigm
- Users
  - UNESP researchers, students, and software developers
  - SPRACE project (São Paulo Research and Analysis Center)
    - Caltech, Fermilab, CERN
    - São Paulo CMS Tier-2 Facility

UNESP Center for Scientific Computing

SPRACE - CMS Tier2 Facility

- 144 worker nodes
  - Physical/Logical CPUs: 288/1088
  - HEPSpec06: 13698
- 02 head nodes
- 04 auxiliary servers
- 12 storage servers
  - 1 PB (raw), 0.85 PB (effective): 81% usage
- CSC Network
  - LAN: 1 Gbps & 10 Gbps
  - MAN: 10 Gbps & 100 Gbps
  - WAN: 4x10Gbps & 100 Gbps

Intel ® Partnership

- IPCC (Intel Parallel Computing Center)
  - Vectorization of Geant (**GE**ometry **AN**d **T**racking)

- Intel Modern Code
  - Workshops and Tutorials
    - ❑ High Performance Computing (HPC)
    - ❑ Data Science
  - HPC Consultancy

Agenda

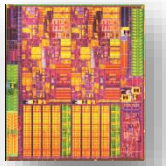
- NCC Presentation
- Hybrid Parallel Architectures
- Protein Structure Prediction / Profrager Advisor
- Evaluating Profrager with Advisor
- MultiCore optimization
- MultiCore/ManyCore optimization
- Evaluation and Results

Exploring paralelism

- Different Kinds of Parallelism:
  - Node Level Parallelism (Cluster)

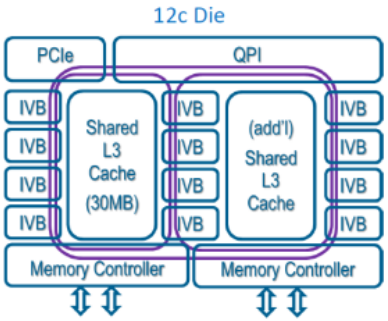
  - Thread/Task Level Parallelism (Multi & Many Core)

  - Data Level Parallelism via SIMD (Intel® AVX & Intel® MIC)



  - Instruction Level Parallelism (Processor Architecture)

Data Level Parallelism via SIMD



C/C++ ou Fortran Code

Multithreading parallelism

Thread 0 / Core 0

Thread 1/ Core 0

Thread 2 / Core 1

⋮

Thread 40 / Core 20

Thread 0/Core 0

Thread 1/Core 0

Thread 2/Core 0

⋮

Thread 244 /Core 61

Micro-architecture optimization

Vector Processor Unit per Core

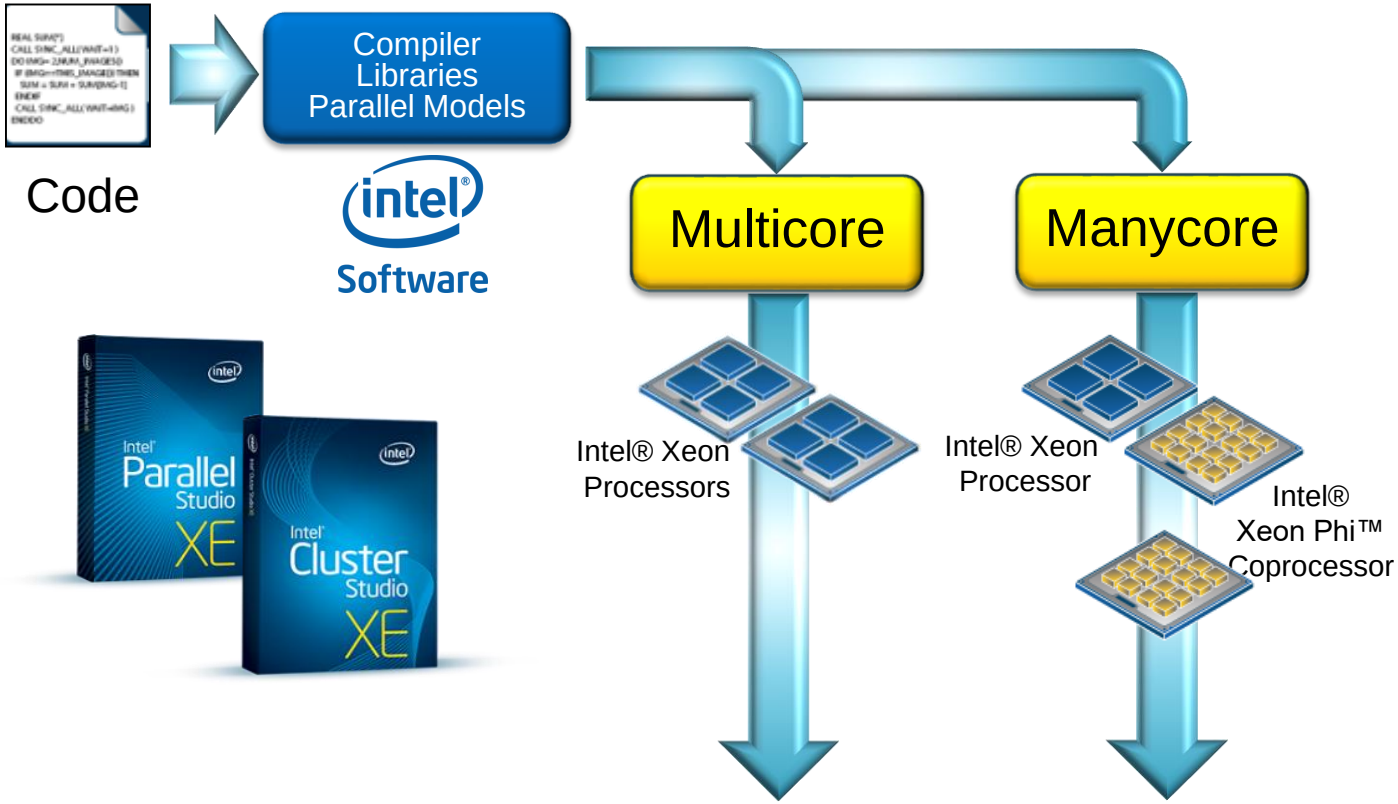
128 Bits

256 Bits

Vector Processor Unit per Core

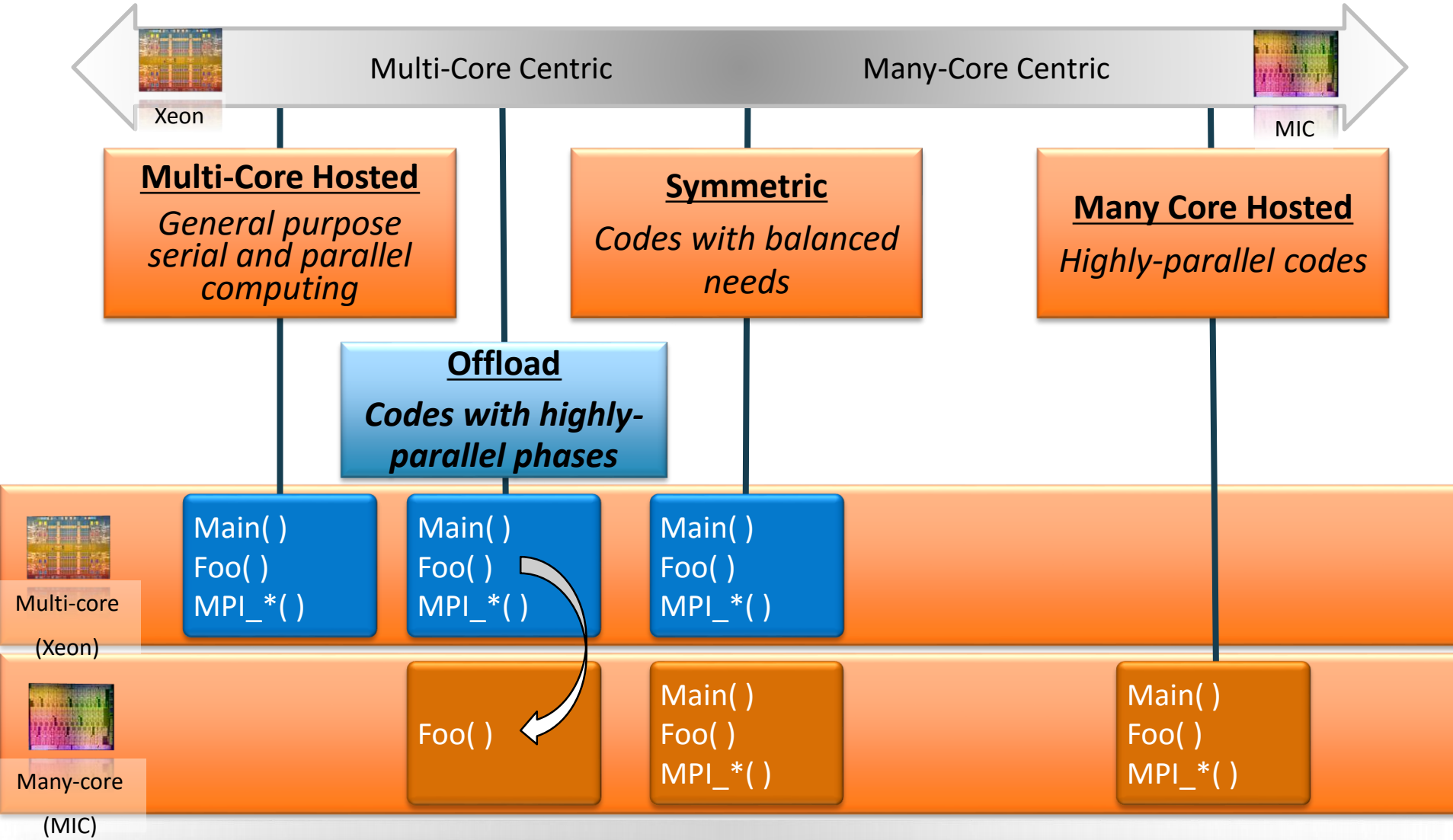
512 Bits

Programming Models



Standards Programming Models
Vectorize, Parallelize, & Optimize

Programming Models



Range of models to meet application needs

Agenda

- NCC Presentation
- Hybrid Parallel Architectures
- Protein Structure Prediction / Profrager Advisor
- Evaluating Profrager with Advisor
- MultiCore optimization
- MultiCore/ManyCore optimization
- Evaluation and Results

Protein Structure Prediction (PSP)

- Protein structure prediction in a challenging task in bioinformatics;
  - Consists in the prediction of the three-dimensional structure of a protein from its amino acid sequence;
  - The knowledge about the structure of a protein helps the understanding of the protein's function;

- Applications: Development of new drugs, Design of novel enzymes, Projects related with the study of genome, etc;

- Fragment libraries is one of the strategies employed by several PSP methods.

Profrager

A tool for the generation of a fragment library from a database of experimentally determined structures and a target protein sequence **[1]**;

- the objective is to simplify the complexity of PSP by reducing the conformational search space;
- Information contained within the fragments is used to build the whole tridimensional structure of the target protein.

[1] dos Santos, K. B., De Oliveira, R. T. R., Custodio, F. L., Dardenne, L. E. ; 'Profrager Web Server: fragment libraries generation for protein structure prediction'; 2015; The 16th International Conference on Bioinformatics & Computational Biology; p. 38-42;

Profrager Algorithm

READ Input Files

loop 1 - loop all positions of the target sequence protein amino acid (Fasta File)

loop 2 - loop all sequence of structural database (db.db) Protein Data Bank (PDB)

createFragments()

fim loop2
sort fragments
loop 6 – print frags to output file
end loop 1

Profrager Algorithm - createFragments

```
loop 3 - loop all positions of each database sequence
   loop 4 loop from database sequence[position] to database sequence
[position[fraglen] ]
      obtain matrix score according to blossun62 matrix

   create fragment from database[position]

   search compatible Geometries;

   loop 5 - loop all positions retrieved from compatible Geometries
      compare it against psipred (GEO.INDEX) == (fasta.horiz)
      executa sum score_PSIPRED
   end loop 5

   if (score_PSIPRED+score_matrix) > score min -> save frag
end loop3
```

Agenda

- NCC Presentation
- Hybrid Parallel Architectures
- Protein Structure Prediction / Profrager Advisor
- Evaluating Profrager with Advisor
- MultiCore optimization
- MultiCore/ManyCore optimization
- Evaluation and Results

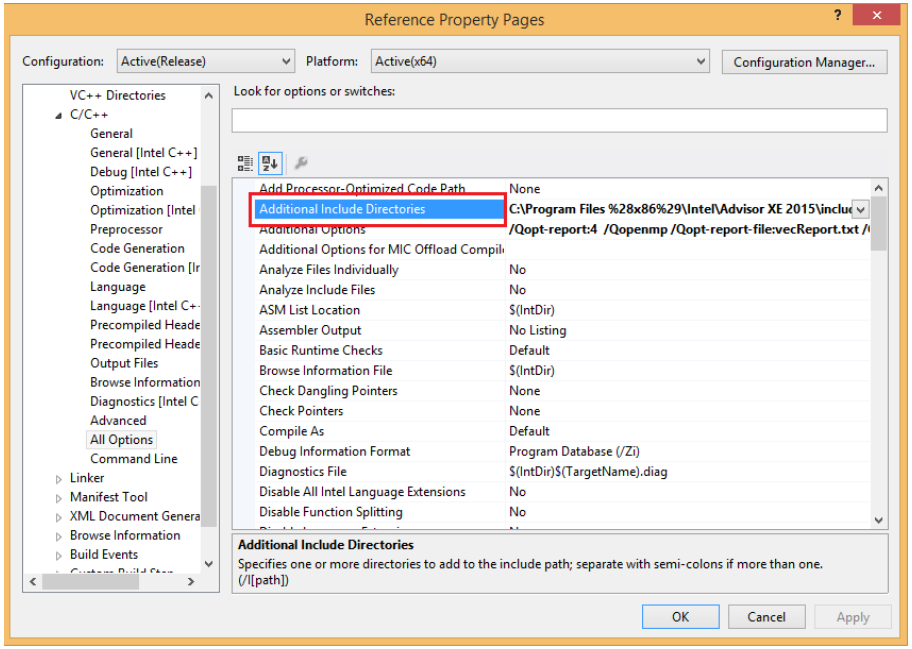
Identifying Parallelization Opportunities

- Evaluate multi-threading parallelization

- Intel® Advisor XE
  - ❑ Performance modeling using several frameworks for multi-threading in processors and co-processors:
    - OpenMP, Intel® Cilk ™ Plus, Intel® Threading Bulding Blocks
    - C, C++, Fortran (OpenMP only) e C# (Microsoft TPL)
  - ❑ Identify parallel opportunities
  - ❑ Scalability prediction: amount of threads/performance gains
  - ❑ Correctness (deadlocks, race condition)

Identifying Parallelization Opportunities

- Intel Advisor steps:
  - 1º - Include headers
  - #include "advisor-annotate.h“
  - 2º - add include reference ; link library



<u>Linux – compiling / link with Advisor</u>

icpc -O2 -openmp
02_ReferenceVersion.cpp
-o 02_ReferenceVersion
-I/opt/intel/advisor_xe/include/
-L/opt/intel/advisor_xe/lib64/

Identifying Parallelization Opportunities

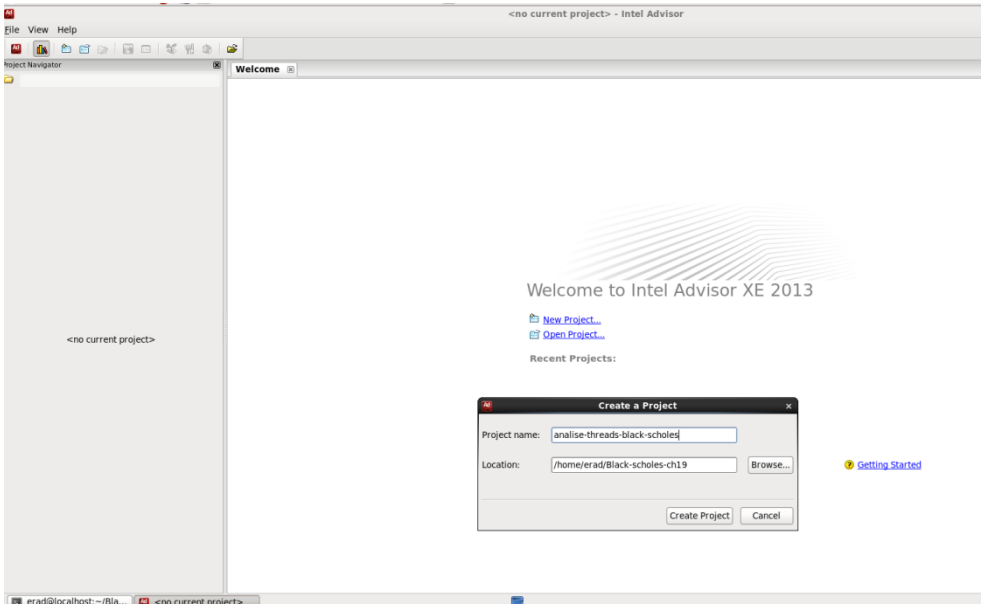
Intel Advisor steps

3º - Executing Advisor

Linux

$ advixe-gui &

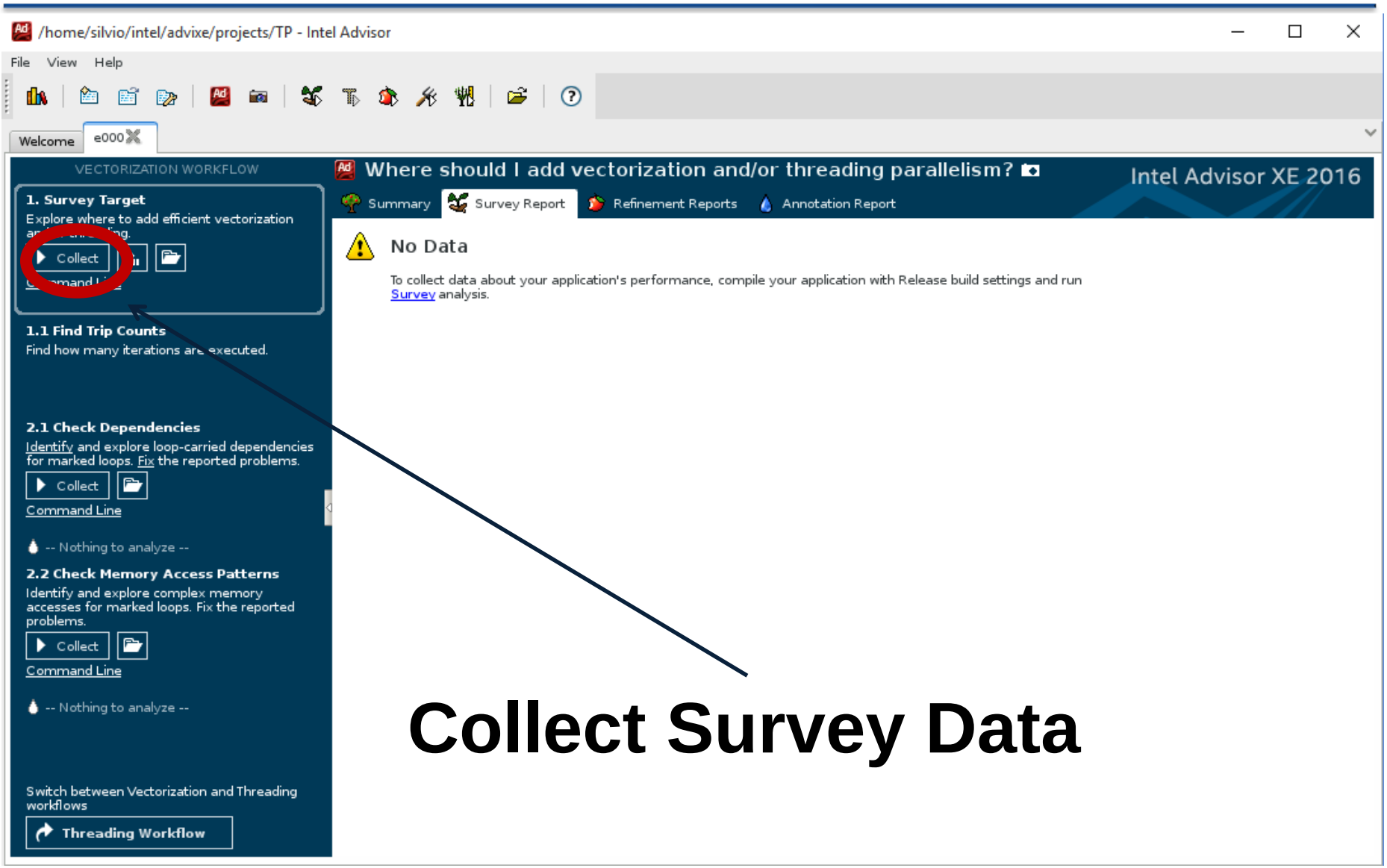
Create new project



Identifying Parallelization Opportunities

- Intel Advisor Analysis:
  - Survey
    - Vectorization of loops: detailed information about vectorization;
    - Total Time: time Elapsed in each loop considering the time involved in internal loops;
    - Self Time: time Elapsed in each loop unconsidering the time involved in internal loops;
  - Suitability
    - Speedup gains obtained parallelizing annotated loops;

Intel Advisor - Survey Data



Collect Survey Data

Intel Advisor - Survey Data

Higher instruction set architecture (ISA) available

Consider recompiling your application using a higher ISA.

Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops: Vector ISA	Vectorized Loops: Gain Estimate	Vectorized Loops: VL (Vector Length)	Vectorized Loops: Compiler Estimated Gain	Instruction Set Analysis: Traits	Instruction Set Analysis: Data Types	Advanced	Location
[loop in Transpose at Transpose.cc:20]		40.470s	40.470s	Vectorized (B...		SSE	3.43x	2; 4	3.43x		Float32; Float64	Unrolled by 4	Transpose.cc:
[loop in Transpose at Transpose.cc:20]		40.470s	40.470s	Vectorized (B...		SSE		2; 4	3.43x		Float32; Float64	Unrolled by 4	Transpose.cc:
[loop in __kmp_launch_thread at kmp_runtime.c:5900]		24.499s	24.820s	Scalar									kmp_runtime
[loop in VerifyTransposed at Main.cc:24]	1 Data type conversions present	0.171s	0.171s	Vectorized (B...		SSE2	2.73x	2; 4	2.73x	Type Conv...	Float32; Float64; Int32	Unrolled by 4	Main.cc:24
[loop in VerifyTransposed at Main.cc:24]	1 Data type conversions present	0.171s	0.171s	Vectorized (B...		SSE2		2; 4	2.73x	Type Conv...	Float32; Float64; Int32	Unrolled by 4	Main.cc:24
[loop in Transpose at Transpose.cc:20]	**1 Ineffective peeled/remainder loop ...**	**0.080s**	**0.080s**	**Vectorized ...**		**SSE**	**1.86x**	**2; 4**	**1.86x**		**Float32; Float64**		**Transpose.**
[loop in Transpose at Transpose.cc:20]		0.050s	0.050s	Vectorized (R...		SSE		2; 4	1.86x		Float32; Float64		Transpose.cc:
[loop in Transpose at Transpose.cc:20]		0.030s	0.030s	Remainder									Transpose.cc:
[loop in Transpose at Transpose.cc:17]		0.030s	40.500s	Scalar	inner loop was al...								Transpose.cc:
[loop in start_thread]		0.000s	24.820s	Scalar									
[loop in __libc_start_main]		0.000s	40.670s	Scalar									
[loop in main at Main.cc:74]		0.000s	40.600s	Scalar	loop with multipl...						Float32; Float64		Main.cc:74
[loop in VerifyTransposed at Main.cc:23]		0.000s	0.171s	Scalar						Unpacks	Float32; Float64; Int32; Int64		Main.cc:23
[loop in [OpenMP worker] at z_Linux_util.c:786]		0.000s	24.820s	Scalar									z_Linux_util.c

Source | Top Down | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: Transpose.cc:20 Transpose

Line	Source	Total Time	%	Loop Time	%	Traits
7	// You are free to use, modify and distribute this code as long as you acknowledge					
8	// the above mentioned publication.					
9	// (c) Colfax International, 2013					
10						
11	#include "Transpose.h"					
12	#include <cstdlib>					
13						
14						
15	void Transpose(FTYPE* const A, const int n) {					
16						
17	⊞ for (int j = 0; j < n; j++) {			40.499s		
18						
19						
20	⊞ for (int i = 0; i < j; i++) {	0.280s		40.550s		
21						
22	const FTYPE c = A[i*n + j];	29.950s				
23	A[i*n + j] = A[j*n + i];	10.180s				
24	A[j*n + i] = c;	0.190s				
25	}					
26						
27	}					
28						
29	}					
30						

Intel Advisor – Check Suitability

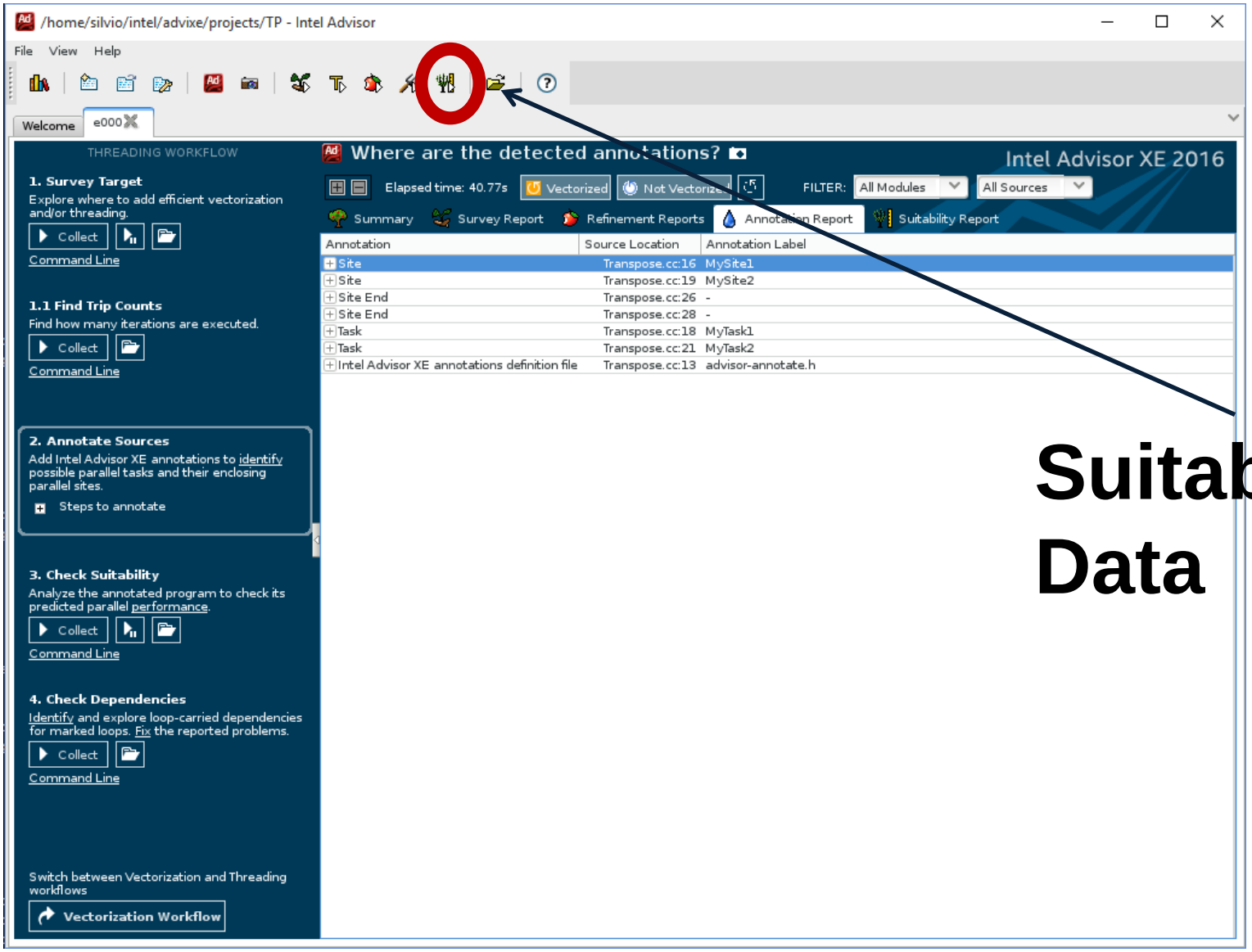
- Inserting advisor **Annotations key words** for Check Suitability:
  - **ANNOTATE_SITE_BEGIN(id)**: before beginning of loop;
  - **ANNOTATE_ITERATION_TASK(id)**: first line inside the loop;
  - **ANNOTATE_SITE_END()**: after end of loop;

- Example:

```
ANNOTATE_SITE_BEGIN( MySite2 );
for (int i = 0; i < j; i++) {
  ANNOTATE_ITERATION_TASK( MyTask2 );
  const FTYPE c = A[i*n + j];
  A[i*n + j] = A[j*n + i];
  A[j*n + i] = c;
}
ANNOTATE_SITE_END();
```

- Recompile application;

Intel Advisor – Check Suitability



Suitability Data

Intel Advisor – Check Suitability

What are the performance implications of the annotated sites?

Summary | Survey Report | Annotation Report | Suitability Report | Correctness Report

Maximum Program Gain For All Sites: 3.77x

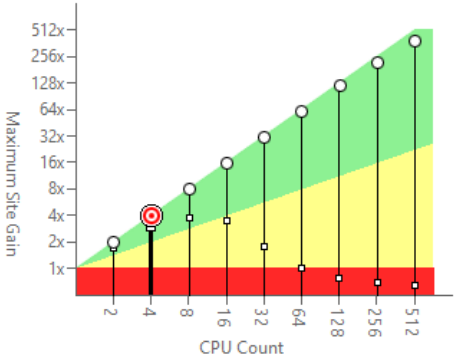
Serial time: 44.0725s
Predicted Parallel time: 11.7049s

Target System: CPU | Threading Model: Intel Cilk Plus | CPU Count: 4

Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances			Site Instance Metrics, Parallel Time
			Total Serial Time	Total Parallel Time	Site Gain	
MySite1	02_ReferenceVersion.cpp:55	3.77x	43.18s	10.81s	3.99x	10.81s

Site Performance Scalability | Site Details

Scalability of Maximum Site Gain



0.2% Load Imbalance: 0.0175s

Min Task Time: 0s
Max Task Time: 0.0029s

0.0% Runtime Overhead: < 0.0001s

Region (Site) Overhead: < 0.0001s
Task Overhead: < 0.0001s
Lock Overhead: 0s

0.0% Lock Contention: 0s

Total Parallel Time: 10.81s

Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 60000000

0.008x
0.040x
0.200x
1x (60000000)
5x
25x
125x

Avg. Iteration (Task) Duration: < 0.0001s

0.008x
0.040x
0.200x
1x (< 0.0001s)
5x
25x
125x

Apply

Runtime Modeling

Type of Change	Gain Benefit if Checked
☐ Reduce Site Overhead	
☑ Reduce Task Overhead	
☐ Reduce Lock Overhead	
☐ Reduce Lock Contention	
☑ Enable Task Chunking	

Agenda

- NCC Presentation
- Hybrid Parallel Architectures
- Protein Structure Prediction / Profrager Advisor
- Evaluating Profrager with Advisor
- MultiCore optimization
- MultiCore/ManyCore optimization
- Evaluation and Results

Profrager Survey Data

- Profrager Vectorization:
  - Only one Loop was automatically vectorized:

Loops	Vector Issues	Self Time	Total Time	Loop Type
[loop in fasta_sequence::find_blosum62 at blast_fasta.h:368]	**2 Possible inefficient memory access patterns present**	**4.680s**	**25.180s**	**Vectorized ...**
[loop in fragmento::fragmento at blast_fasta.h:435]		n/a	n/a	Vectorized (B...
[loop in std::__unguarded_partition<std::reverse_iterator<__gnu_cxx::__normal_itera...		0.000s	2.770s	Scalar Versions
[loop in main at frag_blasta.original.cpp:411]		27.490s	36.189s	Scalar
[loop in _intel_fast_memcmp]		17.899s	17.899s	Scalar
[loop in _int_free]		11.360s	11.360s	Scalar
[loop in main at frag_blasta.original.cpp:354]	1 Data type conversions present	8.720s	255.276s	Scalar
[loop in std::_Rb_tree<std::string, std::pair<std::string const, std::string>, std::_Selec...		8.030s	72.968s	Scalar
[loop in std::string::assign]		3.960s	9.780s	Scalar
[loop in std::string::string]		1.970s	1.970s	Scalar
[loop in std::transform<__gnu_cxx::__normal_iterator<char*, std::string>, __gnu_cx...	1 Assumed dependency present	1.350s	6.970s	Scalar
[loop in std::string::reserve]		0.910s	5.440s	Scalar
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>>...	1 System function call(s) present	0.870s	1.050s	Scalar
[loop in ___strtod_l_internal]		0.850s	0.850s	Scalar
[loop in memcpy]		0.530s	0.530s	Scalar
[loop in malloc_consolidate]		0.360s	0.360s	Scalar
[loop in ___strtod_l_internal]		0.310s	0.310s	Scalar
[loop in str_to_mpn.isra.0]		0.170s	0.170s	Scalar
[loop in _int_malloc]		0.150s	0.150s	Scalar
[loop in ___strtod_l_internal]		0.150s	0.430s	Scalar
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>>...		0.140s	0.250s	Scalar
[loop in ___strtod_l_internal]		0.090s	0.090s	Scalar
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>>...	1 System function call(s) present	0.090s	0.110s	Scalar
[loop in std::__unguarded_partition<std::reverse_iterator<__gnu_cxx::__normal_itera...		0.070s	52.119s	Scalar
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>>...		0.070s	1.500s	Scalar
[loop in std::__unguarded_insertion_sort<std::reverse_iterator<__gnu_cxx::__normal...	1 Assumed dependency present	0.070s	10.050s	Scalar
[loop in std::istream::sentry::sentry]		0.060s	0.060s	Scalar
[loop in memcpy]		0.050s	0.050s	Scalar
[loop in memcpy]		0.050s	0.050s	Scalar
[loop in std::istream::sentry::sentry]		0.050s	0.050s	Scalar
[loop in round_and_return]		0.050s	0.050s	Scalar
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>>...		0.040s	0.390s	Scalar
[loop in ___strtod_l_internal]		0.040s	0.040s	Scalar

```
for(i=0; i<size1; i++ )
[Vectorized (Body) loop in fasta_sequence::find_blosum62 at blast_fasta.h:368]
    Vectorized AVX; AVX2; AVX2GATHER Loop processing Int32; Int64; UInt128; UInt64 data type(s) having Gathers; Inserts; Extracts; Shifts operations
    Loop was unrolled by 1
[Remainder loop in fasta_sequence::find_blosum62 at blast_fasta.h:368]
    Scalar Remainder Loop. Not vectorized
    Loop was unrolled by 1
[Not executed loop in fasta_sequence::find_blosum62 at blast_fasta.h:368]
    Scalar Peeled loop. Not vectorized
    Loop was unrolled by 1
[Scalar loop in fasta_sequence::find_blosum62 at blast_fasta.h:368]
    Scalar Loop. Not vectorized
    No loop transformations were applied
```

Profrager Survey Data

- Listing by Total Time;

Loops	Vector Issues	Self Time	Total Time
[loop in __libc_start_main]		0.000s	345.815s
[loop in __libc_start_main]		0.000s	345.815s
[loop in main at frag_blasta.original.cpp:339]		0.000s	331.985s
[loop in main at frag_blasta.original.cpp:342]	1 Assumed dependency present	0.000s	269.196s
[loop in main at frag_blasta.original.cpp:344]		0.010s	255.456s
[loop in main at frag_blasta.original.cpp:342]		0.000s	255.456s
[loop in main at frag_blasta.original.cpp:354]	1 Data type conversions present	8.720s	255.276s
[loop in std::__introsort_loop<std::reverse_iterator<__gnu_cxx::__normal_iterator<fr ...		0.020s	226.946s
[loop in std::_Rb_tree<std::string, std::pair<std::string const, std::string>, std::_Selec ...		8.030s	72.968s
[loop in std::__unguarded_partition<std::reverse_iterator<__gnu_cxx::__normal_itera ...		0.070s	52.119s
[loop in main at frag_blasta.original.cpp:411]		**27.490s**	**36.189s**
[loop in fasta_sequence::find_blosum62 at blast_fasta.h:368]	2 Possible inefficient memory access patterns present	4.680s	25.180s
[loop in fasta_sequence::find_blosum62 at blast_fasta.h:368]		0.000s	20.500s
[loop in _intel_fast_memcmp]		17.899s	17.899s
[loop in main at frag_blasta.original.cpp:288]		0.000s	13.560s
[loop in main at frag_blasta.original.cpp:291]		0.000s	13.560s
[loop in _int_free]		11.360s	11.360s
[loop in std::__unguarded_insertion_sort<std::reverse_iterator<__gnu_cxx::__normal ...	1 Assumed dependency present	0.070s	10.050s
[loop in std::string::assign]		3.960s	9.780s
[loop in std::transform<__gnu_cxx::__normal_iterator<char*, std::string>, __gnu_cx ...	1 Assumed dependency present	1.350s	6.970s
[loop in std::string::reserve]		0.910s	5.440s
[loop in read_geo at geo_file.h:205]		0.000s	4.470s
[loop in std::__unguarded_partition<std::reverse_iterator<__gnu_cxx::__normal_itera ...		0.000s	2.770s
[loop in std::string::string]		1.970s	1.970s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...		0.070s	1.500s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...		0.000s	1.500s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...	1 System function call(s) present	0.870s	1.050s
[loop in ___strtod_l_internal]		0.850s	0.850s
[loop in memcpy]		0.530s	0.530s
[loop in ___strtod_l_internal]		0.150s	0.430s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...		0.040s	0.390s
[loop in malloc_consolidate]		0.010s	0.370s
[loop in std::getline<char, std::char_traits<char>, std::allocator<char>>]		0.010s	0.360s

Profrager Survey Data

Listing by Self Time;

Loops		Vector Issues	Self Time	Total Time
[loop in main at frag_blasta.original.cpp:411]			27.490s	36.189s
[loop in _intel_fast_memcmp]			17.899s	17.899s
[loop in _int_free]			11.360s	11.360s
[loop in main at frag_blasta.original.cpp:354]		1 Data type conversions present	8.720s	255.276s
[loop in std::_Rb_tree<std::string, std::pair<std::string const, std::string>, std::_Selec...			8.030s	72.968s
[loop in fasta_sequence::find_blosum62 at blast_fasta.h:368]		2 Possible inefficient memory access patterns present	4.680s	25.180s
[loop in std::string::assign]			3.960s	9.780s
[loop in std::string::string]			1.970s	1.970s
[loop in std::transform<__gnu_cxx::__normal_iterator<char*, std::string>, __gnu_cx...		1 Assumed dependency present	1.350s	6.970s
[loop in std::string::reserve]			0.910s	5.440s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...		1 System function call(s) present	0.870s	1.050s
[loop in ___strtod_l_internal]			0.850s	0.850s
[loop in memcpy]			0.530s	0.530s
[loop in malloc_consolidate]			0.360s	0.360s
[loop in ___strtod_l_internal]			0.310s	0.310s
[loop in str_to_mpn.isra.0]			0.170s	0.170s
[loop in _int_malloc]			0.150s	0.150s
[loop in ___strtod_l_internal]			0.150s	0.430s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...			0.140s	0.250s
[loop in ___strtod_l_internal]			0.090s	0.090s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...		1 System function call(s) present	0.090s	0.110s
[loop in std::__unguarded_partition<std::reverse_iterator<__gnu_cxx::__normal_itera...			0.070s	52.119s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...			0.070s	1.500s
[loop in std::__unguarded_insertion_sort<std::reverse_iterator<__gnu_cxx::__normal...		1 Assumed dependency present	0.070s	10.050s
[loop in std::istream::sentry::sentry]			0.060s	0.060s
[loop in memcpy]			0.050s	0.050s
[loop in memcpy]			0.050s	0.050s
[loop in std::istream::sentry::sentry]			0.050s	0.050s
[loop in round_and_return]			0.050s	0.050s
[loop in std::num_get<char, std::istreambuf_iterator<char, std::char_traits<char>> ...			0.040s	0.390s
[loop in ___strtod_l_internal]			0.040s	0.040s
[loop in _int_malloc]			0.030s	0.060s
[loop in _int_malloc]			0.030s	0.030s

Profrager Annotations

Site	frag_blasta.original.2.cpp:340	MySite1

```
338 //      cout <<"# Fields: Query id, Subject id, % identity, alignment length, mismatches, gap openings, q.
339     cerr << "\nReading fragments from \"" << bfile << "\" DB.\n";
340     ANNOTATE_SITE_BEGIN( MySite1 );
341     for(size_t qs = 0; qs<=query.lenght()-frag_len; qs++)//posição do query
342     //for (size_t qs = 0; qs <= 2; qs++)//posição do query
```

Site	frag_blasta.original.2.cpp:347	MySite2

```
345         FRAGS.clear();
346         cerr << "\rFragment position " << qs+1 << " of " << query.lenght()-frag_len;
347             ANNOTATE_SITE_BEGIN( MySite2 );
348         for(size_t subject=0; subject<DB.size(); subject++)
349         {
```

Site	frag_blasta.original.2.cpp:360	MySite3

```
358
359 //              calcula o score BLOSSUM62
360             ANNOTATE_SITE_BEGIN( MySite3 );
361             for (size_t i = 1; i <= DB[subject].seq.size() - frag_len - 1; i++)
362                 //for(size_t i=0; i<=DB[subject].seq.size()-frag_len; i++)//vê a 1º seq do banco de dados.
```

Site	frag_blasta.original.2.cpp:420	MySite4

```
418                     size_t pq = frag.pos_start_query - 1;
419                     int confi = 0, score_psip = 0;
420                     ANNOTATE_SITE_BEGIN( MySite4 );
421                     for (size_t ps = frag.pos_start_subject - 1; ps < frag.pos_end_subject; ps++, pq++)
422                     {
```

Site End	frag_blasta.original.2.cpp:434	-
Site End	frag_blasta.original.2.cpp:450	-
Site End	frag_blasta.original.2.cpp:452	-
Site End	frag_blasta.original.2.cpp:481	-
Task	frag_blasta.original.2.cpp:344	MyTask1
Task	frag_blasta.original.2.cpp:350	MyTask2
Task	frag_blasta.original.2.cpp:365	MyTask3
Task	frag_blasta.original.2.cpp:423	MyTask4
Intel Advisor XE annotations definition file	frag_blasta.cpp:51	advisor-annotate.h
Intel Advisor XE annotations definition file	frag_blasta.original.2.cpp:34	advisor-annotate.h

Profrager Suitability Data

Maximum Program Gain For All Sites: 12.14x

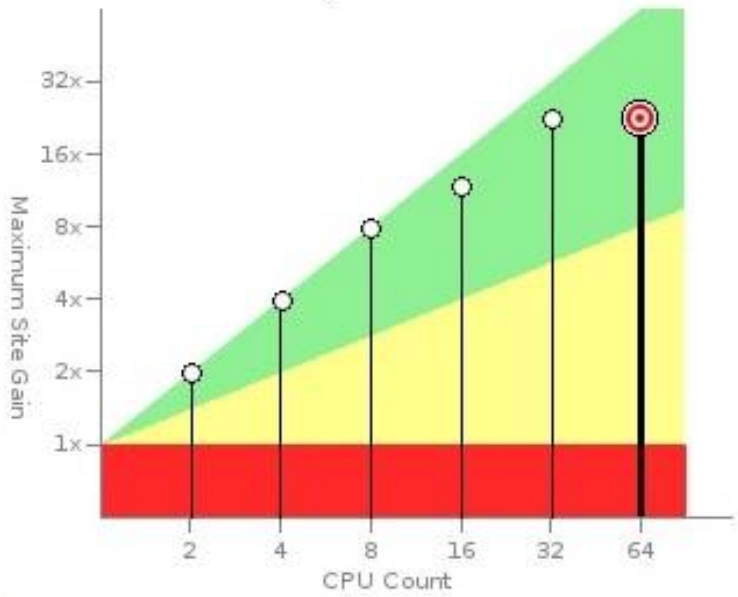
Serial time: 347.4891s
Predicted Parallel time: 28.6243s

Target System: CPU Threading Model: OpenMP

Site Label	Source Location	Impact to Program Gain	Combined Site Total Serial Time
MySite1	frag_blasta.original.2.cpp:340	11.50x	332.00s
MySite2	frag_blasta.original.2.cpp:347	2.11x	291.42s
MySite3	frag_blasta.original.2.cpp:360	0.52x	291.20s
MySite4	frag_blasta.original.2.cpp:420	0.02x	140.46s

Site Performance Scalability | Site Details

Scalability of Maximum Site Gain



Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 24

0.008x
0.040x
0.200x
1x (24)
5x
25x
125x

Avg. Iteration (Task) Duration: 13.83s

0.008x
0.040x
0.200x
1x (13.83s)
5x
25x
125x

Apply

Runtime Moc

Type of Char

- Reduce Site
- Reduce Task
- Reduce Lock
- Reduce Lock
- Enable Task

Profrager Suitability Data

Maximum Program Gain For All Sites: 12.14x

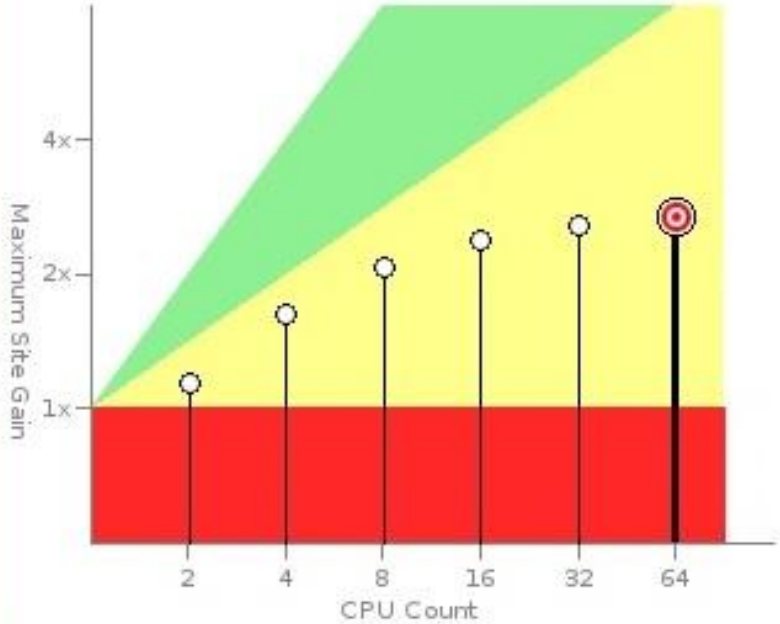
Serial time: 347.4891s
Predicted Parallel time: 28.6243s

Target System: CPU Threading Model: OpenMP

Site Label	Source Location	Impact to Program Gain	Combined Site Total Serial Time
MySite1	frag_blasta.original.2.cpp:340	11.50x	332.00s
MySite2	frag_blasta.original.2.cpp:347	2.11x	291.42s
MySite3	frag_blasta.original.2.cpp:360	0.52x	291.20s
MySite4	frag_blasta.original.2.cpp:420	0.02x	140.46s

Site Performance Scalability | Site Details

Scalability of Maximum Site Gain



Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 18886

0.008x
0.040x
0.200x
1x (18886)
5x
25x
125x

Avg. Iteration (Task) Duration: 0.0006s

0.008x
0.040x
0.200x
1x (0.0006s)
5x
25x
125x

Apply

Runtime Moc

Type of Char

Reduce Site
Reduce Task
Reduce Lock
Reduce Lock
Enable Task

Profrager Suitability Data

Maximum Program Gain For All Sites: 18.51x

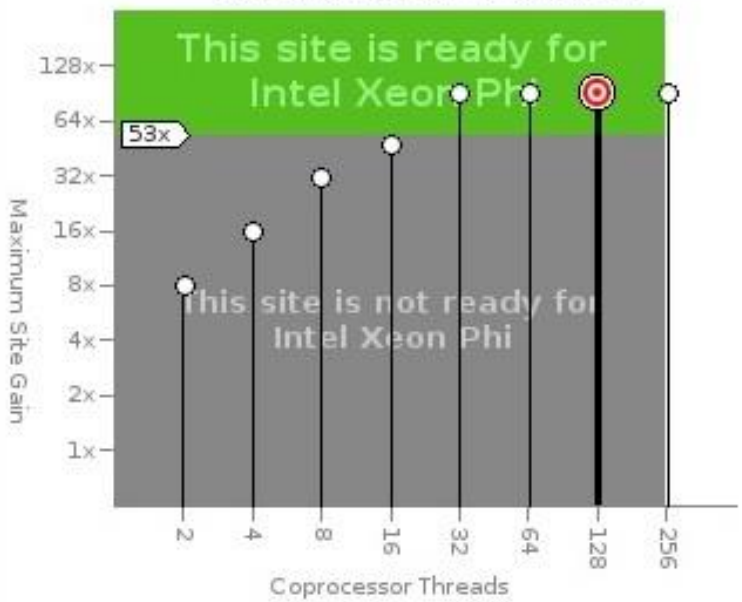
Serial time: 3474.8906s
Predicted Parallel time: 187.7634s

Target System: Intel Xeon Phi Threading Model: OpenMP

Site Label	Source Location	Impact to Program Gain	Combined Site Total Serial Tim
MySite1	frag_blasta.original.2.cpp:340	18.12x	3319.95s
MySite2	frag_blasta.original.2.cpp:347	0x	4338.13s
MySite3	frag_blasta.original.2.cpp:360	1.26x	4336.00s
MySite4	frag_blasta.original.2.cpp:420	0.03x	2358.66s

Site Performance Scalability | Site Details

Scalability of Maximum Site Gain



Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 24

0.008x
0.040x
0.200x
1x (24)
5x
25x
125x

Avg. Iteration (Task) Duration: 138.33s

0.008x
0.040x
0.200x
1x (138.33s)
5x
25x
125x

Apply

Runtime Mod

Type of Cha

- Reduce Site
- Reduce Task
- Reduce Lock
- Reduce Lock
- Enable Task

Agenda

- NCC Presentation
- Hybrid Parallel Architectures
- Protein Structure Prediction / Profrager Advisor
- Evaluating Profrager with Advisor
- MultiCore optimization
- MultiCore/ManyCore optimization
- Evaluation and Results

Profrager Optimizations

- Two optimizations was developed to **loop 1**:
  - 1. Using OpenMP to spread iterations of loop1 among Host Cores;
  - 2. Using MPI to spread iterations of loop1 among devices and then OpenMP to spread iterations of loop1 among device cores;
- Offload model can not be used because Profrager is based on C++ std::vector, that is not bitwise copyble.

Profrager Optimization - OpenMP

Main loop parallelized using **#pragma omp parallel for**:

#pragma omp parallel for

loop 1 - loop all positions of protein amino acid sequence (Fasta File)

  loop 2 - loop all sequence of structural database (db.db)

    createFragments()

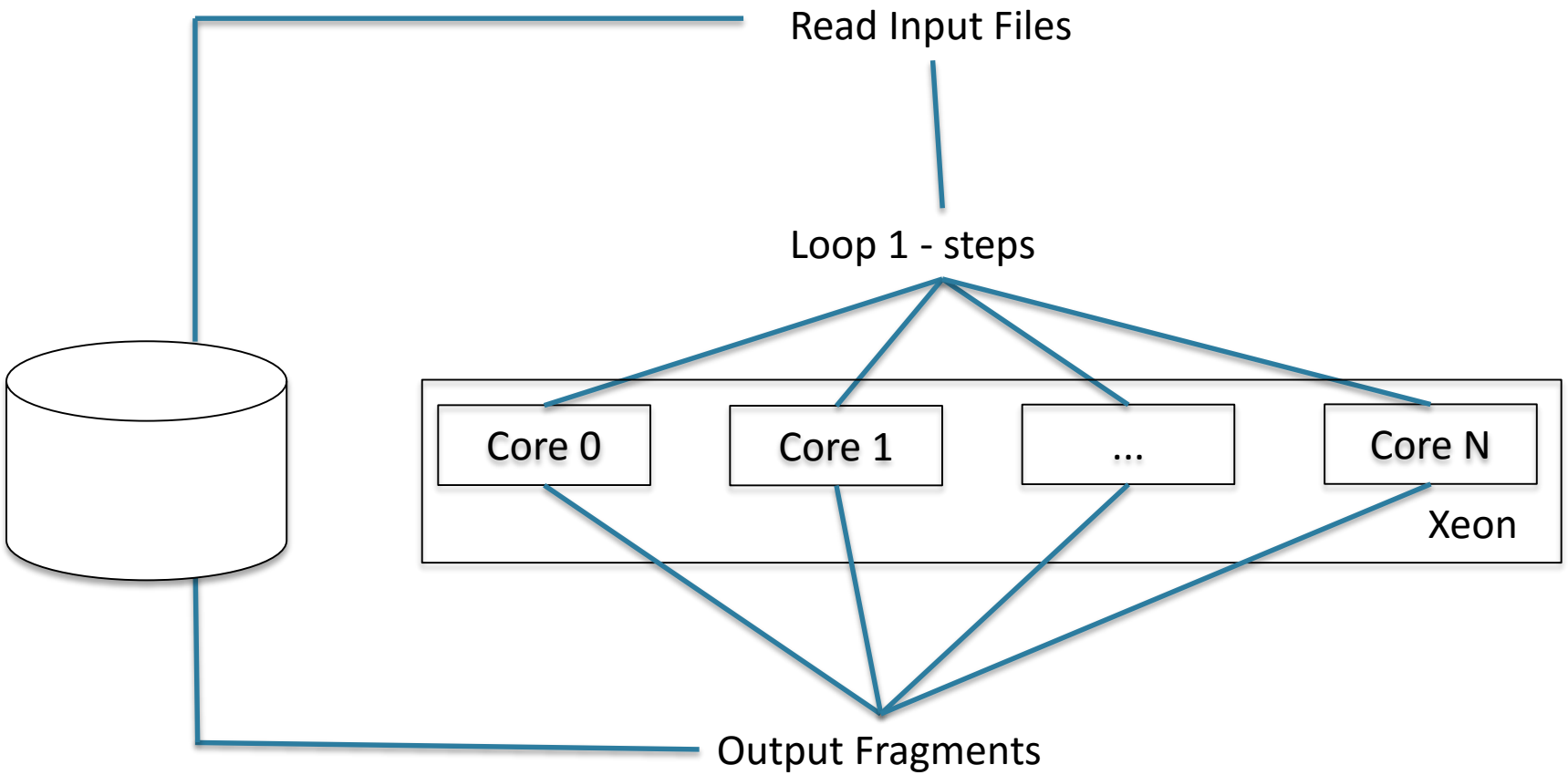
  end loop2

  sort fragments

  loop 6 – print frags to output file

end loop 1

Profrager Optimization - OpenMP



Profrager Optimization – MPI-OpenMP

- MPI/OpenMP model
  - One rank was created for host and and one for each device;
  - In each device the loop iterations are paralelized using OpenMP;
- I/O operations performed on rank 0 only:
  - Read Input files;
  - Output fragments;
- Main loop parallelized using MPI
  - Inside each rank the execution of loops 1 is parallelized using #pragma omp parallel for

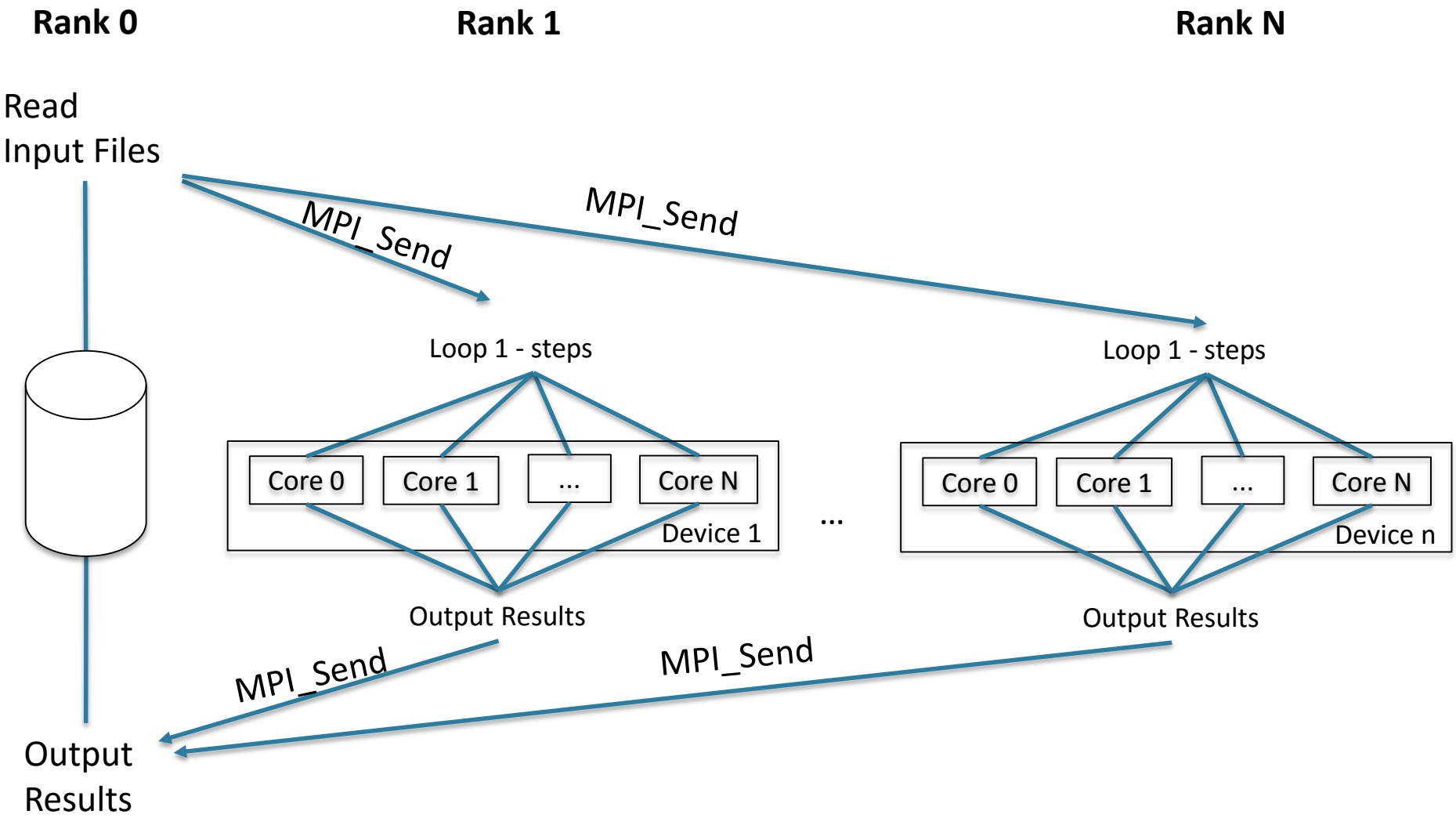
Profrager Optimization – MPI-OpenMP

```
If (rank == 0)
   READ Input Files
   mpi_send Inputfile
else
   mpisend Inputfile

#pragma omp parallel for
loop 1 - loop all positions of protein amino acid sequence (Fasta File)

If (rank == 0)
   Mpi_recv(frags);
   loop 6 – print frags to output file
} else {
   Mpi_send(frags);
}
```

Profrager Optimization - MPI-OpenMP



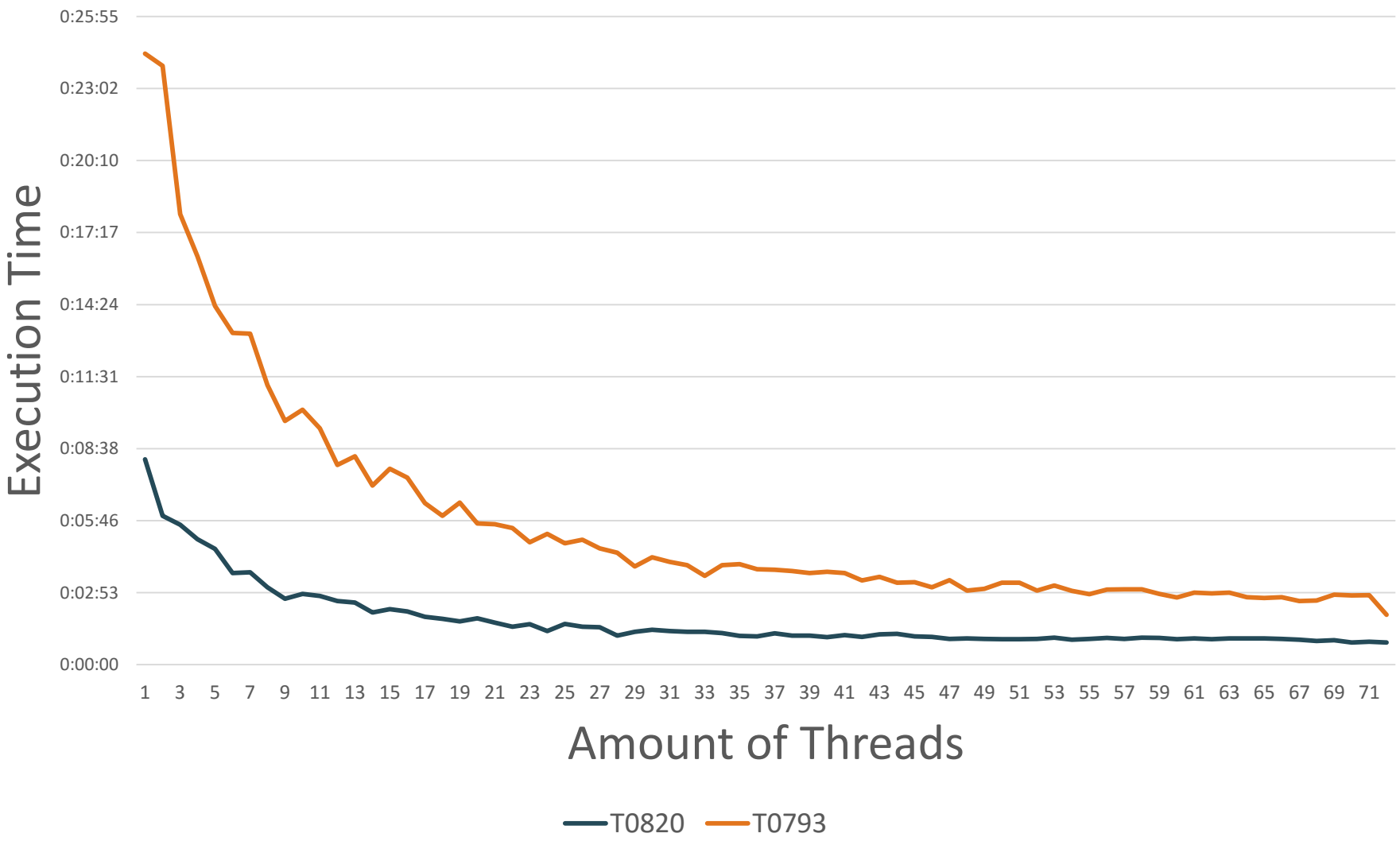
Agenda

- NCC Presentation
- Hybrid Parallel Architectures
- Protein Structure Prediction / Profrager Advisor
- Evaluating Profrager with Advisor
- MultiCore optimization
- MultiCore/ManyCore optimization
- Evaluation and Results

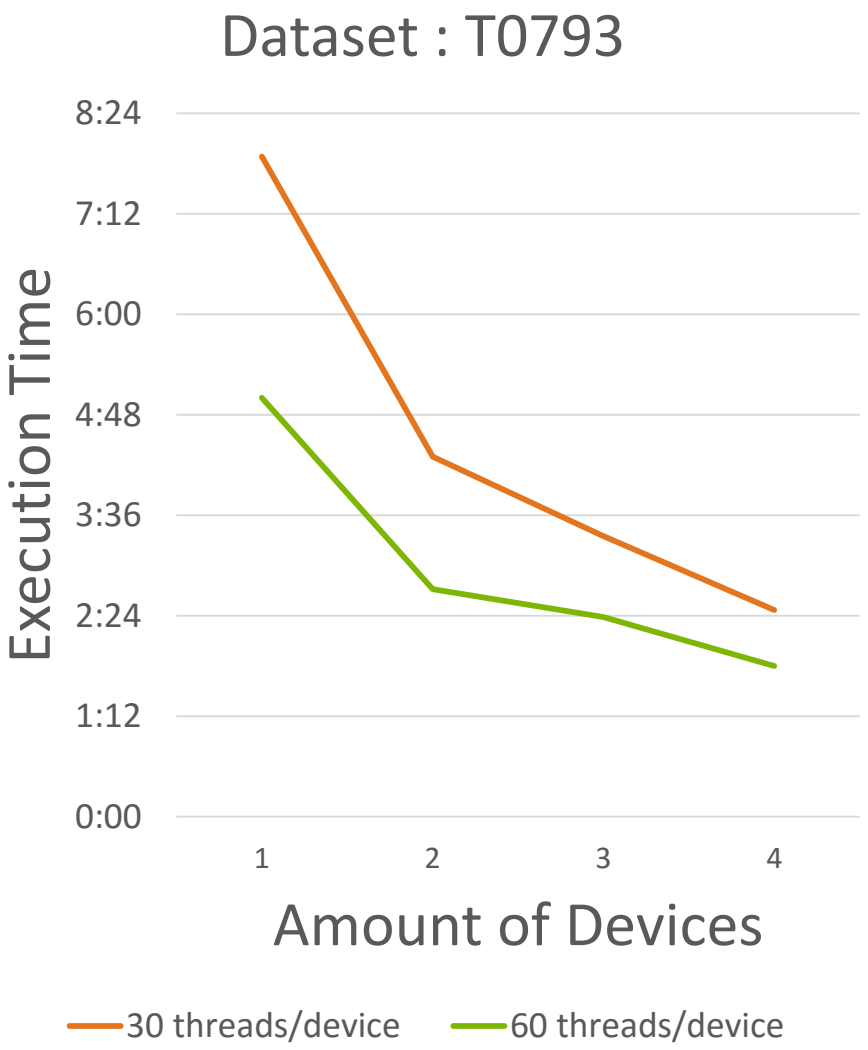
Evaluation

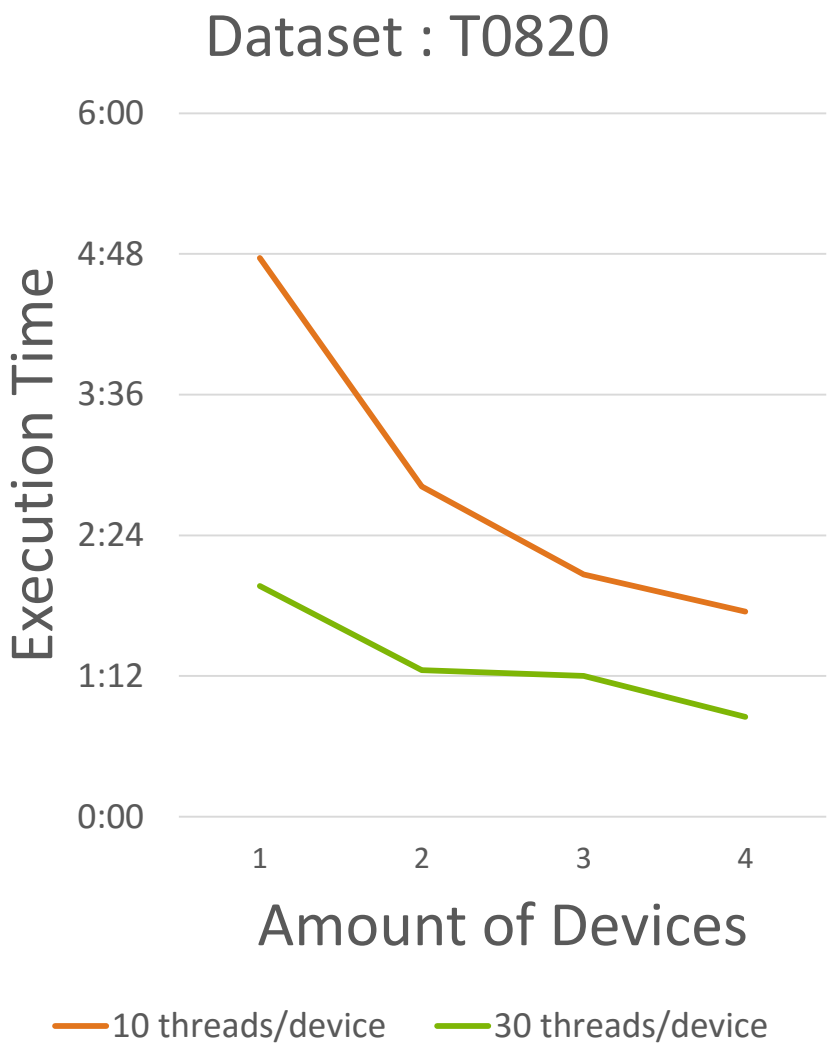
- Hardware
  - Host:
    - Two processors with 18 cores - 36 physical cores
    - 36 cores with Hyperthreading - 72 logical cores
    - 128 GB Ram Memory;
  - Devices: four Intel Xeon Phi Cards:
    - 61 physical cores - 4 Hardware threads - 228 logical cores
    - 16 GB Ram Memory
- Workload
  - Datasets (serial time):
    - 0820 : 7 min 19 seconds
    - 0793 : 28 min 21 seconds
- Tests with datasets 0820 and 0793:
  - optimization OpenMP
  - optimization OpenMP-MPI

Optimization OpenMP – Results

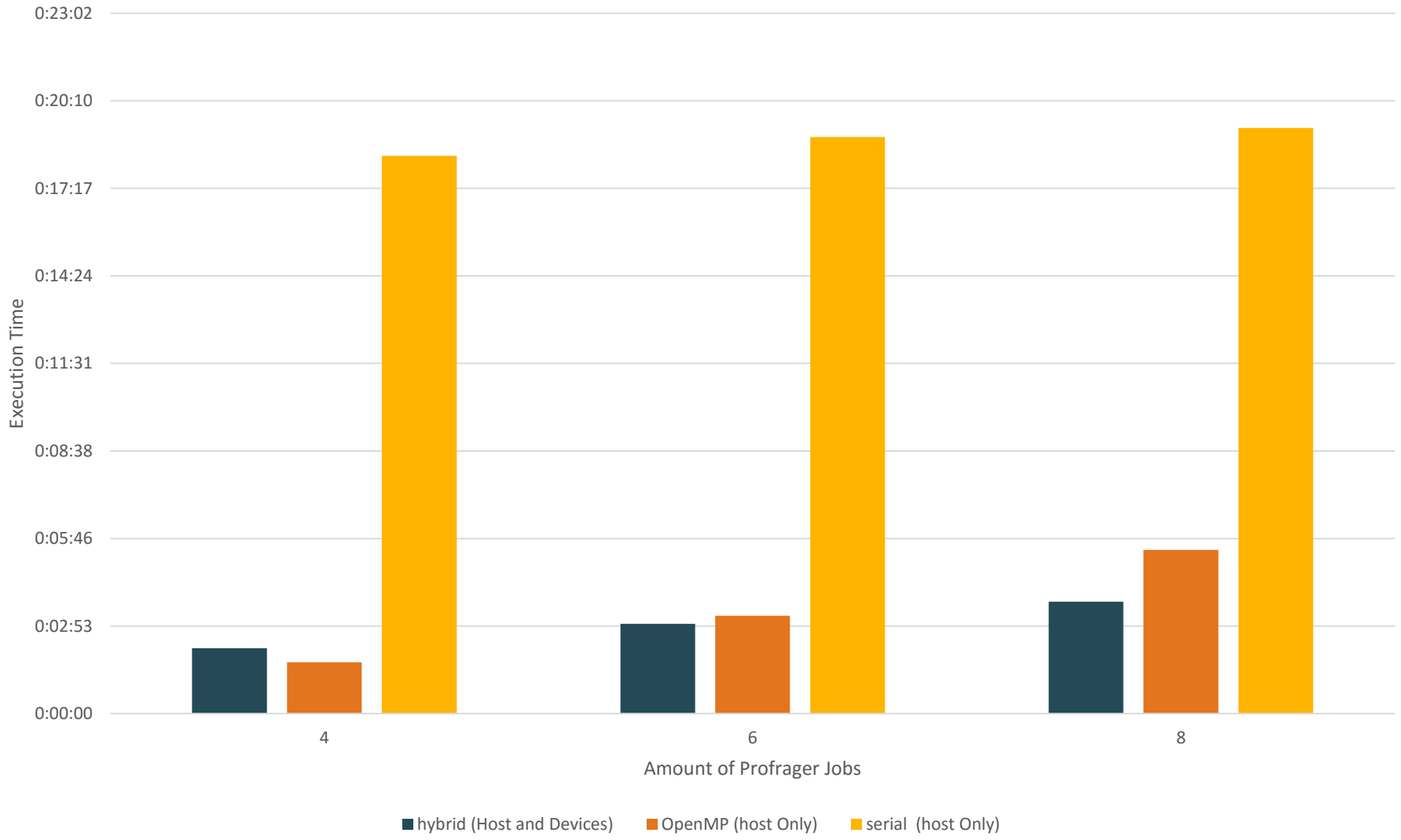


Optimization MPI-OpenMP – Results





Executing several profrager jobs



Discussion

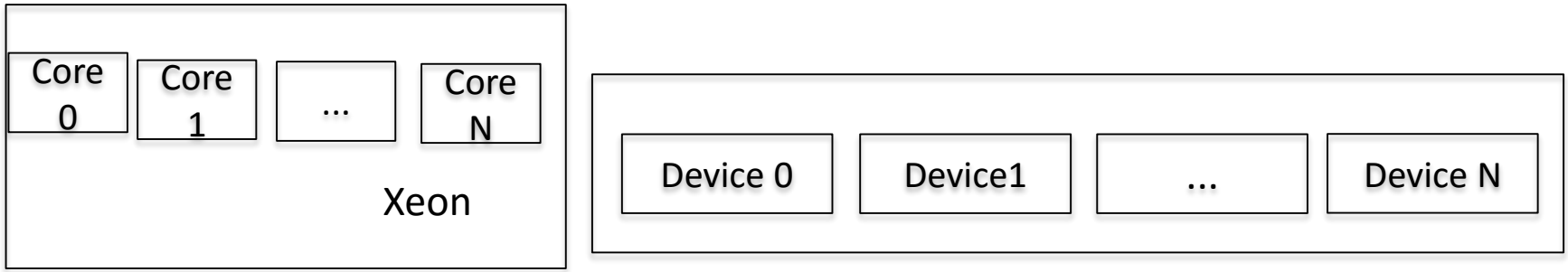
- Memory constraints
  - Experiments cannot be executed with 244 logical threads;
- I/O
  - I/O operations on Xeon Phi present very low performance;
- MPI-OpenMP
  - traditional model that can be used in hybrid parallel architectures;
- Profiling tools can guides parallelization and vectorization;
- Workloads with several Profrager jobs can be accelerated by scheduling jobs on host and on devices;

- Sobras

Intel Xeon / Intel Xeon Phi

- Execution part of iterations on Intel Xeon using OpenMP and part of iterations on Intel Xeon Phi using MPI/OpenMP

Intel Xeon / Intel Xeon Phi



BLOSUM

- BLOSUM (**BLO**cks **SU**bstitution **M**atrix) is a substitution matrix used for sequence alignment of proteins.
- BLOSUM matrices are used to score alignments between protein sequences.

fasta

- **FASTA format** is a text-based format for representing either nucleotide sequences or peptide sequences, in which nucleotides or amino acids are represented using single-letter codes.

Performance Optimization

- Discover hotspots;
- Identify parallelization opportunities;
- Apply optimizations;

Sequential Time

- One Iteration
- 30 iterations:

Intel Advisor

- Discover HotSpots

Intel Advisor

- Annotations

Intel Advisor

- Scalability Prediction

Intel Xeon Optimization

- OpenMP
- Main loop parallelized using #pragma omp parallel for