INTEL® HPC DEVELOPER CONFERENCE
FUEL YOUR INSIGHT

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

Silvio Stanzani, Rogério Iope, Raphael Cóbe

São Paulo State University (UNESP)

November 2016

Outline

- Introduction - UNESP Center for Scientific Computing
- Protein Structure Prediction / Profrager
- Hybrid Parallel Architectures
- Evaluating Profrager with Advisor
- MultiCore/ManyCore optimization
- Evaluation and Results

unesp
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

São Paulo State

- Area: ~United Kingdom
- Population: ~Spain
- Economy GDP: 2x Switzerland
- Investment in S&E: 1.66% GDP (~Argentina)

UNESP (São Paulo State University)

- 2nd largest (in size) among all Brazilian universities
- A successful model of a multicampus university

UNESP Center for Scientific Computing

Consolidates scientific computing resources for UNESP researchers

- HPC resources - Grid / Cloud computing paradigms
- Datacenter facility (120m^2)

Users

- UNESP researchers, students, and software developers
- SPRACE (São Paulo Research and Analysis Center): HEP researchers
  - São Paulo LHC/CMS Tier-2 Facility
  - Collaboration with Fermilab, CERN

UNESP Center for Scientific Computing

SPRACE Project

São Paulo Research and Analysis Center

- High Energy Physics (HEP) group
  - Part of the CERN LHC/CMS experiment (Tier-2 class HPC cluster)
  - Funded by São Paulo Research Foundation (FAPESP)
  - Close collaboration with Fermilab and CERN
  - Started operating in March 2004
- Physics Analysis and Monte Carlo simulation for CMS
  - Physics beyond the Standard Model (extra-dimensions, dark matter)
  - Heavy-ion collisions (strong interaction at high density and temperature)

GridUNESP Project (spin-off of SPRACE)

First Campus Grid in Latin America

- Provides HPC resources to > 375 researchers
- Distributed computing infrastructure (7 distinct UNESP campi)
  - one main cluster and 6 smaller, secondary systems
- > 400 compute nodes (~90 TFlops); ~300 TB high-end storage system
- Partnership with US Open Science Grid [https://www.opensciencegrid.org/]
  - first VO outside US
- ANSP Grid Certificate Authority operation & management
  - issues X.509 digital certificates for Grid/Cloud computing & Web Services

Intel® Parallel Computing Center @ UNESP

- Started in November 2014
- Goal: R&D efforts to adapt HEP software tools and explore manycore architectures
- Parallelization / vectorization of GEANT (geant.cern.ch)
  - a toolkit for the simulation of the passage of particles through matter
- Broad impact
  - extremely important tool for HEP and other S&E fields of socio-economic impact (Medical applications, space physics & Engineering)
- Partnership
  - CERN Geant-V development team, Fermilab Computing Division

Intel® Modern Code Partner Program @ UNESP

- Started in May 2015
- Goals:
  - Regular parallel programming and code modernization training workshops using Intel hardware and Intel software development tools
  - Training activities on Data Science using Intel DAAL and similar tools, aiming to explore confluences of HPC and Big Data
  - Technical support to local developer community
- Intel MCP representative for Latin America
  - https://software.intel.com/en-us/modern-code/live-workshops
- Computing resources (Xeon, KNC / KNL) fully dedicated to R&E activities

Intel® Modern Code Partner Program @ UNESP

- Training sessions being delivered in regional events and even abroad
  - Hamburg (INFIERI school), Cartagena (CCGrid'16), Porto (VecPar'16)
- Partial results (since May 2015)
  - 23 training sessions delivered
  - 1340 participants (350 followed hands-on activities)
- New training activity (introduced in 2016)
  - Introduction to Data Science (with hands-on activities using Intel DAAL)
- Goal to the end of the 2nd year of the project (May 2017)
  - Total number of training sessions: 36 (~1.5 session per month)
  - Total number of participants: 1500 (~40 participants per session)

Profrager Optimization

Protein Structure Prediction (PSP)

Protein structure prediction is a challenging task in bioinformatics

- It 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 on the understanding of the protein's function;

Applications

- Development of new drugs,
- Design of novel enzymes,
- Projects related with the study of Genome, etc

The use of 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]**;

- Developed at the Brazilian “National Laboratory for Scientific Computing” (LNCC) – https://www.lncc.br/sinapad/Profrager/

- Criteria to select the sequences to compose a Library is based on scores;
  - Scores are obtained using
    - Substitution Matrices (e.g. Blossun62)
    - Structural Databases (e.g. Protein Data Bank (PDB))
    - Secondary Structural Databases (Geometry Databases)

- Commonly a PSP experiment will need several libraries

[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) (PDB)

  createFragments()

  end loop2

  sort fragments

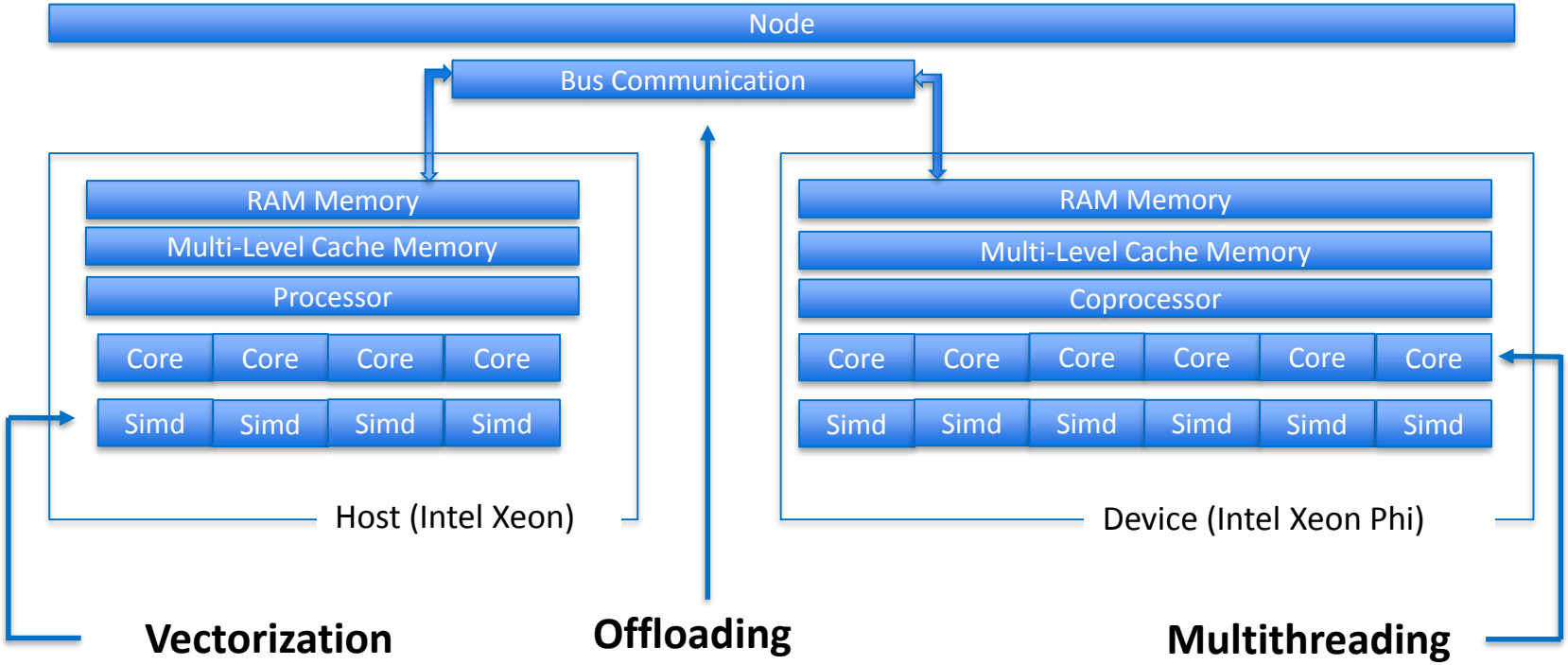
  loop 6 – print frags to output file

end loop 1

Profrager Algorithm - createFragments

```
loop 3 - loop all positions of each structural database sequence
    obtain score_matrix according to blossun62 matrix
      create new fragment
    search compatible Geometries;
    loop 4 - loop all positions retrieved from compatible Geometries
      obtain score_PSIPRED;
    end loop 4
    if (score_PSIPRED+score_matrix) > score min -> save frag
end loop3
```

Hybrid Parallel Architectures



Vectorization

Offloading

- Offload Pragmas
- MPI/OpenMP Model

Multithreading

Identifying Parallelization Opportunities

- Intel® Advisor XE
  - Identify parallel opportunities
  - Scalability prediction: amount of threads/performance gains
- Survey
  - Vectorization of loops: detailed information about vectorization
  - Total Time: elapsed time in each loop considering the time involved in internal loops
  - Self Time: elapsed time in each loop without internal loops
- Suitability
  - Speedup gains obtained parallelizing annotated loops

Profrager Survey Data - Total Time and Self Time

Loop 1 – T: 331 s – S: 0s

Loop 2 – T: 255 s - S: 0s

Loop 3 – T: 255 s - S: 8s

Loop 4 – T: 36 s - S: 27s

Loops		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]			269.196s
[loop in main at frag_blasta.original.cpp:344]			255.456s
[loop in main at frag_blasta.original.cpp:342]			255.456s
[loop in main at frag_blasta.original.cpp:354]			255.276s
[loop in std::__introsort_loop<std::reverse_iterator<__gnu_cxx::__normal_iterator<fr ...			226.946s
[loop in std::_Rb_tree<std::string, std::pair<std::string const, std::string>, std::_Selec			72.968s
[loop in std::__unguarded_partition<std::reverse_iterator<__gnu_cxx:: ...		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 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

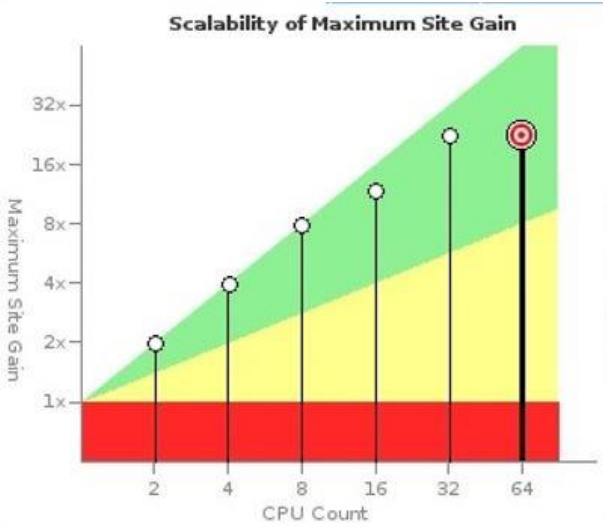
Profrager Suitability Data

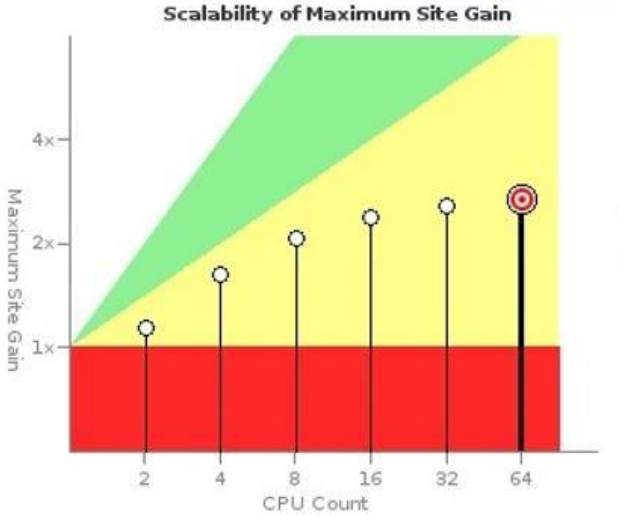
Maximum Program Gain For All Sites: 12.14x

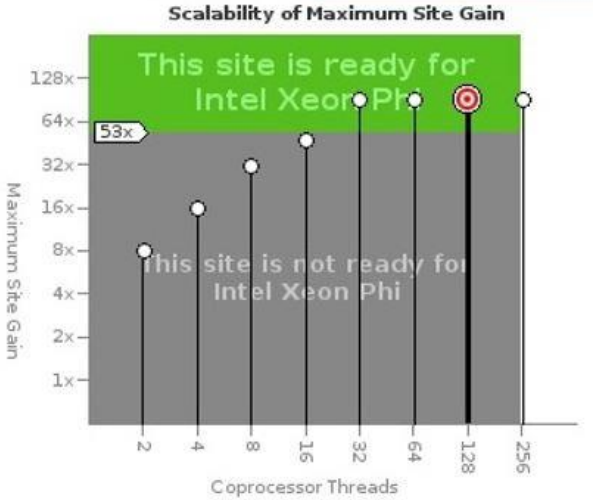
Serial time:	347.4891s
Predicted Parallel time:	28.6243s

Target System: CPU Threading Model: OpenMP

Site Label	Source Location	Impact to Program Gain	Combined Site I 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







Profrager Optimizations

Two optimizations were developed to parallelize **loop 1**: **[1]**

- 1. Using OpenMP to execute **loop 1** iterations in multiple threads on Host;
- 2. Using MPI to execute **loop 1** iterations in multiple devices and OpenMP to execute iterations in multiple threads on Devices;

Offload model cannot be used because Profrager is based on C++ std::vector, that is not bitwise copyable.

[1] Silvio Stanzani , Raphael Cobe, Rogério Iope, Igor Freitas, Laurent Dardenne, Fábio Custódio **"Optimization of ProFrager, a Protein Structure and Function Prediction Tool"** in https://software.intel.com/en-us/articles/optimization-of-profrager-a-protein-structure-and-function-prediction-tool

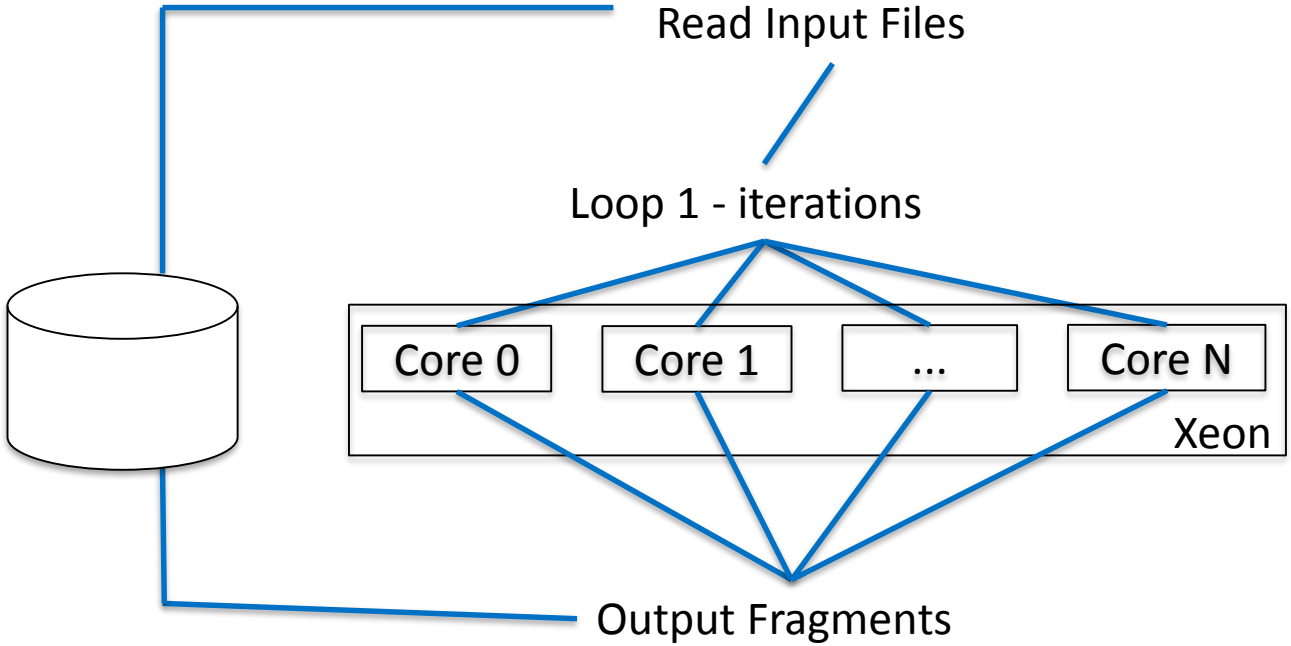
Profrager Optimization - OpenMP

```
#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

Rank 0 is mapped to host

- Performs I/O operations only
  - Read Input files;
  - Output fragments;
- Data Transfer among ranks using MPI_SEND and MPI_RECV

Rank 1 .. N are mapped to each device

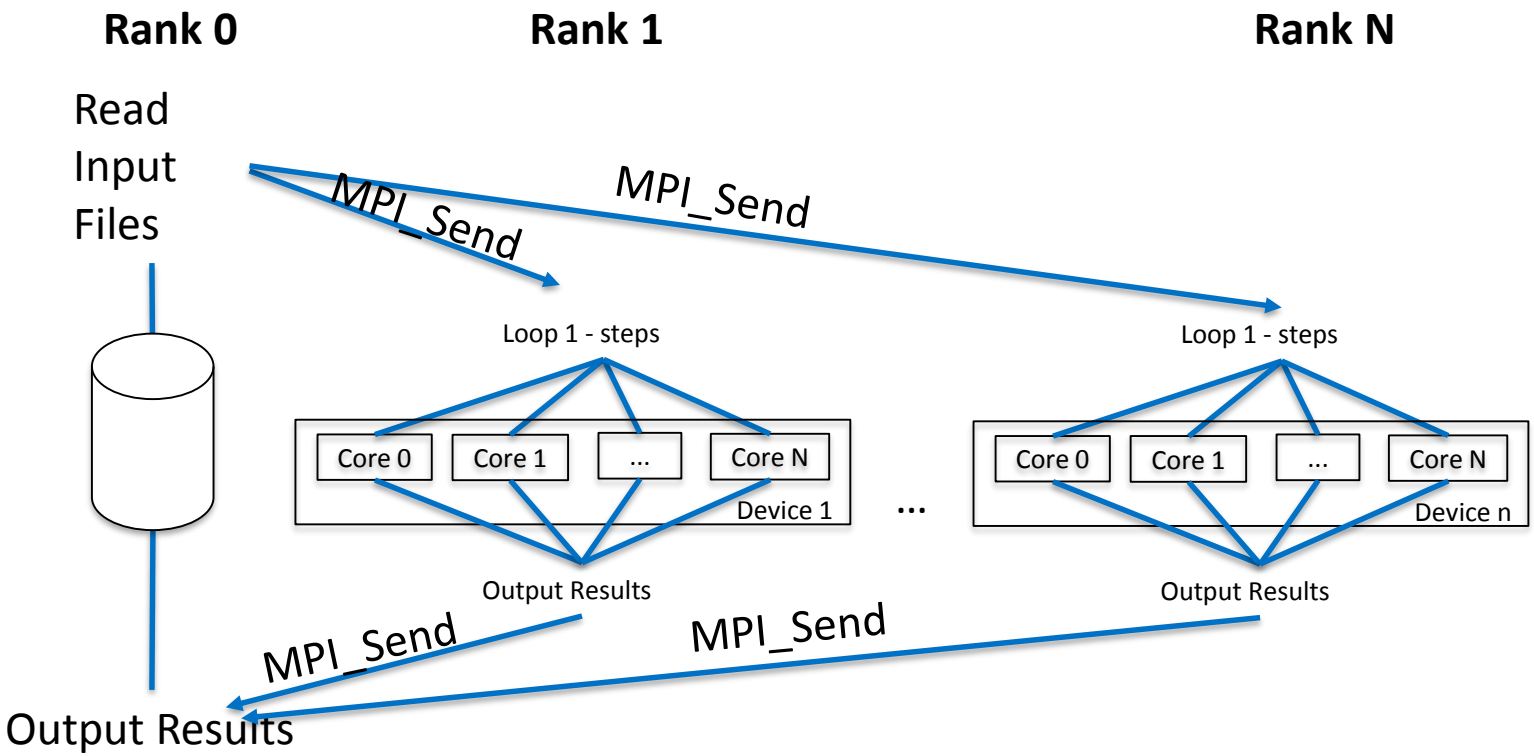
- **loop1** iterations is parallelized using OpenMP;

Profrager Optimization - MPI-OpenMP

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

```
If (rank > 0)
    #pragma omp parallel for
    loop 1 - loop all positions of protein amino acid sequence (Fasta File)
        loop 6 –  mpi_send(frags);
    end loop 1
else
    while mpi_recv(frags);
        print frags to output file
```

Profrager Optimization - MPI-OpenMP



Evaluation

Hardware

- Host:
  - Two Intel Xeon processors with 18 cores each (E5-2699 v3) → total 36 physical cores
  - 36 cores with Hyper-threading → 72 logical cores
  - 128 GB RAM Memory;
- Devices: four Intel Xeon Phi Cards (5110P)
  - 61 physical cores - 4 Hardware threads - 228 logical cores
  - 16 GB RAM Memory

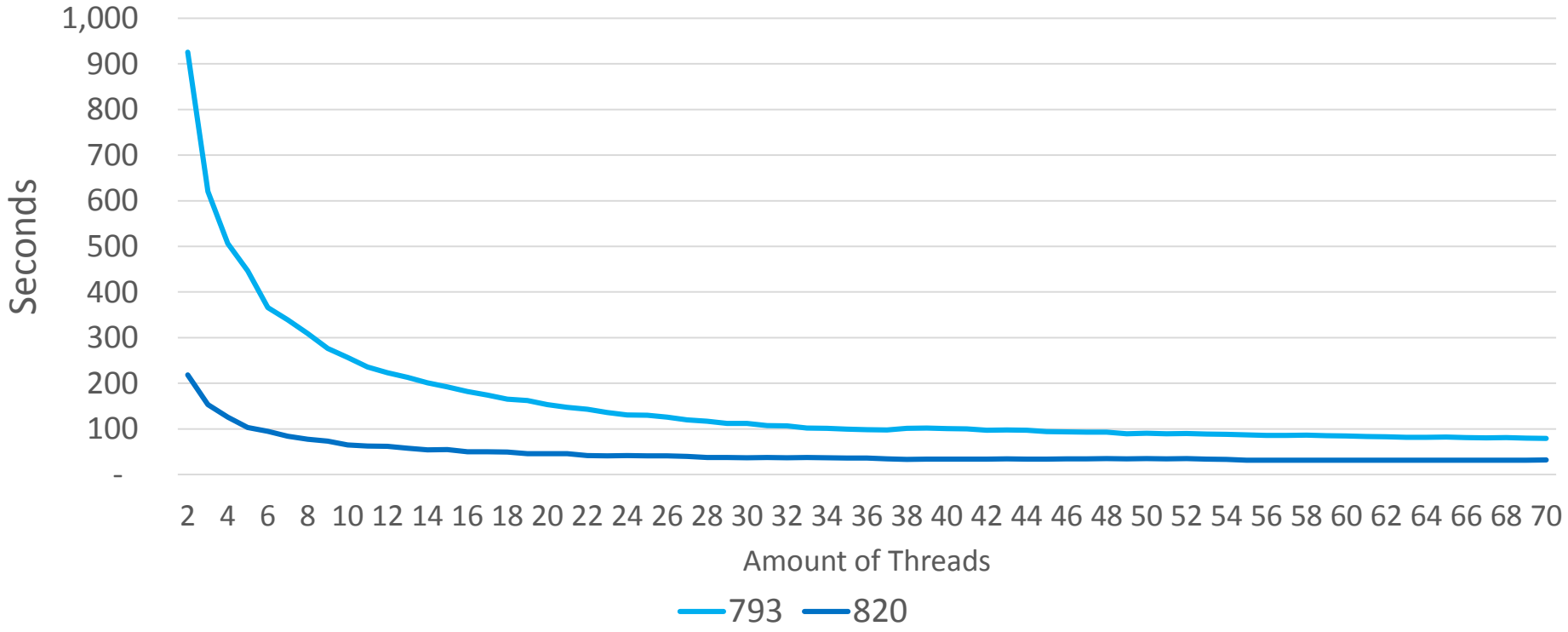
Workload

- Datasets (serial time):
  - 0820 : 7 min 19 seconds
  - 0793 : 28 min 21 seconds

Tests

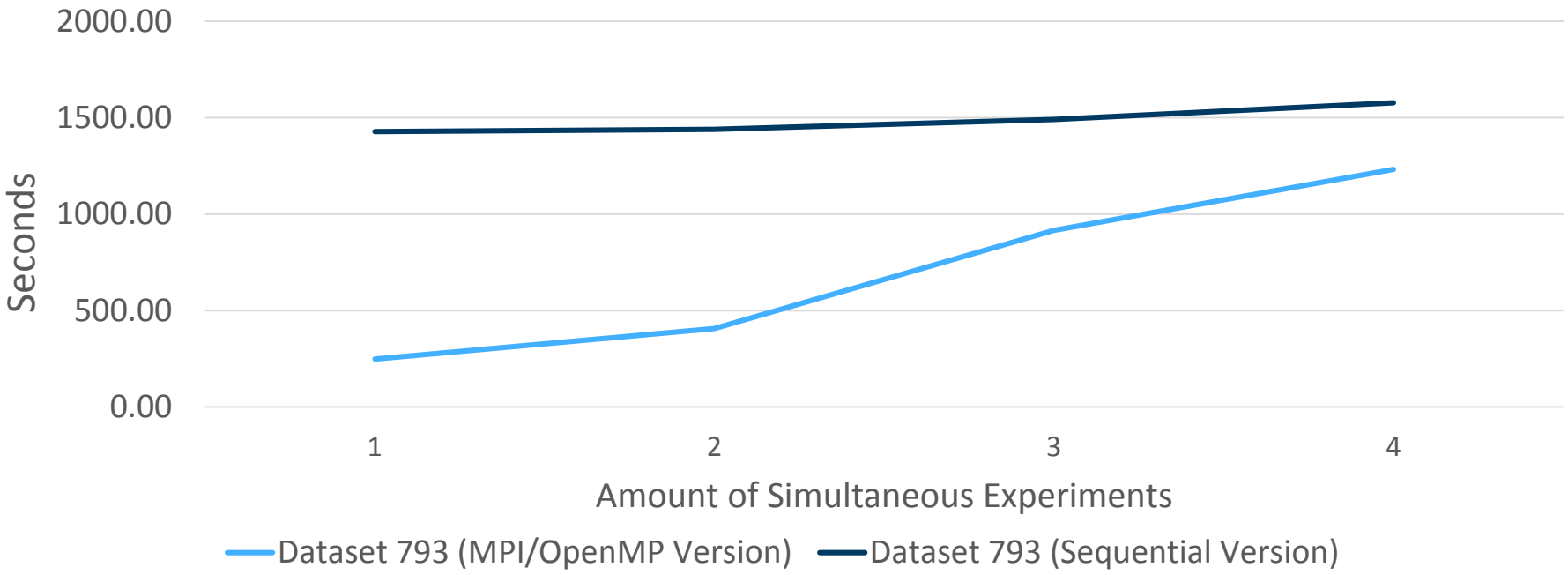
- Execution of a Single Profrager Experiment using OpenMP optimization
- Execution of multiple Profrager experiments using OpenMP-MPI optimization
- Execution of multiple Profrager experiments using OpenMP-MPI optimization and OpenMP optimization

OpenMP Optimization - Execution of a Single Profrager Experiment



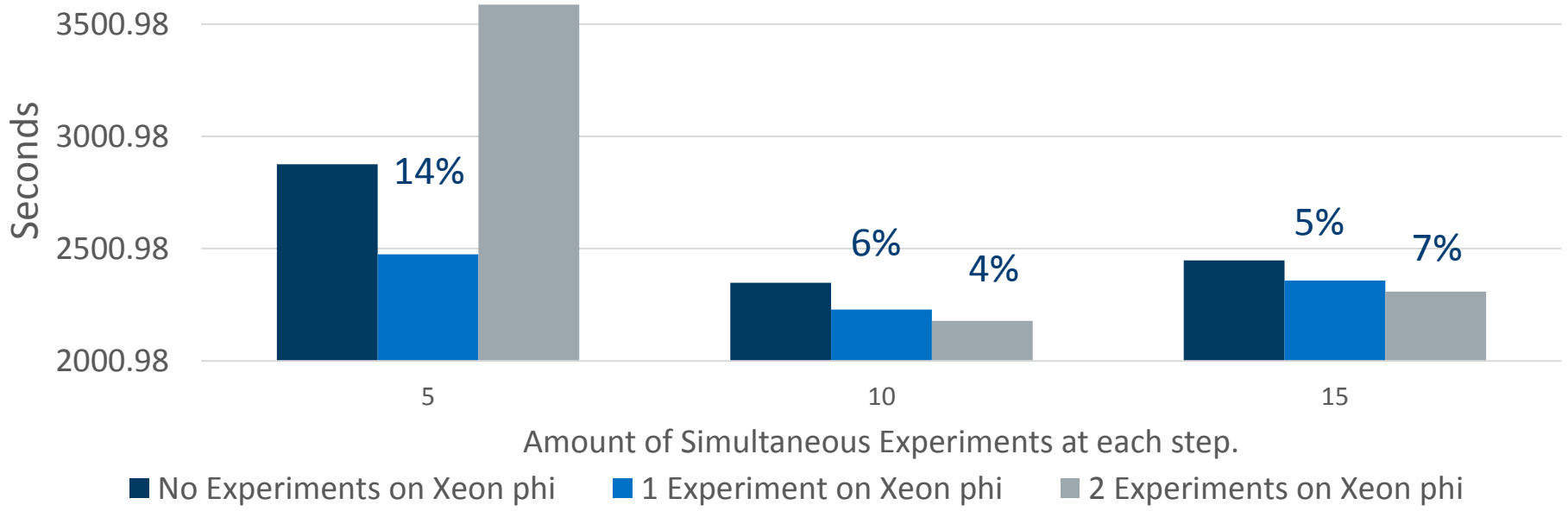
Speedup Achieved: 14x for Dataset 820 and 22x for Dataset 793.

MPI-OpenMP Optimization - Execution of multiple Profrager experiments



Executing four experiments of Dataset 793 on Xeon Phi is faster than executing one sequential experiment.

OpenMP and MPI-OpenMP Optimization - Execution of 30 Profrager experiments at steps



Experiments with Dataset 793 showed that Intel Xeon Phi improved the performance compared to the utilization of Intel Xeon Processor only.

Extra slides

Profrager Survey Data

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

Profrager Suitability Data - Xeon

Maximum Program Gain For All Sites: 12.14x

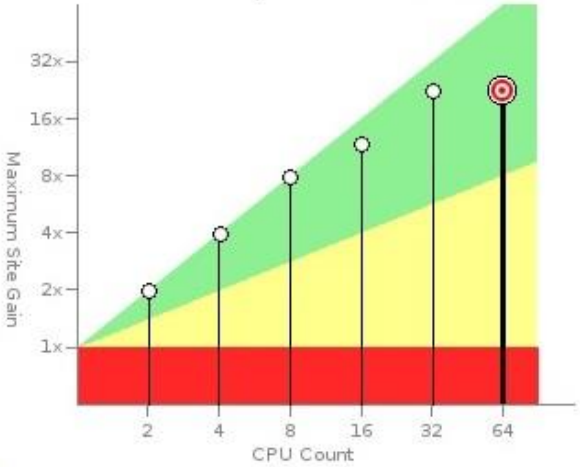
Serial time: 347.4891s
Predicted Parallel time: 28.6243s

Target System: CPU Threading Model: OpenMP

Site Label	Source Location	Impact to Program Gain	Combined Site I 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 - Xeon

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 - Xeon Phi

Maximum Program Gain For All Sites: 18.51x

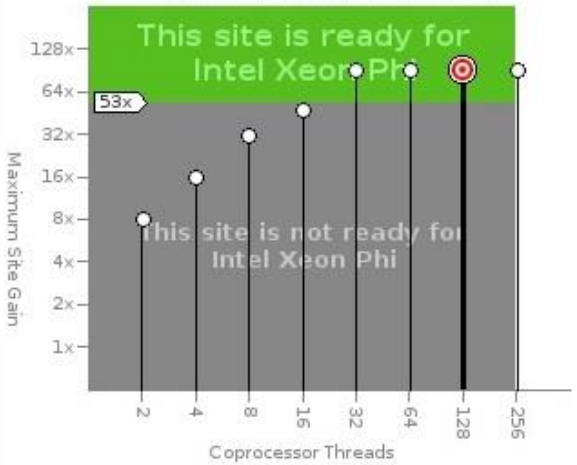
Serial time: 3474.8906s
Predicted Parallel time: 187.7634s

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

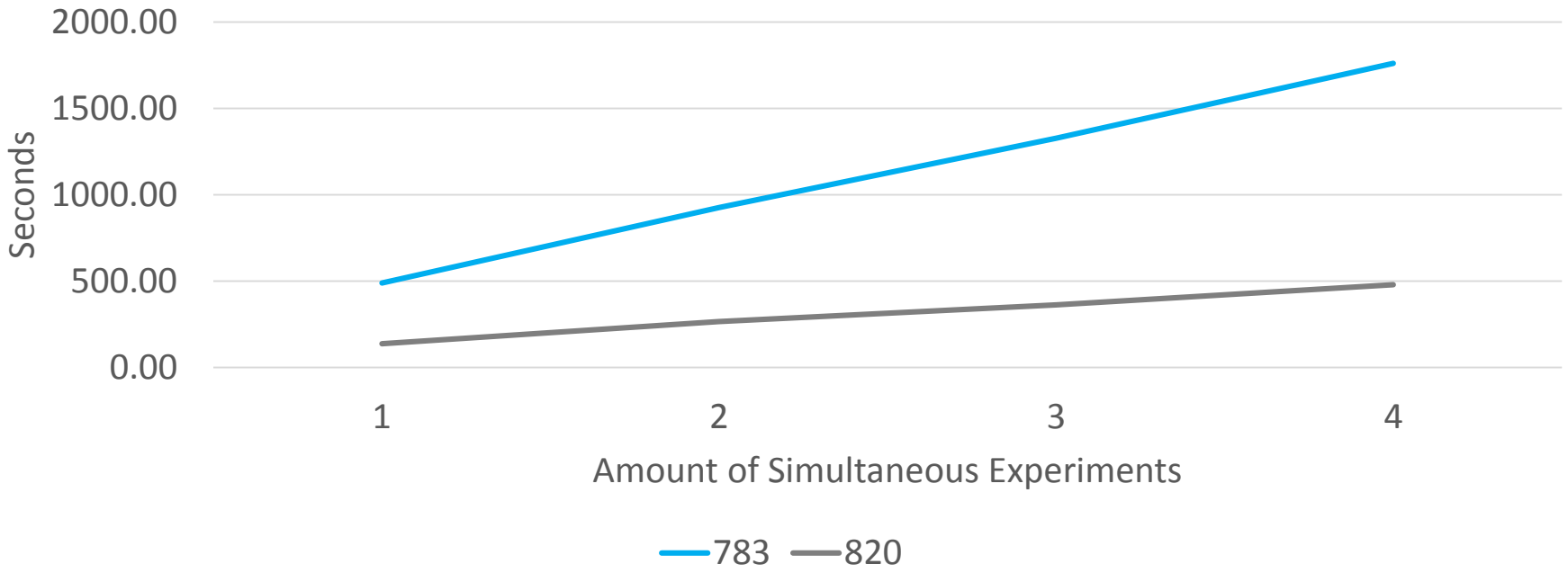
Apply

Runtime Mod

Type of Cha

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

MPI-OpenMP Optimization - Test 2



Executing four experiments of Dataset 793 on Xeon Phi is faster than executing one sequential experiment.

Tier-2 Availability and Reliablity Report

CMS

September 2016

Federation Summary - Sorted by Name

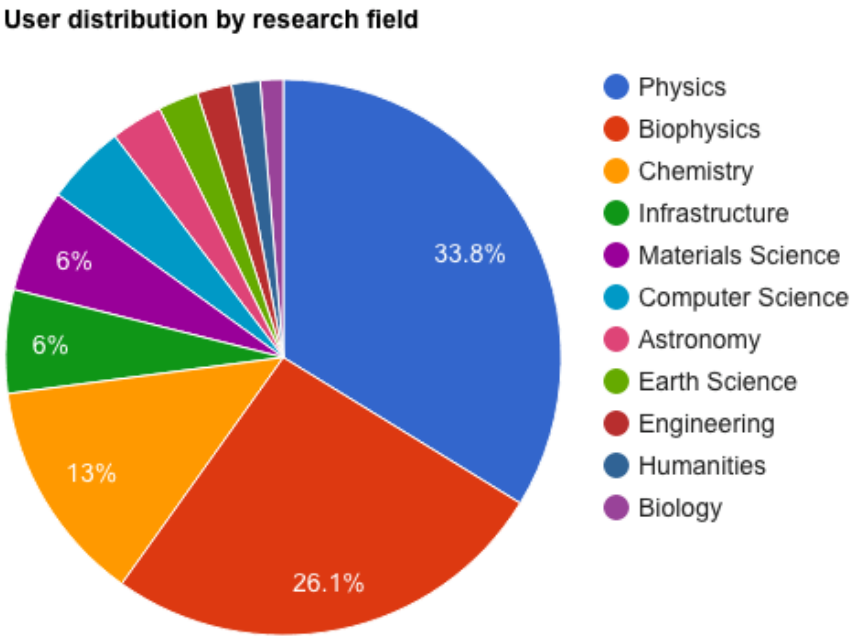
Color coding: N/A <30% <60% <90% >=90%

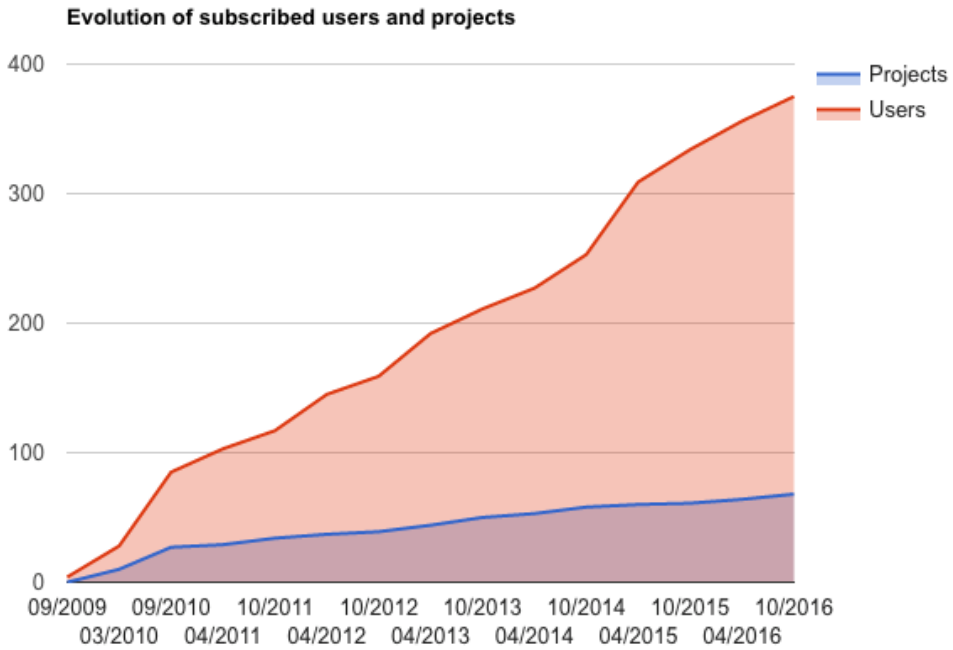
Availability Algorithm: (OSG-CE + CREAM-CE + ARC-CE + HTCONDOR-CE) * (all SRMv2 + all OSG-SRMv2)

Federation	Availability	Reliability
AT-HEPHY-VIENNA-UIBK	92%	92%
BE-TIER2	96%	96%
BR-SP-SPRACE	99%	99%
CERN-PROD	100%	100%
CH-CHIPP-CSCS	100%	100%
CN-IHEP	100%	100%
DE-DESY-RWTH-CMS-T2	98%	98%
EE-NICPB	88%	88%
ES-CMS-T2	82%	87%
FI-HIP-T2	100%	100%
FR-GRIF	100%	100%
FR-IN2P3-CC-T2	96%	99%
FR-IN2P3-IPHC	94%	94%
GR-Ioannina-HEP	96%	96%
HU-HGCC-T2	97%	99%
IN-INDIACMS-TIFR	95%	95%
IT-INFN-T2	96%	96%
KR-KNU-T2	100%	100%

Federation	Availability	Reliability
PK-CMS-T2	98%	98%
PL-TIER2-WLCG	N/A	N/A
PT-LIP-LCG-Tier2	98%	98%
RU-RDIG	96%	96%
T2-LATINAMERICA	90%	90%
T2_US_Caltech	89%	89%
T2_US_Florida	96%	96%
T2_US_MIT	100%	100%
T2_US_Nebraska	100%	100%
T2_US_Purdue	100%	100%
T2_US_UCSD	99%	99%
T2_US_Wisconsin	100%	100%
TH-Tier2	N/A	N/A
TR-Tier2-federation	99%	99%
TW-CMS-T2	98%	98%
UA-Tier2-Federation	97%	99%
UK-London-Tier2	100%	100%
UK-SouthGrid	94%	94%

GridUNESP Project (spin-off of SPRACE project)





General-purpose infrastructure:
users from distinct research fields

68+ projects, 375+ users
(as of Oct 2016)