Profiling and Optimization

Lochan Khedkar

Outline

- What is Profiling
- Sampling based profiling
- Instrumentation based profiling
- Flow of Profiling and Analysis
- Profilers available for performance analysis
- OpenMP Profiling using Vtune
- Analysis of OpenMP application
- Optimization
- GNU Profiler Next Generation
- OpenMP analysis using GProfNG
- Tuning and Analysis Utility Profiler
- Profiling MPI application using Tuning and Analysis utility (TAU)Profiler

What is Profiling

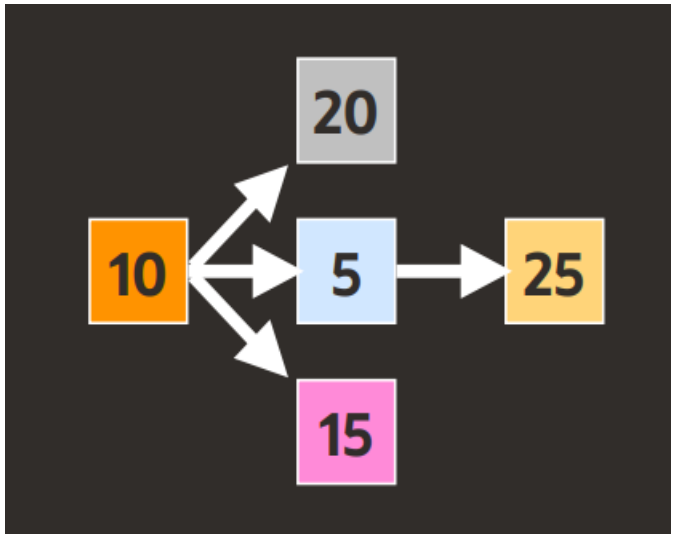
- Application profiling is the process of analysing and measuring the performance of a software application in order to identify and diagnose performance issues or bottlenecks.
- Profiling tools help developers to identify areas of code that are taking a long time to execute, or that are using an excessive amount of system resources like memory, CPU or disk I/O.
- The purpose of profiling is to identify any bottlenecks or areas of the application that can be optimized to improve its performance, scalability, and reliability.
- By profiling their applications, developers can ensure that they are delivering high-quality software that performs well and meets the needs of their users.
- Profiling involves collecting data on various aspects of the application and its analysis.
- Implemented through either
  - Sampling based profiling
  - Instrumentation based Profiling

Sampling Based Profiling

- With sampling, the executable is stopped at regular intervals. Each time it is halted, key information is gathered and stored.
- At specified intervals, the Sampling method collects information about the functions that are executing in your application. While the program is interrupted the profiler grabs a snapshot of its current state.
- Sampling is a low cost to the profiler and has little effect on the execution of the application being profiled.
- If you need accurate measurements of call times or are looking for performance issues in an application , then sampling based profiler are useful.
- Sampling has less accuracy in the number of calls

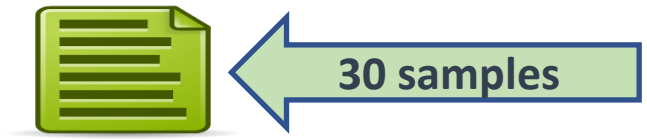
Inclusive Vs Exclusive CPU Time

- This is an important concept in profiling tools
- The inclusive metric includes all callees underneath the caller - For example, all the CPU time accumulated when executing a function
- The exclusive metric excludes everything outside the caller - For example, the CPU time accumulated outside of calling other functions



Function	Inclusive time	Exclusive time
A	75	10
B	20	20
C	30	5
D	15	15
E	25	25

```
Void Alpha()
{
```



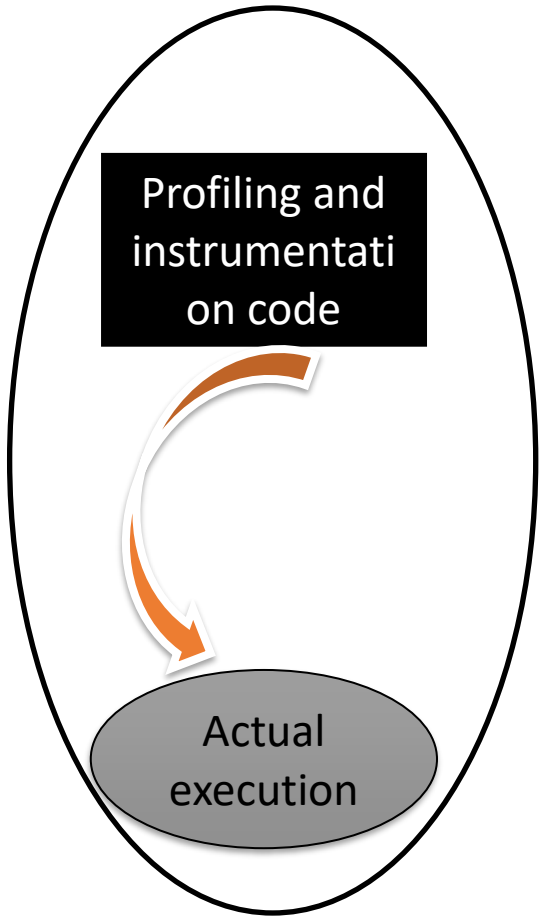

```
 Beta();

}

Void Beta()
{
```


```
}
```

Functions	Inclusive	Exclusive
Alpha	80	30
Beta	50	50

Instrumentation based Profiling

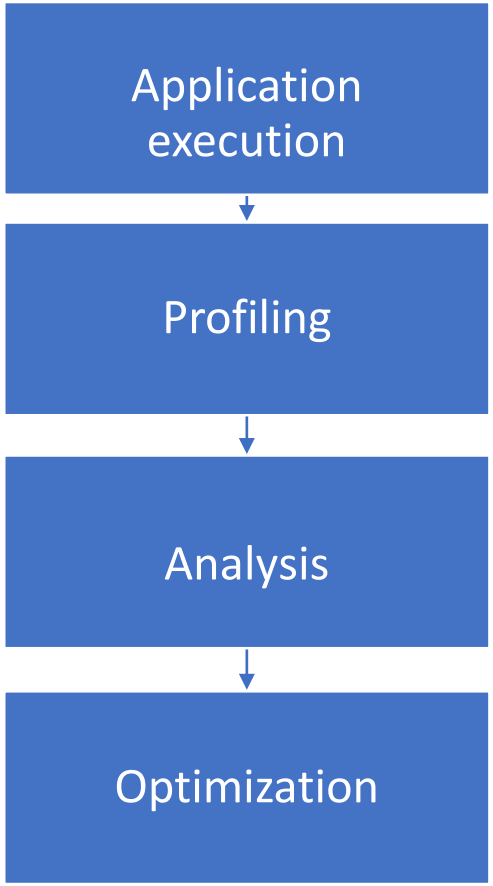
- The first and earliest type are instrumenting profilers.
- Data collection is done by tools that either injecting code into a binary file that captures timing information or by using callback hooks to collect and emit exact timing and call count information while an application runs.
- The instrumentation method has a high overhead when compared to sampling-based approaches.
- Instrumentation profiling is that you can get exact call counts on how many times your functions were called.
- Instrumentation can be achieved during
  - Compile time Instrumentation
  - Binary Instrumentation




```
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



Profilers Available for performance analysis

- Intel's Vtune
- GNU Profiler(gprof)
- GNU Profiler Next Generation(GProfNG) by Oracle
- Tuning and analysis Utility Profiler by Oregon University

OpenMP Profiling using Vtune

Installation steps for vtune

- wget https://registrationcenter-download.intel.com/akdlm/IRC_NAS/56d0db2b-1ff1-4abe-857a-72ca9be22bd3/l_oneapi_vtune_p_2024.0.1.14_offline.sh
- chmod u+x l_oneapi_vtune_p_2024.0.1.14_offline.sh
- ./ l_oneapi_vtune_p_2024.0.1.14_offline.sh

Hotspot analysis using Vtune

gcc –g –fopenmp –o mat_mult mat_mult.c
vtune -collect hotspots ./mat_mult

```
vtune: Executing actions 75 % Generating a report                              Elapsed Time: 54.413s
    CPU Time: 31.760s
        Effective Time: 31.110s
        Spin Time: 0.640s
            Imbalance or Serial Spinning: 0.640s
            Lock Contention: 0s
            Other: 0s
        Overhead Time: 0.010s
            Creation: 0.010s
            Scheduling: 0s
            Reduction: 0s
            Atomics: 0s
            Other: 0s
    Total Thread Count: 20
    Paused Time: 0s

Top Hotspots
Function                   Module        CPU Time  % of CPU Time(%)
-------------------------  ------------  --------  ----------------
printf                     libc.so.6      25.906s             81.6%
multiply._omp_fn.0         mm              3.520s             11.1%
putchar                    libc.so.6       1.644s              5.2%
gomp_simple_barrier_wait   libgomp.so.1    0.290s              0.9%
gomp_team_barrier_wait_end libgomp.so.1    0.220s              0.7%
[Others]                   N/A             0.180s              0.6%
Effective CPU Utilization: 0.4%
 | The metric value is low, which may signal a poor logical CPU cores
 | utilization caused by load imbalance, threading runtime overhead, contended
 | synchronization, or thread/process underutilization. Explore sub-metrics to
 | estimate the efficiency of MPI and OpenMP parallelism or run the Locks and
 | Waits analysis to identify parallel bottlenecks for other parallel runtimes.
 |
    Average Effective CPU Utilization: 0.169 out of 40
Collection and Platform Info
    Application Command Line: ./mm
    Operating System: 3.10.0-1160.el7.x86_64 #@@#################################################################################################@@# #
```

Analysis

- Effective CPU execution time is 32.249 before optimization.
- Presence of spin time in Imbalance or serial spinning category indicates potential area for optimization.
- Also overhead time is crucial for optimal performance and should be as low as possible.
- Thread count should be appropriate for your workload and does not lead to excessive overhead.
- Logical core utilization value indicate poor CPU utilization.
- Low CPU utilization caused due to load imbalance, threading runtime overhead, synchronization and thread under utilization
- printf and multiply. _omp_fn.o consume maximum CPU time and needs to be optimize.
- It indicates that parallel region multiply._omp_fn.o can be optimized further.

Optimization

- Use thread affinity to bind threads to specific CPU and avoid using large no of threads for reducing thread overheads.
- Experiment with chunk size in schedule clause to find the optimal balancing between load balancing and overheads.
- Consider using collapse clause to combine multiple nested loops into single loop for improved parallelization.
- Enable compiler optimization flags to allow compiler to perform additional optimization.
- Utilize SIMD (single instructions multiple data by enabling compiler vectorization flags
- Experiment with loop unrolling to reduce loop overhead and improve instruction level parallelism.

Hotspot analysis using Vtune after optimization

```
vtune: Executing actions 75 % Generating a report                              Elapsed Time: 51.335s
    CPU Time: 34.408s
        Effective Time: 34.078s
        Spin Time: 0.330s
            Imbalance or Serial Spinning: 0.330s
            Lock Contention: 0s
            Other: 0s
        Overhead Time: 0s
            Creation: 0s
            Scheduling: 0s
            Reduction: 0s
            Atomics: 0s
            Other: 0s
    Total Thread Count: 20
    Paused Time: 0s

Top Hotspots
Function                   Module        CPU Time  % of CPU Time(%)
-------------------------  ------------  --------  ----------------
printf                     libc.so.6      26.972s             78.4%
multiply._omp_fn.0         mm              5.388s             15.7%
putchar                    libc.so.6       1.688s              4.9%
gomp_simple_barrier_wait   libgomp.so.1    0.130s              0.4%
gomp_team_barrier_wait_end libgomp.so.1    0.100s              0.3%
[Others]                   N/A             0.130s              0.4%
Effective CPU Utilization: 0.5%
 | The metric value is low, which may signal a poor logical CPU cores
 | utilization caused by load imbalance, threading runtime overhead, contended
 | synchronization, or thread/process underutilization. Explore sub-metrics to
 | estimate the efficiency of MPI and OpenMP parallelism or run the Locks and
 | Waits analysis to identify parallel bottlenecks for other parallel runtimes.
 |
    Average Effective CPU Utilization: 0.214 out of 40
Collection and Platform Info
    Application Command Line: ./mm
    Operating System: 3.10.0-1160.el7.x86_64 #@@##########################################################################################@@# #
```

GNU Profiler Next Generation(GProfNG) by Oracle

Installation

- wget https://ftp.gnu.org/gnu/binutils/binutils-2.42.tar.gz
- tar -xvf binutils-2.42.tar.gz
- ./configure –prefix=/install_directory –with-gmp=/gmp_install_directory
- make ;make install

Openmp Analysis with GProfNG

- gcc –g –fopenmp –o mat_mult mat_mult.c
- Collect the data using following command
- gprofng collect app ./mat_mult
- Display the hotspot functions in mat_mult.c application using following command
- gprofng display text –functions test.2.er

```
[lochannk@login02 oneapi]$ gprofng display text -functions test.2.er/
Functions sorted by metric: Exclusive Total CPU Time

Excl. Total    Incl. Total     Name
CPU            CPU
 sec.       %   sec.       %
7.035  100.00   7.035  100.00   <Total>
3.342   47.51   3.342   47.51   multiply._omp_fn.0
2.822   40.11   2.822   40.11   <static>@0xefa99 (<libc-2.17.so>)
0.250    3.56   3.252   46.23   __printf_fp_l
0.120    1.71   3.422   48.65   vfprintf
0.100    1.42   0.100    1.42   __mpn_divrem
0.070    1.00   2.892   41.11   _IO_new_file_overflow
0.070    1.00   0.170    2.42   hack_digit.13666
0.060    0.85   0.060    0.85   __GI_memcpy
0.060    0.85   3.653   51.92   multiply
0.040    0.57   0.040    0.57   _IO_new_file_xsputn
0.040    0.57   0.040    0.57   gomp_team_barrier_wait_end
0.020    0.28   0.020    0.28   __overflow
0.020    0.28   0.020    0.28   gomp_barrier_wait_end
0.010    0.14   0.010    0.14   __GI_strlen
0.010    0.14   0.010    0.14   strchrnul
0.       0.     2.822   40.11   _IO_new_do_write
0.       0.     2.822   40.11   _IO_new_file_write
0.       0.     3.653   51.92   __libc_start_main
0.       0.     0.380    5.41   collector_root
0.       0.     0.020    0.28   gomp_team_end
0.       0.     0.380    5.41   gomp_thread_start
0.       0.     3.653   51.92   main
0.       0.     3.422   48.65   printf
0.       0.     0.150    2.13   putchar
```

Openmp thread analysis using GProfNG

- Threads analysis is done by following command
- gprofng display text –threads test.2.er/

```
[lochannk@login02 oneapi]$ gprofng display text -threads test.2.er/
Objects sorted by metric: Exclusive Total CPU Time

Excl. Total     Name
CPU
 sec.       %
7.035 100.00    <Total>
3.653  51.92    Process 1, Thread 1
3.342  47.51    Process 1, Thread 3
0.020   0.28    Process 1, Thread 2
0.020   0.28    Process 1, Thread 4
```

- gprofng display text –thread_select 3 –function

```
[lochannk@login02 oneapi]$ gprofng display text -thread_select 2 -functions test.2.er/
Exp Sel Total
=== === =====
  1 2       4
Functions sorted by metric: Exclusive Total CPU Time

Excl. Total   Incl. Total     Name
CPU           CPU
 sec.      %   sec.      %
0.020 100.00  0.020 100.00   <Total>
0.010  50.00  0.010  50.00   gomp_barrier_wait_end
0.010  50.00  0.010  50.00   gomp_team_barrier_wait_end
0.      0.    0.020 100.00   collector_root
0.      0.    0.020 100.00   gomp_thread_start
```

Profiling MPI application using Tuning and Analysis utility (TAU)Profiler

Installation

- wget http://tau.uoregon.edu/tau.tgz
- tar -xvf tau.tgz
- /configure -c++=mpicxx -cc=mpicc -fortran=mpif90 -tag=openmpi -mpi -bfd=download - pdt=/home/lochannk/pdtoolkit-3.25.1 -papi=/opt/cray/pe/papi/6.0.0.2
- make
- make install

MPI application functions analysis

- mpicc –g –o matrix_mult mpi_mm.c
- mpirun –np 2 tau_exec –T openmpi ./matrix_mult

```
[lochannk@login02 oneapi]$ pprof
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.813          388           1           1     388802 .TAU application
 99.8           21          387           1          11     387989 taupreload_main
 60.1          233          233           1           0     233500 MPI_Init()
 24.5           95           95           1           0      95352 MPI_Finalize()
  7.7           30           30           3           0      10010 MPI_Recv()
  2.1            8            8           4           0       2006 MPI_Send()
  0.0        0.008        0.008           1           0          8 MPI_Comm_size()
  0.0        0.005        0.005           1           0          5 MPI_Comm_rank()

NODE 1;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.625          386           1           1     386785 .TAU application
 99.8           30          386           1          11     386160 taupreload_main
 60.5          233          233           1           0     233876 MPI_Init()
 28.0          108          108           1           0     108387 MPI_Finalize()
  3.6           13           13           4           0       3442 MPI_Recv()
  0.0        0.102        0.102           3           0         34 MPI_Send()
  0.0        0.004        0.004           1           0          4 MPI_Comm_rank()
  0.0        0.003        0.003           1           0          3 MPI_Comm_size()
```

MPI performance counter analysis

- For performance counter analysis set export TAU_METRICS=“ TIME,PAPI_TOT_CYC”
- mpirun –np 2 tau_exec –T openmpi ./matrix_m

```
[lochannk@login02 oneapi]$ cd MULTI__PAPI_TOT_CYC/
[lochannk@login02 MULTI__PAPI_TOT_CYC]$ pprof
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs Count/Call Name
              counts total counts
---------------------------------------------------------------------------------------
100.0   3.835E+05   2.122E+08           1           1  212226018 .TAU application
 99.8   2.258E+07   2.118E+08           1          11  211842488 taupreload_main
 35.2   7.474E+07   7.474E+07           3           0   24914871 MPI_Recv()
 33.4   7.079E+07   7.079E+07           1           0   70791802 MPI_Init()
 13.1   2.788E+07   2.788E+07           4           0    6969219 MPI_Send()
  7.5   1.583E+07   1.583E+07           1           0   15830377 MPI_Finalize()
  0.0   1.117E+04   1.117E+04           1           0      11170 MPI_Comm_rank()
  0.0        5965        5965           1           0       5965 MPI_Comm_size()

NODE 1;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs Count/Call Name
              counts total counts
---------------------------------------------------------------------------------------
100.0   3.638E+05   1.896E+08           1           1  189624364 .TAU application
 99.8    7.45E+07   1.893E+08           1          11  189260527 taupreload_main
 37.3   7.077E+07   7.077E+07           1           0   70774385 MPI_Init()
 14.6   2.776E+07   2.776E+07           4           0    6941070 MPI_Recv()
  8.5   1.607E+07   1.607E+07           1           0   16065442 MPI_Finalize()
  0.1   1.383E+05   1.383E+05           3           0      46086 MPI_Send()
  0.0   1.149E+04   1.149E+04           1           0      11489 MPI_Comm_rank()
  0.0        6370        6370           1           0       6370 MPI_Comm_size()
```

THANK YOU !