CDAC HPC Profiler

Abhishek Patil

Outline

- Introduction
- What is Profiling
- Inclusive vs Exclusive profiling
- Sampling profiler
- Instrumenting profiler
- Key Features
- Application Run/Demo
- Screenshot
- Outcome of the experiments

Introduction

- What is Profiling
  - Profiling is the measurement of which parts of your application are consuming a particular computational resource of interest. This could be which methods are using the most CPU time, which lines allocate the most objects, where your CPU cache misses are coming from, etc.
- Reflection of summary information during execution – time consumed, function calls ,
- Reflects performance behaviour of program entities

  functions , loops, basic blocks
- Very good for low cost performance assessment
- Helps to understand performance bottlenecks and hotspots
- Implemented through either
  - Sampling
  - Measurement

Inclusive Vs Exclusive Profiling

```
int main()
{

/* takes 100 seconds */

 f1(); /* takes 20 seconds */

/* other work */
f2(); /* takes 50 seconds */
f1(); /* takes 20 seconds */
/* other work */
}
```

- Inclusive time for main - 100 seconds
- Exclusive time for main - 100-20-50-20-10 seconds

Brief info about Sampling

- The most common type of profiler is the ***sampling* profiler**.

They work **by interrupting the application** under test periodically in proportion to **the consumption of the resource** we're interested in.

While the program is interrupted the **profiler grabs a snapsho**t of its current state, which includes where in the code it is.

After the **state is captured** the program continues. For the method timing example earlier, a sampling profiler would interrupt the program after a certain amount of time had elapsed and capture its state.

It would then aggregate those samples over time to produce a **statistical picture of the state of the application**. You could use the ***percent of samples*** that contained a method of interest to calculate *how much time was spent in that method* (though not the duration of that method).

Sampling based profiling

```
Void Alpha()
{

  Beta();
}

Void Beta()
{

}
```

30 samples

Functions	Inclusive	Exclusive
Alpha	80	30
Beta	50	50

50 samples

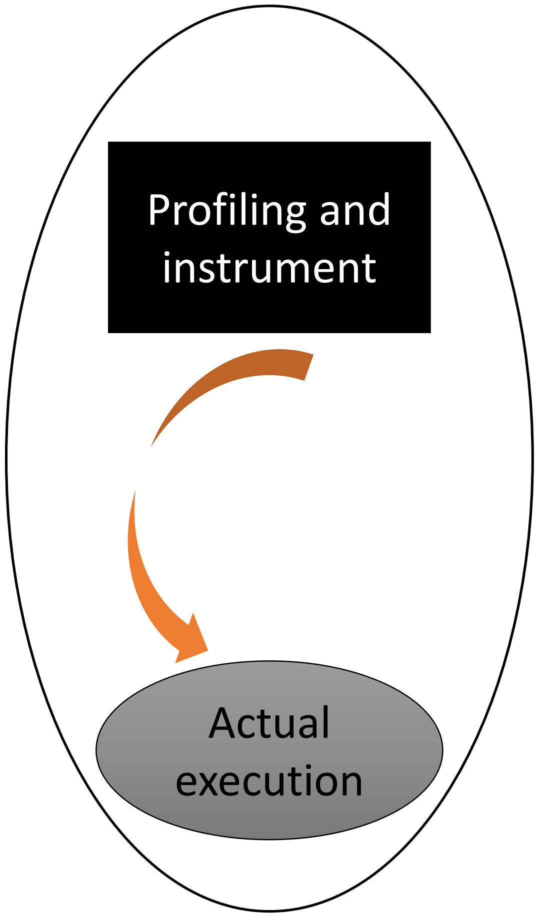
Brief info about Instrumenting

- The first and earliest type are instrumenting profilers.

- They work by *instrumenting* the program under test in order to collect information about the **resource of interest**.

- For example if you wanted to **calculate how much time** methods were taking to execute an instrumenting profiler would **add instructions** to the **beginning and end** of each method to capture the current time which could then be used to reconstruct the **duration spent inside each method**.

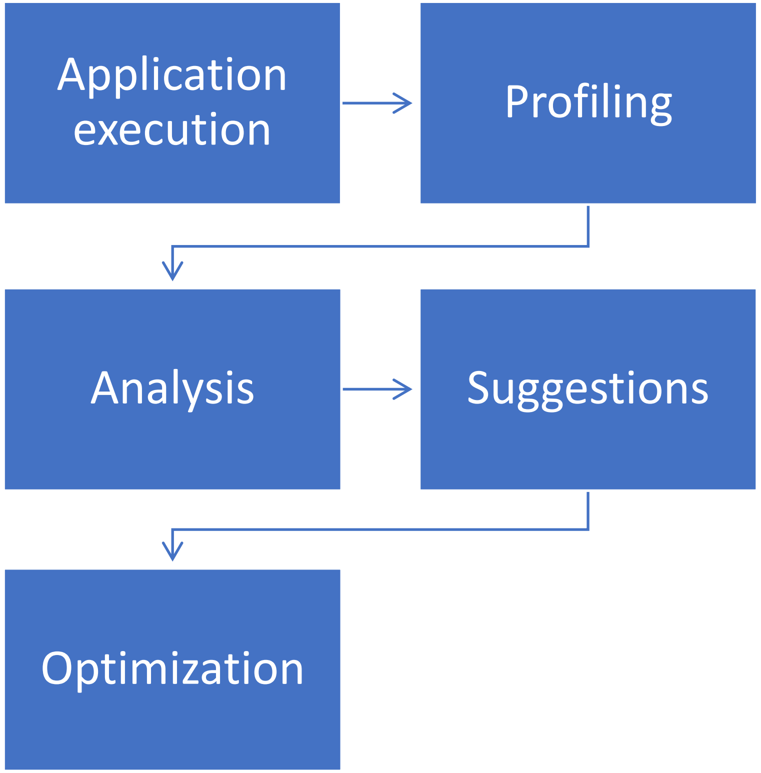
Instrumentation based Profiling




```
T=E*F;
For (I=1;I<N;I++)
{
V[I]=C[I]*B[I];
A[I]=C(2I+4);
}
```

```
T=E*F;
Instrumentation code
For (I=1;I<N;I++)
{
V[I]=C[I]*B[I];
A[I]=C(2I+4);
}
Instrumentation code
```

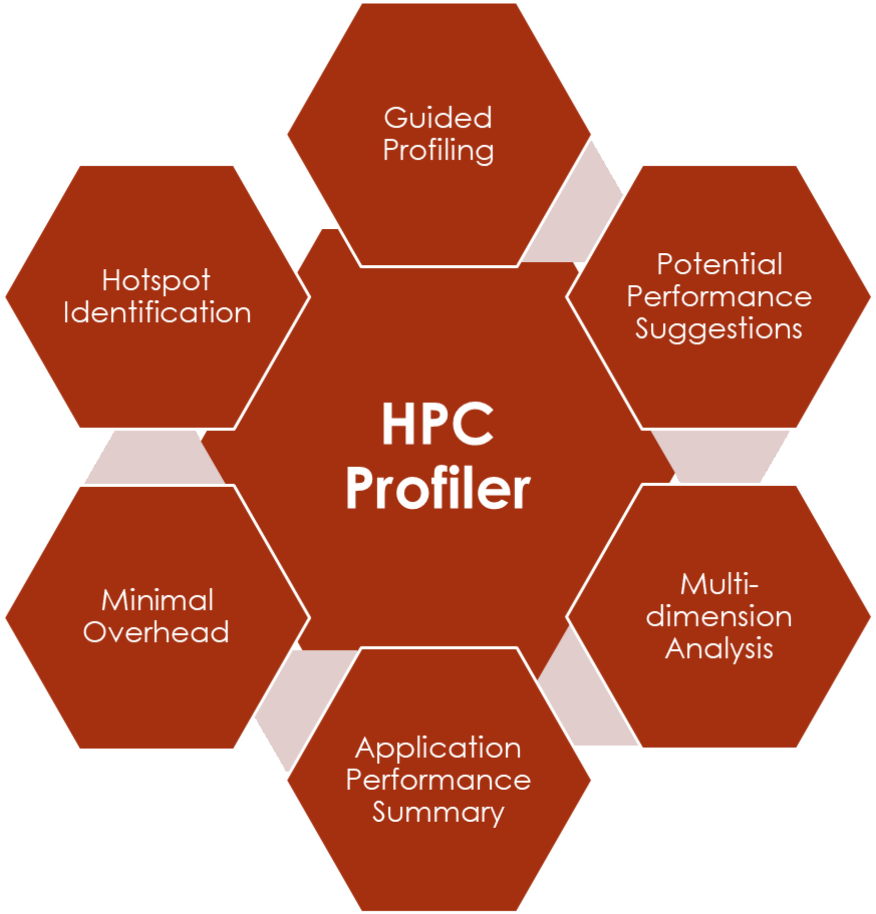
Flow of Profiling and Analysis



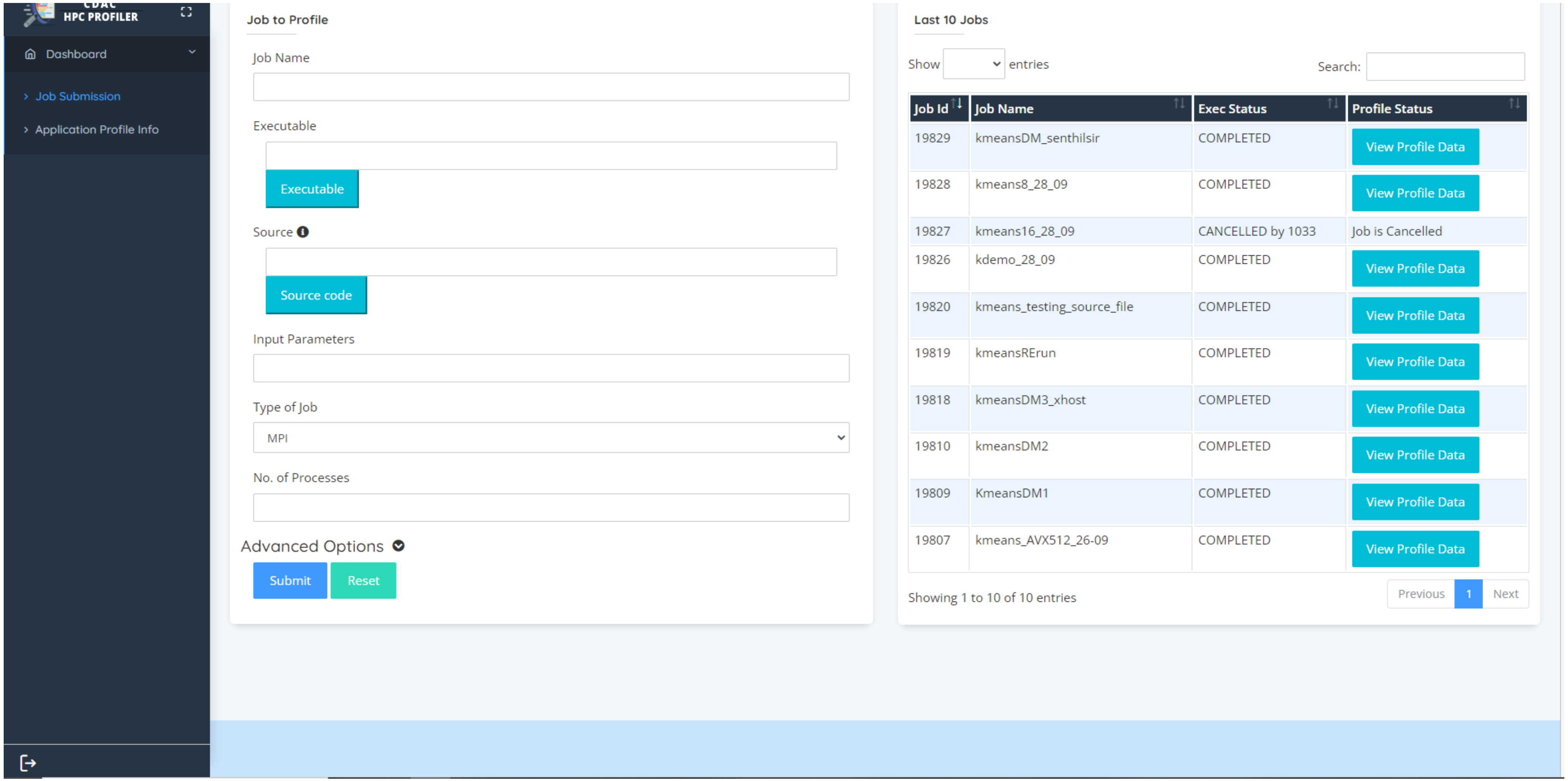
What can a HPC Profiler tell us

- Processes wise analysis – threads, user functions, loops and blocks
- Memory usage
- Timing information
- Parallel calls (and other library calls)
- Other detailed information like slow sections in the source code , vectorization analyses, etc.

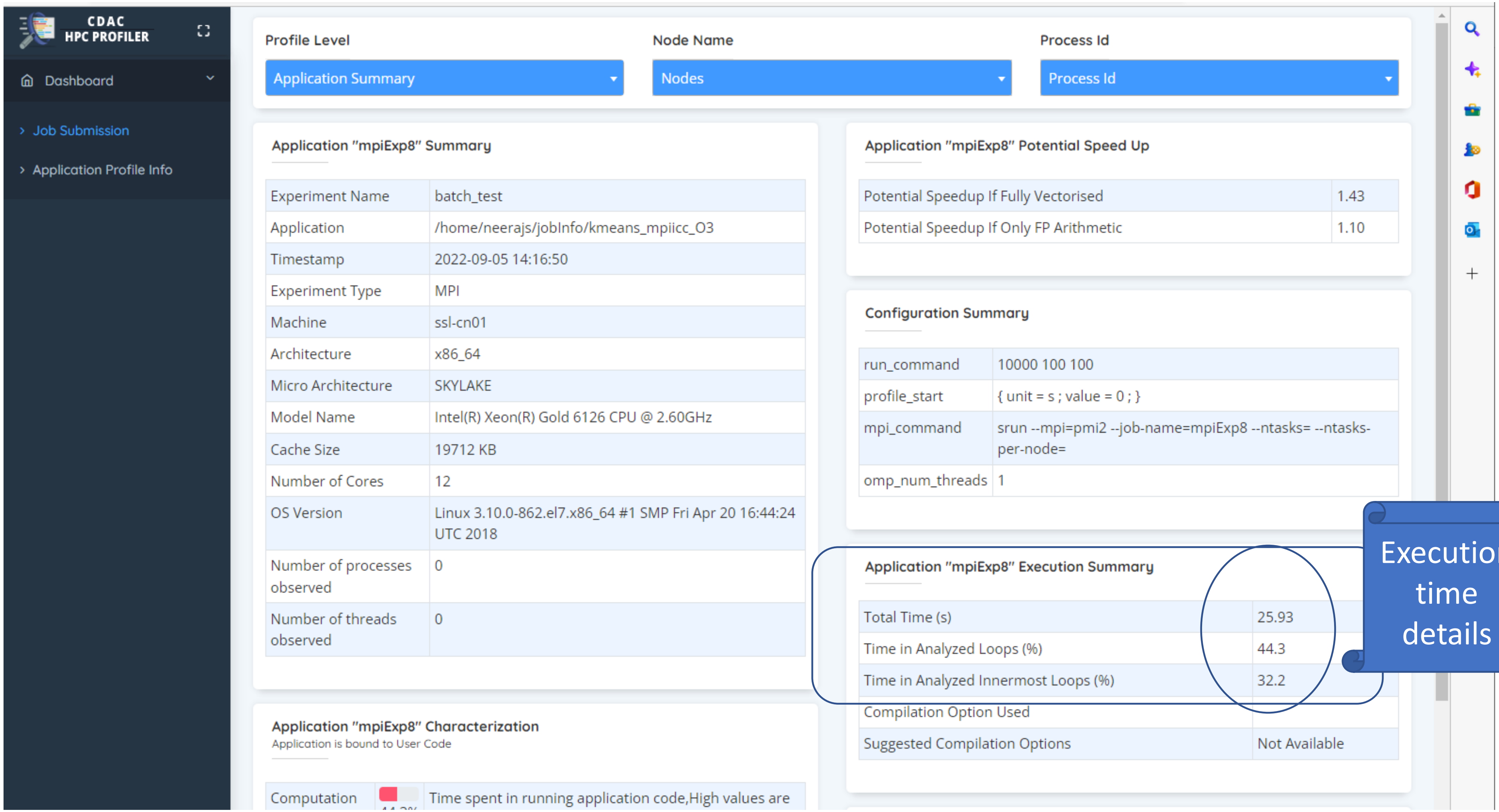
HPC Profiler – Key features



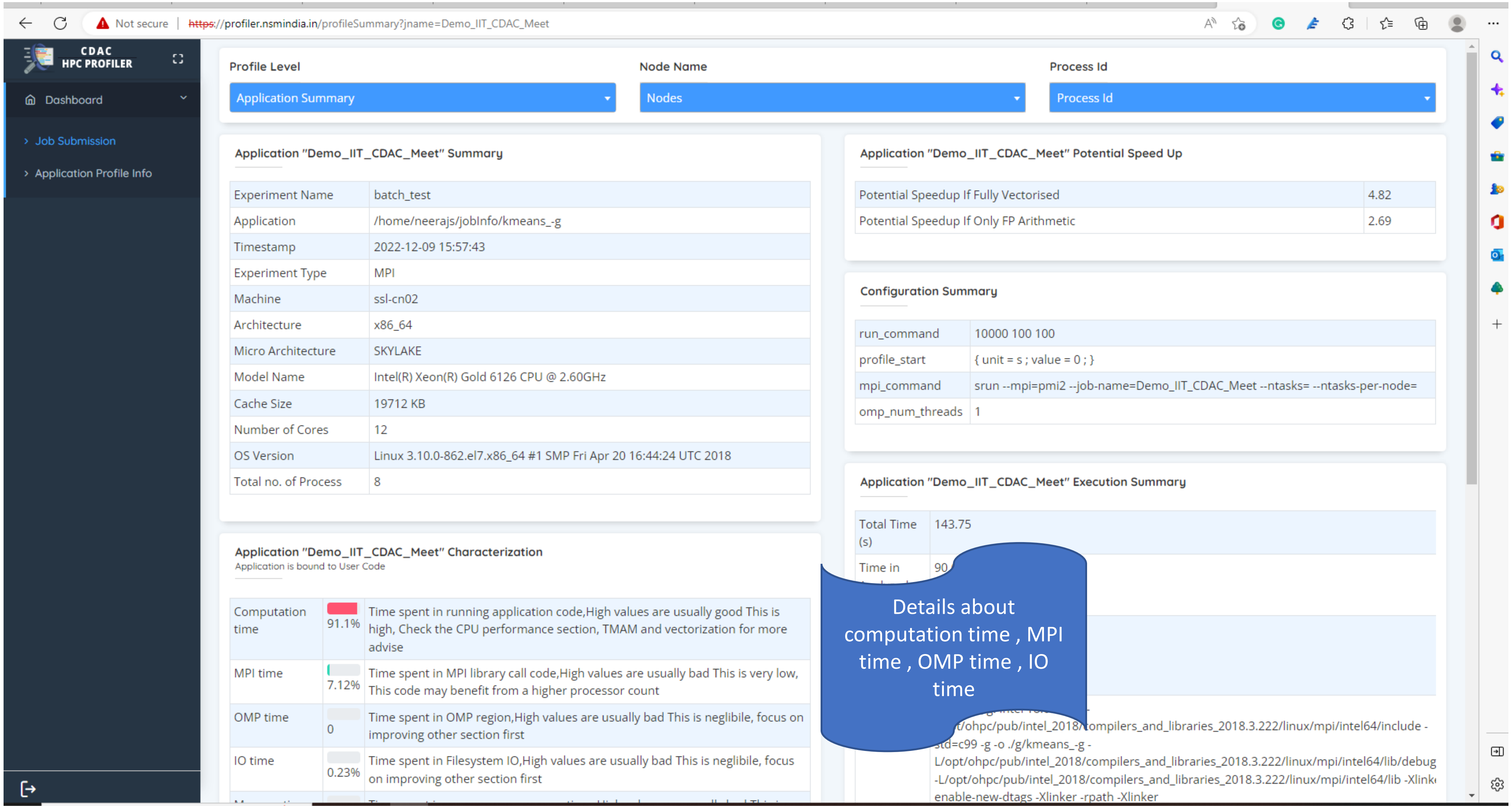
Index page :-


CDAC
HPC PROFILER
Dashboard
Job Submission
Application Profile Info
Job to Profile
Job Name
Executable
Executable
Source
Source code
Input Parameters
Type of Job
MPI
No. of Processes
Advanced Options
Submit
Reset
Last 10 Jobs
Show
entries
Search:
Job Id
Job Name
Exec Status
Profile Status
19829
kmeansDM_senthilsir
COMPLETED
View Profile Data
19828
kmeans8_28_09
COMPLETED
View Profile Data
19827
kmeans16_28_09
CANCELLED by 1033
Job is Cancelled
19826
kdemo_28_09
COMPLETED
View Profile Data
19820
kmeans_testing_source_file
COMPLETED
View Profile Data
19819
kmeansRErun
COMPLETED
View Profile Data
19818
kmeansDM3_xhost
COMPLETED
View Profile Data
19810
kmeansDM2
COMPLETED
View Profile Data
19809
KmeansDM1
COMPLETED
View Profile Data
19807
kmeans_AVX512_26-09
COMPLETED
View Profile Data
Showing 1 to 10 of 10 entries
Previous
1
Next

Application summary page :-

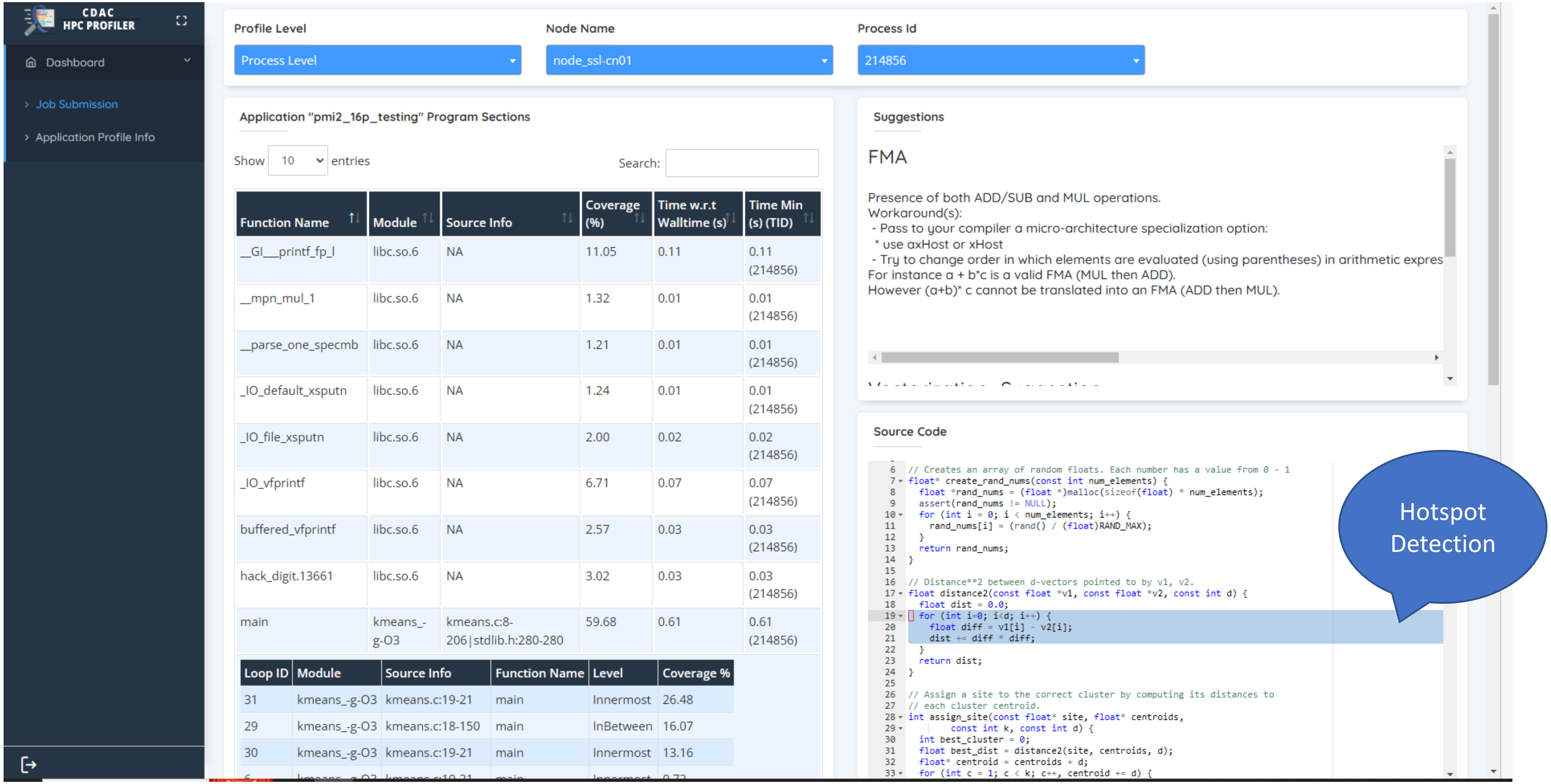


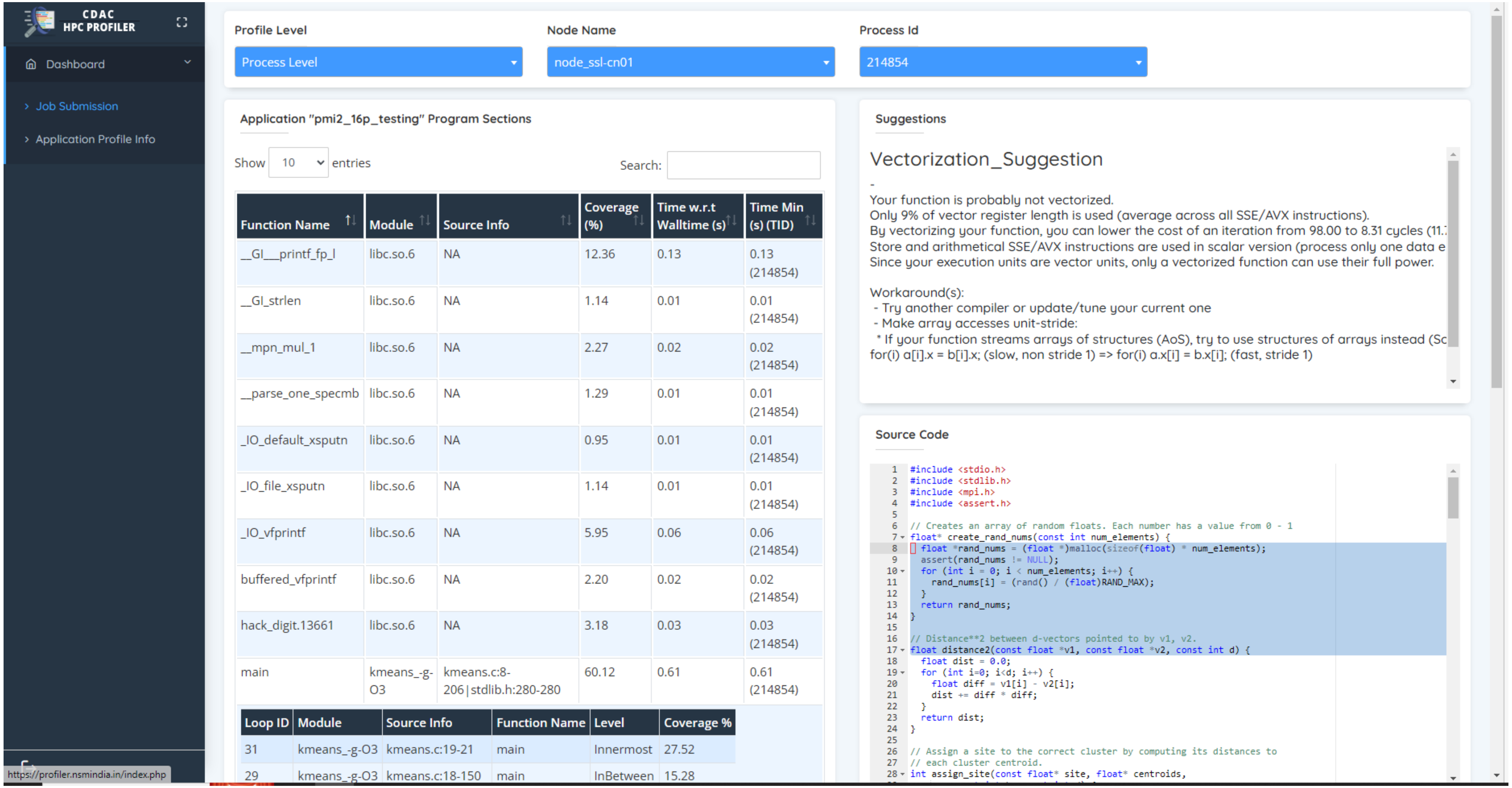
Execution time details



Details about computation time , MPI time , OMP time , IO time

Process level info page :-




CDAC
HPC PROFILER
Dashboard
Job Submission
Application Profile Info
Profile Level
Node Name
Process Id
Process Level
node_ssl-cn01
214854
Application "pmi2_16p_testing" Program Sections
Show 10 entries
Search:
Function Name | Module | Source Info | Coverage (%) | Time w.r.t Walltime (s) | Time Min (s) (TID)
__GI__printf_fp_l | libc.so.6 | NA | 12.36 | 0.13 | 0.13 (214854)
__GI_strlen | libc.so.6 | NA | 1.14 | 0.01 | 0.01 (214854)
__mpn_mul_1 | libc.so.6 | NA | 2.27 | 0.02 | 0.02 (214854)
__parse_one_specmb | libc.so.6 | NA | 1.29 | 0.01 | 0.01 (214854)
_IO_default_xsputn | libc.so.6 | NA | 0.95 | 0.01 | 0.01 (214854)
_IO_file_xsputn | libc.so.6 | NA | 1.14 | 0.01 | 0.01 (214854)
_IO_vfprintf | libc.so.6 | NA | 5.95 | 0.06 | 0.06 (214854)
buffered_vfprintf | libc.so.6 | NA | 2.20 | 0.02 | 0.02 (214854)
hack_digit.13661 | libc.so.6 | NA | 3.18 | 0.03 | 0.03 (214854)
main | kmeans_-g-O3 | kmeans.c:8-206|stdlib.h:280-280 | 60.12 | 0.61 | 0.61 (214854)
Loop ID | Module | Source Info | Function Name | Level | Coverage %
31 | kmeans_-g-O3 | kmeans.c:19-21 | main | Innermost | 27.52
29 | kmeans_-g-O3 | kmeans.c:18-150 | main | InBetween | 15.28
https://profiler.nsmindia.in/index.php
Suggestions
Vectorization_Suggestion
-
Your function is probably not vectorized.
Only 9% of vector register length is used (average across all SSE/AVX instructions).
By vectorizing your function, you can lower the cost of an iteration from 98.00 to 8.31 cycles (11.
Store and arithmetical SSE/AVX instructions are used in scalar version (process only one data e
Since your execution units are vector units, only a vectorized function can use their full power.
Workaround(s):
- Try another compiler or update/tune your current one
- Make array accesses unit-stride:
* If your function streams arrays of structures (AoS), try to use structures of arrays instead (Sc
for(i) a[i].x = b[i].x; (slow, non stride 1) => for(i) a.x[i] = b.x[i]; (fast, stride 1)
Source Code
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <assert.h>
// Creates an array of random floats. Each number has a value from 0 - 1
float* create_rand_nums(const int num_elements) {
float *rand_nums = (float *)malloc(sizeof(float) * num_elements);
assert(rand_nums != NULL);
for (int i = 0; i < num_elements; i++) {
rand_nums[i] = (rand() / (float)RAND_MAX);
}
return rand_nums;
}
// Distance**2 between d-vectors pointed to by v1, v2.
float distance2(const float *v1, const float *v2, const int d) {
float dist = 0.0;
for (int i=0; i<d; i++) {
float diff = v1[i] - v2[i];
dist += diff * diff;
}
return dist;
}
// Assign a site to the correct cluster by computing its distances to
// each cluster centroid.
int assign_site(const float* site, float* centroids,

The result after implementing suggestions

Not secure | https://profiler.nsmindia.in/profileSummary?jname=retestMPI2

CDAC
HPC PROFILER

- Dashboard
- Job Submission
- Application Profile Info

Application "retestMPI2" Summary

Experiment Name	batch_test
Application	/home/neerajs/jobInfo/kmeans_-g-O3-xHost
Timestamp	2022-09-05 14:44:06
Experiment Type	MPI
Machine	ssl-cn01
Architecture	x86_64
Micro Architecture	SKYLAKE
Model Name	Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz
Cache Size	19712 KB
Number of Cores	12
OS Version	Linux 3.10.0-862.el7.x86_64 #1 SMP Fri Apr 20 16:44:24 UTC 2018
Number of processes observed	0
Number of threads observed	0

Application "retestMPI2" Characterization

Application is bound to User Code

Computation time	26.8%	Time spent in running application code,High values are usually good This is very low, focus on improving other section first
MPI time	59.7%	Time spent in MPI library call code,High values are usually bad This is high, Check the MPI breakdown for improvements
OMP time	0	Time spent in OMP region,High values are usually bad This is neglibile, focus on improving other section first
IO time	1.93%	Time spent in Filesystem IO,High values are usually bad This is neglibile, focus on improving other section first

Application "retestMPI2" Potential Speed Up

Potential Speedup If Fully Vectorised	1.03
Potential Speedup If Only FP Arithmetic	1.13

Configuration Summary

run_command	10000 100 100
profile_start	{ unit = s ; value = 0 ; }
mpi_command	srun --mpi=pmi2 --job-name=retestMPI2 --ntasks= --ntasks-per-node=
omp_num_threads	1

Application "retestMPI2" Execution Summary

Total Time (s)	16.13
Time in Analyzed Loops (%)	26.9
Time in Analyzed Innermost Loops (%)	11.9
Compilation Option Used	kmeans_-g-O3-xHost: Intel 18.0.3.222 -I/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux std=c99 -g -O3 -xHost -o ./g/kmeans_-g-O3-xHost -L/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linu -L/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linu enable-new-dtags -Xlinker -rpath -Xlinker /opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux -Xlinker -rpath -Xlinker /opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux rpath -Xlinker /opt/intel/mpi-rt/2017.0.0/intel64/lib/debug_mt -Xlink

THANK YOU !

Feedback form link : bit.ly/hpcprof