



# 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 snapshot 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()  
{



← 30 samples

Beta();  
}

Functions	Inclusive	Exclusive
Alpha	80	30
Beta	50	50

Void Beta()  
{



← 50 samples

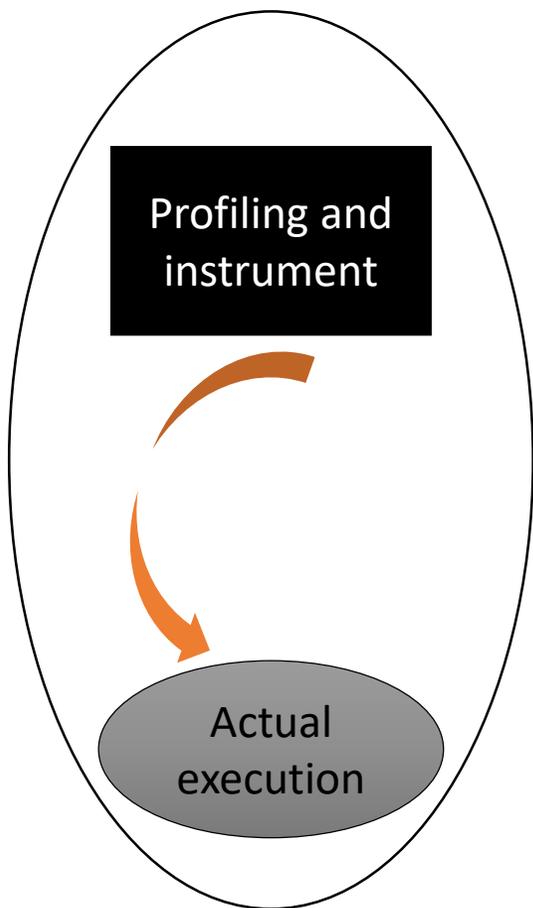
}



# 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

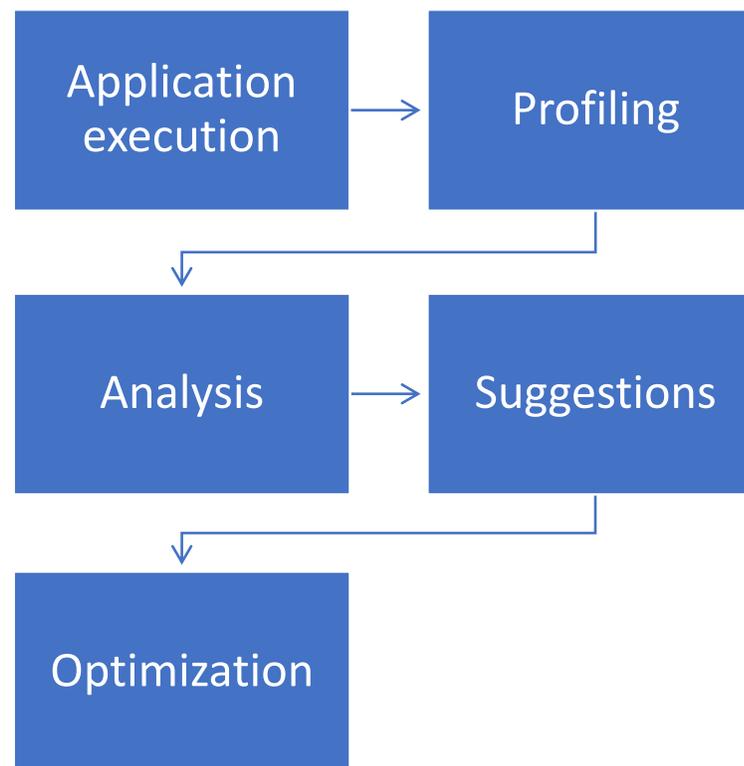


```
T=E*F;  
For (I=1;I<N;I++)  
{  
V[I]=C[I]*B[I];  
A[I]=C(2I+4);  
}
```

```
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





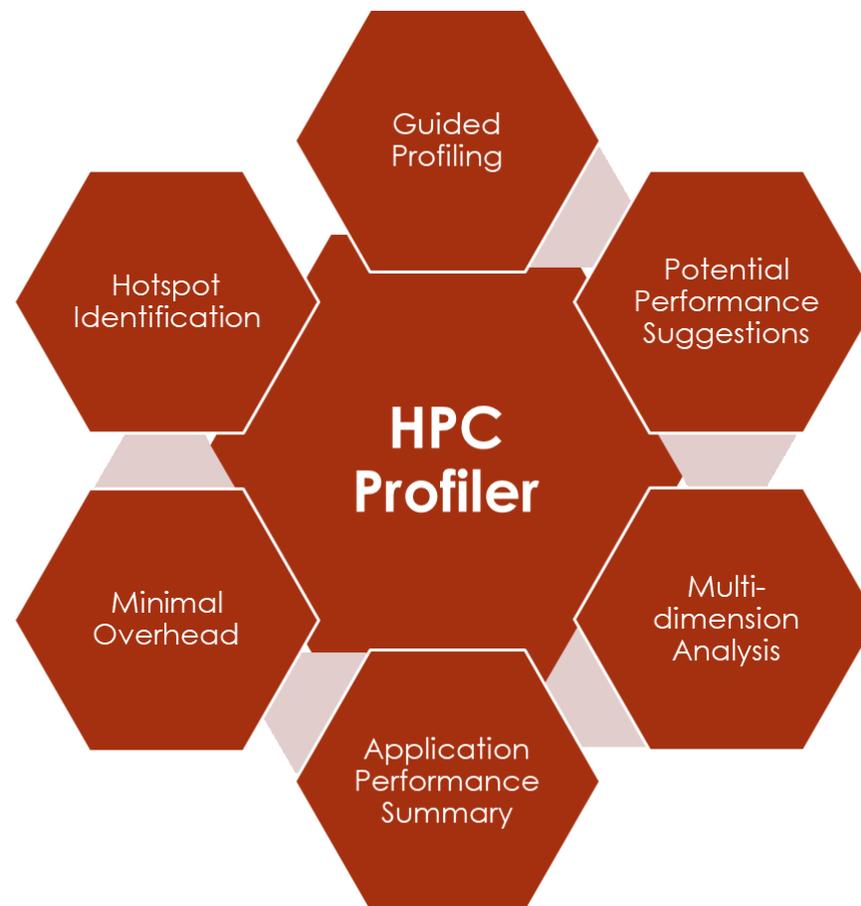
# What can a HPC Profiler tell us



- 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 :-

**CUAC HPC PROFILER**

- Dashboard
- Job Submission
- Application Profile Info

### Job to Profile

Job Name

Executable   
[Executable](#)

Source   
[Source code](#)

Input Parameters

Type of Job

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	<a href="#">View Profile Data</a>
19828	kmeans8_28_09	COMPLETED	<a href="#">View Profile Data</a>
19827	kmeans16_28_09	CANCELLED by 1033	Job is Cancelled
19826	kdemo_28_09	COMPLETED	<a href="#">View Profile Data</a>
19820	kmeans_testing_source_file	COMPLETED	<a href="#">View Profile Data</a>
19819	kmeansRErun	COMPLETED	<a href="#">View Profile Data</a>
19818	kmeansDM3_xhost	COMPLETED	<a href="#">View Profile Data</a>
19810	kmeansDM2	COMPLETED	<a href="#">View Profile Data</a>
19809	KmeansDM1	COMPLETED	<a href="#">View Profile Data</a>
19807	kmeans_AVX512_26-09	COMPLETED	<a href="#">View Profile Data</a>

Showing 1 to 10 of 10 entries [Previous](#) [1](#) [Next](#)



# Application summary page :-

**CDAC HPC PROFILER**

Dashboard

Job Submission

Application Profile Info

Profile Level
Node Name
Process Id

Application Summary
Nodes
Process Id

### Application "mpiExp8" Summary

Experiment Name	batch_test
Application	/home/neerajs/jobInfo/kmeans_mpiicc_O3
Timestamp	2022-09-05 14:16:50
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 "mpiExp8" Potential Speed Up

Potential Speedup If Fully Vectorised	1.43
Potential Speedup If Only FP Arithmetic	1.10

### Configuration Summary

run_command	10000 100 100
profile_start	{ unit = s ; value = 0 ; }
mpi_command	srunc --mpi=pmi2 --job-name=mpiExp8 --ntasks= --ntasks-per-node=
omp_num_threads	1

### Application "mpiExp8" Execution Summary

Total Time (s)	25.93
Time in Analyzed Loops (%)	44.3
Time in Analyzed Innermost Loops (%)	32.2
Compilation Option Used	
Suggested Compilation Options	Not Available

### Application "mpiExp8" Characterization

Application is bound to User Code

Computation ■ 44.3% Time spent in running application code, High values are

Execution time details



Not secure | [https://profiler.nsmindia.in/profileSummary?jname=Demo\\_IIT\\_CDAC\\_Meet](https://profiler.nsmindia.in/profileSummary?jname=Demo_IIT_CDAC_Meet)

CDAC HPC PROFILER

- Dashboard
- Job Submission
- Application Profile Info

Profile Level: Application Summary | Node Name: Nodes | Process Id: Process Id

### Application "Demo\_IIT\_CDAC\_Meet" Summary

Experiment Name	batch_test
Application	/home/neerajs/jobInfo/kmeans_-g
Timestamp	2022-12-09 15:57:43
Experiment Type	MPI
Machine	ssl-cn02
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
Total no. of Process	8

### Application "Demo\_IIT\_CDAC\_Meet" Potential Speed Up

Potential Speedup If Fully Vectorised	4.82
Potential Speedup If Only FP Arithmetic	2.69

### Configuration Summary

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

### Application "Demo\_IIT\_CDAC\_Meet" Execution Summary

Total Time (s)	143.75
Time in	90

### Application "Demo\_IIT\_CDAC\_Meet" Characterization

Application is bound to User Code

Computation time	91.1%	Time spent in running application code, High values are usually good This is high, Check the CPU performance section, TMAM and vectorization for more advise
MPI time	7.12%	Time spent in MPI library call code, High values are usually bad This is very low, This code may benefit from a higher processor count
OMP time	0	Time spent in OMP region, High values are usually bad This is negligble, focus on improving other section first
IO time	0.23%	Time spent in Filesystem IO, High values are usually bad This is negligble, focus on improving other section first

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

```

/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux/mpi/intel64/include -
std=c99 -g -o ./g/kmeans_-g -
-L/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux/mpi/intel64/lib/debug
-L/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux/mpi/intel64/lib -Xlink
enable-new-dtags -Xlinker -rpath -Xlinker

```



# Process level info page :-

**CDAC HPC PROFILER**

- Dashboard
- Job Submission
- Application Profile Info

Profile Level: Process Level
Node Name: node\_ssl-cn01
Process Id: 214856

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	11.05	0.11	0.11 (214856)
__mpn_mul_1	libc.so.6	NA	1.32	0.01	0.01 (214856)
__parse_one_specmb	libc.so.6	NA	1.21	0.01	0.01 (214856)
_IO_default_xsputn	libc.so.6	NA	1.24	0.01	0.01 (214856)
_IO_file_xsputn	libc.so.6	NA	2.00	0.02	0.02 (214856)
_IO_vfprintf	libc.so.6	NA	6.71	0.07	0.07 (214856)
buffered_vfprintf	libc.so.6	NA	2.57	0.03	0.03 (214856)
hack_digit.13661	libc.so.6	NA	3.02	0.03	0.03 (214856)
main	kmeans_g-03	kmeans.c:8-206   stdlib.h:280-280	59.68	0.61	0.61 (214856)

Loop ID	Module	Source Info	Function Name	Level	Coverage %
31	kmeans_g-03	kmeans.c:19-21	main	Innermost	26.48
29	kmeans_g-03	kmeans.c:18-150	main	InBetween	16.07
30	kmeans_g-03	kmeans.c:19-21	main	Innermost	13.16

**Suggestions**

**FMA**

Presence of both ADD/SUB and MUL operations.  
Workaround(s):

- Pass to your compiler a micro-architecture specialization option:
  - \* use axHost or xHost
- Try to change order in which elements are evaluated (using parentheses) in arithmetic expres

For instance a + b\*c is a valid FMA (MUL then ADD).  
However (a+b)\* c cannot be translated into an FMA (ADD then MUL).

---

**Source Code**

```

6 // Creates an array of random floats. Each number has a value from 0 - 1
7 float* create_rand_nums(const int num_elements) {
8     float *rand_nums = (float *)malloc(sizeof(float) * num_elements);
9     assert(rand_nums != NULL);
10    for (int i = 0; i < num_elements; i++) {
11        rand_nums[i] = (rand() / (float)RAND_MAX);
12    }
13    return rand_nums;
14 }
15
16 // Distance**2 between d-vectors pointed to by v1, v2.
17 float distance2(const float *v1, const float *v2, const int d) {
18     float dist = 0.0;
19    for (int i=0; i<d; i++) {
20        float diff = v1[i] - v2[i];
21        dist += diff * diff;
22    }
23    return dist;
24 }
25
26 // Assign a site to the correct cluster by computing its distances to
27 // each cluster centroid.
28 int assign_site(const float* site, float* centroids,
29               const int k, const int d) {
30     int best_cluster = 0;
31     float best_dist = distance2(site, centroids, d);
32     float* centroid = centroids + d;
33     for (int c = 1; c < k; c++, centroid += d) {

```



**CDAC HPC PROFILER**

- Dashboard
- Job Submission
- Application Profile Info

<https://profiler.nsmindia.in/index.php>

Profile Level

Process Level

Node Name

node\_ssl-cn01

Process Id

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

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 (11.7x). Store and arithmetical SSE/AVX instructions are used in scalar version (process only one data element). 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 (SoA): 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

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <mpi.h>
4 #include <assert.h>
5
6 // Creates an array of random floats. Each number has a value from 0 - 1
7 float* create_rand_nums(const int num_elements) {
8     float *rand_nums = (float *)malloc(sizeof(float) * num_elements);
9     assert(rand_nums != NULL);
10    for (int i = 0; i < num_elements; i++) {
11        rand_nums[i] = (rand() / (float)RAND_MAX);
12    }
13    return rand_nums;
14 }
15
16 // Distance**2 between d-vectors pointed to by v1, v2.
17 float distance2(const float *v1, const float *v2, const int d) {
18     float dist = 0.0;
19     for (int i=0; i<d; i++) {
20         float diff = v1[i] - v2[i];
21         dist += diff * diff;
22     }
23     return dist;
24 }
25
26 // Assign a site to the correct cluster by computing its distances to
27 // each cluster centroid.
28 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 negligible, focus on improving other section first
IO time	1.93%	Time spent in Filesystem IO, High values are usually bad This is negligible, focus on improving other section first

### Application "retestMPI2" Potential Speedup

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	srunk --mpi=pmpi2 --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 - /opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux/std=c99 -g -O3 -xHost -o ./g/kmeans_-g-O3-xHost - /opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux -L/opt/ohpc/pub/intel_2018/compilers_and_libraries_2018.3.222/linux -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 -Xlinker



THANK YOU !

Feedback form link : [bit.ly/hpcprof](https://bit.ly/hpcprof)