Assignment No.:

Title: Implement odd-even merge sort using HPC infrastructure using python/scala/java/c++.

Aim: To perform odd-even merge sort using HPC infrastructure

Objective: To implement concept of odd even merge sort using the standards of parallelism like openMP or MPI.

Theory:

Odd-even Merge Sort –

- The odd-even merge sort algorithm was developed by K.E. Batcher. It is based on a merge Algorithm that merges 2 sorted sequences.
- In contrast to merge sort, this algorithm is not data dependent.

Algorithm:

Merge Sorting

1) Merge Algorithm:

Input: Sequence $a_0 \cdots a_{n-1}$

Output: Sorted sequence

Method: if $n \geq 2$ then

- Apply odd even merge $(n/2)$ recursively to the even subsequence $a_0, a_2, a_4, \cdots a_{n-2}$ & to odd subsequence $a_1, a_3, a_5, \cdots a_{n-1}$
- Comparision for all i.e $\{1, 3, 5, \cdots, n-1\}$ $[a_i : a_{i+1}]$

- Else comparison $[a_{i+1} : a_i]$

Sorting Algorithm:
Input: Sequence $a_0 \cdots a_{n-1}$ (n power of 2)
Output: Sorted sequence
Method: if $n>1$ then
- Apply odd-even merge sort $(n/2)$ to the two halves $a_0 \cdots a_{n/2-1}$ & $a_{n/2} \cdots a_{n-1}$ of the sequence
- Odd-even Merge (n)

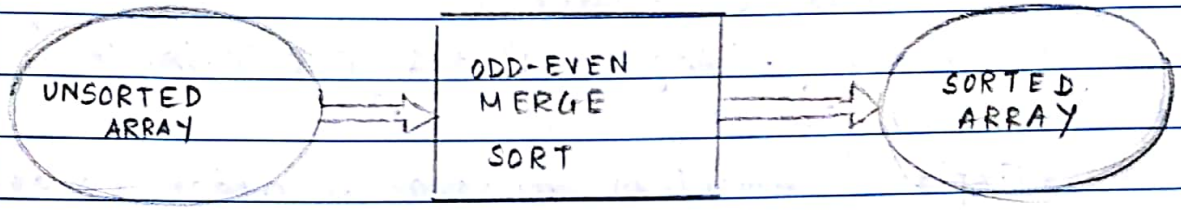
2) Parallelism in odd-even merge sort using OpenMP or MPI.

Using OpenMP: Partition() - This method is used for partitioning odd-even array & then it calls the merge sort for each partition.
Merge() - This method is used for combining 2 sorted arrays that is even odd & form the final single sorted array.
#pragma-omp-parallel_section: Independent section can be parallelized by using this directive.
#pragma-omp-section: It is used to divide the section among the threads.
#pragma-omp-parallel-for: The for loop can be parallelized by using this method.

Input : Unsorted array.

Output: Sorted array.

Mathematical Model:



INPUT PROCESS OUTPUT

S = { I, O, F, Sc, Fc }

I = Input = { Unsorted Array }

O = Output = { Sorted Array }

F = Function = { partition(), comparision(), merge(), sorting(), display() }

Sc = Success Cases = { ① Sorted array as output.
② Parallelism works well }

Fc = Failure Cases = { ① OpenMP flag not used to compile.
② Power failure
③ Array is not sorted as expected. }

Platform : Ubuntu 16.04

Conclusion : Thus, we successfully studied and implemented odd-even merge sort using HPC.

*. FAQ's :-

1) Explain complexity of odd-even merge sort approach.

Ans - Let T(n) be the number of comparisons performed by odd-even merge(n). Then we have for n>2

$T(n) = 2 \cdot T(n/2) + n/2 - 1$

with $T(2) = 1$, we have

$T(n) = n/2 \cdot (\log(n) - 1) + 1 \quad (= O(n \cdot \log(n))$

2) Explain concept of recursion in odd-even merge sort approach.

Ans - By recursive application of the merge algorithm, the sorting algorithm of odd-even merge sort is formed

Algorithm odd even merge sort (n)

Input: sequence, $a_0, a_1 \dots a_{n-1}$

Output: sorted sequence

Method:

if $n > 1$ then

(i) Apply odd even merge sort $(n/2)$ recursively to the two halves $a_0, \dots a_{n/2-1}$ & $a_{n/2} \dots a_{n-1}$ of sequence

(ii) odd even merge (n)

Hence, Number of comparisions of odd-even merge sort is $O(n \log(n)^2)$.

Therefore, recursion helps to keep the track of previous context when we keep on dividing the problem into sub-problems.

3) What is the difference between odd-even mergesort & mergesort?

Ans— Sorting algorithm mergesort produces a sorted sequence by sorting its 2 halves & merging them. It has a time complexity of O(nlogn). Mergesort is optimal.

In contrast to mergesort, this algorithm is not data dependent. The same comparisions are performed regardless of the actual data. Therefore, odd-even mergesort can be implemented as a sorting network. It is not asymptotically optimal, however it allows a distributed implementation.

Assignment No. :

Title: Write a program to check task distribution using gprof.

Aim: To study features of profiler such as gprof & analyze task distribution.

Objective: To analyze task distribution using gprof profiler.

Theory:

Features of gprof :-

1) gprof time profiler gives detail time statistics for each sub-routine.
2) It creates relative graph for all sub-routines.
3) It analyzes the program bottleneck.
4) It requires recompilation of the code. Compiler options & libraries provide wrappers for each routine call and provides periodic sampling of programming.

Gprof :-

Functioning of Gprof :-

Gprof will change executable of your program (this is called instrumenting the code) to store some book-keeping information e.g. how many times a function is called.

The statistical profiling bit comes from snooping the program counter regularly to get a sample of what your code is doing.

Gprof does both, it instruments the code & collects samples from looking at the program counter.

Profiler outputs:

Flat profile:

- time your program spent in each function.
- how many times that function is called.
- information on which function burn most of the cycles is clearly indicated here.

Call graph:

- shows for each function :
  - which functions called it
  - which other functions it called
  - how many times it was called.
- an estimate of how much time was spent in the subroutines of each function.
- suggests places where you might try to eliminate function calls that use a lot of time.

Key points -

- Before you can profile your program, you must first recompile it specifically for profiling
- To do so, add the -pg argument to the compiler's commandline.
- This creates an executable called abc from the source abc.cpp with debugging & profiling turned on.

- Once your program has been compiled with profiling turned on, running the program to completion causes a file named gmon.out to be created in the current directory.
- gprof works by analyzing data collected during the execution of your program after your program has finished running.
- gmon.out holds this data in a gprof-readable format.

Information related to gmon.out :-

- If gmon.out already exists, it will be overwritten
- The program must exit normally. Pressing ctrl+c or killing program is not a normal exit.
- Since you are trying to analyze your program in a real-world situation, you should run program exactly the same way as you normally would (same inputs, command line arguments, etc).
- Run gprof as follows —

$ gprof program-name [data-file] [> output-file]

- (If you don't specify the name of data-file then gmon.out is assumed. Following gprof command with "> output-file" causes the output of gprof to be saved to output-file so that you can examine it later).

Input: Provide the file (binary.cpp) to gprof for analysis.

Output: Detailed analysis in the form of flat profile & call graph.

Steps of profiling :- (Phases)

1) Compiling for profiling
2) Creating gmon.out
3) Running gprof
4) Analyzing gprof's output
5) Interpreting the flat profile.
6) Interpreting the call graph.

1) Compiling for profiling -

$ g++ -pg -o prog prog.cpp

2) Creating gmon.out

Run the program (prog) to completion. A gmon.out file is created. Gprof ~~consists~~ works by analyzing data collected during execution of your program, gmon.out holds this data in gprof-readable data.

3) Running gprof

$ gprof prog [data-file] [> output-file]

gmon.out is the default data-file

4) The output file can be analysed with a text editor. Two kinds of analysis are performed: flat profile & call graph.

5) Interpreting the flat profile

Flat profile shows the total amount of time your program spent executing each function. At the end of the profile, legend describes columns.

6) Interpreting the call graph

Call graph shows how much time was spent in each function & its children. From this information, you can find functions that, while they themselves may not have used much time, called other functions that did use unusual amounts of time.

Platform: Ubuntu 16.04

Conclusion: Task distribution is carried out successfully using gprof.

*. FAQ's:

1) Why are profiles used ?

Ans— Profiling is an important aspect of software programming. Through profiling, one can determine the parts in program code that are time-consuming & need to be rewritten (by spotting bugs). This helps make your program execution faster, which is desired in HPC.

2) What is the significance of gmon.out file ?

Ans— Once program has been compiled with profiling turned on, running program to completion causes a file named gmon.out. gprof works by analyzing data collected during the execution of your program after your program has finished execution. gmon.out holds this data in a gprof-readable format.
- If gmon.out already exits, it is overwritten.
- Program must exit normally.
- The program has to run in real world scenario. (same inputs, command line arguments, etc).

3) What are limitations of gprof ?

Ans— Gprof gives no indication of parts of your program that are limited by I/O or swapping bandwidth. Samples of the program counter are taken at fixed intervals of run time the time measurements in gprof output say nothing about time that your program was not running.

Title: Implement OBST tree search using HPC task subdivision. Merge the results to get the final result.

Aim: Implement OBST tree search using HPC task subdivision (Open MP). Merge the results to get the final result.

Objective: To understand the concept of OBST.

OBST-Theory -

An optimal binary search tree is a binary search tree for which nodes are arranged on levels such that the tree cost is minimum. For the purpose of better presentation of binary search tree, we will consider 'extended binary search trees' which have the keys stored at their internal nodes. Suppose n-keys '$k_1, k_2 \cdots k_n$' are stored at internal nodes of a binary search tree. It is assumed that the keys are given in sorted order so that $k_1 < k_2 < k_3 \cdots < k_n$. An extended binary search tree is obtained from binary search tree by adding successor nodes to each of its terminal nodes as indicated in the following figure

$$c[i,j] = \min_{i<k\leq j} \{ c(i,k-1) + c(k,j) + p(k) + w(i,k-1) + w(k,j) \}$$

$$c[i,j] = \min_{i<k\leq j} \{ c(i,k-1) + c(k,j) \} + w(i,j)$$

Advantage –

The major advantage of binary search trees over other data structures is that the related sorting algorithms & search algorithms such as inorder traversal can be very efficient, they are also easy to code.

Disadvantage –

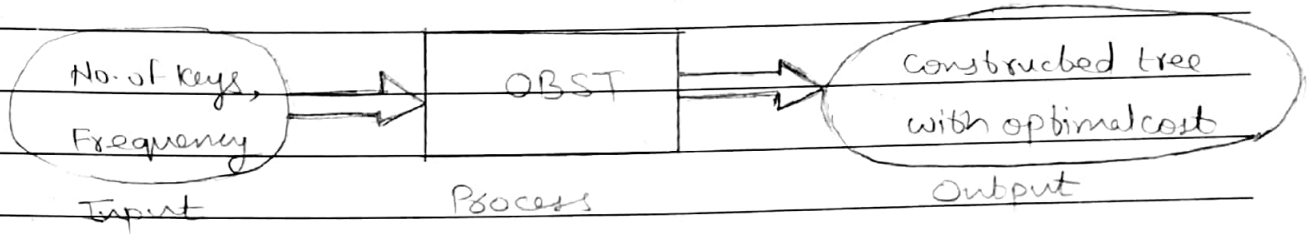
1) The shape of the binary search tree totally depends on the order of insertions & deletions, & can become degenerate.
2) When inserting or searching for an element in binary search tree, the key of each visited node has to be compared with the key of the element to be inserted or found.
3) The keys in the binary search tree may be long & run time may increase.
4) After a long intermixed sequence of random insertion & deletion, the expected height of the tree approaches square root of number of key, $\sqrt{n}$ which grows much faster than $\log n$.

Analysis :- The OBST has a time complexity of $O(n^3)$. Its space efficiency is only $O(n^2)$. It can possibly be lowered to complexity of $O(n^2)$ with smaller recursive functions & smaller range of values.

Input: Elements to be added in the tree.

Output: OBST with minimum cost.

Mathematical Model:



Let S be the system such that-

S = { I, O, Fn, Sc, Fc }

I = Input = { No. of keys, Frequency }

O = Output = { Tree with optimal cost }

Fn = Function = { Function performing OBST construction }

Sc = Success Cases = { (i) Optimal tree is constructed.
(ii) Desired output is obtained. }

Fc = Failure Cases = { (i) Element not inserted correctly.
(ii) Tree is not optimal. }

Platform:- Ubuntu 16.04

Conclusion:- Thus, we successfully implemented an OBST using HPC task subdivision.

* FAQ's :-

1) Difference between OBST & BST?

Ans- An OBST, some times called weight balanced binary tree is a BST which provides the smallest possible search time for a given sequence of access. OBSTs are generally divided into 2 types: static & dynamic. In static optimality, the tree cannot be modified after it has been constructed. In dynamic optimality, the tree can be modified by permitting tree rotations.

2) What is the complexity of OBST?

Ans- The OBST has a time complexity of $O(n^3)$. Its space efficiency is $O(n^2)$. It can possibly be lowered to complexity of $O(n^2)$.

3) What is the point in OBST where parallelism can be implemented?

Ans- Binary search trees can be constructed parallely for each permutation of the given sequence. Cost of each BST can be calculated concurrently.

Title: Using Divide & Conquer strategies design for concurrent Quick sort using C++.

Aim: Design a class using divide & conquer strategies for concurrent quick sort using C++.

Objective: To identify scope & parallelism [OpenMP] in quicksort algorithm & to enhance the performance using HPC.

Theory:

Quick Sort Complexity Analysis -

1) Worst-case running time:

When quicksort always has the most unbalanced on time for some constant c, the recursive call on n-1 elements takes c(n-1) time, the recursive call on n-2 element takes c(n-2) time, & so on.

- In big Oh Notation quicksort's worst case running time is $O(n^2)$.

2) Best-case running time:

Quicksort's best case occurs when the partitions are as evenly balanced as possible: their sizes either are equally. Using Big Oh notation, we get the same result as for merge sort $O(n \log n)$.

3) Average-case running time:

It is also $O(n \log n)$.

* Open MP-

Open MP is an API that supports Multiplatfoom shared memory multiprocessing programming in C, C++ & fortran.

Open MP uses a portable, scalable model that gives programmer a simple & flexible interface for developing parallel application for platform ranging from the standard desktop computer to the supercomputer.

An application built with the hybrid model of parallel programming run on a computer cluster using both OpenMP & MPI.

	Function	Description
1.	omp_set_num_threads	No. of threads used for parallelism.
2.	omp_get_num_threads	No. of threads in parallel execution
3.	omp_get_thread_num	Return thread numbers
4.	omp_get_wtime	Returns elapsed wall-clock
5.	open_in_parallel	Return non-zero if it called in parallel region
6.	omp_get_num_proc	Return no. of processor that could be assigned to program

Input: Array of Integer.

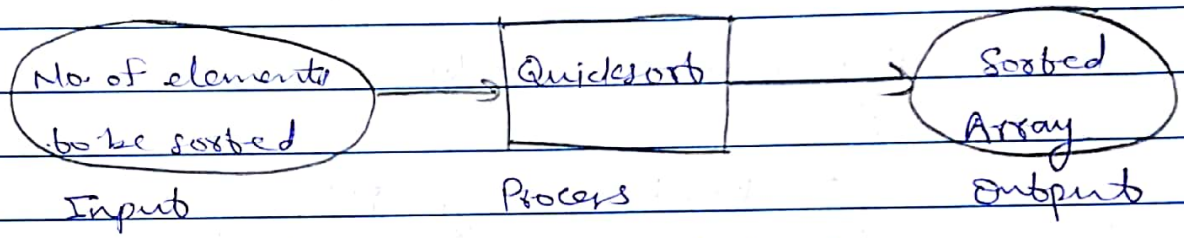
Output: Sorted Array.

* Algorithm:

```
Quicksort( array, low, high) {
If low < high
   J = partition (array, low, high)
   # pragma omp parallel sections
   {
      # pragma omp section
      {
         Quicksort ( array, low, J-1)
      }
      # pragma omp section
      {
         Quicksort( array, J+1, high )
      }
}
Partition (array, low, high) {
   key = arr[low];
   I = low+1;
   J = high;
while ( I < high && key >= arr[I])
I++;
while (key < arr[J])
   J--;
if (I<J)
   temp=arr[I];
   arr[I];
else
```

```
else
 arr[j] = temp;
 temp = arr[low];
 arr[low] = arr[j];
 arr[j] = temp;
return(j);
}
```

* Mathematical Model :-



Let S be the system such that

S = { I, O, Fn, Sc, Fc }

I = Input set = { No. of elements to be sorted }

O = Output set = { elements of quick sort (sorted) }

Fn = Function set = { sorting operation }

Sc = Success set = { sorted elements }

Fc = Failure set = { sorting fails }

Platform : Linux Ubuntu 12.04.

Conclusion : Concurrent Quicksort is implemented in C++ using divide & conquer strategies.

* FAQ's —

1. How pivot element is selected? How does it affect the complexity?

Ans— The pivot element can be the first, last, middle element or can be chosen randomly. We should avoid choosing the first or last element if the array is sorted because it will take O(n²) time (worst case). So to avoid worst case sorting, the pivot should be middle or randomly chosen.

2. What is the use of OpenMP?

Ans— (i) OpenMP is a specification for a set of compiler directives, library routines, & environment variables that can be used to specify high-level parallelism in Fortran & C/C++ programs.

(ii) It is simple, no need to deal with message passing as in MPI.

(iii) Unified code for both serial & parallel applications.

(iv) Used for both coarse-grained & fine grained parallelism.

(v) In SIMD mode of computation, OpenMP is used as it can have big performance advantage over MPI.

(vi) Used on various accelerators such as GPGPU.

3. What is meant by a multithreaded application?

Ans — (i) Multithreading is an execution model allows multiple threads to exist within the context of single process.

(ii) Threading allows an application to remain responsive without the use of a ~~match~~ catch all application loop. when doing lengthy operations.

(iii) Multithreaded application can perform multiple functions in parallel by use of multiprocessor systems.

(iv) Such applications can perform large number of operations in significantly less amount of time compared to serial applications.

* FAQ's:

1) List the important features of MPI (with all MPI function standards).

Ans— (i) General:
- Communicators combine context & group for message security.
- Thread safety.

(ii) Point-to-point communication:
- Structured buffers & derived data types, heterogeneity.
- Modes: normal (blocking & non-blocking), synchronous, ready (to allow access to fast protocol), buffered.

(iii) Collective:
- Both built-in & user-defined collective operations.
- Large number of data movement routines.
- Sub-groups defined directly or by topology.

* MPI function standards—

a) MPI_Init :- Initiates the MPI execution environment.
Syntax: int MPI_Init (int *argc, char ***argv);

b) MPI_Comm_size : Determines number of processes in group.
Syntax: int MPI_Comm_size (MPI_Comm comm, int *size);

c) MPI_Comm_rank : Determines rank of process in a communicator.
Syntax: int MPI_Comm_rank (MPI_Comm comm, int *rank);

d) MPI_Send : It performs basic send operation
Syntax: MPI_Send(void *buffer, int count,
MPI_Datatype dtype, int dest, int tag, MPI_Comm comm);

e) MPI_Recv : Performs basic receive operation
Syntax: int MPI_Recv(void *buff, int count,
MPI_Datatype dtype, int src, int tag, MPI_Comm comm,
MPI_Status *status);

2) How does binary search tree work for incorporated parallelism & Explain BST complexity.

Ans- (i) In a BST, all the elements to the left of the root are smaller than the root element & to right of it are greater.

(ii) If a single BST is to be constructed, then incorporating parallelism ~~int~~ in it will be different.

(iii) If multiple BST's are to be constructed, different elements as root each time parallely can be incorporated here to construct the left-right subtrees parallely.

(iv) Searching of multiple elements in the same & can be done parallely.

(v) Average time complexity of BST is O(log n).
Worst case time complexity of BST is O(n)

3) Explain compilation & execution of BST using a single & multiple machine.

Ans- (i) Single machine: mpicc bst.c
mpiexec -np 4 ./a.out

(ii) multiple machines: mpicc bst.c
mpiexec -np 16 -f file.txt ./a.out

Consider there are 4 nodes in cluster with each node having 4 cores.
-f flag is used to specify the machine file. The machine file has node names & corresponding cores of machine that can be used.

* FAQ's:

1) Describe any two ways to calculate pi.

Ans- Method 1: Calculating pi using an infinite series.

a) Mathematicians have found Gregory-Leibniz series to calculate the value of pi to a great number of decimal places.

b) $\pi = \left(\frac{4}{1}\right) - \left(\frac{4}{3}\right) + \left(\frac{4}{5}\right) - \left(\frac{4}{7}\right) + \left(\frac{4}{9}\right) - \cdots$

c) Though not very efficient, it will get closer to pi with every iteration, accurately producing pi to 10 correct decimal places using direct summation of the series requires about five billion terms

d) We also have Nilkanta series:

$$\pi = 3 + \frac{4}{2*3*4} - \frac{4}{4*5*6} + \frac{4}{6*7*8} - \cdots$$

Method 2: Arcsine function (Sine Inverse function

a) Pick any number x in the range $[-1, 1]$

b) Pi = 2 * (Arcsine (sqrt $(1-x^2)$) + abs (Arcsine (x)))

c) The range bound on x is because, $\sin^{-1}$ is undefined for values that are outside of range $[-1, 1]$.

2) What is difference between OpenMP & MPI?

Ans-

OpenMP	MPI
(i) Its a way to program on shared memory devices.	(i) Its a way to program on distributed memory devices.
(ii) Parallelism occurs where every parallel thread has access to all data.	(ii) Parallelism occurs where every parallel process is working in its memory space in isolation from others.
(iii) Eg.: Splitting 'for' loop amongst different threads.	(iii) Eg.: Each process tackles an independent subproblem of a global problem based on the process ID (rank).

3) What is MPI? What are the mandatory routines for PI assignment?

Ans- (a) MPI is a language independent communication protocol used to program parallel computers.

(b) Supports both point-to-point & collective communication.

(c) It is a model standard for communication among processes running a parallel program on distributed ~~among~~ memory system.

(d) MPI has the advantage on NUMA architectures over shared space models.

(e) MPI implementations consists of set of routines callable from FORTRAN, C or C++ & from any language capable of interfacing with such routines libraries.

The routines used for Pi calculation of MPI are :-
(i) MPI_Wtime (ii) MPI_Bcast (iii) MPI_Reduce.

graph

* FAQ's:

1) What is the difference of machine file flag in MPICH2 & OpenMP?

Ans- An application built using hybrid model of parallel programming can run on a computer cluster using both OpenMP & MPI such that OpenMP is used for parallelism within a node while MPI is used for parallelism between nodes.
Hence, machine file is used only in MPI with -f flag.

2) What is scalability in a cluster?

Ans- Scalability in a clustering means that as you increase the number of data objects, the time to perform clustering should roughly scale to the order of complexity of the algorithm.
- Clustering of computers is based on the concepts of modular growth. To scale a cluster from hundreds of uniprocessor nodes is a non-trivial task. The scalability could be limited by multicore chip technology, cluster topology packaging method, Power consumption memory wall, i/o bottlenecks, latency tolerance, etc. The purpose is to achieve scalable performance constrained by the factors.

3) What is Amdahl's law?

Ans— (i) Amdahl's law can be formulated in following way:

$$\text{Slatency}(S) = \frac{1}{1-P+P/s}$$

where,

- Slatency is the theoretical speedup of the execution of the whole task.
- S is the speed up of the part of the task that benefits from improved system resources.
- P is the proportion of execution time that the part benefitting from improved resources originally occupied.

Furthermore,

$$\text{Slatency}(S) \leq \frac{1}{1-P}$$

$$\lim_{S \to \infty} \text{Slatency}(S) = \frac{1}{1-P}$$

(ii) It gives potential speedup on parallel computing system.

(iii) This law applies only to the cases where the problem size is fixed.

booth

* FAQ's:

1) What is the use of Booth's multiplication algorithm?

Ans- (i) Booth multiplication algorithm multiplies two signed binary numbers in 2's complement notation.

(ii) Booth algorithm takes advantage of the fact that the time taken for multiplication depends on the number of 1's in the ~~multiplication~~ multiplier.

(iii) Booth's algorithm is used in ALU of computer to calculate multiplication of signed & unsigned numbers in binary ~~form~~.

(iv) This actually tells how computer internally calculates signed number multiplication.

2) ~~Explain~~ Explain MPI_COMM_WORLD & MPI_Comm_rank.

Ans- MPI_COMM_WORLD is a global communicator group.

- MPI_COMM_WORLD is a default communicator through ~~with~~ which every process can communicate with every other.

- MPI_Comm_rank is a routine that gives (returns) ID (rank) of a particular process that ~~is~~ executes this routine.

- The rank is returned or stored in the parameter passed to the routine.

- The processes are ordered from 0 to n-1, where n is no. of processes in a communicator group.

Syntax : MPI_Comm_rank(MPI_Comm comm, int *rank);

3) Distinguish between Booth's & modified Booth's.

Ans—

Booth's	Modified Booth's
a) Multiplication algorithm that multiplies 2 signed binary numbers in 2's complement notation.	a) Multiplication algorithm having certain advantages over the old version of Booth's.
b) No partial product stage.	b) Includes partial product stage called as – Recording. – Reducing the partial product in two rows. – addition (that gives final product).
c) Every column is shifted & added with 0/1.	c) Every second column is taken & multiplied by 0/1 or +2 or -1 or -2.
d) Partial product can not be reduced further.	d) Partial product is reduced to its half.

* FAQ's:

1) What MPI routines are used to implement strassen's approach?

Ans- MPI routines used to implement strassen's approach:
(i) MPI_Init
(ii) MPI_Comm_rank
(iii) MPI_Comm_size
(iv) MPI_Bcast
(v) MPI_Barrier
(vi) MPI_Send
(vii) MPI_Recv
(viii) MPI_Finalize

2) What is the complexity of strassen's approach?

Ans- Time complexity of strassen's approach is $O(n^{2.80})$

3) Explain the difference between strassen's matrix multiplication & normal matrix multiplication.

Ans → Strassen's Matrix Multiplication	Normal Matrix Multiplication
(i) Time Complexity = $O(n^{2.80})$	(i) Time Complexity = $O(n^3)$
(ii) It involves 7 multiplications & 18 additions.	(ii) It involves 8 multiplications & 4 addition
(iii) It requires additional temporary memory storage.	(iii) It doesn't require additional temporary storage.

MAEER's MIT

Assignment No.:

* Title: Develop a stack sampling using threads using VTune Amplifier.

* Aim: Develop a stack sampling using threads using VTune Amplifier.

* Objective: (i) To understand concept of VTune Amplifier.
(ii) To effectively use multi-core or distributed, concurrent/parallel environments.

* Theory:

VTune Amplifier features:

1) Use memory efficiently: Tune data structures & NUMA.
2) Optimize for high speed storage; I/O & compute imbalance.
3) Intel media ~~SDK~~ SDK integration: Meaningful media stack metrics.
4) Low overhead Java, python & go: Managed + native code
5) Containers: Docker, Mesos, LXC
6) Basic profiling: Hotspots.
7) Threading Analysis:-
Concurrency, locks & waits
OpenMP, Intel ~~Thread~~ Threading building blocks.
8) Microarchitecture analysis: cache, branch prediction
9) Vectorization + Intel Advisor: FLOPS estimates.
10) MPI + Intel Trace Analyzer & Collector: Scalability, imbalance, overhead.

> Procedure for stack sampling analysis:

- Sampling is a statistical profiling method that shows you the functions that are doing most of the user mode work in the application.
- Sampling is a good place to start to look for areas to speed up your application.
- At specified intervals, the sampling method collects information about the functions that are executing in your application.
- After you finish a profiling run, the summary view of the profiling data shows shows the most active function call tree, called the Hot Path, where the most of the work in the application was performed.
- View also lists the functions that were performing the most individual work, & provides a timeline graph you can use to focus on specific segments of the sampling session.

> Configure Analysis:

- Analysis types needs to be selected to configure the analysis setting.
- Switch to Analysis Type tab.
- From left pane, select an analysis type from the list like Basic Hotspots, Advanced Hotspots, Concurrency, locks & waits, etc and the select & set the proper parameters to begin analysis by clicking on the start button on the right.

> Thread Profiling:

- The number of active threads corresponds to the concurrency level of an application.
- By comparing the concurrency level with the number of processors, VTune Amplifier classifies how an application utilizes the processors in the system.
- It defines default utilization ranges depending on the number of processor cores & displays the thread concurrency in the Summary & Bottom-up window. You can change the utilization thresholds by dragging the slider in the Summary window.
- Thread concurrency may be higher than CPU usage if threads are in the runnable state & not consuming CPU time.
- VTune Amplifier defines the Target concurrency level for your application by default to the number of physical cores.

* Input: Give any multithreaded, parallel files (binary.cpp) to VTune Amplifier for Sampling.

* Output: VTune Amplifier provides basic hotspot analysis of program (Summary, Bottom-up analysis, top-bottom tree), give detailed information of program analysis.

* Steps to install & operate VTune Amplifier :
- Extract the archive to a directory.
- Within the directory, run sudo ./install.sh (root permission are required)
- Press Enter to continue.
- Keep pressing space to scroll to the end of the license agreement, then enter the string 'accept'. Enter.
- Press 2, then enter, to select evaluation
- Press 2, then enter, to not participate in intel's program.
- Press enter to continue, wait for it to install.
- Press enter to continue.
- Note down the post installation instructions. There should be a line saying 'bash users: source /opt/intel/vtune_amplifier_xe_2016.0.444464/amplxe-vars.sh'
- Version number may be different. Copy the string from the source
- Type in source ~/.bashrc.
- Open ~/.bashrc in gedit or any text edibor. Paste the above line at the end. There should be no newline between 'source' & version, it should be one line.
- Run echo 0 | sudo tee /proc/sys/kernel/yama/ptrace_scope.
- Run amplxe-gui. VTune Amplifier should start.

- Select new project. Enter any project name.
- There should be two tabs in right panel.
- Analysis Target & Analysis Type.
- In analysis Target, in Application, browse for the output file to your program (a.out).
- In analysis Type, select Advanced Hotspots. Hit start.
- Your program will execute. In case it requires input look at the terminal from which you started VTune & enter it there. After the program executes, you will get a detailed report.

Platform: Ubuntu 16.04

Conclusion: Stack sampling was carried out successfully using VTune Amplifier.

* FAQ's:

1) Why VTune Amplifier is used?

Ans- VTune Amplifier performance profiler is a commercial application for software performance analysis of 32 & 64-bit x86 based machines. It assists in various kinds of code profiling including stack sampling, thread profiling & hardware event sampling. Tool can be used to analyze thread & storage performance. The profiler result consists of details such as time spent in each subroutine which can be drilled down to the instruction level.

2) What are types of sampling?

Ans- Software Sampling: Works on x86 compatible processors & gives both the locations where time is spent & call stack is used.
Hardware-event sampling: This uses the on chip performance monitoring unit. It can find specific tuning opportunities like cache misses & branch mispredictions.

3) How performance of a parallel program can be improvised using VTune Amplifier?

Ans→ Each analysis type provides a set of performance metrics that helps you sort out the problems in your code & understand how

to optimize it.

- Compute intensive application analysis branch introduces analysis types based on application that are compute sensitive HPC performance characterization evaluates compute-sensitive or throughput applications for floating point operation & memory efficiency.
- The platform analysis branch introduces analysis types monitoring CPU, GPU & power usage of application program.
- Algorithm analysis we use the collected data to understand the inefficiencies in your current algorithms & improve application performance.