Ben-Gurion University
of the Negev

home assignment #1

The work was written by

Shahaf Zohar 205978000

Iris Eting 209027333

1. Our code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mpi.h"

// Function to generate unique seeds for random number generation
void generateUniqueSeeds(int rank, int* seeds, int numProcesses);

// Function to perform Monte Carlo simulation
void performMonteCarloSimulation(int rank, int* seeds, int numProcesses, int
attempts, int* successes);

// Function to print results
void printResults(int rank, int master, int totalSuccesses, int attempts,
double startTime);

int main(int argc, char *argv[]) {
    // MPI variables
    int master = 0;
    int rank;
    int numProcesses;

    // Number of Monte Carlo attempts
    int attempts = 900000000;

    // Count of successful attempts across all processes
    int successes = 0;

    // Constant for PI with 25 decimal places
    double PI25DT = 3.141592653589793238462643;

    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numProcesses);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Allocate memory for unique seeds
    int* uniqueSeeds = (int*)malloc(sizeof(int) * numProcesses);

    // Generate unique seeds for random number generation
    generateUniqueSeeds(rank, uniqueSeeds, numProcesses);

    // Record start time
    double startTime = MPI_Wtime();

    // Perform Monte Carlo simulation
    performMonteCarloSimulation(rank, uniqueSeeds, numProcesses, attempts,
&successes);

    // Reduce successes across all processes
    int totalSuccesses = 0;
    MPI_Reduce(&successes, &totalSuccesses, 1, MPI_INT, MPI_SUM, master,
MPI_COMM_WORLD);

    // Print results
    printResults(rank, master, totalSuccesses, attempts, startTime);
    // Free allocated memory
    free(uniqueSeeds);

    // Finalize MPI
    MPI_Finalize();

    return 0;
}

// Function to generate unique seeds for random number generation
void generateUniqueSeeds(int rank, int* seeds, int numProcesses) {
    // Only rank 0 generates unique seeds
    if (rank == 0) {
        for (int i = 0; i < numProcesses; i++) {
            seeds[i] = rand();
        }
    }

    // Broadcast generated seeds to all processes
    MPI_Bcast(seeds, numProcesses, MPI_INT, 0, MPI_COMM_WORLD);
}

// Function to perform Monte Carlo simulation
void performMonteCarloSimulation(int rank, int* seeds, int numProcesses, int
attempts, int* successes) {
    // Seed the random number generator with the unique seed for each process
    srand(seeds[rank]);

    // Perform Monte Carlo simulation for the assigned number of attempts
    for (int i = 0; i < (attempts / numProcesses); i++) {
        double x = (double)rand() / RAND_MAX * 2 - 1;
        double y = (double)rand() / RAND_MAX * 2 - 1;
        double distance = x * x + y * y;

        // Check if the point is inside the unit circle
        if (distance < 1) {
            (*successes)++;
        }
    }
}

// Function to print results
void printResults(int rank, int master, int totalSuccesses, int attempts,
double startTime) {
    // Only the master process prints the final results
    if (rank == 0) {
        double endTime = MPI_Wtime();
        double PI25DT = 3.141592653589793238462643;
        double pi = 4 * ((double)totalSuccesses / attempts);

        printf("Total HITS across all processes = %d\n", totalSuccesses);
        printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi -
PI25DT));
        printf("Computation time = %f seconds\n", endTime - startTime);
        fflush(stdout);
    }
}
```

2. A brief explanation of how to run the code:

Compilation of the program:

```
mpicc Hw1.c -o a
```

Running the program after compiling and selecting several processes:

```
mpirun -np <number_of_processes> ./a
```

Using Scalasca

A performance analysis tool for parallel programs:

```
scalasca -instrument mpicc -o cpi_scalasca ./Hw1.c
```

Running the program with Scalasca and several processes can be added:

```
scalasca -analyze mpirun -np <number_of_processes> ./cpi_scalasca
```

After running the previous command, a Scalasca file is created. This command runs the Scalasca file:

```
scalasca -examine ./scorep_cpi_scalasca_<number_of_processes>_sum
```

Using jumpshot:

Compilation of the program:

```
tau_cc.sh -o Hw1_tau ./Hw1.c
```

Running the program after compiling and selecting several processes:

```
mpirun -np <number_of_processes> ./Hw1_tau
```

Run paraprof

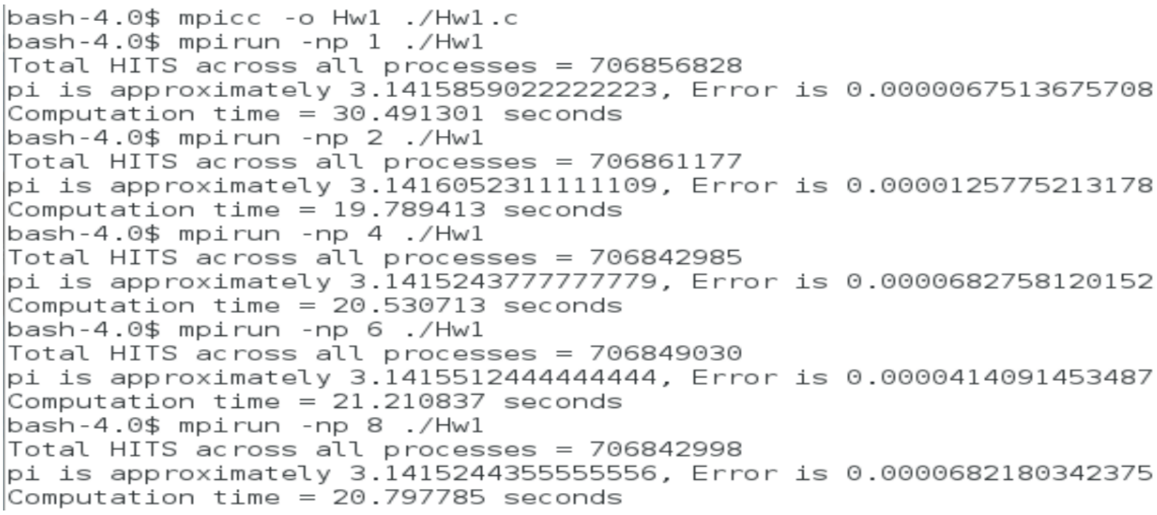
```
paraprof
```

close and run this command to prompt jumpshot

```
tau_treemerge.pl
tau2slog2 tau.trc tau.edf
jumpshot ./tau.slog2
```

3.

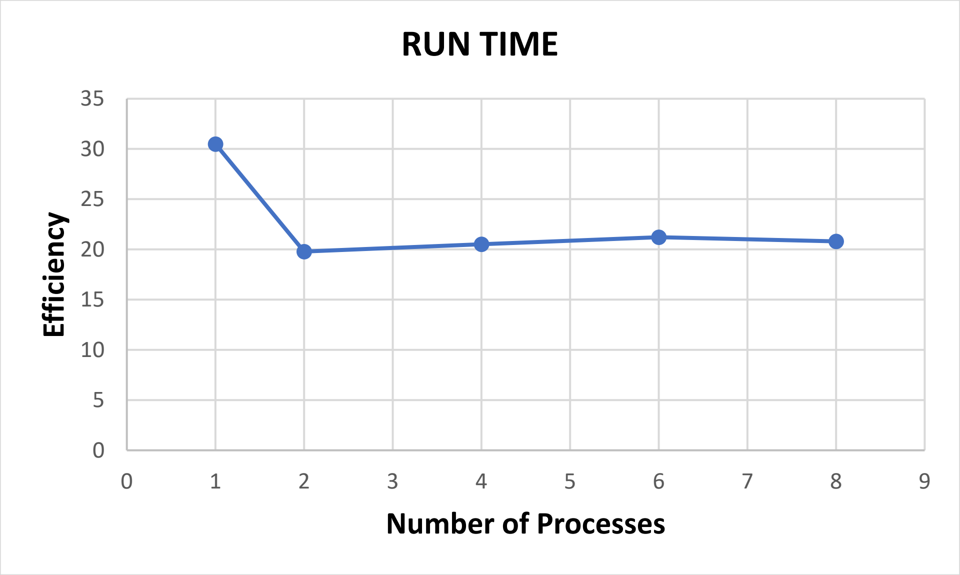
MPIRUN results of the code run for 1,2,4,6,8 processes:



```
bash-4.0$ mpicc -o Hw1 ./Hw1.c
bash-4.0$ mpirun -np 1 ./Hw1
Total HITS across all processes = 706856828
pi is approximately 3.1415859022222223, Error is 0.0000067513675708
Computation time = 30.491301 seconds
bash-4.0$ mpirun -np 2 ./Hw1
Total HITS across all processes = 706861177
pi is approximately 3.1416052311111109, Error is 0.0000125775213178
Computation time = 19.789413 seconds
bash-4.0$ mpirun -np 4 ./Hw1
Total HITS across all processes = 706842985
pi is approximately 3.1415243777777779, Error is 0.0000682758120152
Computation time = 20.530713 seconds
bash-4.0$ mpirun -np 6 ./Hw1
Total HITS across all processes = 706849030
pi is approximately 3.1415512444444444, Error is 0.0000414091453487
Computation time = 21.210837 seconds
bash-4.0$ mpirun -np 8 ./Hw1
Total HITS across all processes = 706842998
pi is approximately 3.1415244355555556, Error is 0.0000682180342375
Computation time = 20.797785 seconds
```

Number of Processes	RUN TIME	Speed Up	Efficiency
1	30.491	1	1
2	19.789	1.540805	0.770403
4	20.53	1.485192	0.371298
6	21.21	1.437577	0.239596
8	20.797	1.466125	0.183266

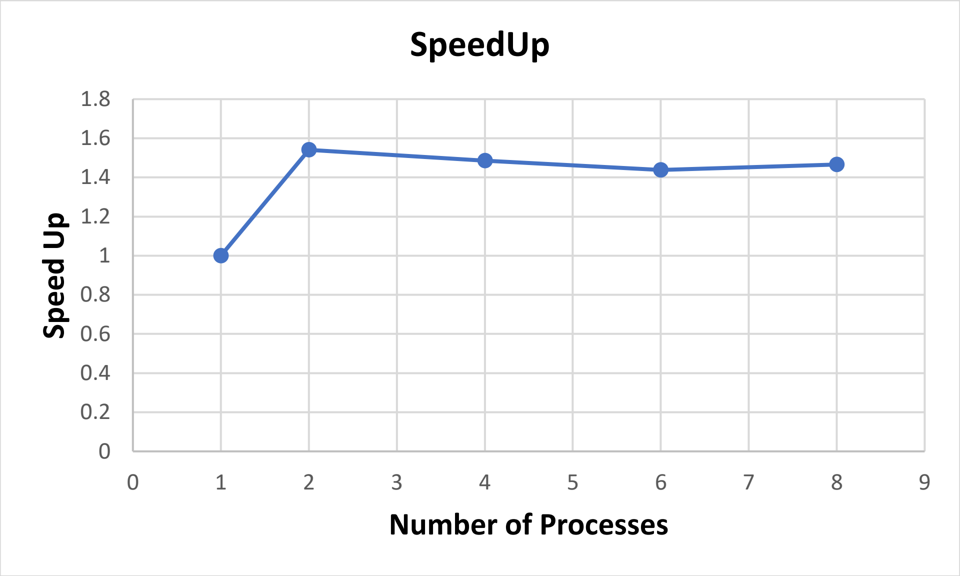
Run Time graph:



As you can see in the graph, the shortest run time is achieved by running the program with 2 processes, and it is 19.789 seconds. The worst run time is for running the program with a single process 30.491 seconds.

Speed Up graph:

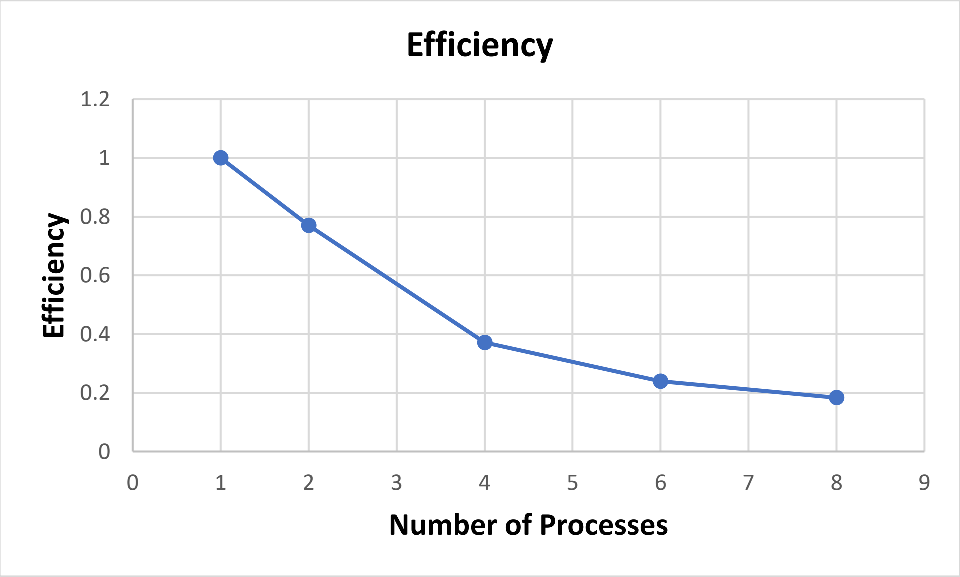
Speed Up = $S(p) = \frac{t_s}{t_p}$



As you can see in the graph, the best Speed Up is achieved by running the program with 2 processes, and it is S(p=2) = 1.54.

Efficiency graph:

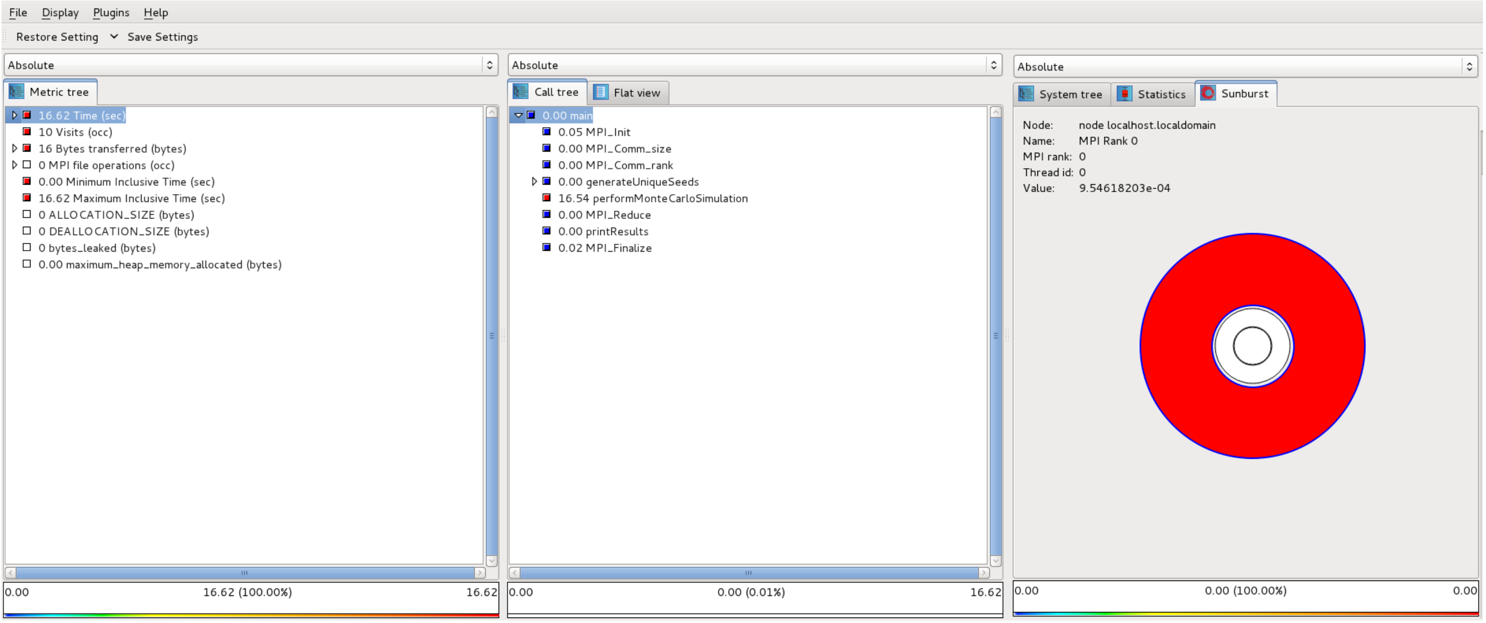
Efficiency = $E(p) = \frac{t_s}{t * t_p}$



As you can see in the graph, the best Efficiency is achieved by running the program with 1 process, and it is E = 1. The next best efficiency is achieved by running the program with 2 process E(P) = 0.77.

4) After running the Scalasca commands the following files were created:

For one process:



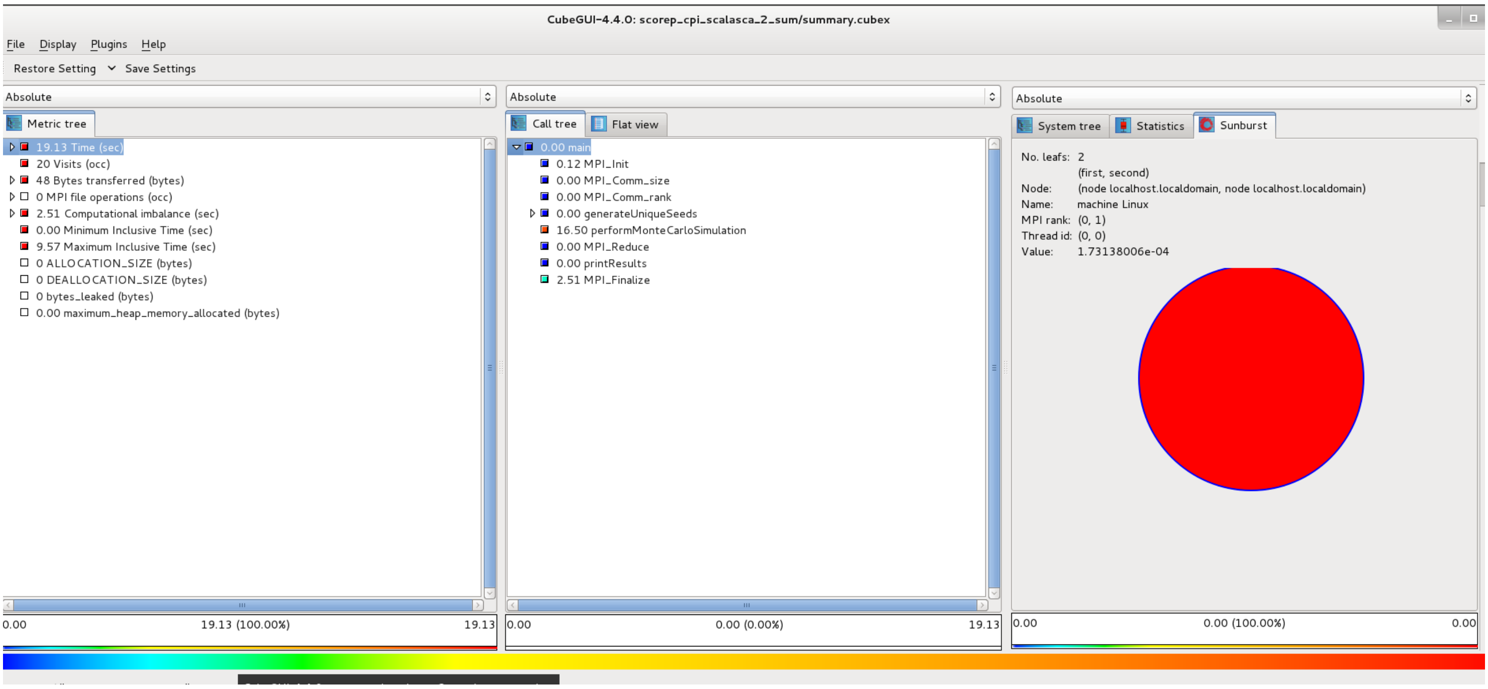
```
16.62 Time (sec)
10 Visits (occ)
16 Bytes transferred (bytes)
0 MPI file operations (occ)
0.00 Minimum Inclusive Time (sec)
16.62 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
```

```
0.00 main
  0.05 MPI_Init
  0.00 MPI_Comm_size
  0.00 MPI_Comm_rank
  0.00 generateUniqueSeeds
  16.54 performMonteCarloSimulation
  0.00 MPI_Reduce
  0.00 printResults
  0.02 MPI_Finalize
```

```
Node:      node localhost.localdomain
Name:      MPI Rank 0
MPI rank:  0
Thread id: 0
Value:     9.54618203e-04
```

```
0.00    16.62 (100.00%)    16.62
0.00    0.00 (0.01%)       16.62
0.00    0.00 (100.00%)     0.00
```

For two Process:


CubeGUI-4.4.0: scorep_cpi_scalasca_2_sum/summary.cubex
File Display Plugins Help
Restore Setting Save Settings
Absolute
Metric tree
Call tree
Flat view
System tree
Statistics
Sunburst

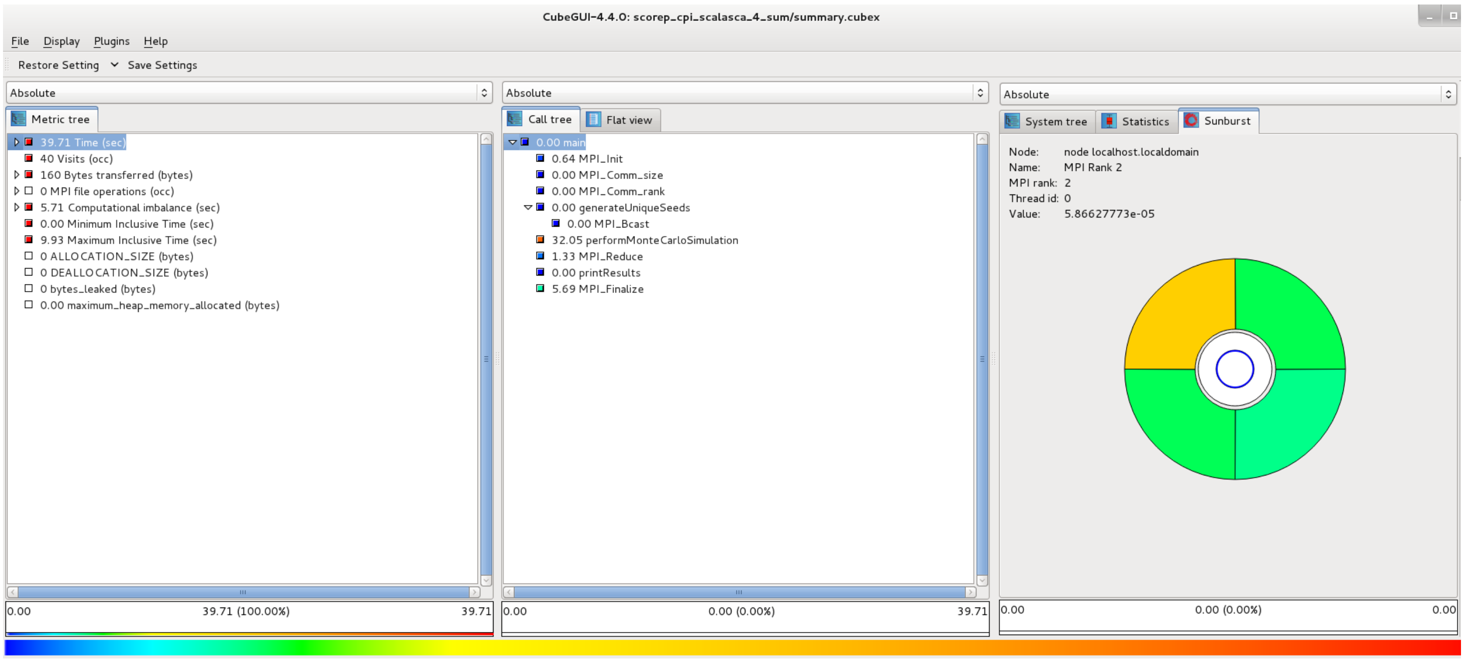
```
19.13 Time (sec)
20 Visits (occ)
48 Bytes transferred (bytes)
0 MPI file operations (occ)
2.51 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
9.57 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
```

```
0.00 main
  0.12 MPI_Init
  0.00 MPI_Comm_size
  0.00 MPI_Comm_rank
  0.00 generateUniqueSeeds
  16.50 performMonteCarloSimulation
  0.00 MPI_Reduce
  0.00 printResults
  2.51 MPI_Finalize
```

```
No. leafs: 2
           (first, second)
Node:      (node localhost.localdomain, node localhost.localdomain)
Name:      machine Linux
MPI rank:  (0, 1)
Thread id: (0, 0)
Value:     1.73138006e-04
```

```
0.00    19.13 (100.00%)    19.13
0.00    0.00 (0.00%)       19.13
0.00    0.00 (100.00%)     0.00
```

For four processes:



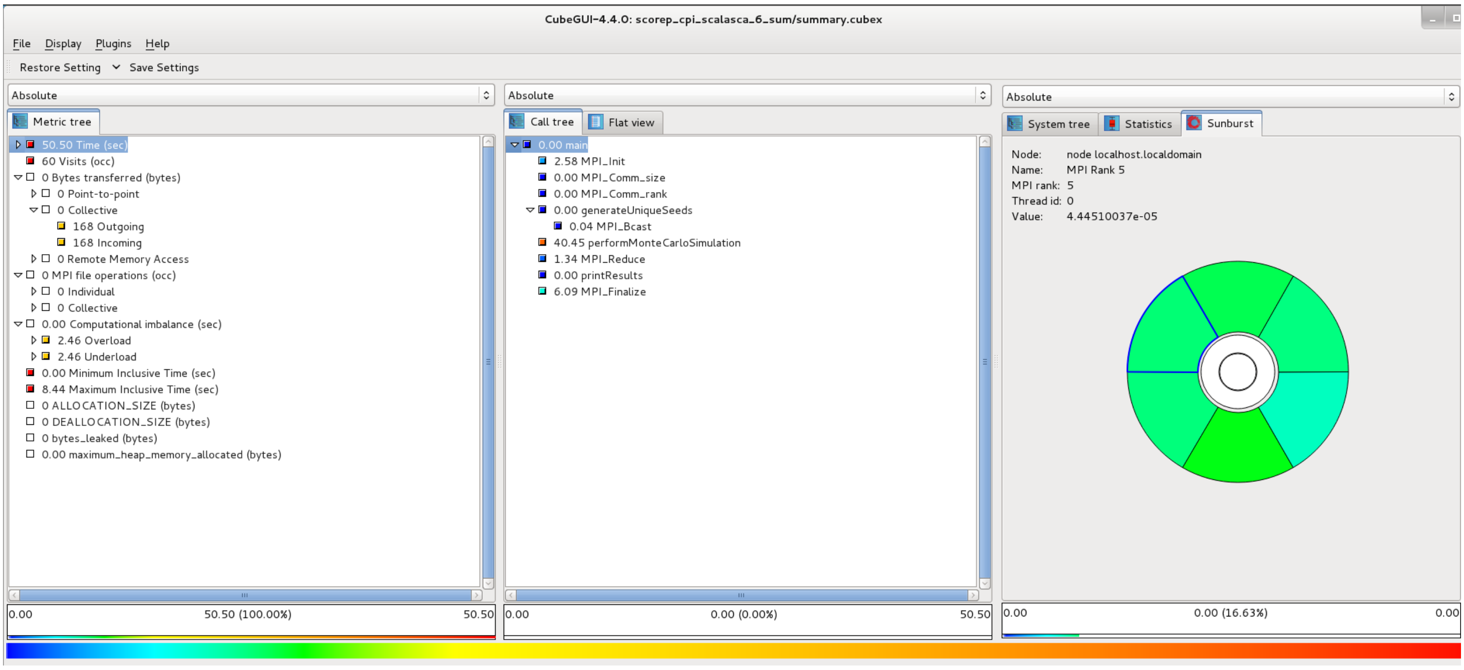
```
39.71 Time (sec)
40 Visits (occ)
160 Bytes transferred (bytes)
0 MPI file operations (occ)
5.71 Computational imbalance (sec)
0.00 Minimum Inclusive Time (sec)
9.93 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
```

```
0.00 main
  0.64 MPI_Init
  0.00 MPI_Comm_size
  0.00 MPI_Comm_rank
  0.00 generateUniqueSeeds
    0.00 MPI_Bcast
  32.05 performMonteCarloSimulation
  1.33 MPI_Reduce
  0.00 printResults
  5.69 MPI_Finalize
```

```
Node:      node localhost.localdomain
Name:      MPI Rank 2
MPI rank:  2
Thread id: 0
Value:     5.86627773e-05
```

```
0.00    39.71 (100.00%)    39.71
0.00    0.00 (0.00%)       39.71
0.00    0.00 (0.00%)       0.00
```

For six processes:



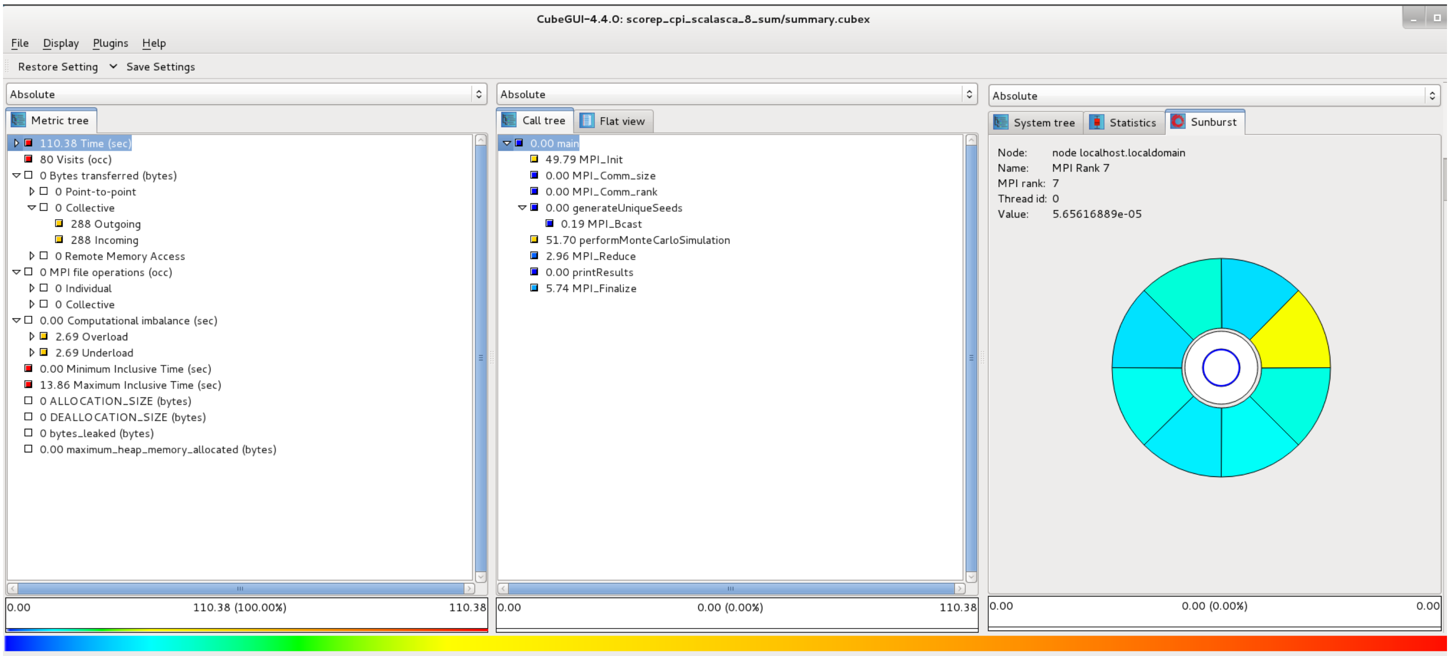
```
50.50 Time (sec)
60 Visits (occ)
0 Bytes transferred (bytes)
  0 Point-to-point
  0 Collective
    168 Outgoing
    168 Incoming
  0 Remote Memory Access
0 MPI file operations (occ)
  0 Individual
  0 Collective
0.00 Computational imbalance (sec)
  2.46 Overload
  2.46 Underload
0.00 Minimum Inclusive Time (sec)
8.44 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
```

```
0.00 main
  2.58 MPI_Init
  0.00 MPI_Comm_size
  0.00 MPI_Comm_rank
  0.00 generateUniqueSeeds
    0.04 MPI_Bcast
  40.45 performMonteCarloSimulation
  1.34 MPI_Reduce
  0.00 printResults
  6.09 MPI_Finalize
```

```
Node:      node localhost.localdomain
Name:      MPI Rank 5
MPI rank:  5
Thread id: 0
Value:     4.44510037e-05
```

```
0.00    50.50 (100.00%)    50.50
0.00    0.00 (0.00%)       50.50
0.00    0.00 (16.63%)      0.00
```

For eight processes:



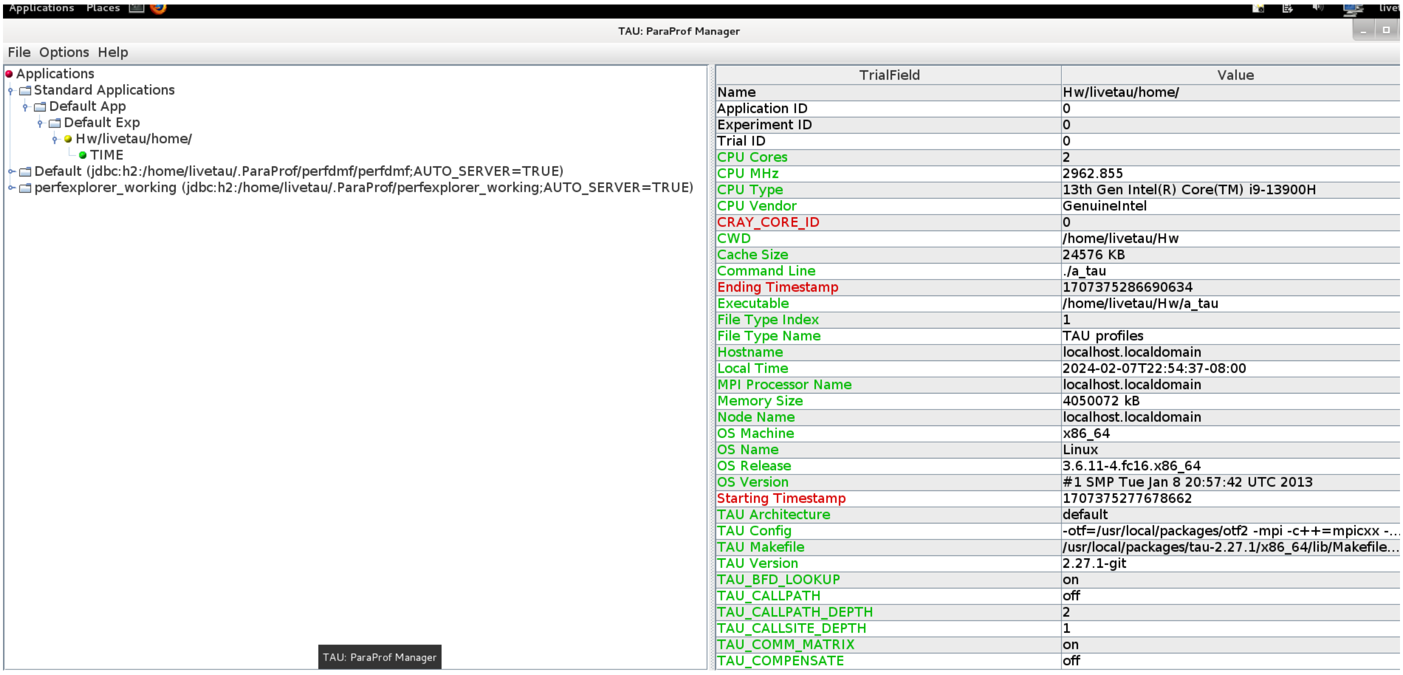
CubeGUI-4.4.0: scorep_cpi_scalasca_8_sum/summary.cubex

```
▷ ■ 110.38 Time (sec)
  ■ 80 Visits (occ)
▽ □ 0 Bytes transferred (bytes)
    ▷ □ 0 Point-to-point
    ▽ □ 0 Collective
          ■ 288 Outgoing
          ■ 288 Incoming
    ▷ □ 0 Remote Memory Access
▽ □ 0 MPI file operations (occ)
    ▷ □ 0 Individual
    ▷ □ 0 Collective
▽ □ 0.00 Computational imbalance (sec)
    ▷ ■ 2.69 Overload
    ▷ ■ 2.69 Underload
  ■ 0.00 Minimum Inclusive Time (sec)
  ■ 13.86 Maximum Inclusive Time (sec)
  □ 0 ALLOCATION_SIZE (bytes)
  □ 0 DEALLOCATION_SIZE (bytes)
  □ 0 bytes_leaked (bytes)
  □ 0.00 maximum_heap_memory_allocated (bytes)
```

```
▽ ■ 0.00 main
      ■ 49.79 MPI_Init
      ■ 0.00 MPI_Comm_size
      ■ 0.00 MPI_Comm_rank
    ▽ ■ 0.00 generateUniqueSeeds
          ■ 0.19 MPI_Bcast
      ■ 51.70 performMonteCarloSimulation
      ■ 2.96 MPI_Reduce
      ■ 0.00 printResults
      ■ 5.74 MPI_Finalize
```

```
Node:      node localhost.localdomain
Name:      MPI Rank 7
MPI rank:  7
Thread id: 0
Value:     5.65616889e-05
```

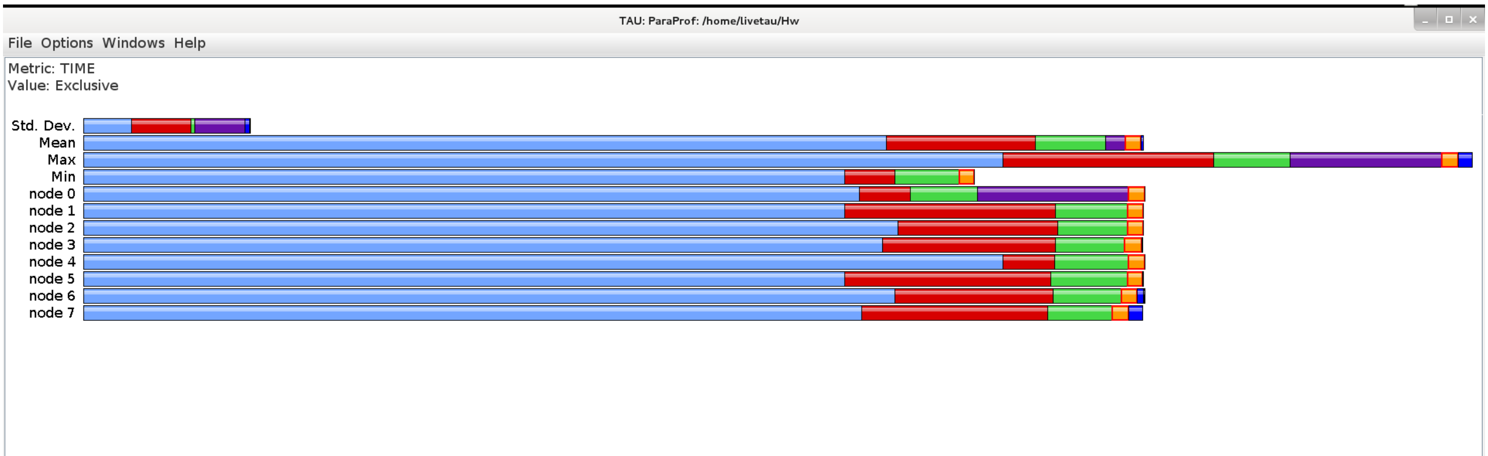
Paraprof :



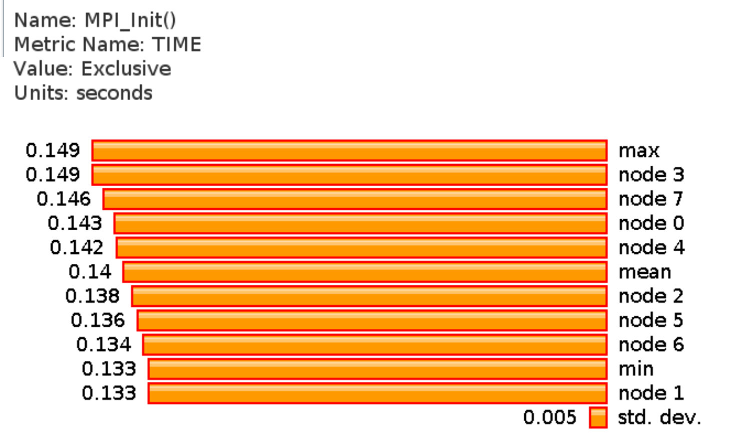
```
● Applications
  ○ Standard Applications
     ○ Default App
        ○ Default Exp
           ○ Hw/livetau/home/
              ● TIME
  ○ Default (jdbc:h2:/home/livetau/.ParaProf/perfdmf/perfdmf;AUTO_SERVER=TRUE)
  ○ perfexplorer_working (jdbc:h2:/home/livetau/.ParaProf/perfexplorer_working;AUTO_SERVER=TRUE)
```

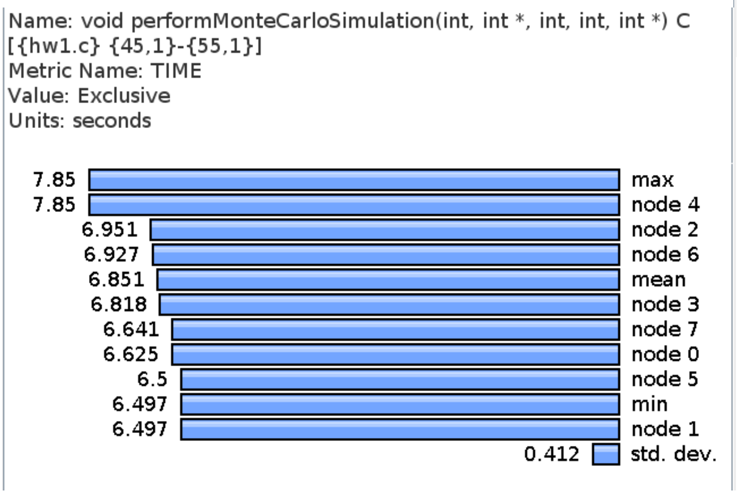
TrialField	Value
Name	Hw/livetau/home/
Application ID	0
Experiment ID	0
Trial ID	0
CPU Cores	2
CPU MHz	2962.855
CPU Type	13th Gen Intel(R) Core(TM) i9-13900H
CPU Vendor	GenuineIntel
CRAY_CORE_ID	0
CWD	/home/livetau/Hw
Cache Size	24576 KB
Command Line	./a_tau
Ending Timestamp	1707375286690634
Executable	/home/livetau/Hw/a_tau
File Type Index	1
File Type Name	TAU profiles
Hostname	localhost.localdomain
Local Time	2024-02-07T22:54:37-08:00
MPI Processor Name	localhost.localdomain
Memory Size	4050072 kB
Node Name	localhost.localdomain
OS Machine	x86_64
OS Name	Linux
OS Release	3.6.11-4.fc16.x86_64
OS Version	#1 SMP Tue Jan 8 20:57:42 UTC 2013
Starting Timestamp	1707375277678662
TAU Architecture	default
TAU Config	-otf=/usr/local/packages/otf2 -mpi -c++=mpicxx -...
TAU Makefile	/usr/local/packages/tau-2.27.1/x86_64/lib/Makefile...
TAU Version	2.27.1-git
TAU_BFD_LOOKUP	on
TAU_CALLPATH	off
TAU_CALLPATH_DEPTH	2
TAU_CALLSITE_DEPTH	1
TAU_COMM_MATRIX	on
TAU_COMPENSATE	off

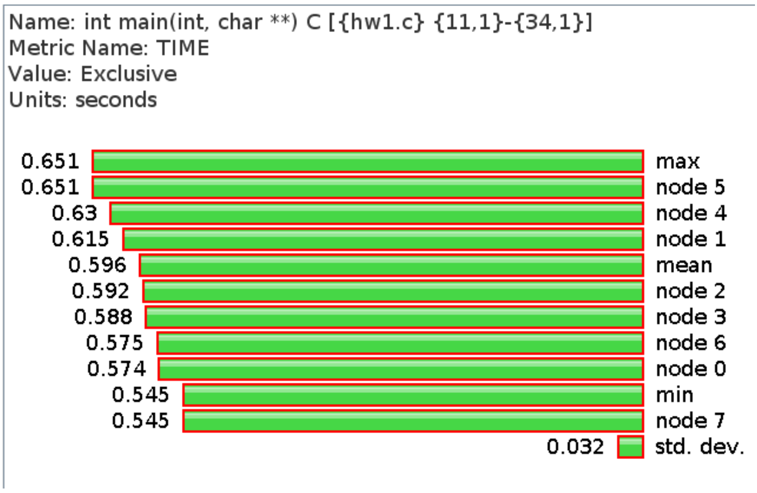
Statistics on the performance of the code in the system according to the number of processes of 8:

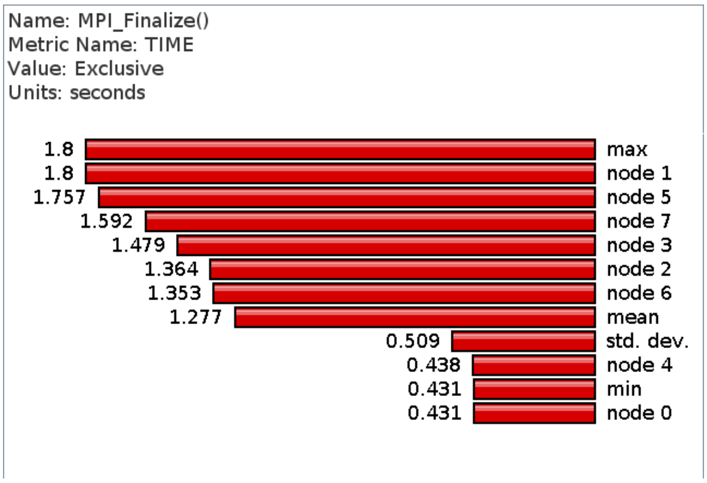


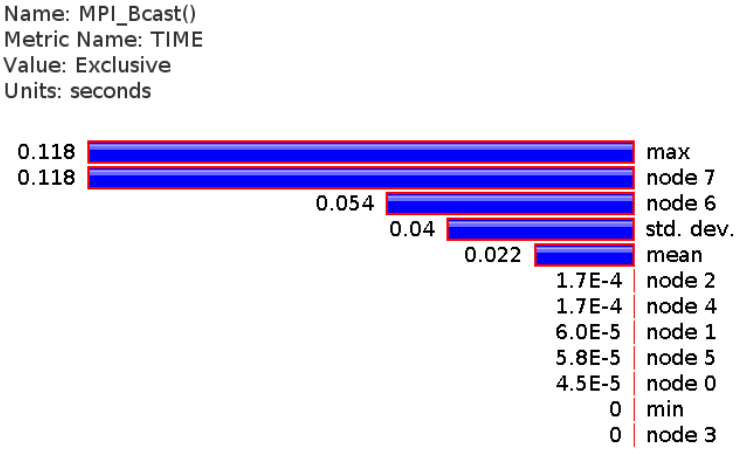
Metric: TIME
Value: Exclusive

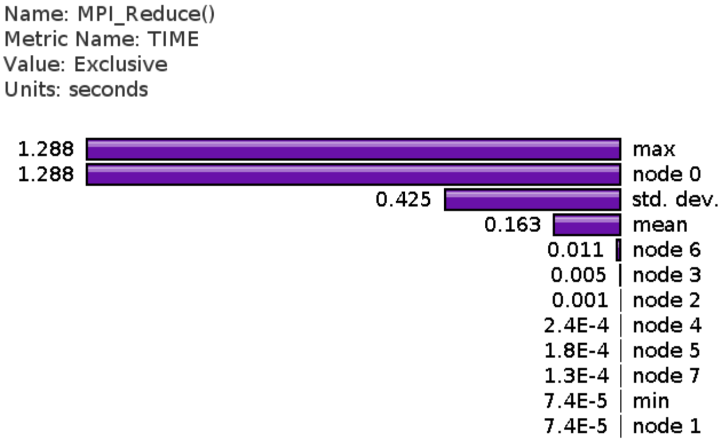

Name: MPI_Init()
Metric Name: TIME
Value: Exclusive
Units: seconds


Name: void performMonteCarloSimulation(int, int *, int, int, int *) C [{hw1.c} {45,1}-{55,1}]
Metric Name: TIME
Value: Exclusive
Units: seconds

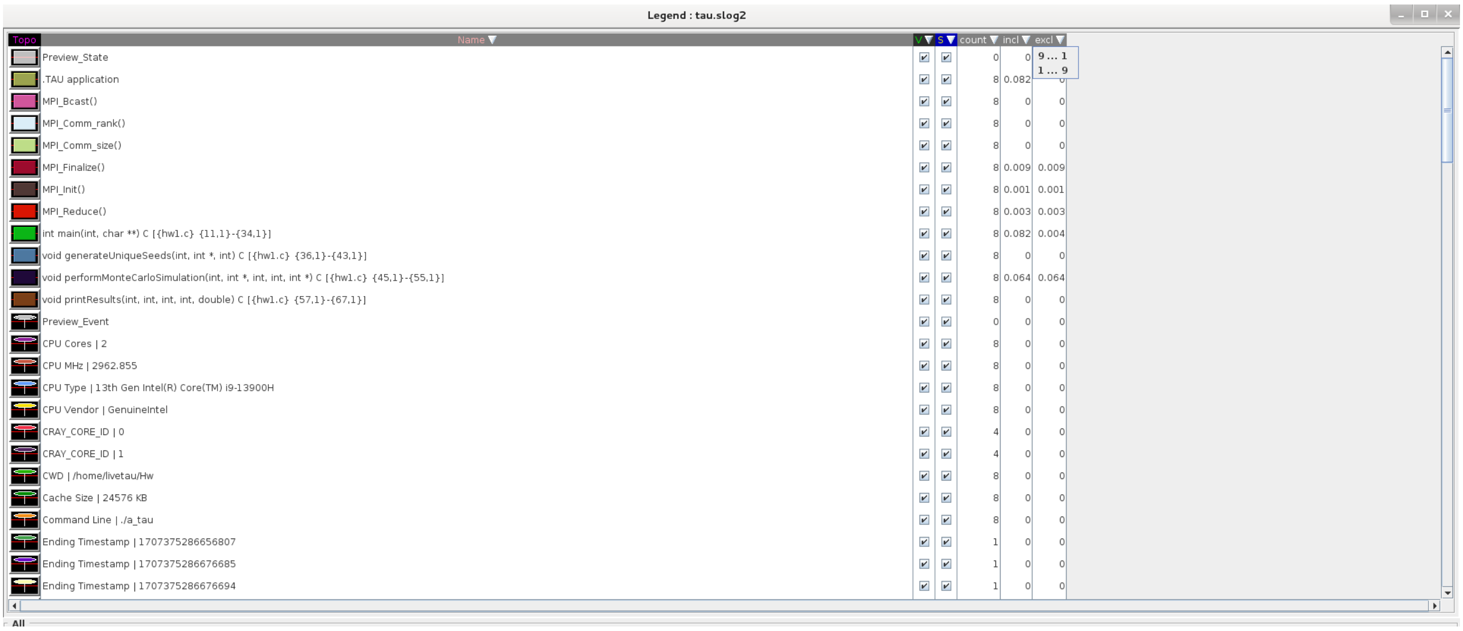

Name: int main(int, char **) C [{hw1.c} {11,1}-{34,1}]
Metric Name: TIME
Value: Exclusive
Units: seconds


Name: MPI_Finalize()
Metric Name: TIME
Value: Exclusive
Units: seconds


Name: MPI_Bcast()
Metric Name: TIME
Value: Exclusive
Units: seconds

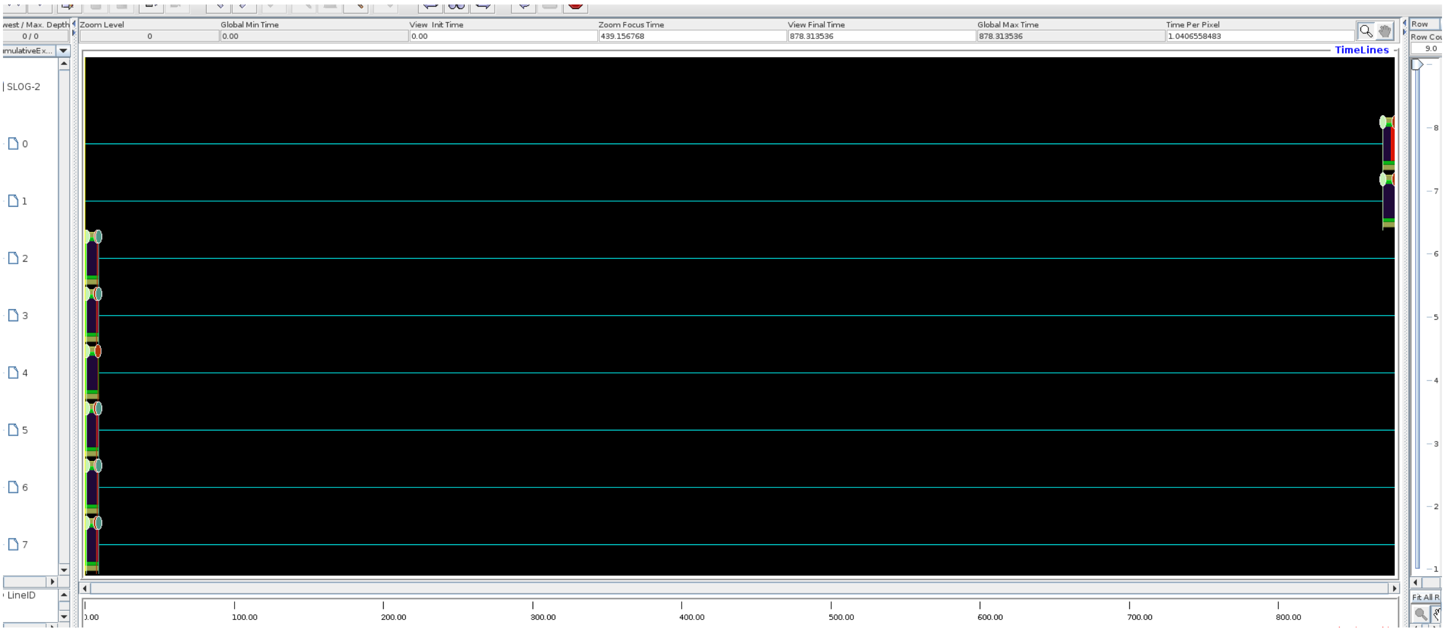

Name: MPI_Reduce()
Metric Name: TIME
Value: Exclusive
Units: seconds

JumpShot:

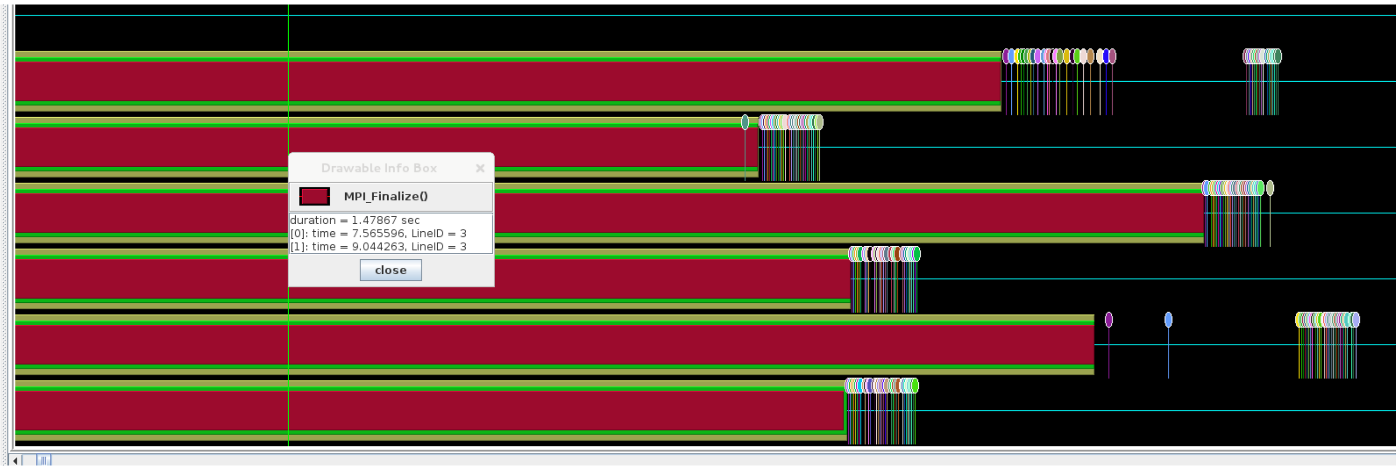


Legend : tau.slog2

Topo	Name	V	S	count	incl	excl
■	Preview_State	☑	☑	0	0	[illegible]
■	.TAU application	☑	☑	8	0.082	0
■	MPI_Bcast()	☑	☑	8	0	0
■	MPI_Comm_rank()	☑	☑	8	0	0
■	MPI_Comm_size()	☑	☑	8	0	0
■	MPI_Finalize()	☑	☑	8	0.009	0.009
■	MPI_Init()	☑	☑	8	0.001	0.001
■	MPI_Reduce()	☑	☑	8	0.003	0.003
■	int main(int, char **) C [{hw1.c} {11,1}-{34,1}]	☑	☑	8	0.082	0.004
■	void generateUniqueSeeds(int, int *, int) C [{hw1.c} {36,1}-{43,1}]	☑	☑	8	0	0
■	void performMonteCarloSimulation(int, int *, int, int, int *) C [{hw1.c} {45,1}-{55,1}]	☑	☑	8	0.064	0.064
■	void printResults(int, int, int, int, double) C [{hw1.c} {57,1}-{67,1}]	☑	☑	8	0	0
●	Preview_Event	☑	☑	0	0	0
●	CPU Cores | 2	☑	☑	8	0	0
●	CPU MHz | 2962.855	☑	☑	8	0	0
●	CPU Type | 13th Gen Intel(R) Core(TM) i9-13900H	☑	☑	8	0	0
●	CPU Vendor | GenuineIntel	☑	☑	8	0	0
●	CRAY_CORE_ID | 0	☑	☑	4	0	0
●	CRAY_CORE_ID | 1	☑	☑	4	0	0
●	CWD | /home/livetau/Hw	☑	☑	8	0	0
●	Cache Size | 24576 KB	☑	☑	8	0	0
●	Command Line | ./a_tau	☑	☑	8	0	0
●	Ending Timestamp | 1707375286656807	☑	☑	1	0	0
●	Ending Timestamp | 1707375286676685	☑	☑	1	0	0
●	Ending Timestamp | 1707375286676694	☑	☑	1	0	0



west / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel
0 / 0	0	0.00	0.00	439.156768	878.313536	878.313536	1.0406558483



Drawable Info Box

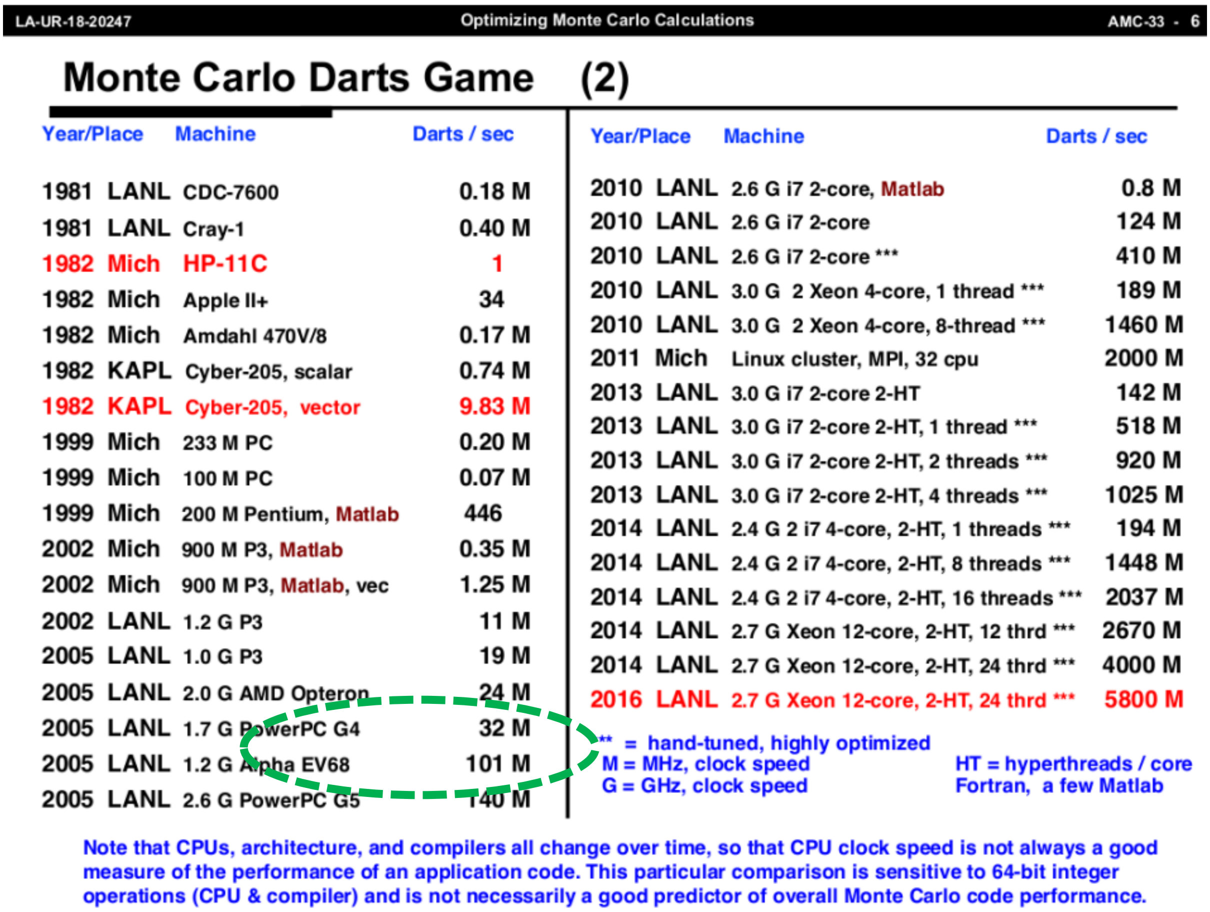
MPI_Finalize()

duration = 1.47867 sec
[0]: time = 7.565596, LineID = 3
[1]: time = 9.044263, LineID = 3

5) Our best execution reached an amount of **45,479,812** darts per second.

Our program uses 900000000 darts attempts. 900000000/19.789 = 45479812.02.

Meaning we stand at about 45.5M darts per second. Comparing to the following table:



LA-UR-18-20247 Optimizing Monte Carlo Calculations AMC-33 - 6

Monte Carlo Darts Game (2)

Year/Place	Machine	Darts / sec
1981 LANL	CDC-7600	0.18 M
1981 LANL	Cray-1	0.40 M
1982 Mich	HP-11C	1
1982 Mich	Apple II+	34
1982 Mich	Amdahl 470V/8	0.17 M
1982 KAPL	Cyber-205, scalar	0.74 M
1982 KAPL	Cyber-205, vector	9.83 M
1999 Mich	233 M PC	0.20 M
1999 Mich	100 M PC	0.07 M
1999 Mich	200 M Pentium, Matlab	446
2002 Mich	900 M P3, Matlab	0.35 M
2002 Mich	900 M P3, Matlab, vec	1.25 M
2002 LANL	1.2 G P3	11 M
2005 LANL	1.0 G P3	19 M
2005 LANL	2.0 G AMD Opteron	24 M
2005 LANL	1.7 G PowerPC G4	32 M
2005 LANL	1.2 G Alpha EV68	101 M
2005 LANL	2.6 G PowerPC G5	140 M

Year/Place	Machine	Darts / sec
2010 LANL	2.6 G i7 2-core, Matlab	0.8 M
2010 LANL	2.6 G i7 2-core	124 M
2010 LANL	2.6 G i7 2-core ***	410 M
2010 LANL	3.0 G 2 Xeon 4-core, 1 thread ***	189 M
2010 LANL	3.0 G 2 Xeon 4-core, 8-thread ***	1460 M
2011 Mich	Linux cluster, MPI, 32 cpu	2000 M
2013 LANL	3.0 G i7 2-core 2-HT	142 M
2013 LANL	3.0 G i7 2-core 2-HT, 1 thread ***	518 M
2013 LANL	3.0 G i7 2-core 2-HT, 2 threads ***	920 M
2013 LANL	3.0 G i7 2-core 2-HT, 4 threads ***	1025 M
2014 LANL	2.4 G 2 i7 4-core, 2-HT, 1 threads ***	194 M
2014 LANL	2.4 G 2 i7 4-core, 2-HT, 8 threads ***	1448 M
2014 LANL	2.4 G 2 i7 4-core, 2-HT, 16 threads ***	2037 M
2014 LANL	2.7 G Xeon 12-core, 2-HT, 12 thrd ***	2670 M
2014 LANL	2.7 G Xeon 12-core, 2-HT, 24 thrd ***	4000 M
2016 LANL	2.7 G Xeon 12-core, 2-HT, 24 thrd ***	5800 M

*** = hand-tuned, highly optimized
M = MHz, clock speed
G = GHz, clock speed

HT = hyperthreads / core
Fortran, a few Matlab

Note that CPUs, architecture, and compilers all change over time, so that CPU clock speed is not always a good measure of the performance of an application code. This particular comparison is sensitive to 64-bit integer operations (CPU & compiler) and is not necessarily a good predictor of overall Monte Carlo code performance.

We fit the table at around the year 2005, where there was a leap from 32M darts per second to 101M darts per second.

6) **Conclusions:**

Looking at these graphs and the overall results we would recommend to the potential user to use this program with the run of 2 processes if his need is time sensitive. That is because the 2 processes run had the best Speed Up and Run Time, and relatively high Efficiency. If the users need is not time sensitive, then we would recommend using 1 process due to its max Efficiency.