RV COLLEGE OF ENGINEERING®

(Autonomous Institution Affiliated to VTU, Belagavi)

Department of Computer Science & Engineering

Bengaluru – 560 059

Laboratory Certificate

This is to certify that Mr./Ms. _____ has satisfactorily completed the course of experiments in Practical _____ prescribed by the department of _____ during the year _____.

Name of the Candidate: _____

USN No: _____ Semester: _____

Marks	
Maximum	**Obtained**

Signature of the Staff in-charge **Head of the Department**

Date:

Vision and Mission of the Department of Computer Science and Engineering

Vision

To achieve leadership in the field of Computer Science & Engineering by strengthening fundamentals and facilitating interdisciplinary sustainable research to meet the ever growing needs of the society.

Mission

- To evolve continually as a centre of excellence in quality education in Computers and allied fields.
- To develop state of the art infrastructure and create environment capable for interdisciplinary research and skill enhancement.
- To collaborate with industries and institutions at national and international levels to enhance research in emerging areas.
- To develop professionals having social concern to become leaders in top-notch industries and/or become entrepreneurs with good ethics.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's):

PEO1: Develop Graduates capable of applying the principles of mathematics, science, core engineering and Computer Science to solve real-world problems in interdisciplinary domains.

PEO2: To develop the ability among graduates to analyze and understand current pedagogical techniques, industry accepted computing practices and state-of-art technology.

PEO3: To develop graduates who will exhibit cultural awareness, teamwork with professional ethics, effective communication skills and appropriately apply knowledge of societal impacts of computing technology.

PEO4: To prepare graduates with a capability to successfully get employed in the right role/become entrepreneurs to achieve higher career goals or takeup higher education in pursuit of lifelong learning.

PROGRAM OUTCOMES:

Engineering Graduates will be able to:

PO1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

PSO1: System Analysis and Design

The student will:

1. Recognize and understand the dynamic nature of developments in computer architecture, data organization and analytical methods.
2. Learn the applicability of various systems software elements for solving real-world design problems.
3. Identify the various analysis & design methodologies for facilitating development of high quality system software products with focus on performance optimization.
4. Display good team participation, communication, project management and document skills.

PSO2: Product Development

The student will:

1. Demonstrate knowledge of the ability to write programs and integrate them resulting in state-of –art hardware/software products in the domains of embedded systems, databases /data analytics, network/web systems and mobile products.
2. Participate in teams for planning and implementing solutions to cater to business – specific requirements displaying good team dynamics and professional ethics.
3. Employ state-of-art methodologies for product development and testing / validation with focus on optimization and quality related aspects

Course Outcomes

CO1. Explore the fundamentals of high performance computing concepts.

CO2. Analyze the performance of parallel programming.
CO3. Design parallel computing constructs for different applications.
CO4. Demonstrate Parallel computing concepts for suitable applications.

INDEX

Sl. No	Contents	COs.	Date of submission	Marks / Remarks	Signature
	Working examples on OpenMP Directives, Environment variables and runtime libraries.	1,2,3,4			
1	a)Write an OpenMp program that computes the value of PI using Monto-Carlo Algorithm. b)Write a MPI program that computes the value of PI using Monto-Carlo Algorithm.	1,2,4			
2	Write an OpenMP program that computes a simple matrix-matrix multiplication using dynamic memory allocation. a) Illustrate the correctness of the program. b) Justify the inference when outer "for" loop is parallelized with and without using the explicit data scope variables.	1,2,4			
3	A) Write an OpenMP program for Cache unfriendly sieve of Eratosthenes and Cache friendly Sieve of Eratosthenes for enumerating prime numbers upto N and prove the correctness. B)Write an OpenMP program for Cache unfriendly sieve of Eratosthenes and Cache friendly and parallel Sieve of Eratosthenes for enumerating prime numbers upto N and prove the correctness.	1,2,3,4			
4	Write an OpenMP program to convert a color image to black and white image. a) Demonstrate the performance of different scheduling techniques for varying chunk values b) Analyze the scheduling patterns by assigning a single color value for an image for each thread	1,2,3,4			
5	Write a MPI program that has a total of 4 processes. Process with rank 1, 2 and 3 should send the following messages respectively to the process with rank 0: HELLO, CSE, RVCE	2,3,4			
6	Write an OpenMP program for Word search in a file and illustrate the performance using different sizes of file.	1,2,3			
7	SAXPY (Single precision real Alpha X plus Y): Write an OpenCL program to compute A = alpha*B + C, where alpha is a constant and A, B, and C are vectors of an arbitrary size n.				

Part B

The students are encouraged to implement additional Innovative Experiments (IE) or small project in the lab in CUDA/OpenCL. It will be evaluated for 20 marks.

Parallel Architecture and Distributed Computing

Objective:

1. Review the evolution of parallelism in Computer Architecture.
2. Understand the basic ideas of vector processing, multiprocessing and GPU computing.
3. Focus on performance of different processor architectures
4. Review the advantages of thread level parallelism
5. Exposure to basics of parallel Programming using OpenMP, MPI and OpenACC.

Recommended Systems/Software Requirements:

- Quad core processor
- Linux Operating system with GNU C Compiler (gcc) package installed.

Prerequisites:

Programming in C,Data structures using C, Computer organization, Computer Architecture, Operating System, Microprocessor.

Course Outcomes Vs Program Outcome												
	PO1	PO 2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO1 0	PO1 1	PO1 2
CO1	**-**	**-**	**1**	**1**	**1**	**-**	**-**	**-**	**-**	**-**	**-**	**1**
CO2	**-**	**2**	**2**	**2**	**2**	**-**	**-**	**-**	**1**	**1**	**-**	**2**
CO3	**2**	**2**	**2**	**2**	**2**	**1**	**-**	**1**	**1**	**1**	**-**	**2**
CO4	**2**	**2**	**2**	**2**	**2**	**1**	**1**	**1**	**1**	**1**	**-**	**2**
Course Vs Program Outcome												

COURSE	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
	2	2	3	3	2	1	2	1	2	2	2	3

Do's and Don'ts in the Laboratory

Do's…….

1. ComePreparedtothelab.
2. Usethecomputers and other hardware foracademicpurposesonly.
3. Followthelabexercisecyclesasinstructedbythedepartment.
4. Keepthechairsbacktoitspositionbeforeyouleave.
5. Keepyourlabclean.

Don'ts…….

1. Do not handle any equipment before reading the Instructions/Instruction manuals.
2. Read carefully the power ratings of the equipment before it is switched on whether ratings 230 V/50Hz.
3. Strictly observe the instructions given by the Teacher/Lab Instructor.

Parallel Architecture and Distributed Computing Lab Rubrics (16CS71)

Lab Write-up and Execution rubrics (Max: 6 marks)

		Excellent	Good	Poor
a.	**Understanding of problem and parallel program writing** **(2 Marks)**	Writing an efficient parallel program for the given sequential program along with appropriate data scoping (2)	Writing an efficient parallel program for the given sequential program without data scoping/ with inappropriate data scoping (< 2 to >=1)	Not able to write parallel Program (0)
b.	**Execution** **(2 Marks)**	Demonstrate • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (2)	Demonstrating any one • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (< 2 to >=1)	No Execution(0)
c.	**Result tabulation and Graph** **(2 Marks)**	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results and inference. (2)	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results without inference (< 2 to >=1)	No documentation. (0)

Viva rubrics (Max: 4 marks)

		Excellent	Good	Poor
a.	**Inference Justification(1)**	Completely justify the obtained inference.(1)	Incomplete Justification for the inference(0.5)	Unable to explain the inferences. (0)
b.	**Understanding of Parallel construct** **(2 Marks)**	Explains the usage of all parallel programming constructs. (2)	Adequately explains the usage of all parallel programming constructs (< 2 to >=1)	Unable to explain concepts. (0)
c.	**Application of parallel constructs (1)**	Identifying an appropriate parallel construct for given problem (1)		Did not solve problem.(0)

Schedule of experiments

Sl. No	Name of Experiment	No. Of labs
	Working examples on OpenMP Directives, Environment variables and runtime libraries.	01
1	a)Write an OpenMp program that computes the value of PI using Monto-Carlo Algorithm. b)Write a MPI program that computes the value of PI using Monto-Carlo Algorithm.	01
2	Write an OpenMP program that computes a simple matrix-matrix multiplication using dynamic memory allocation. a) Illustrate the correctness of the program. b) Justify the inference when outer "for" loop is parallelized with and without using the explicit data scope variables.	
3	A) Write an OpenMP program for Cache unfriendly sieve of Eratosthenes and Cache friendly Sieve of Eratosthenes for enumerating prime numbers upto N and prove the correctness. B)Write an OpenMP program for Cache unfriendly sieve of Eratosthenes and Cache friendly and parallel Sieve of Eratosthenes for enumerating prime numbers upto N and prove the correctness.	01
4	Write an OpenMP program to convert a color image to black and white image. a) Demonstrate the performance of different scheduling techniques for varying chunk values b) Analyze the scheduling patterns by assigning a single color value for an image for each thread	01
5	Write an OpenMP parallel program for Points Classification. Prove the correctness of sequential program with that of parallel.	01
6	Write an OpenMP program for Word search in a file and illustrate the performance using different sizes of file.	01
7		

Part B

Student has to develop in groups or individually a project on real world problem using CUDA

Introduction to OpenMP

Applications with adequate single-processor performance on high-end systems often enjoy a significant cost advantage when implemented in parallel on systems utilizing multiple microprocessors. The language extension to implement single processor programs on multi processors system is OpenMP, designed to make it very easy to write multithreaded code. OpenMP is a set of compiler directives to express shared memory parallelism. These directives have been defined for Fortran, C, and C++. In addition to directives, OpenMP also includes a small set of runtime library routines and environment variables. The language extensions called Application Programming Interface (API) in OpenMP fall into one of three categories Compiler Directives, Library routines and Environment Variables.

Compiler Directives:

- Parallel Construct
- Data Environment
- Work Sharing
- Synchronization

PARALLEL Region Construct

- A parallel region is a block of code that will be executed by multiple threads
- Main thread creates a team of threads and becomes the master of the team. The master is a member of that team and has thread id 0 within that team
- Starting from the beginning of this parallel region, the code is duplicated and all threads will execute that code.
- There is an implied barrier at the end of a parallel section.
- If any thread terminates within a parallel region, all threads in the team terminate.

The number of threads in a parallel region is determined by the following factors, in order of precedence:

- Evaluation of the IF clause
- Setting of the NUM_THREADS clause
- Use of the omp_set_num_threads() library function
- Setting of the OMP_NUM_THREADS environment variable
- Implementation default 11 usually the number of CPUs on a node, though it could be dynamic.

Data Scoping Attribute Clauses:

The OpenMP Data Scope Attribute Clauses are used to explicitly define how variables should be scoped. They explicitly define how variables should be scoped. They include:

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- SHARED
- DEFAULT
- REDUCTION
- COPYIN

Data Scope Attribute Clauses are used in conjunction with several directives (PARALLEL, DO/for, and SECTIONS) to control the scoping of enclosed variables.

- **private clause-private(list)**
  This declares variables in its list to be private to each thread
- **firstprivate clause- firstprivate(list)**
  A private initialized copy of B is created before the parallel region begins. The copy of each thread gets the same value
- **SHARED Clause-shared(list)**
  A shared variable exists in only one memory location and all threads can read or write to that address
- **DEFAULT Clause- default (shared | none)**
  Specify default scope for all variables in the lexical extent.
- **LASTPRIVATE Clause- lastprivate (list)**
  Value from the last loop iteration assigned the original variable object.
- **COPYIN Clause-- copyin (list)**
  Initialized with value from master thread. Used for threadprivate variables
- **COPYPRIVATE Clause - copyprivate (list)**
  Used to broadcast values of single thread to all instances of the private variables. Associated with the SINGLE directive.

REDUCTION Clause- reduction (operator: list)

Variables which needed to be shared & modified by all the processors.The number of threads in a parallel region is determined by the following factors, in order of precedence:

- Evaluation of the IF clause
- Setting of the NUM_THREADS clause
- Use of the omp_set_num_threads() library function
- Setting of the OMP_NUM_THREADS environment variable
- Implementation default 13 usually the number of CPUs on a node, though it could be dynamic

WorkSharing Constructs: Divides execution of code region among members of the team. Worksharing constructs do not launch new threads.

Types of worksharing

- **FOR** data parallelism
- **SECTIONS** functional parallelism
- **SINGLE** serializes a section of code

FOR Directive

FOR directive specifies that the iterations of the loop immediately following it must be executed in parallel by the team.

Clauses:

SCHEDULE: Describes how iterations of the loop are divided among the threads in the team. The default schedule is implementation dependent.

STATIC:

DYNAMIC:

GUIDED:

RUNTIME: Determined by an environment variable OMP_SCHEDULE.

NO WAIT / no wait: Threads do not synchronize at the end of the parallel loop.

ORDERED: Specifies that the iterations of the loop must be executed as they would be in a serial program

a) **SECTION:** Each SECTION is executed once by a thread in the team
Clauses:NOWAIT: implied barrier exists at the end of a SECTIONS directive, unless this clause is used.

b) **Single Directive:** The enclosed code is to be executed by only one thread in the team. May be useful when dealing with sections of code that are not thread safe (such as I/O)

c) **Combined Parallel WorkSharing Constructs:** These directives behave identical to individual parallel directives types
    - Parallel for
    - parallel sections

Synchronization Constructs

- **MASTER** Directive
- **CRITICAL** Directive
- **BARRIER** Directive
- **ATOMIC** Directive
- **FLUSH** Directive
- **ORDERED** Directive

- **MASTER Directive**

Executed only by master thread of the team. All other threads on the team skip this section of code. There is no implied barrier associated with this directive.

- **CRITICAL Directive**

The CRITICAL directive specifies a region of code that must be executed by only one thread at a time.

- **BARRIERDirective**

On reaching BARRIER directive, a thread will wait at that point until all other threads have reached that barrier.All threads then resume executing in parallel the code that follows the barrier.

- **ATOMIC** Directive

Specifies that a specific memory location must be updated atomically.Avoids simultaneous update from many threads.

- **FLUSH Directive**

Identifies a synchronization point at which the implementation must provide a consistent view of memory.

- **ORDERED Directive**

Must appear within for or parallel for directive. Only 1 thread at a time is allowed into an ordered section. The thread executes the iterations in the same order as the iterations are executed in sequential.

Runtime Library routines:

Execution environment routines that can be used to control and to query the parallel execution environment. Lock routines that can be used to synchronize access to data

- OMP_SET_NUM_THREADS[void omp_set_num_threads(intnum_threads)]:
- omp_set_num_threads routine affects the number of threads to be used for subsequent parallel regions
- OMP_GET_NUM_THREADS[(intomp_get_num_threads(void)] : Returns the number of threads in the current team.
- OMP_GET_THREAD_NUM [intomp_get_thread_num()]: Returns the thread ID of the thread
- OMP_GET_NUM_PROCS [intomp_get_num_procs()] : To get the number of processors
- OMP_IN_PARALLEL [intomp_in_parallel()]: Determine if the section of code which is executing is parallel or not.
- OMP_SET_DYNAMIC[void omp_set_dynamic(intval)]: Enables or disables dynamic adjustment (by the run time system) of the number of threads available for execution of parallel regions.
- OMP_GET_DYNAMIC[intomp_get_dynamic()] :Determine if dynamic thread adjustment is enabled or not.
- OMP_SET_NESTED[void omp_set_nested(int nested)]: Used to enable or disable nested parallelism
- OMP_GET_NESTED [intomp_get_nested ()]

Environment Variables:

- OMP_SCHEDULE
  - setenv OMP_SCHEDULE "guided"
  - setenv OMP_SCHEDULE "dynamic"
- OMP_NUM_THREADS
  - setenv OMP_NUM_THREADS 8
- OMP_DYNAMIC
  - setenv OMP_DYNAMIC TRUE
- OMP_NESTED
  - setenv OMP_NESTED TRUE

An Interface Specification:

M P I: Message Passing Interface

MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be.

MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.

Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be:

- Practical
- Portable
- Efficient
- Flexible

The MPI standard has gone through a number of revisions, with the most recent version being MPI-3.x. Interface specifications have been defined for C and Fortran90 language bindings: C++ bindings from MPI-1 are removed in MPI-3. MPI-3 also provides support for Fortran 2003 and 2008 features

Actual MPI library implementations differ in which version and features of the MPI standard they support. Developers/users will need to be aware of this.

Header File:

#include<mpi.h>

MPI Program Compilation:

$ mpicc hello2.c -o hello2

MPI Program **Execution:**

$ **mpirun -np 4 hello2**

Format of MPI Calls:

- C names are case sensitive; Fortran names are not.
- Programs must not declare variables or functions with names beginning with the prefix MPI_ or PMPI_ (profiling interface).

C Binding	
Format:	`rc = MPI_Xxxxx(parameter, ... )`
Example:	`rc = MPI_Bsend(&buf,count,type,dest,tag,comm)`
Error code:	Returned as "rc". MPI_SUCCESS if successful

Communicators and Groups:

- MPI uses objects called communicators and groups to define which collection of processes may communicate with each other.
- Most MPI routines require you to specify a communicator as an argument.
- Communicators and groups will be covered in more detail later. For now, simply use **MPI_COMM_WORLD** whenever a communicator is required - it is the predefined communicator that includes all of your MPI processes.

Rank:

- Within a communicator, every process has its own unique, integer identifier assigned by the system when the process initializes. A rank is sometimes also called a "task ID". Ranks are contiguous and begin at zero.
- Used by the programmer to specify the source and destination of messages. Often used conditionally by the application to control program execution (if rank=0 do this / if rank=1 do that).

Error Handling:

- Most MPI routines include a return/error code parameter, as described in the "Format of MPI Calls" section above.
- However, according to the MPI standard, the default behavior of an MPI call is to abort if there is an error. This means you will probably not be able to capture a return/error code other than MPI_SUCCESS (zero).
- The standard does provide a means to override this default error handler. You can also consult the error handling section of the relevant MPI Standard documentation located at http://www.mpi-forum.org/docs/.
- The types of errors displayed to the user are implementation dependent.

MPI Routines:

1. MPI_Init

Initializes the MPI execution environment. This function must be called in every MPI program, must be called before any other MPI functions and must be called only once in an MPI program. For C programs, MPI_Init may be used to pass the command line arguments to all processes, although this is not required by the standard and is implementation dependent.

MPI_Init (&argc,&argv)
MPI_INIT (ierr)

2. MPI_Comm_size

Returns the total number of MPI processes in the specified communicator, such as MPI_COMM_WORLD. If the communicator is MPI_COMM_WORLD, then it represents the number of MPI tasks available to your application.

MPI_Comm_size (comm,&size)
MPI_COMM_SIZE (comm,size,ierr)

3. MPI_Comm_rank

Returns the rank of the calling MPI process within the specified communicator. Initially, each process will be assigned a unique integer rank between 0 and number of tasks - 1 within the communicator MPI_COMM_WORLD. This rank is often referred to as a task ID. If a process becomes associated with other communicators, it will have a unique rank within each of these as well.
MPI_Comm_rank (comm,&rank)
MPI_COMM_RANK (comm,rank,ierr)

4. MPI_Get_processor_name

Returns the processor name. Also returns the length of the name. The buffer for "name" must be at least MPI_MAX_PROCESSOR_NAME characters in size. What is returned into "name" is implementation dependent - may not be the same as the output of the "hostname" or "host" shell commands.

MPI_Get_processor_name(&name,&resultlength)
MPI_GET_PROCESSOR_NAME (name,resultlength,ierr)

5. MPI_Get_version

Returns the version and subversion of the MPI standard that's implemented by the library.

MPI_Get_version(&version,&subversion)
MPI_GET_VERSION (version,subversion,ierr)

6. MPI_Wtime

Returns an elapsed wall clock time in seconds (double precision) on the calling processor.

MPI_Wtime ()
MPI_WTIME ()

7. MPI_Finalize

Terminates the MPI execution environment. This function should be the last MPI routine called in every MPI program - no other MPI routines may be called after it.

MPI_Finalize ()
MPI_FINALIZE (ierr)

8. MPI_Send

Basic blocking send operation. Routine returns only after the application buffer in the sending task is free for reuse. Note that this routine may be implemented differently on different systems. The MPI standard permits the use of a system buffer but does not require it. Some implementations may actually use a synchronous send (discussed below) to implement the basic blocking send.

MPI_Send(&buf,count,datatype,dest,tag,comm)
MPI_SEND(buf,count,datatype,dest,tag,comm,ierr)
MPI_Recv

Receive a message and block until the requested data is available in the application buffer in the receiving task.

MPI_Recv(&buf,count,datatype,source,tag,comm,&status)
MPI_RECV (buf,count,datatype,source,tag,comm,status,ierr)

9. MPI_Bcast

Data movement operation.Broadcasts (sends) a message from the process with rank "root" to all other processes in the group.

MPI_Bcast (&buffer,count,datatype,root,comm)

MPI_BCAST (buffer,count,datatype,root,comm,ierr)

10. MPI_Scatter

Data movement operation.Distributes distinct messages from a single source task to each task in the group.

MPI_Scatter (&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,root,comm)

MPI_SCATTER (sendbuf,sendcnt,sendtype,recvbuf, recvcnt,recvtype,root,comm,ierr)

11. MPI_Gather

Data movement operation.Gathers distinct messages from each task in the group to a single destination task. This routine is the reverse operation of MPI_Scatter.

MPI_Gather (&sendbuf,sendcnt,sendtype,&recvbuf, recvcount,recvtype,root,comm)

MPI_GATHER (sendbuf,sendcnt,sendtype,recvbuf, recvcount,recvtype,root,comm,ierr)

12. MPI_Reduce

Collective computation operation. Applies a reduction operation on all tasks in the group and places the result in one task.

MPI_Reduce (&sendbuf,&recvbuf,count,datatype,op,root,comm)

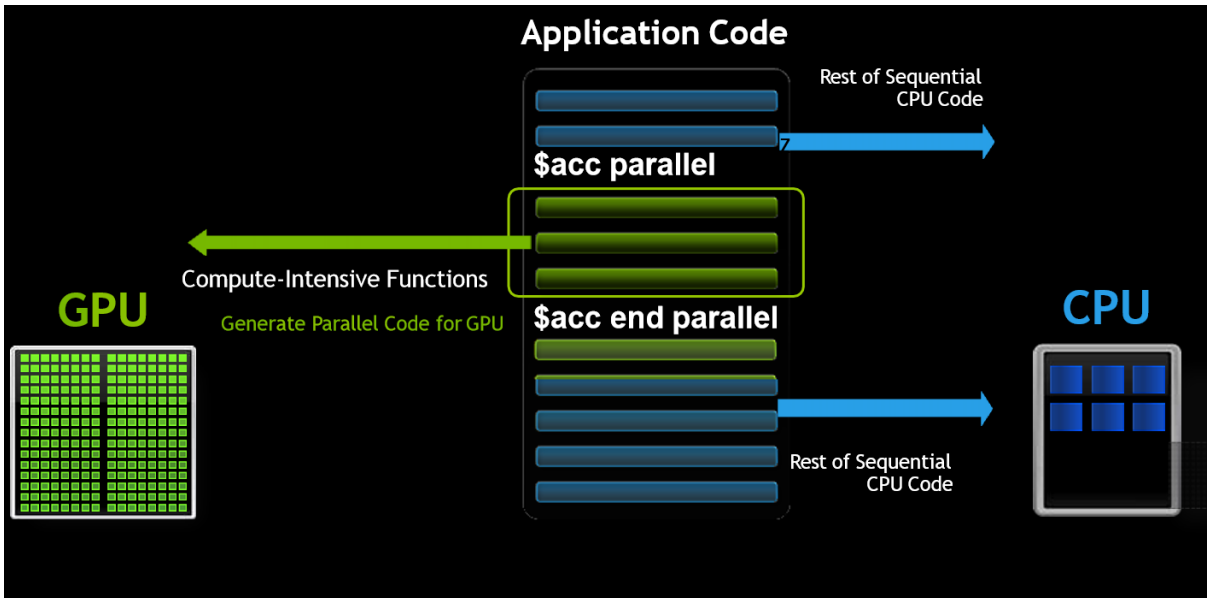
MPI_REDUCE (sendbuf,recvbuf,count,datatype,op,root,comm,ierr)

OpenAcc:

What are OpenACC Directives?

Designed for multicore CPUs & many core GPUs. Portable compiler hints Compiler to parallelizes code.

OpenACC Execution Model:



Guidelines for recording the outputs:

- Tabulate the results by varying input size and number of threads

Input size	Execution Time- Number of Threads						
	1	2	3	4	8	16	32
1000							
10000							
100000							
1000000							

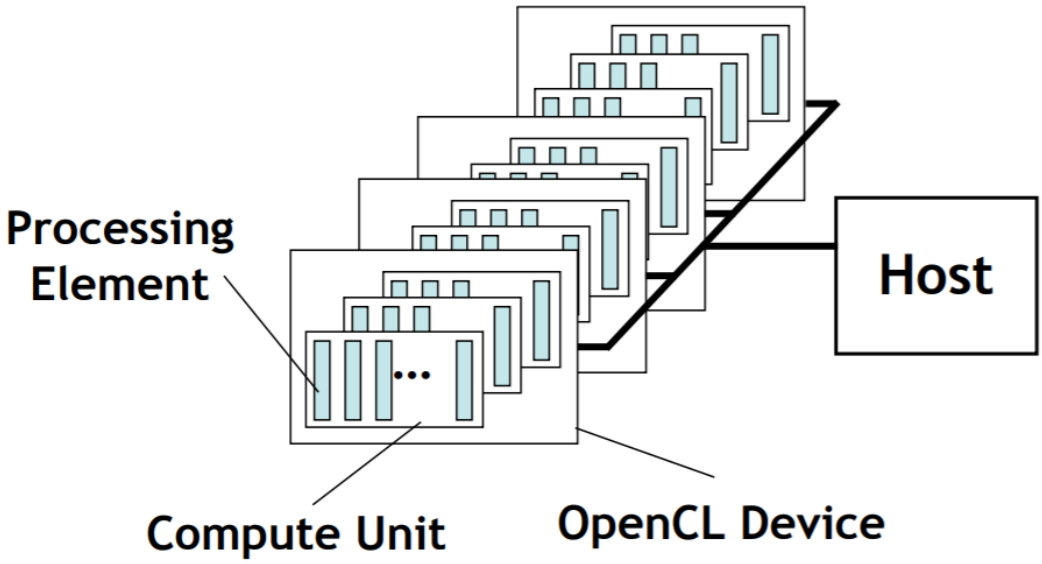
- Plot the following graphs

1. Number of threads Vs Execution time
2. Input size vs execution time

OpenCL:

OpenCL Embedded profile for mobile and embedded silicon – Relaxes some data type and precision requirements – Avoids the need for a separate "ES" specification. Khronos APIs provide computing support for imaging & graphics – Enabling advanced applications in, e.g., Augmented Reality. OpenCL will enable parallel computing in new markets – Mobile phones, cars, avionics

OpenCL Platform Model:



One Host and one or more OpenCL Devices – Each OpenCL Device is composed of one or more Compute Units . Each Compute Unit is divided into one or more Processing Elements . Memory divided into host memory and device memory.

The BIG idea behind OpenCL

Replace loops with functions (a kernel) executing at each point in a problem domain – E.g., process a 1024x1024 image with one kernel invocation per pixel or 1024x1024=1,048,576 kernel executions.

Traditional loops

```
void
mul(const  int n,
    const  float *a,
    const  float *b,
           float *c)
{
  int i;
  for (i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

OpenCL

```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)
{
  int id = get_global_id(0);
  c[id]  = a[id] * b[id];
}
// execute over n work-items
```

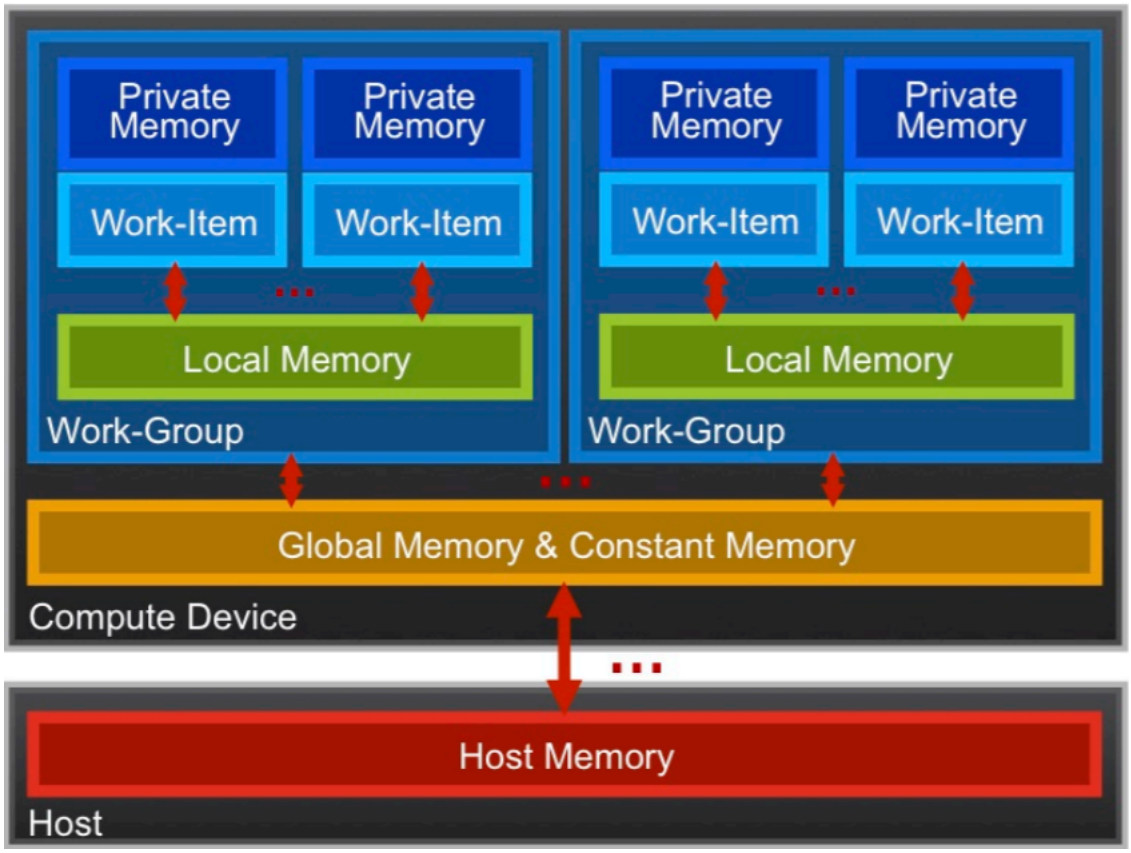
OpenCL Memory model:

Private Memory – Per work-item

• Local Memory – Shared within a work-group

• Global Memory Constant Memory – Visible to all work-groups

• Host memory – On the CPU Memory management is explicit: You are responsible for moving data from host global local and back



Execution model (kernels)

• OpenCL execution model … define a problem domain and execute an instance of a kernel for each point in the domain

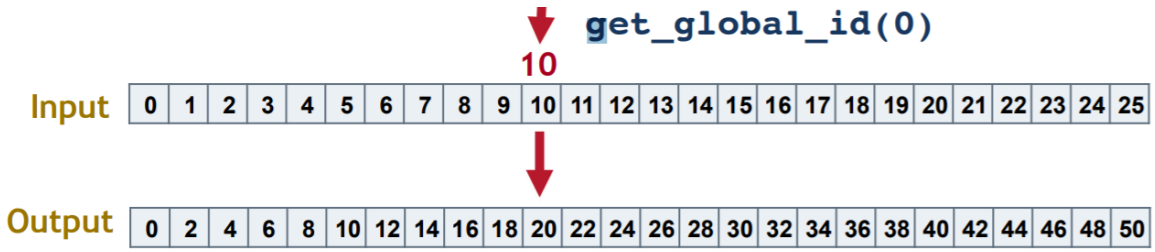
```
__kernel void times_two( __global float* input, __global float* output)

{

int i = get_global_id(0);

output[i] = 2.0f * input[i];

}
```


get_global_id(0)
10
Input
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Output
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50

Experiment No. 1:

****

PADP Lab, RVCE, CSE Aug 2021
Example :PI Calculation.
Objective :PI Calculation using Monto-Carlo Algorithm(OpenMp,MPI).
Input : User has to set OMP_NUM_THREADS environment variable for n number of threads and has to specify the number of intervals
Output : Each thread calculates the partial sum and then the master thread prints the final PI value and time taken forcomputation of PI value.
****

```
/* Program to compute Pi using Monte Carlo methods */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#define PI 3.1415926538837211
main(intargc,char **argv)
{

        double          x, totalsum, h;

printf("\n\t\t Enter number of intervals\n");
        scanf("%d", &Noofintervals);


        /* No. of intervals should be positive integer */
        if (Noofintervals<= 0) {
                printf("\n\t\t Number of intervals should be positive integer\n");
                exit(1);
        }
        totalsum = 0.0;

                h = 1.0 / Noofintervals;

                for (i = 1; i <Noofintervals + 1; i = i+1) {
                x = h * (i + 0.5);
                totalsum = totalsum + 4.0/(1.0 + x * x);
        }
        totalsum = totalsum * h;
        printf("\n\t\t Calculated PI :  \t%1.15lf  \n\t\t Error : \t%1.16lf\n", totalsum,
fabs(totalsum - PI));*/
        printf("\n\t\t Calculated PI              : %1.15lf",totalsum );

}
```

Output:

Input size	**Execution Time- Number of Threads**			
	1	**2**	**4**	**8**

Plot the graphs for the following

1. Number of threads vs Execution time
2. Input size vs execution time

Observation:

		Excellent	Good	Poor
a.	**Understanding of problem and parallel program writing** **(2 Marks)**	Writing an efficient parallel program for the given sequential program along with appropriate data scoping (2)	Writing an efficient parallel program for the given sequential program without data scoping/ with inappropriate data scoping (< 2 to >=1)	Not able to write parallel Program (0)
b.	**Execution** **(2 Marks)**	Demonstrate • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (2)	Demonstrating any one • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (< 2 to >=1)	No Execution(0)
c.	**Result tabulation and Graph** **(2 Marks)**	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results and inference. (2)	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results without inference (< 2 to >=1)	No documentation. (0)

Viva rubrics (Max: 4 marks)

		Excellent	Good	Poor
a.	**Inference Justification(1)**	Completely justify the obtained inference.(1)	Incomplete Justification for the inference(0.5)	Unable to explain the inferences. (0)
b.	**Understanding of Parallel construct** **(2 Marks)**	Explains the usage of all parallel programming constructs. (2)	Adequately explains the usage of all parallel programming constructs (< 2 to >=1)	Unable to explain concepts. (0)
c.	**Application of parallel constructs (1)**	Identifying an appropriate parallel construct for given problem (1)		Did not solve problem.(0)

Experiment No. 2

****

PADP Lab, RVCE, CSE Aug 2021

Example :Matrix Multiplication

Objective :Write an OpenMP program that computes a simple matrix-matrix multiplication using dynamic memory allocation.

a) Illustrate the correctness of the program.

b) Justify the inference when outer "for" loop is parallelized with and without using the explicit data scope variables.

Input : Number of threads, Matrix Size

Output :Output Matrix, Time.

****

```
#include <stdio.h>
int main()
{
int m, n, p, q, c, d, k, sum = 0;
int first[10][10], second[10][10], multiply[10][10];

printf("Enter the number of rows and columns of first matrix\n");
scanf("%d%d", &m, &n);
printf("Enter the elements of first matrix\n");
for (  c = 0 ; c < m ; c++ )
for ( d = 0 ; d < n ; d++ )
scanf("%d", &first[c][d]);
printf("Enter the number of rows and columns of second matrix\n");
scanf("%d%d", &p, &q);
if ( n != p )
printf("Matrices with entered orders can't be multiplied with each other.\n");
else
{
printf("Enter the elements of second matrix\n");
for ( c = 0 ; c < p ; c++ )
for ( d = 0 ; d < q ; d++ )
scanf("%d", &second[c][d]);
for ( c = 0 ; c < m ; c++ )
 {
for ( d = 0 ; d < q ; d++ )
     {
for ( k = 0 ; k < p ; k++ )
{
sum = sum + first[c][k]*second[k][d];
}
multiply[c][d] = sum;
sum = 0;
     }
```

```
    }

printf("Product of entered matrices:-\n");
for ( c = 0 ; c < m ; c++ )
    {
for ( d = 0 ; d < q ; d++ )
printf("%d\t", multiply[c][d]);

printf("\n");
    }
  }

return 0;
}
```

Output:

Input size	**Execution Time- Number of Threads**			
	1	**2**	**4**	**8**

Plot the graphs for the following

1. Number of threads vs Execution time
2. Input size vs execution time

Observation:

		Excellent	Good	Poor
a.	**Understanding of problem and parallel program writing** **(2 Marks)**	Writing an efficient parallel program for the given sequential program along with appropriate data scoping (2)	Writing an efficient parallel program for the given sequential program without data scoping/ with inappropriate data scoping (< 2 to >=1)	Not able to write parallel Program (0)
b.	**Execution** **(2 Marks)**	Demonstrate • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (2)	Demonstrating any one • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (< 2 to >=1)	No Execution(0)
c.	**Result tabulation and Graph** **(2 Marks)**	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results and inference. (2)	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results without inference (< 2 to >=1)	No documentation. (0)

Viva rubrics (Max: 4 marks)

		Excellent	Good	Poor
a.	**Inference Justification(1)**	Completely justify the obtained inference.(1)	Incomplete Justification for the inference(0.5)	Unable to explain the inferences. (0)
b.	**Understanding of Parallel construct** **(2 Marks)**	Explains the usage of all parallel programming constructs. (2)	Adequately explains the usage of all parallel programming constructs (< 2 to >=1)	Unable to explain concepts. (0)
c.	**Application of parallel constructs (1)**	Identifying an appropriate parallel construct for given problem (1)		Did not solve problem.(0)

Experiment No. 3

****

PADP Lab, RVCE, CSE Aug 2021

Example :Cache_sieve.c

Objective : Write a program for Cache unfriendly sieve of Eratosthenes and Cache friendly Sieve of Eratosthenes for enumerating prime numbers upto N and prove the correctness.

****

```
#include<math.h>
#include<string.h>
#include<omp.h>
#include<iostream>
using namespace std;
double t = 0.0;
inline long Strike( bool composite[], long i,long stride, long limit ) {
        for( ; i<=limit; i+=stride )
                composite[i] = true;
        return i;
}

longCacheUnfriendlySieve( long n )
 {
        long count = 0;
        long m = (long)sqrt((double)n);
        bool* composite = new bool[n+1];
        memset( composite, 0, n );
        t = omp_get_wtime();
        for( long i=2; i<=m; ++i )
                if( !composite[i] ) {
                        ++count;
                        // Strike walks array of size n here.
                        Strike( composite, 2*i, i, n );
}
        for( long i=m+1; i<=n; ++i )
                if( !composite[i] ){
                ++count;
                }
        t = omp_get_wtime() - t;
        delete[] composite;
        return count;
}
longCacheFriendlySieve( long n )
{
long count = 0;
long m = (long)sqrt((double)n);
bool* composite = new bool[n+1];
memset( composite, 0, n );
long* factor = new long[m];
long* striker = new long[m];
```

```
longn_factor = 0;
    t= omp_get_wtime();
for( long i=2; i<=m; ++i )
if( !composite[i] )
{
     ++count;
striker[n_factor] = Strike( composite, 2*i, i, m );
factor[n_factor++] = i;
}
  // Chops sieve into windows of size ≈ sqrt(n)
for( long window=m+1; window<=n; window+=m )
{
long limit = min(window+m-1,n);
for( long k=0; k<n_factor; ++k )
// Strike walks window of size sqrt(n) here.
striker[k] = Strike( composite, striker[k], factor[k],limit );
for( long i=window; i<=limit; ++i )
if( !composite[i] )
  ++count;
}
   t = omp_get_wtime() - t;
delete[] striker;
delete[] factor;
delete[] composite;
return count;
}
int main()
{
        long count = CacheUnfriendlySieve(100000000);
long count = CacheFriendlySieve(100000000);
        cout<< count;
        cout<< "Time : "<<t<<endl;
}
```

Output:

Input size	**Execution Time**		
			Cache Parallel Sieve

Plot the graphs for the following

1. Cache friendly Vs Cache unfriendly for different input sizes.

Observation:

		Excellent	Good	Poor
a.	**Understanding of problem and parallel program writing** **(2 Marks)**	Writing an efficient parallel program for the given sequential program along with appropriate data scoping (2)	Writing an efficient parallel program for the given sequential program without data scoping/ with inappropriate data scoping (< 2 to >=1)	Not able to write parallel Program (0)
b.	**Execution** **(2 Marks)**	Demonstrate • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (2)	Demonstrating any one • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (< 2 to >=1)	No Execution(0)
c.	**Result tabulation and Graph** **(2 Marks)**	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results and inference. (2)	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results without inference (< 2 to >=1)	No documentation. (0)

Viva rubrics (Max: 4 marks)

		Excellent	Good	Poor
a.	**Inference Justification(1)**	Completely justify the obtained inference.(1)	Incomplete Justification for the inference(0.5)	Unable to explain the inferences. (0)
b.	**Understanding of Parallel construct** **(2 Marks)**	Explains the usage of all parallel programming constructs. (2)	Adequately explains the usage of all parallel programming constructs (< 2 to >=1)	Unable to explain concepts. (0)
c.	**Application of parallel constructs (1)**	Identifying an appropriate parallel construct for given problem (1)		Did not solve problem.(0)

Experiment No. 4

****

PADP Lab, RVCE, CSE Aug 2021
Name: Negative_image.c

Objective: Write a program to find the negative of an image and demonstrate the following
a). Use default data scope variables and justify the inference.

b). Use Critical section to prove the correctness using default data scope variables

Use appropriate data scope variables to prove the correctness.
Input:image in png format
Output:negative of the given image
****

```
#include <stdio.h>
#include <error.h>
#include <gd.h>
#include<string.h>
int main(intargc, char **argv)
{
 FILE *fp = {0};
gdImagePtrimg;
char *iname = NULL;
char *oname = NULL;
intcolor, x, y, w, h,i=0;
int red, green, blue;
color = x = y = w = h = 0;
red = green = blue = 0;
if(argc != 3)
error(1, 0, "Usage: gdnegat input.png output.png");
else
{
iname = argv[1];
oname = argv[2];
 }
if((fp = fopen(iname, "r")) == NULL)
error(1, 0, "Error - fopen(): %s", iname);
else
img = gdImageCreateFromPng(fp);
w = gdImageSX(img);
h = gdImageSY(img);
double t = omp_get_wtime();
for(x = 0; x < w; x++) {
for(y = 0; y < h; y++) {
color = x+0;
color=gdImageGetPixel(img, x, y);
red   = 255 - gdImageRed(img, color);
green = 255 - gdImageGreen(img, color);
blue  = 255 - gdImageBlue(img, color);
color = gdImageColorAllocate(img, red, green, blue);
```

```
gdImageSetPixel(img, x, y, color);
}
}
if((fp = fopen(oname, "w")) == NULL)
error(1, 0, "Error - fopen(): %s", oname);
else
 {
gdImagePng(img, fp);
fclose(fp);
}
t = omp_get_wtime() - t;
gdImageDestroy(img);
printf("Time taken = %g",t);
return 0;
}
```

Output:

	Execution Time- Number of Threads			
	1	**2**	**4**	**8**
Data Scope				
Critical Section				

Plot the graphs for the following

1. Number of threads vs Execution time
2. Input size vs execution time

Observation:

		Excellent	Good	Poor
a.	**Understanding of problem and parallel program writing** **(2 Marks)**	Writing an efficient parallel program for the given sequential program along with appropriate data scoping (2)	Writing an efficient parallel program for the given sequential program without data scoping/ with inappropriate data scoping (< 2 to >=1)	Not able to write parallel Program (0)
b.	**Execution** **(2 Marks)**	Demonstrate • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (2)	Demonstrating any one • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (< 2 to >=1)	No Execution(0)
c.	**Result tabulation and Graph** **(2 Marks)**	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results and inference. (2)	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results without inference (< 2 to >=1)	No documentation. (0)

Viva rubrics (Max: 4 marks)

		Excellent	Good	Poor
a.	**Inference Justification(1)**	Completely justify the obtained inference.(1)	Incomplete Justification for the inference(0.5)	Unable to explain the inferences. (0)
b.	**Understanding of Parallel construct** **(2 Marks)**	Explains the usage of all parallel programming constructs. (2)	Adequately explains the usage of all parallel programming constructs (< 2 to >=1)	Unable to explain concepts. (0)
c.	**Application of parallel constructs (1)**	Identifying an appropriate parallel construct for given problem (1)		Did not solve problem.(0)

Experiment No. 5

****

PADP Lab, RVCE, CSE Aug 2021

Example: MPI Message passing

Objective: Write a MPI program that has a total of 4 processes. Process with rank 1, 2 and 3 should send the following messages respectively to the process with rank 0: HELLO, CSE, RVCE

****

****

```
#include <string.h>
#define BUFFER_SIZE 32

int main(int argc,char *argv[])
{
        int  MyRank,Numprocs, Destination, iproc;
        int  tag = 0;
        int  Root = 0, temp = 1;
        char Message[BUFFER_SIZE];
        MPI_Init(&argc,&argv);
        MPI_Status status;

        MPI_Comm_rank(MPI_COMM_WORLD,&MyRank);
        MPI_Comm_size(MPI_COMM_WORLD,&Numprocs);

        /* print host name, and send message from process with rank 0 to all other processes
*/
        if(MyRank == 0)       {
                system("hostname");
                strcpy(Message, "Hello India");
                for (temp=1; temp<Numprocs;temp++)
                {
                        MPI_Send(Message, BUFFER_SIZE, MPI_CHAR, temp,
tag,MPI_COMM_WORLD);
                }
        }
        else {
                system("hostname");
                MPI_Recv(Message, BUFFER_SIZE, MPI_CHAR, Root,
tag,MPI_COMM_WORLD, &status);
                printf("\n%s in process with rank %d from Process with rank %d\n",
Message,MyRank,Root);
        }

        MPI_Finalize();
}
```

		Excellent	Good	Poor
a.	**Understanding of problem and parallel program writing** **(2 Marks)**	Writing an efficient parallel program for the given sequential program along with appropriate data scoping (2)	Writing an efficient parallel program for the given sequential program without data scoping/ with inappropriate data scoping (< 2 to >=1)	Not able to write parallel Program (0)
b.	**Execution** **(2 Marks)**	Demonstrate • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (2)	Demonstrating any one • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (< 2 to >=1)	No Execution(0)
c.	**Result tabulation and Graph** **(2 Marks)**	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results and inference. (2)	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results without inference (< 2 to >=1)	No documentation. (0)

Viva rubrics (Max: 4 marks)

		Excellent	Good	Poor
a.	**Inference Justification(1)**	Completely justify the obtained inference.(1)	Incomplete Justification for the inference(0.5)	Unable to explain the inferences. (0)
b.	**Understanding of Parallel construct** **(2 Marks)**	Explains the usage of all parallel programming constructs. (2)	Adequately explains the usage of all parallel programming constructs (< 2 to >=1)	Unable to explain concepts. (0)
c.	**Application of parallel constructs (1)**	Identifying an appropriate parallel construct for given problem (1)		Did not solve problem.(0)

Experiment No. 6

****
PADP Lab, RVCE, CSE Aug 2021
Example : Wordsearch.c
Objective : Write a parallel program for Word search in a file. Justify the inference.
****

```
#include<stdio.h>
#include<omp.h>
#include<string.h>
#define COUNT 10
#define FILE_NAME "words.txt"
charsearch_words[20][COUNT] =
{"The","around","graphics","from","by","be","any","which","various","mount"};
long counts[COUNT];
intline_c = 0;
intis_alpha(char c)
{
return ((c >= 65 && c <= 90) || (c >= 97 && c <= 122));
}
intis_equal(char* a,const char* key, intignore_case){
char b[20];
strcpy(b,key);
intlen_a = strlen(a),len_b = strlen(b);
if(len_a != len_b)
{
return 0;
  }
if(ignore_case != 0)
{
int i;
for(i=0;i<len_a;i++)
{
if(a[i] > 90)
{
a[i] -= 32;
}
 }
for(i=0;i<len_b;i++)
{
if(b[i] > 90)
{
b[i] -= 32;
}
}
        }
        return (strcmp(a,b) == 0);
}
voidread_word(char *temp, FILE *fp)
```

```
{
        int i=0;
        charch;
        while((ch = fgetc(fp)) != EOF&&is_alpha(ch) == 0)
{

        }
        while(ch != EOF
                &&is_alpha(ch) != 0)
{
                temp[i++] = ch;
                ch = fgetc(fp);
        }
        temp[i] = '\0';
}

longdetermine_count(const char *file_name, const char *key, intignore_case)
{
        intkey_index=0,key_len = strlen(key);
        longword_count=0;
        charch;
        FILE *fp = fopen(file_name,"r");
        char temp[40];
        int i=0;
        while(feof(fp) == 0)
{
        read_word(temp,fp);
        if(is_equal(temp,key,ignore_case) != 0)
{
        word_count++;
}
        //printf("%s ",temp);
      }
        //printf("\nWord %s: %ld",key,word_count);
        returnword_count;
       }
void main()
{
        int i;
        for(i=0;i<COUNT;i++)
{
        counts[i] = 0;
}
        double t = omp_get_wtime();
        for(i=0;i<COUNT;i++)
{
                counts[i] = determine_count(FILE_NAME,search_words[i],1);
        }
        t = omp_get_wtime() - t;
```

```
        for(i=0;i<COUNT;i++)
{
                printf("\n%s: %ld",search_words[i],counts[i]);
        }
        printf("\nTime Taken: %lf\n",t);
}
```

Output:

Input size	**Execution Time- Number of Threads**			
	1	**2**	**4**	**8**

Plot the graphs for the following

1. Number of threads vs Execution time
2. Input size vs execution time

Observation:

		Excellent	Good	Poor
a.	**Understanding of problem and parallel program writing** **(2 Marks)**	Writing an efficient parallel program for the given sequential program along with appropriate data scoping (2)	Writing an efficient parallel program for the given sequential program without data scoping/ with inappropriate data scoping (< 2 to >=1)	Not able to write parallel Program (0)
b.	**Execution** **(2 Marks)**	Demonstrate • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (2)	Demonstrating any one • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (< 2 to >=1)	No Execution(0)
c.	**Result tabulation and Graph** **(2 Marks)**	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results and inference. (2)	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results without inference (< 2 to >=1)	No documentation. (0)

Viva rubrics (Max: 4 marks)

		Excellent	Good	Poor
a.	**Inference Justification(1)**	Completely justify the obtained inference.(1)	Incomplete Justification for the inference(0.5)	Unable to explain the inferences. (0)
b.	**Understanding of Parallel construct** **(2 Marks)**	Explains the usage of all parallel programming constructs. (2)	Adequately explains the usage of all parallel programming constructs (< 2 to >=1)	Unable to explain concepts. (0)
c.	**Application of parallel constructs (1)**	Identifying an appropriate parallel construct for given problem (1)		Did not solve problem.(0)

PADP Lab, RVCE, CSE Oct 2021
Example: OpenCL
Objective:SAXPY (Single precision real Alpha X plus Y): Write an OpenCL program to compute A = alpha*B + C, where alpha is a constant and A, B, and C are vectors of an arbitrary size n.

```
#include <stdio.h>
#include <stdlib.h>
#ifdef __APPLE__
#include <OpenCL/cl.h>
#else
#include <CL/cl.h>
#endif
#define VECTOR_SIZE 1024

//OpenCL kernel which is run for every work item created.
const char *saxpy_kernel =
"__kernel                                   \n"
"voidsaxpy_kernel(float alpha,     \n"
"                 __global float *A,       \n"
"                 __global float *B,       \n"
"                 __global float *C)       \n"
"{                                          \n"
"    //Get the index of the work-item       \n"
"    int index = get_global_id(0);          \n"
"    C[index] = alpha* A[index] + B[index]; \n"
"}                                          \n";

int main(void) {
int i;
  // Allocate space for vectors A, B and C
float alpha = 2.0;
float *A = (float*)malloc(sizeof(float)*VECTOR_SIZE);
float *B = (float*)malloc(sizeof(float)*VECTOR_SIZE);
float *C = (float*)malloc(sizeof(float)*VECTOR_SIZE);
for(i = 0; i < VECTOR_SIZE; i++)
  {
    A[i] = i;
    B[i] = VECTOR_SIZE - i;
    C[i] = 0;
  }

  // Get platform and device information
cl_platform_id * platforms = NULL;
cl_uintnum_platforms;
  //Set up the Platform
cl_intclStatus = clGetPlatformIDs(0, NULL, &num_platforms);
```

```
platforms = (cl_platform_id *)
malloc(sizeof(cl_platform_id)*num_platforms);
clStatus = clGetPlatformIDs(num_platforms, platforms, NULL);

  //Get the devices list and choose the device you want to run on
cl_device_id     *device_list = NULL;
cl_uintnum_devices;

clStatus = clGetDeviceIDs( platforms[0], CL_DEVICE_TYPE_GPU, 0,NULL,
&num_devices);
device_list = (cl_device_id *)
malloc(sizeof(cl_device_id)*num_devices);
clStatus = clGetDeviceIDs( platforms[0],CL_DEVICE_TYPE_GPU, num_devices,
device_list, NULL);

  // Create one OpenCL context for each device in the platform
cl_context context;
context = clCreateContext( NULL, num_devices, device_list, NULL, NULL, &clStatus);

  // Create a command queue
cl_command_queuecommand_queue = clCreateCommandQueue(context, device_list[0], 0,
&clStatus);

  // Create memory buffers on the device for each vector
cl_memA_clmem = clCreateBuffer(context, CL_MEM_READ_ONLY,VECTOR_SIZE *
sizeof(float), NULL, &clStatus);
cl_memB_clmem = clCreateBuffer(context, CL_MEM_READ_ONLY,VECTOR_SIZE *
sizeof(float), NULL, &clStatus);
cl_memC_clmem = clCreateBuffer(context, CL_MEM_WRITE_ONLY,VECTOR_SIZE *
sizeof(float), NULL, &clStatus);

  // Copy the Buffer A and B to the device
clStatus = clEnqueueWriteBuffer(command_queue, A_clmem, CL_TRUE, 0,
VECTOR_SIZE * sizeof(float), A, 0, NULL, NULL);
clStatus = clEnqueueWriteBuffer(command_queue, B_clmem, CL_TRUE, 0,
VECTOR_SIZE * sizeof(float), B, 0, NULL, NULL);

  // Create a program from the kernel source
cl_program program = clCreateProgramWithSource(context, 1,(const char
**)&saxpy_kernel, NULL, &clStatus);

  // Build the program
clStatus = clBuildProgram(program, 1, device_list, NULL, NULL, NULL);

  // Create the OpenCL kernel
cl_kernel kernel = clCreateKernel(program, "saxpy_kernel", &clStatus);

  // Set the arguments of the kernel
clStatus = clSetKernelArg(kernel, 0, sizeof(float), (void *)&alpha);
```

```
clStatus = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *)&A_clmem);
clStatus = clSetKernelArg(kernel, 2, sizeof(cl_mem), (void *)&B_clmem);
clStatus = clSetKernelArg(kernel, 3, sizeof(cl_mem), (void *)&C_clmem);

  // Execute the OpenCL kernel on the list
size_tglobal_size = VECTOR_SIZE; // Process the entire lists
size_tlocal_size = 64;           // Process one item at a time
clStatus = clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL, &global_size,
&local_size, 0, NULL, NULL);

  // Read the cl memory C_clmem on device to the host variable C
clStatus = clEnqueueReadBuffer(command_queue, C_clmem, CL_TRUE, 0,
VECTOR_SIZE * sizeof(float), C, 0, NULL, NULL);

  // Clean up and wait for all the comands to complete.
clStatus = clFlush(command_queue);
clStatus = clFinish(command_queue);

  // Display the result to the screen
for(i = 0; i < VECTOR_SIZE; i++)
printf("%f * %f + %f = %f\n", alpha, A[i], B[i], C[i]);

  // Finally release all OpenCL allocated objects and host buffers.
clStatus = clReleaseKernel(kernel);
clStatus = clReleaseProgram(program);
clStatus = clReleaseMemObject(A_clmem);
clStatus = clReleaseMemObject(B_clmem);
clStatus = clReleaseMemObject(C_clmem);
clStatus = clReleaseCommandQueue(command_queue);
clStatus = clReleaseContext(context);
free(A);
free(B);
free(C);
free(platforms);
free(device_list);
return 0;
}
```

Output:

Input size	**Execution Time- Number of Threads**			
	1	**2**	**4**	**8**

Plot the graphs for the following

1. Number of threads vs Execution time
2. Input size vs execution time

Observation:

		Excellent	Good	Poor
a.	**Understanding of problem and parallel program writing** **(2 Marks)**	Writing an efficient parallel program for the given sequential program along with appropriate data scoping (2)	Writing an efficient parallel program for the given sequential program without data scoping/ with inappropriate data scoping (< 2 to >=1)	Not able to write parallel Program (0)
b.	**Execution** **(2 Marks)**	Demonstrate • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (2)	Demonstrating any one • The correctness of parallel program with that of sequential. • Performance improvement of parallel program with sequential. (< 2 to >=1)	No Execution(0)
c.	**Result tabulation and Graph** **(2 Marks)**	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results and inference. (2)	Tabulate the results for different input sizes and by varying the number of threads. Plot the graph for tabulated results without inference (< 2 to >=1)	No documentation. (0)

Viva rubrics (Max: 4 marks)

		Excellent	Good	Poor
a.	**Inference Justification(1)**	Completely justify the obtained inference.(1)	Incomplete Justification for the inference(0.5)	Unable to explain the inferences. (0)
b.	**Understanding of Parallel construct** **(2 Marks)**	Explains the usage of all parallel programming constructs. (2)	Adequately explains the usage of all parallel programming constructs (< 2 to >=1)	Unable to explain concepts. (0)
c.	**Application of parallel constructs (1)**	Identifying an appropriate parallel construct for given problem (1)		Did not solve problem.(0)

Part B

Student has to develop in groups or individually a project to solve real world problem using CUDA