



CDAC HPC Profiler

Abhishek Patil



Outline

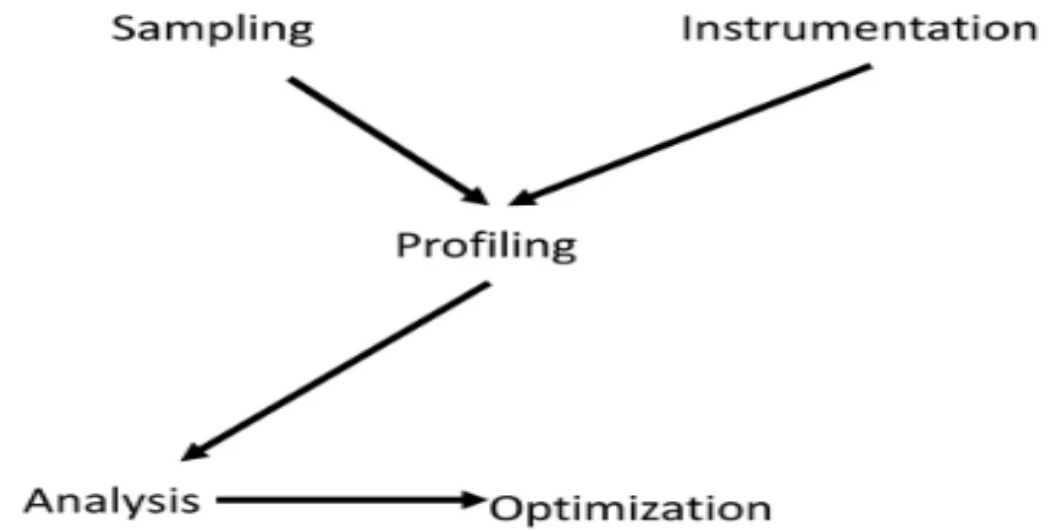
- Introduction
- What is Profiling
- Key Features
- Application Run/Demo
- Screenshots
- Outcome of the experiments



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

Big Picture of Profiling





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).



Brief info about Instrumenting

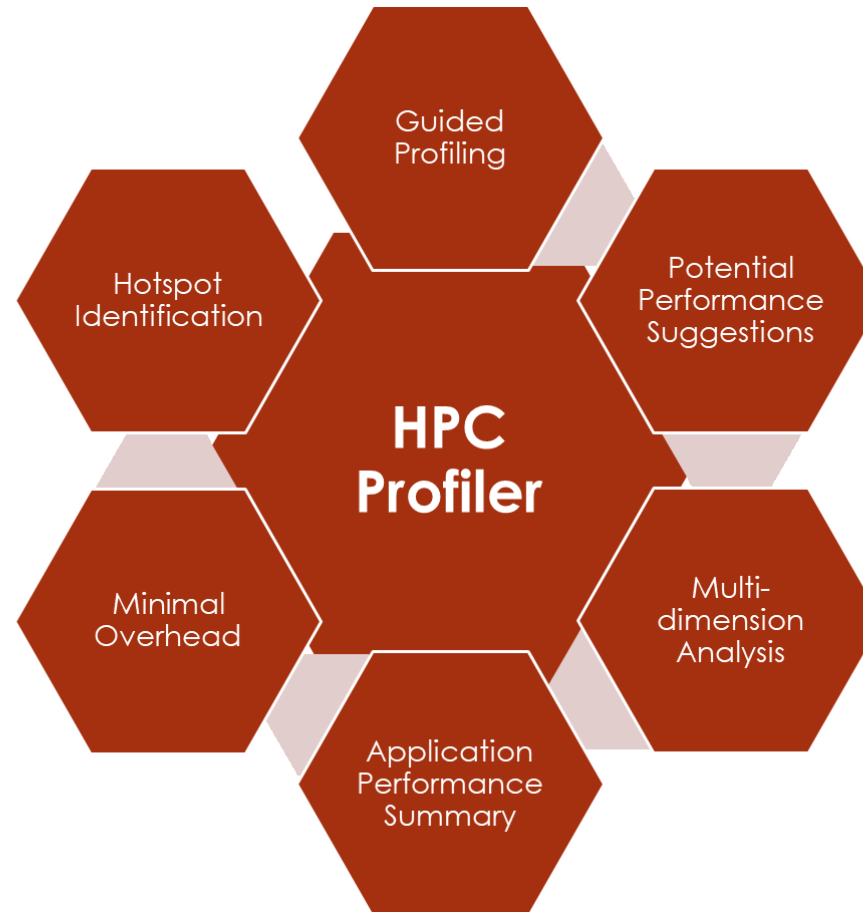
- The first and earliest type are instrumenting profilers.
- They work by *instrumenting* the program under test in order to collect information about the **resource of interest**.
- For example if you wanted to **calculate how much time** methods were taking to execute an instrumenting profiler would **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**.

What can a 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 function profiles , vectorization analyses, etc.

HPC Profiler – Key features





Screenshots

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 :-



CDAC
HPC PROFILER

Dashboard

Job Submission

Application Profile Info

Profile Level

Application Summary

Node Name

Nodes

Process Id

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" Characterization

Application is bound to User Code

Computation

Time spent in running application code, High values are

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	srunk --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



Cluster level page :-

> Application Profile Info

Show 10 entries
Search:

Function Name	Source Info	Coverage (%)								
distance2	kmeans.c:17-23	87.92								
<table border="1"> <thead> <tr> <th>Loop ID</th> <th>Source Info</th> <th>Level</th> <th>Coverage %</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>kmeans.c:19-21</td> <td>Single</td> <td>87.43</td> </tr> </tbody> </table>	Loop ID	Source Info	Level	Coverage %	1	kmeans.c:19-21	Single	87.43		
Loop ID	Source Info	Level	Coverage %							
1	kmeans.c:19-21	Single	87.43							
MPIDI_CH3I_MRAIL_Cq_poll_ib	NA	2.51								
assign_site	kmeans.c:29-40	2.22								
MPIDI_CH3I_MRAIL_Get_next_vbuf	NA	1.43								
MPIDI_CH3I_SMP_read_progress	NA	1.18								
add_site	kmeans.c:45-49	0.90								
pthread_spin_lock	NA	0.79								
MPIDI_CH3I_SMP_pull_header	NA	0.70								
.text@start	NA	0.55								
MPIDI_CH3I_Progress_test	NA	0.31								

Showing 1 to 10 of 164 entries

Previous
1
2
3
4
5
...
17
Next

☒ Function Name
☐ Module
☒ Source Info
☒ Coverage
☐ Time w.r.t Walltime

Application Categorization

Vectorization_Suggestion

Your function is probably not vectorized. Only 10% 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 6.00 to 0.50 cycles (12.0 Store and arithmetical SSE/AVX instructions are used in scalar version (process only one data el 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

```

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,
29               const int k, const int d) {
30   int best_cluster = 0;

```

Process level info page :-



> Application Profile Info

Show 10 entries

Search:

Function Name	Module	Source Info	Coverage (%)	Time w.r.t Walltime (s)	Time Min (s) (TID)
.text@start	libmlx5.so.1.3.15	NA	0.56	0.81	0.81 (414681)
add_site	kmeans_g	kmeans.c:45-49	0.89	1.28	1.28 (414681)
Loop ID	Module	Source Info	Function Name	Level	Coverage %
3	kmeans_g	kmeans.c:46-47	add_site	Single	0.88
Data not available for this function					
assign_site	kmeans_g	kmeans.c:29-40	2.18	3.13	3.13 (414681)
distance2	kmeans_g	kmeans.c:17-23	87.20	125.10	125.10 (414681)
MPIDI_CH3I_MRAIL_Cq_poll_ib	libmpi.so.12.1.1	NA	2.91	4.18	4.18 (414681)
MPIDI_CH3I_MRAIL_Get_next_vbuf	libmpi.so.12.1.1	NA	1.48	2.12	2.12 (414681)
MPIDI_CH3I_Progress_test	libmpi.so.12.1.1	NA	0.55	0.79	0.79 (414681)
MPIDI_CH3I_SMP_pull_header	libmpi.so.12.1.1	NA	0.74	1.06	1.06 (414681)

Vectorization_Suggestion

Your function is 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 5.00 to 0.44 cycles (11.43).
All SSE/AVX instructions are used in scalar version (process only one data element in vector register).
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

```
31 float best_dist = distance2(site, centroids, d);
32 float* centroid = centroids + d;
33 for (int c = 1; c < k; c++, centroid += d) {
34     float dist = distance2(site, centroid, d);
35     if (dist < best_dist) {
36         best_cluster = c;
37         best_dist = dist;
38     }
39 }
40 return best_cluster;
41 }
42
43 // Add a site (vector) into a sum of sites (vector).
44 void add_site(const float * site, float * sum, const int d) {
45     for (int i=0; i<d; i++) {
46         sum[i] += site[i];
47     }
48 }
49
50 // Print the centroids one per line.
51 void print_centroids(float * centroids, const int k, const int d) {
52     float *p = centroids;
53     printf("Centroids:\n");
54     for (int i = 0; i<k; i++) {
55         for (int j = 0; j<d; j++, p++) {
56             printf("%f ", *p);
57         }
58     }
```



The result after implementing suggestions

Outcome of experiment :-



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	srunk --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 - l/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/lin 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 !

For queries contact us at:

nsmss@cdac.in