	Sanjivani Rural Education Society's SANJIVANI COLLEGE OF ENGINEERING (An Autonomous Institution) Kopargaon – 423 603, Maharashtra.	ACAD-F-16 A
Academic Year: 2023-24	**Lab Manual**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		**Date of Preparation : ___________**

Course Code & Course Name	:	**High Performance Computing Laboratory (CO406)**
Name of the Faculty	:	**Dr. B. J. Dange**
Year/Sem	:	**BTech / VII**
Evaluation Scheme	:	**Term Work (TW): 50 Marks**
Teaching Scheme	:	**Practical: 2 Hrs. / Week**
	:	**Credits: 1**

Faculty Signature	:				
Verified by HoD	:		Date of Verification	:	

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Index**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Vision and Mission**	**Revision : 00** **Dated :** ________
Department : Department of Computer Engineering		

Institute Vision

To Develop World Class Professionals through Quality Education.

Institute Mission

To create Academic Excellence in the field of Engineering and Management through Education, Training and Research to improve quality of life of people.

Department Vision

To develop world class engineering professionals with good moral characters and make them capable to exhibit leadership through their engineering ability, creative potential and effective soft skills which will improve the quality of life in society.

Department Mission

1] To impart quality technical education to the students through innovative and interactive teaching learning process to acquire sound technical knowledge, professional competence and to have aptitude for research and development.

2] Develop students as excellent communicators and highly effective team members and leaders with full appreciation of the importance of professional, ethical and social responsibilities.

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**PEOs and PSOs**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		

Program Educational Objectives (PEOs)

- **PEO1:** To prepare the committed and motivated graduates by developing technical competency and research attitude with support of strong academic environment.
- **PEO2:** To train graduates with strong fundamentals, domain knowledge and update with modern technology to design and develop novel products and provide effective solutions for social benefits.
- **PEO3:** To exhibit employability skills, leadership and right attitude to succeed in their professional career.

Program Specific Outcomes (PSOs)

1. **Professional Skills:** The ability to apply knowledge of problem solving, algorithmic analysis, software Engineering, Data Structures, Networking, Database with modern recent trends to provide the effective solutions for Computer Engineering Problems.

2. **Problem-Solving Skills:** The ability to inculcate best practices of software and hardware design for delivering quality products useful for the society.

3. **Successful Career:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths.

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Program Outcomes**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		

Program Outcomes (POs)

Engineering Graduates will be able to:

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

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

3. 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.

4. 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.

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

6. 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.

7. 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.

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

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

10. 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.

11. 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.

12. 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.

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Course Objectives and Outcomes**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		

Course Objectives

1. To understand and apply different parallel programming constructs to implement parallel algorithms.
2. To apply parallel programming tools to write parallel algorithms for sorting algorithms.
3. To implement different data mining algorithms using a parallel approach.
4. To build a cluster environment for implementation of MPI routines.
5. To write a program using MPI routines to implement different algorithms.
6. To understand the GPU architecture and implement CUDA program for real time applications.

Course Outcomes (COs)

On completion of the course, student will be able to–

CO1: Apply parallel algorithms for different algorithms using concurrent or parallel environments.

CO2: Apply parallel algorithms for sorting applications

CO3: Apply parallel computing techniques for data mining algorithms.

CO4: Demonstrate the different steps involved in building a simple Cluster.

CO5: Implement message-passing programs in distributed environments.

CO6: Use GPU architecture using CUDA program for solving real-time applications.

Mapping of Course Outcomes to POs-PSOs

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO 11	PO12	PSO1	PSO 2	PSO3
CO1	2	1	2	3	--	--	--	--	--	2	--	--	3	--	--
CO2	3	2	3	2	--	--	--	--	--	--	--	--	2	--	--
CO3	2	2	3	1	3	--	--	--	--	--	--	--	--	3	2
CO4	3	1	3	3	3	--	--	--	--	--	--	--	3	2	2
CO5	3	1	2	2	3	--	--	--	--	--	--	--	3	2	2
CO6	3	1	2	3	3	3	2	--	--	--	--	--	2	--	3

	Sanjivani Rural Education Society's SANJIVANI COLLEGE OF ENGINEERING (An Autonomous Institution) Kopargaon – 423 603, Maharashtra.	ACAD-F-15 A
Academic Year: 2023-24	**List of Equipment Software's/ Tools**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		

S. No.	**Name of Software/ Tools**
1.	Openmp 5.2
2.	MPICH 4.1.2
3.	CUDA toolkit 10.0
4.	Apache Hadoop 3.4.2
5.	

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**List of Assignments**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		

	Sanjivani Rural Education Society's SANJIVANI COLLEGE OF ENGINEERING **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**List of Assignments**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		

Laboratory Assignment Details

	Sanjivani Rural Education Society's SANJIVANI COLLEGE OF ENGINEERING (An Autonomous Institution) Kopargaon – 423 603, Maharashtra.	ACAD-F-15 A
Academic Year: 2023-24	**Assignment No:1**	**Revision : 00** **Dated :** ________
Department : Department of Computer Engineering		
Title: Vector and Matrix Operations		

Problem Statement

Design parallel algorithm to

1. Add two large vectors
2. Multiply Vector and Matrix
3. Multiply two N × N arrays using n^2 processors

Input: Matrix and Vector

Output: Output of Corresponding operation

Theory :

What is OpenMP

Open specifications for Multi Processing

Long version: Open specifications for MultiProcessing via collaborative work between interested parties from the hardware and software industry, government and academia.

•An Application Program Interface (API) that is used to explicitly direct multi-threaded, shared memory parallelism.

•API components:

–Compiler directives

–Runtime library routines

–Environment variables

•Portability

–API is specified for C/C++ and Fortran

–Implementations on almost all platforms including Unix/Linux and Windows

•Standardization

–Jointly defined and endorsed by major computer hardware and software vendors.

–Possibility to become ANSI standard

Brief History of OpenMP

•In 1991, Parallel Computing Forum (PCF) group invented a set of directives for specifying loop parallelism in Fortran programs.

•X3H5, an ANSI subcommittee developed an ANSI standard based on PCF.

• In 1997, the first version of OpenMP for Fortran was defined by OpenMP Architecture Review Board.

•Binding for C/C++ was introduced later.

•Version 3.1 is available since 2011.

OpenMP Programming Model

•Shared memory, thread-based parallelism

–OpenMP is based on the existence of multiple threads in the shared memory programming paradigm.

–A shared memory process consists of multiple threads.

• Explicit Parallelism

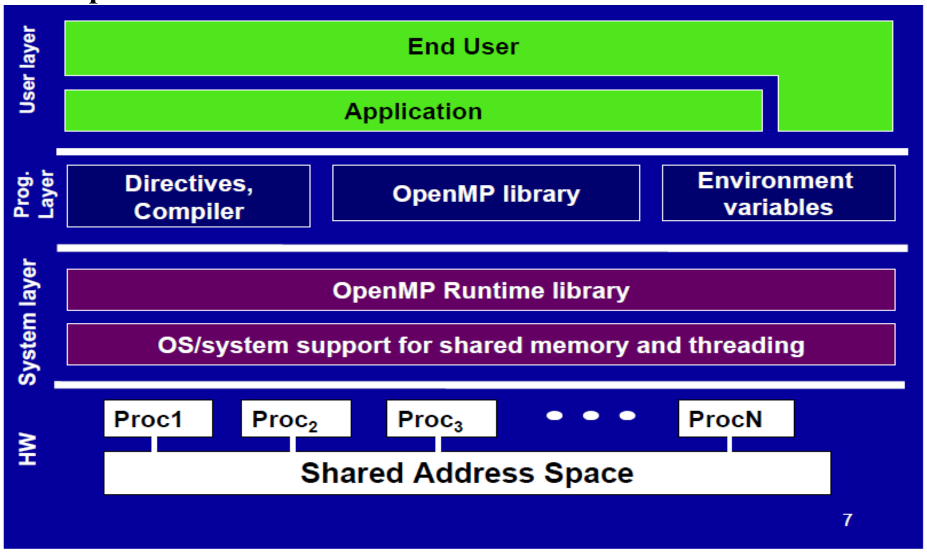
–Programmer has full control over parallelization. OpenMP is not an automatic parallel programming model.

•Compiler directive based

–Most OpenMP parallelism is specified through the use of compiler directives which are embedded in the source code.

Architecture of OpenMP-

-

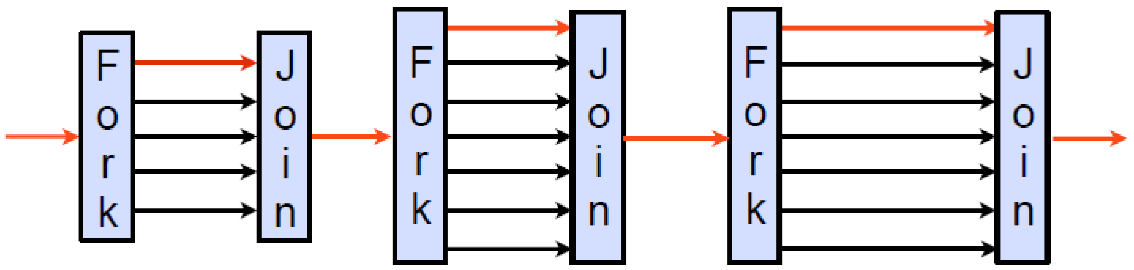


Fork-Join Parallelism •OpenMP program begin as a single process: the *master thread*. The master thread executes sequentially until the first *parallel region* construct is encountered.

•When a parallel region is encountered, master thread

–Create a group of threads by **FORK**.

–Becomes the master of this group of threads, and is assigned the thread id 0 within the group.



•The statement in the program that are enclosed by the *parallel region* construct are then executed in parallel among these threads.

•**JOIN**: When the threads complete executing the statement in the *parallel region* construct, they synchronize and terminate, leaving only the master thread.

OpenMP Code Structure

```
#include <stdlib.h>
#include <stdio.h>
#include "omp.h"
int main()
{
#pragma omp parallel
{
int ID = omp_get_thread_num();
printf("Hello (%d)\n", ID);
printf(" world (%d)\n", ID);
}
}
```

Set # of threads for OpenMP In csh setenv OMP_NUM_THREAD

Compile: g++ -fopenmp hello.c Run: ./a.out

OpenMP Core Syntax

```
#include “omp.h”
int main ()
{
int var1, var2, var3;
// Serial code
. . .
// Beginning of parallel section.
// Fork a team of threads. Specify variable scoping
#pragma omp parallel private(var1, var2) shared(var3)
{
// Parallel section executed by all threads
. . .
// All threads join master thread and disband
}
// Resume serial code . . .
}
```

OpenMP C/C++ Directive Format

OpenMP directive forms

–C/C++ use compiler directives

•Prefix: #pragma omp …

–A directive consists of a directive name followed by *clauses*

Example: #pragma omp parallel default (shared) private (var1, var2)

OpenMP parallel Region Directive

#pragma omp parallel [*clause list*]

Typical clauses in [*clause list*]

•Conditional parallelization

–if (scalar expression)

•Determine whether the parallel construct creates threads

•Degree of concurrency

–num_threads (integer expresson)

•number of threads to create

•Date Scoping

–private (variable list)

•Specifies variables local to each thread

–firstprivate (variable list)

•Similar to the private

•Private variables are initialized to variable value before the parallel directive

–shared (variable list)

•Specifies variables that are shared among all the threads

–default (data scoping specifier)

•Default data scoping specifier may be shared or none

Example:

```
#pragma omp parallel if (is_parallel == 1) num_threads(8) shared (var_b) private (var_a)
firstprivate (var_c) default (none)
{
/* structured block */
}
```

•if (is_parallel == 1) num_threads(8)

–If the value of the variable is_parallel is one, create 8 threads

•shared (var_b)

–Each thread shares a single copy of variable b

•private (var_a) firstprivate (var_c)

–Each thread gets private copies of variable var_a and var_c

–Each private copy of var_c is initialized with the value of var_c in main thread when the parallel directive is encountered

•default (none)

–Default state of a variable is specified as none (rather than shared)

–Singals error if not all variables are specified as shared or private

Another Example Using *for*

•Sequential code to add two vectors

```
for(i=0;i<N;i++)
 {
c[i] = b[i] + a[i];
}
```

•A worksharing for construct to add vectors

```
 #pragma omp parallel
 {
#pragma omp for
{
for(i=0; i<N; i++) {c[i]=b[i]+a[i];
}
}
}
```

or

```
#pragma omp parallel for

{

for(i=0; i<N; i++)

{

c[i]=b[i]+a[i];

}

}
```

Matrix-Vector Multiplication

```
# pragma  omp parallel default (none) \
shared (a, b, c, m,n) private (i,j,sum) num_threads(4)
for(i=0; i < m; i++){
sum = 0.0;
for(j=0; j < n; j++)
sum += b[i][j]*c[j];
a[i] =sum;
}
```

Matrix-Matrix Multiplication

```
// static scheduling of matrix multiplication loops
```

```
#pragma omp parallel default (private) \
shared (a, b, c, dim) num_threads(4)
#pragma omp for schedule(static)
for(i=0; i < dim; i++)
{
for(j=0; j < dim; j++)
{
c[i][j] = 0.0;
for(k=0; j < dim; k++)
c[i][j] += a[i][k]*b[k][j];
}
}
```

Conclusion-Different Matrix and Vector operation is performed using openmp in parallel environment.

Questions:

1. What is OpenMP ?
2. What problem does OpenMP solve ?
3. Why should I use OpenMP ?
4. Which compilers support OpenMP ?
5. Who uses OpenMP ?
6. What languages does OpenMP support ?
7. Is OpenMP scalable
8. Can I use loop-level parallelism ?
9. Can I use nested parallelism ?
10. Can I use task parallelism ?
11. Is it better to parallelize the outer loop ?
12. Can I use OpenMP to program on accelerators ?
13. Can I use OpenMP to program SIMD units ?

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:2**	**Revision : 00** **Dated : _________**
Department : Department of Computer Engineering		
Title: Parallel sorting Algorithms		

Problem Statement

Parallel sorting Algorithms- for Bubble Sort and Merger Sort, based on existing sequential algorithms, design and implement parallel algorithm utilizing all resources available.

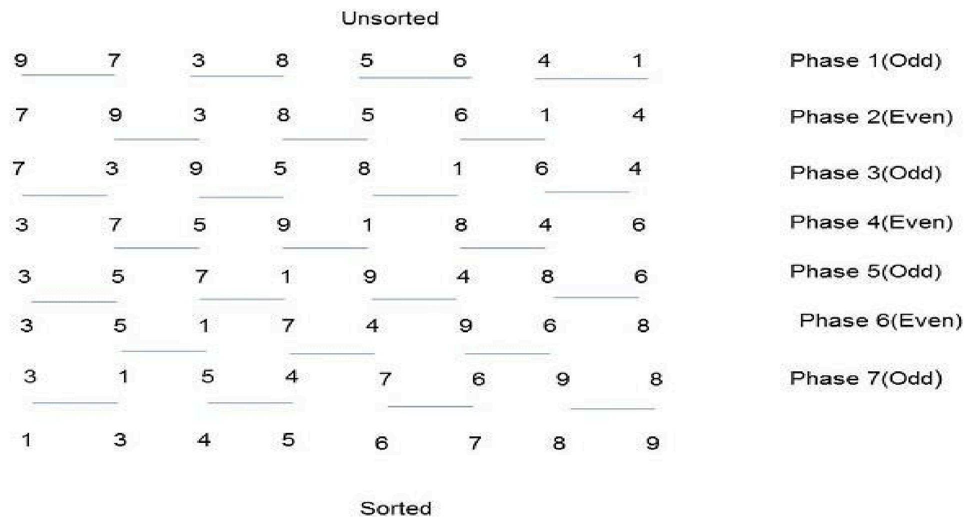
Input: Array Element with no of element.

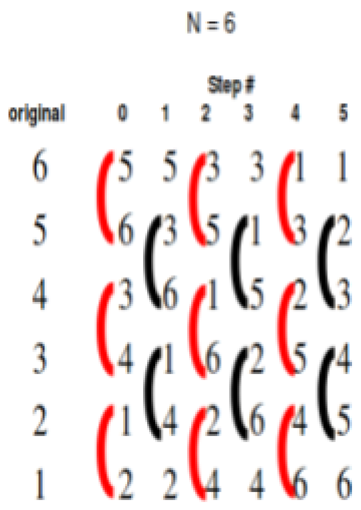
Output: Sorted Array

Theory :

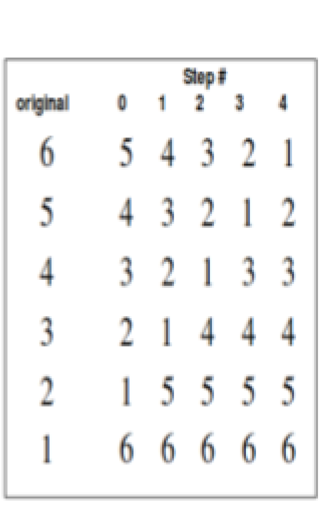
Parallel Bubble Sort-

Odd-Even Transposition Sort is based on the Bubble Sort technique. It compares two adjacent numbers and switches them, if the first number is greater than the second number to get an ascending order list. The opposite case applies for a descending order series. Odd-Even transposition sort operates in two phases − **odd phase** and **even phase**. In both the phases, processes exchange numbers with their adjacent number in the right.



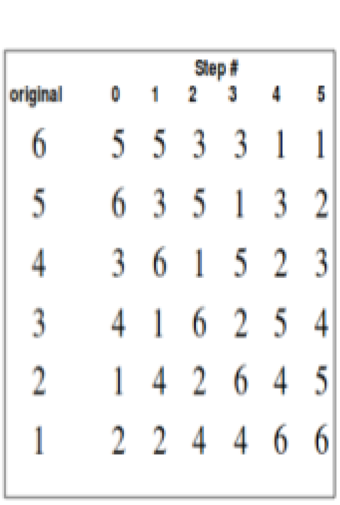


original	0	1	2	3	4	5
6	5	5	3	3	1	1
5	6	3	5	1	3	2
4	3	6	1	5	2	3
3	4	1	6	2	5	4
2	1	4	2	6	4	5
1	2	2	4	4	6	6



original	0	1	2	3	4
6	5	4	3	2	1
5	4	3	2	1	2
4	3	2	1	3	3
3	2	1	4	4	4
2	1	5	5	5	5
1	6	6	6	6	6

Non-threaded



original	0	1	2	3	4	5
6	5	5	3	3	1	1
5	6	3	5	1	3	2
4	3	6	1	5	2	3
3	4	1	6	2	5	4
2	1	4	2	6	4	5
1	2	2	4	4	6	6

Threaded

Bubble Sort Code:

```
#include<iostream>
#include<stdlib.h>
#include<omp.h>
using namespace std;
void bubble(int *, int);
void swap(int &, int &);

void bubble(int *a, int n)  {
 for(  int i = 0;  i < n;  i++ )  {
    int first = i % 2;
    #pragma omp parallel for shared(a,first)   //Creates parallel threads to swap consecutive
elements
    for(  int j = first;  j < n-1;  j += 2  ) {
        if(  a[ j ]  >  a[ j+1 ]  ) {
         swap(  a[ j ],  a[ j+1 ]  );
   }}}}

void swap(int &a, int &b){
 int test; test=a; a=b; b=test;                    //swaps two variables
}

int main(){
 int *a,n;
cout<<"\n enter total no of elements=>"; cin>>n; a=new int[n];
cout<<"\n enter elements=>"; for(int i=0;i<n;i++) {  cin>>a[i]; }
bubble(a,n);
cout<<"\n sorted array is=>"; for(int i=0;i<n;i++) {  cout<<a[i]<<endl; }

return 0;}
```

Parallel Merge Sort-

Merge sort first divides the unsorted list into smallest possible sub-lists, compares it with the adjacent list, and merges it in a sorted order. It implements parallelism very nicely by following the divide and conquer algorithm.
Mergesort is one of the most popular sorting techniques. It is the typical example for demonstrating the divide-and-conquer paradigm.

Merge sort (also commonly spelled mergesort) is an efficient, general-purpose, comparison-based sorting algorithm.

Mergesort has the worst case serial growth as **O(nlogn)**.
Sorting an array: A[p .. r] using mergesort involves three steps.

1) Divide Step

If a given array A has zero or one element, simply return; it is already sorted. Otherwise, split A[p .. r] into two subarrays A[p .. q] and A[q + 1 .. r], each containing about half of the elements of A[p .. r]. That is, q is the halfway point of A[p .. r].

2) Conquer Step

Conquer by recursively sorting the two subarrays A[p .. q] and A[q + 1 .. r].

3) Combine Step

Combine the elements back in A[p .. r] by merging the two sorted subarrays A[p .. q] and A[q + 1 .. r] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r).

We can parallelize the “conquer” step where the array is recursively sorted amongst the left and right subarrays. We can ‘parallely’ sort the left and the right subarrays.

Parallelizing Merge Sort through OpenMP

As stated before, the parallelizable region is the “conquer” part. We need to make sure that the left and the right sub-arrays are sorted simuntaneously. We need to implement both left and right *sections* in parallel.

This can be done in OpenMP using directive:

#pragma omp parallel sections

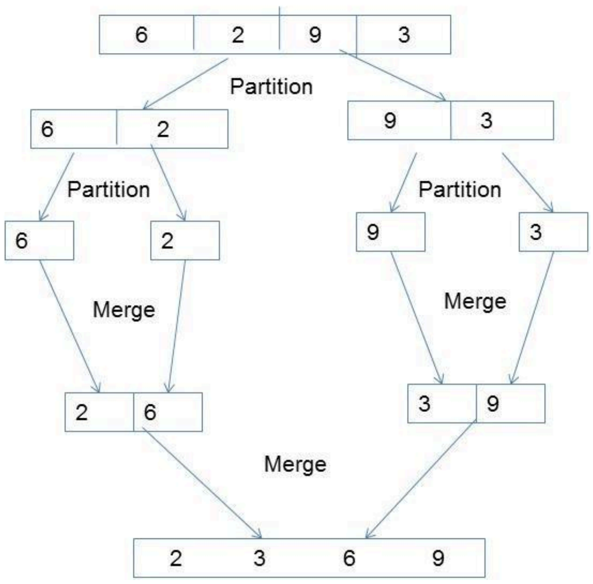
And each section that has to be parallelized should be enclosed with the directive:

#pragma omp section

Now, let's work on parallelizing the both sections through OpenMP

```
if(i<j)
   {
      mid=(i+j)/2;

      #pragma omp parallel sections
      {

         #pragma omp section
         {
            mergesort(a,i,mid);        //left recursion
         }

         #pragma omp section
         {
            mergesort(a,mid+1,j);    //right recursion
         }
      }

      merge(a,i,mid,mid+1,j);    //merging of two sorted sub-arrays
   }
```

The above will parallelize both left and right recursion.



Conclusion-Different parallel sorting algorithm is performed using openmp in a parallel environment.

Questions:

1. What is OpenMP ?
2. What problem does OpenMP solve ?
3. Why should I use OpenMP ?
4. Which compilers support OpenMP ?
5. Who uses OpenMP ?
6. What languages does OpenMP support ?
7. Is OpenMP scalable
8. Can I use loop-level parallelism ?
9. Can I use nested parallelism ?
10. Can I use task parallelism ?
11. Is it better to parallelize the outer loop ?
12. Can I use OpenMP to program on accelerators ?
13. Can I use OpenMP to program SIMD units ?

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:3**	**Revision : 00** **Dated :** ________
Department : Department of Computer Engineering		
Title: Parallel Implementation of the K Nearest Neighbors Classifier using python/openmp		

Problem Statement

Input: No of Element or Object and K Value

Output: Classification of Given Object

Theory :

Introduction to K-Nearest Neighbor (KNN)

KNN is a non-parametric supervised learning technique in which we try to classify the data point to a given category with the help of training set. In simple words, it captures information of all training cases and classifies new cases based on a similarity.
Predictions are made for a new instance (x) by searching through the entire training set for the K most similar cases (neighbors) and summarizing the output variable for those K cases. In classification this is the mode (or most common) class value.

How KNN algorithm works

Suppose we have the height, weight and T-shirt size of some customers and we need to predict the T-shirt size of a new customer given only height and weight information we have. Data including height, weight and T-shirt size information is shown below -

Height (in cms)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M

158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

K-Nearest Neighbor Simplified

Step 1 : Calculate Similarity based on distance function

There are many distance functions but Euclidean is the most commonly used measure. It is mainly used when data is continuous. Manhattan distance is also very common for continuous variables.

Euclidean :

$$d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city - block :

$$d(x, y) = \sum_{i=1}^{m} |x_i - y_i|$$

Distance Functions

The idea to use distance measure is to find the distance (similarity) between new sample and training cases and then finds the k-closest customers to new customer in terms of height and weight.

New customer named 'Monica' has height 161cm and weight 61kg.

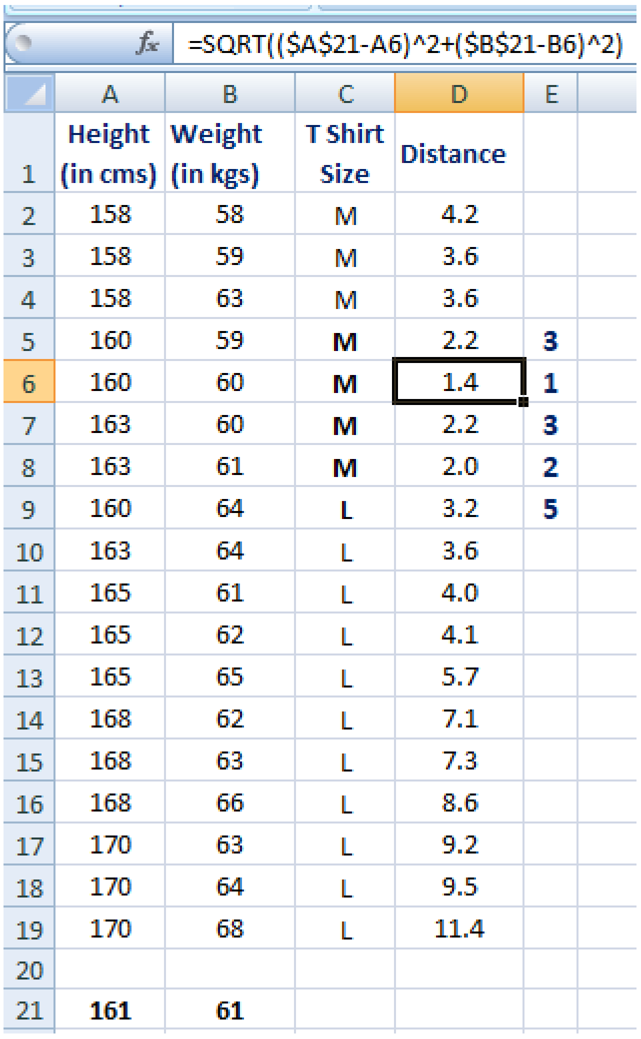
Euclidean distance between first observation and new observation (monica) is as follows -

=SQRT((161-158)^2+(61-58)^2)

Similarly, we will calculate distance of all the training cases with new case and calculates the rank in terms of distance. The smallest distance value will be ranked 1 and considered as nearest neighbor.

Step 2 : Find K-Nearest Neighbors

Let k be 5. Then the algorithm searches for the 5 customers closest to Monica, i.e. most similar to Monica in terms of attributes, and see what categories those 5 customers were in. If 4 of them had 'Medium T shirt sizes' and 1 had 'Large T shirt size' then your best guess for Monica is 'Medium T shirt. See the calculation shown in the snapshot below -

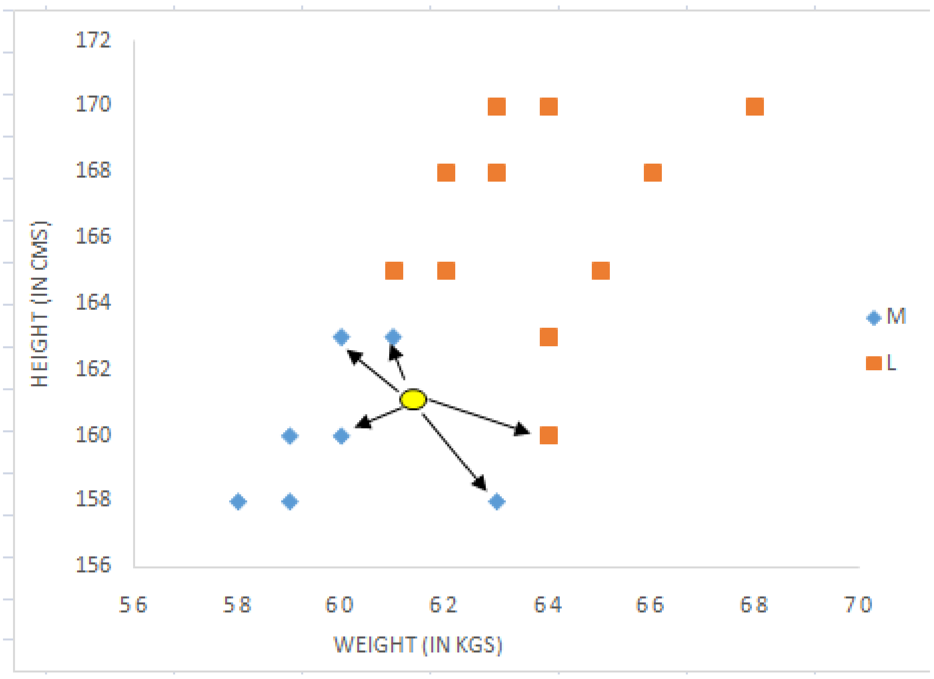


=SQRT((A21-A6)^2+(B21-B6)^2)

	A	B	C	D	E
1	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
2	158	58	M	4.2	
3	158	59	M	3.6	
4	158	63	M	3.6	
5	160	59	**M**	2.2	**3**
6	160	60	**M**	1.4	**1**
7	163	60	**M**	2.2	**3**
8	163	61	**M**	2.0	**2**
9	160	64	**L**	3.2	**5**
10	163	64	L	3.6	
11	165	61	L	4.0	
12	165	62	L	4.1	
13	165	65	L	5.7	
14	168	62	L	7.1	
15	168	63	L	7.3	
16	168	66	L	8.6	
17	170	63	L	9.2	
18	170	64	L	9.5	
19	170	68	L	11.4	
20					
21	**161**	**61**			

Calculate KNN manually

In the graph below, binary dependent variable (T-shirt size) is displayed in blue and orange color. 'Medium T-shirt size' is in blue color and 'Large T-shirt size' in orange color. New customer information is exhibited in yellow circle. Four blue highlighted data points and one orange highlighted data point are close to yellow circle. so the prediction for the new case is blue highlighted data point which is Medium T-shirt size.



KNN: Visual Representation

Assumptions of KNN

1. Standardization

When independent variables in training data are measured in different units, it is important to standardize variables before calculating distance. For example, if one variable is based on height in cms, and the other is based on weight in kgs then height will influence more on the distance calculation. In order to make them comparable we need to standardize them which can be done by any of the following methods :

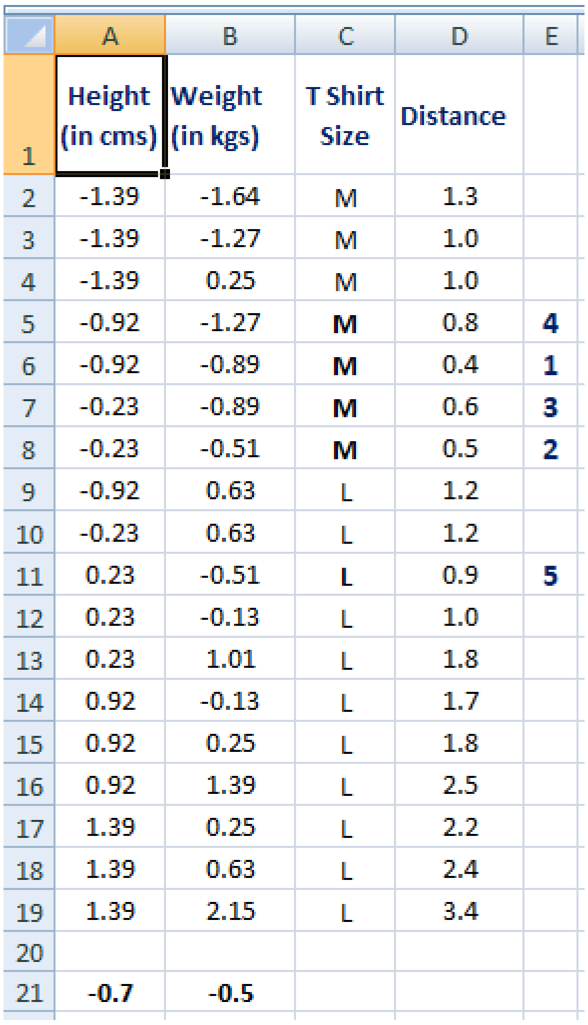
$$Xs = \frac{X - mean}{s.d.}$$

$$Xs = \frac{X - mean}{max - min}$$

$$Xs = \frac{X - min}{\max - min}$$

Standardization

After standardization, 5th closest value got changed as height was dominating earlier before standardization. Hence, it is important to standardize predictors before running K-nearest neighbor algorithm.



	A	B	C	D	E
1	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
2	-1.39	-1.64	M	1.3	
3	-1.39	-1.27	M	1.0	
4	-1.39	0.25	M	1.0	
5	-0.92	-1.27	**M**	0.8	**4**
6	-0.92	-0.89	**M**	0.4	**1**
7	-0.23	-0.89	**M**	0.6	**3**
8	-0.23	-0.51	**M**	0.5	**2**
9	-0.92	0.63	L	1.2	
10	-0.23	0.63	L	1.2	
11	0.23	-0.51	**L**	0.9	**5**
12	0.23	-0.13	L	1.0	
13	0.23	1.01	L	1.8	
14	0.92	-0.13	L	1.7	
15	0.92	0.25	L	1.8	
16	0.92	1.39	L	2.5	
17	1.39	0.25	L	2.2	
18	1.39	0.63	L	2.4	
19	1.39	2.15	L	3.4	
20					
21	**-0.7**	**-0.5**			

Knn after standardization

2. Outlier

Low k-value is sensitive to outliers and a higher K-value is more resilient to outliers as it considers more voters to decide prediction.

K-nearest neighbor algorithm (KNN) is part of supervised learning that has been used in many applications in the field of data mining, statistical pattern recognition and many others.

KNN is a method for classifying objects based on closest training examples in the feature space.

An object is classified by a majority vote of its neighbors. K is always a positive integer. The

neighbors are taken from a set of objects for which the correct classification is known.
It is usual to use the Euclidean distance, though other distance measures such as the Manhattean distance could in principle be used instead.
The algorithm on how to compute the K-nearest neighbors is as follows:

1. Determine the parameter K = number of nearest neighbors beforehand. This value is all up to you.
2. Calculate the distance between the query-instance and all the training samples. You can use any distance algorithm.
3. Sort the distances for all the training samples and determine the nearest neighbor based on the K-th minimum distance.
4. Since this is supervised learning, get all the Categories of your training data for the sorted value which fall under K.
5. Use the majority of nearest neighbors as the prediction value.

Conclusion- KNN implemented using openmp in parallel environment.

Questions:

1. How is KNN different from k-means clustering?
2. In k-NN what will happen when you increase/decrease the value of k?
3. What would be the relation between the time taken by 1-NN,2-NN,3-NN**.**
4. How KNN can be implemented using parallel algorithm?
5. Why k- value is odd?

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:4**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		
Title: Study of MPI Cluster		

Problem Statement

Study of MPI Cluster building steps - MPI Cluster setup and overview of different routines.

a. Different steps to build a MPI cluster over LAN.

b. Master-Slave concept and different MPI routines.

Prerequisites

- **Operating System:** The Operating System is Ubuntu 18.04.
- **MPI:** We could either use OpenMPI or MPICH. OpenMPI (version 2.1.1). List Open MPI available versions

```
apt list -a openmpi-bin

sudo apt-get install openmpi-bin //To install open-mpi
```

Steps to Create an MPI Cluster

Step 1: Configure your hosts file

We are going to communicate between the computers and we don't want to type in the IP addresses every so often. Instead, we can give a name to the various nodes in the network that we wish to communicate with. hosts file is used by the device operating system to map hostnames to IP addresses.

Example of a host file in **master**.

```
sudo nano /etc/hosts
```

Add these host IPs and Worker IPs in that file :

```
#MPI CLUSTERS

172.20.36.120 manager

172.20.36.153 worker1

172.20.36.143 worker2

172.20.36.116 worker3
```

For worker (**slave**) node

Example of a host file for worker2

```
#MPI CLUSTER SETUP

172.20.36.120   manager

172.20.36.143   worker2
```

Step 2: Create a new user

We can operate the cluster using existing users. It's better to create a new user to keep things simpler. Create new user accounts with the same username in all the machines to keep things simple.

To add a new user:

```
sudo adduser mpiuser
```

Making mpiuser a sudoer :

```
sudo usermod -aG sudo mpiuser
```

Step 3: Setting up SSH

Machines are going to be talking over the network via SSH and share data via NFS. Follow the below process for both manager and the worker nodes.

To install ssh in the system.

```
sudo apt-get install openssh-server
```

Log in to the newly created user by

```
su - mpiuser
```

Navigate to ~/.ssh folder and

```
ssh-keygen -t rsa

cd .ssh/

cat id_rsa.pub >> authorized_keys

ssh-copy-id worker1
```

For example:

mpiuser@tele-h81m-s138:~/.ssh$ ssh-copy-id worker2

Now you can connect to worker nodes without entering passwords

```
ssh worker2
```

In worker nodes use

```
ssh-copy-id manager
```

Step 4: Setting up NFS

We share a directory via NFS in the manager which the worker mounts to exchange data.

NFS-Server for the master node :
Install the required packages by

$ sudo apt-get install nfs-kernel-server

We need to create a folder that we will share across the network. In our case, we used "cloud". To export the cloud directory, we need to create an entry in /etc/exports

```
sudo nano /etc/exports
```

Add

*/home/mpiuser/cloud *(rw,sync,no_root_squash,no_subtree_check)*

Instead of *, we can specifically give out the IP address to which we want to share this folder, or we can use *.

For Example:

/home/mpiuser/cloud 172.20.36.121(rw,sync,no_root_squash,no_subtree_check)

After an entry is made, run the following.

```
$ exportfs -a
```

Run the above command, every time any change has been made to /etc/exports.

Use sudo exportfs -a if the above statement doesn't work.

If required, restart the NFS server

$ sudo service nfs-kernel-server restart

> NFS-worker for the client nodes

Install the required packages

```
$ sudo apt-get install nfs-common
```

Create a directory in the worker's machine with the same name – "cloud"

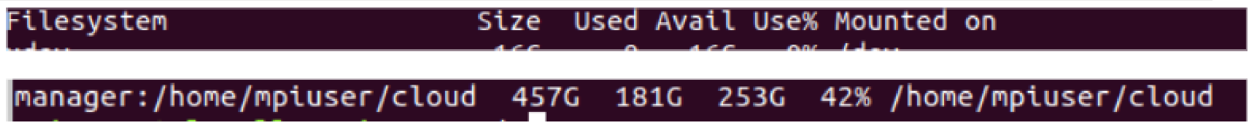
```
$ mkdir cloud
```

And now, mount the shared directory like

$ sudo mount -t nfs manager:/home/mpiuser/cloud ~/cloud

To check the mounted directories,

```
$ df -h
```



```
Filesystem                   Size  Used Avail Use% Mounted on
manager:/home/mpiuser/cloud  457G  181G  253G  42% /home/mpiuser/cloud
```

This is how it would show up

To make the mount permanent so you don't have to manually mount the shared directory every time you do a system reboot, you can create an entry in your file systems table – i.e., /etc/fstab file like this:

```
$ nano /etc/fstab
```

Add

```
#MPI CLUSTER SETUP

manager:/home/mpiuser/cloud /home/mpiuser/cloud nfs
```

Step 5: Running MPI programs

Navigate to the NFS shared directory (“cloud” in our case) and create the files there[or we can paste just the output files). To compile the code, the name of which let’s say is mpi_hello.c, we will have to compile it the way given below, to generate an executable mpi_hello.

```
$ mpicc -o mpi_hello mpi_hello.c
```

To run it only in the master machine, we do

```
$ mpirun -np 2 ./mpi_helloBsend
```

np – No. of processes = 2

To run the code within a cluster

```
$ mpirun -hostfile my_host ./mpi_hello
```

Here, the *my_host* file determines the IP Addresses and number of processes to be run.
Sample Hosts File :

```
manager slots=4 max_slots=40

worker1 slots=4 max_slots=40

worker2  max_slots=40

worker3 slots=4 max_slots=40
```

Alternatively,

```
$ mpirun -np 5 -hosts worker,localhost ./mpi_hello
```

Note: Hostnames can also be substituted with IP addresses.

Conclusion- MPI cluster setup steps are completed and sample program executed successfully.

Questions-

1. What is prerequisite for setup of MPI Cluster?
2. How to run program in MPI?

3. How to run program on multiple processor?
4. Which command is used to compile the MPI Program?
5. What is MPI?

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:5**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		
Title: Parallel implementation of all pair shortest path algorithm using openmp/MPI		

Problem Statement

Input:- Graph with weighted edges

Output- All pairs shortest path distances.

Theory:-

Parallel computing has become essential for solving computationally intensive problems efficiently. Dijkstra's algorithm is a well-known graph algorithm used to find the shortest paths from a single source vertex to all other vertices in a graph. When dealing with large graphs, it becomes necessary to parallelize the algorithm to achieve faster results.

Approaches :

Data Parallelism using OpenMP

Task Parallelism using OpenMP

Approach 1: Data Parallelism using OpenMP

Divide the nodes into chunks and assign each chunk to the thread. Each thread computes the shortest path for its assigned nodes in the parallel.

Syntax:

#include <omp.h>

#pragma omp parallel for

```
for (int i = 0; i < num_nodes; ++i) {

// Compute shortest path for node i

}
```

Implementation :

```
#include <iostream>
#include <vector>
#include <limits>
#include <omp.h>
#define INF std::numeric_limits<int>::max()
void GFG(const std::vector<std::vector<int>>& graph, int start) {
    int num_nodes = graph.size();
    std::vector<int> dist(num_nodes, INF);
    std::vector<bool> visited(num_nodes, false);
    dist[start] = 0;
    #pragma omp parallel for
    for (int i = 0; i < num_nodes; ++i) {
        int u = -1, min_dist = INF;
        // Find the node with the shortest distance
        for (int j = 0; j < num_nodes; ++j) {
            if (!visited[j] && dist[j] < min_dist) {
                u = j;
                min_dist = dist[j];
            }
        }
        if (u == -1) break;
        // Relax adjacent nodes
        for (int v = 0; v < num_nodes; ++v) {
            if (!visited[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
        visited[u] = true;
    }
    // Print the shortest distances
    std::cout << "Vertex \t Distance from Source\n";
    for (int i = 0; i < num_nodes; ++i) {
        std::cout << i << "\t\t" << dist[i] << "\n";
    }
}
int main() {
    std::vector<std::vector<int>> graph = {
        {0, 4, 0, 0, 0, 0, 0, 8, 0},
        {4, 0, 8, 0, 0, 0, 0, 11, 0},
        {0, 8, 0, 7, 0, 4, 0, 0, 2},
        {0, 0, 7, 0, 9, 14, 0, 0, 0},
        {0, 0, 0, 9, 0, 10, 0, 0, 0},
        {0, 0, 4, 14, 10, 0, 2, 0, 0},
        {0, 0, 0, 0, 0, 2, 0, 1, 6},
        {8, 11, 0, 0, 0, 0, 1, 0, 7},
        {0, 0, 2, 0, 0, 0, 6, 7, 0}
    };
    GFG(graph, 0);
    return 0;
}
```

Approach 2: Task Parallelism using OpenMP

Create parallel tasks for each node. Each task computes the shortest path for its assigned node independently.

Syntax:

```
#include <omp.h>

#pragma omp parallel

{

#pragma omp single nowait

{

for (int i = 0; i < num_nodes; ++i) {

#pragma omp task

{

// Compute shortest path for node i

}

}
}

}
```

Implementation :

```
#include <iostream>

#include <limits>

#include <omp.h>

#include <vector>

#define INF std::numeric_limits<int>::max()


void GFG(const std::vector<std::vector<int> >& graph,

        int start)

{

   int num_nodes = graph.size();

   std::vector<int> dist(num_nodes, INF);

   std::vector<bool> visited(num_nodes, false);

   dist[start] = 0;

#pragma omp parallel

   {

#pragma omp single nowait

       {

           for (int i = 0; i < num_nodes; ++i) {

#pragma omp task

               {

                   int u = -1, min_dist = INF;
```

```
                // Find the node with shortest distance
                for (int j = 0; j < num_nodes; ++j) {
                    if (!visited[j]
                        && dist[j] < min_dist) {
                        u = j;
                        min_dist = dist[j];
                    }
                }
                if (u != -1) {
                    // The Relax adjacent nodes
                    for (int v = 0; v < num_nodes;
                         ++v) {
                        if (!visited[v] && graph[u][v]
                            && dist[u] + graph[u][v]
                                   < dist[v]) {
                            dist[v]
                                = dist[u] + graph[u][v];
                        }
                    }
                    visited[u] = true;
                }
            }
        }
    }
}
```

```
    // Print the shortest distances
    std::cout << "Vertex \t Distance from Source\n";
    for (int i = 0; i < num_nodes; ++i) {
        std::cout << i << "\t\t" << dist[i] << "\n";
    }
}
int main()
{
    std::vector<std::vector<int> > graph
        = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
            { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
            { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
            { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
            { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
            { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
            { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
            { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
            { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
    GFG(graph, 0);
    return 0;
}
```

Conclusion:- Parallel all pair shortest path algorithm is implemented using openMP

Questions:-

1. Can you explain the All-Pairs Shortest Path (APSP) problem and why it is important in graph theory?

2. How does OpenMP help in parallelizing the APSP algorithm, and what are the main advantages of using it?
3. Describe how you would parallelize the Floyd-Warshall algorithm for APSP using OpenMP.
4. What are the potential issues with data dependencies when parallelizing the APSP algorithm using OpenMP, and how can they be mitigated?
5. How would you evaluate the performance of your parallel APSP implementation using OpenMP, and what metrics would you consider?

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:6**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		
Title: Program to implement collective communication using MPI routines.		

Problem Statement

Program to implement collective communication using MPI routines.

a. Gather, Scatter and Broadcast operations. b. Matrix-Vector multiplication execution.

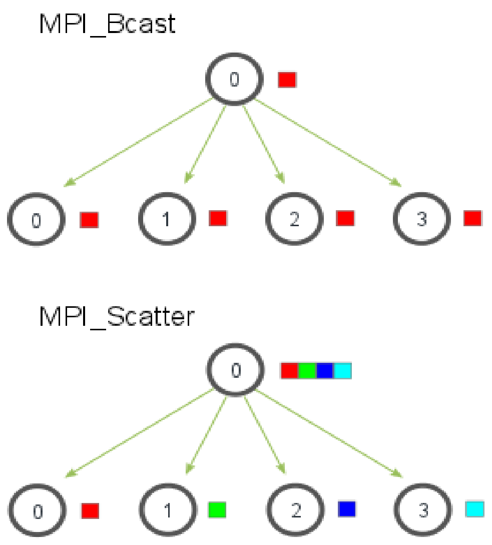
Input:- Matrix and vector with data

Output:- Multiplication of vector and matrix using MPI

Theory:-

An introduction to MPI_Scatter

MPI_Scatter is a collective routine that is very similar to MPI_Bcast. MPI_Scatter involves a designated root process sending data to all processes in a communicator. The primary difference between MPI_Bcast and MPI_Scatter is small but important.



MPI_Bcast sends the *same* piece of data to all processes while MPI_Scatter sends *chunks of an array* to different processes. Check out the illustration below for further clarification. In the illustration, MPI_Bcast takes a single data element at the root process (the red box) and copies it to all other processes. MPI_Scatter takes an array of elements and distributes the elements in the order of process rank. The first element (in red) goes to process zero, the second element (in green) goes to process one, and so on. Although the root process (process zero) contains the entire array of data, MPI_Scatter will copy the appropriate element into the receiving buffer of the process. Here is what the function prototype of MPI_Scatter looks like.

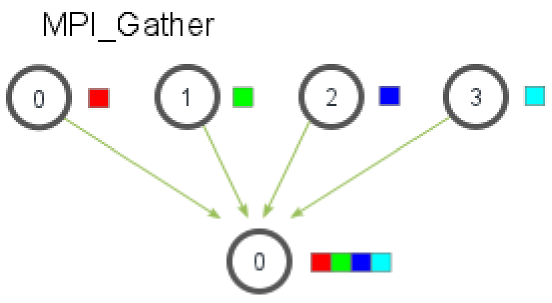
```
MPI_Scatter(
   void* send_data,
   int send_count,
   MPI_Datatype send_datatype,
   void* recv_data,
   int recv_count,
   MPI_Datatype recv_datatype,
   int root,
   MPI_Comm communicator)
```

Yes, the function looks big and scary, but let's examine it in more detail. The first parameter, send_data, is an array of data that resides on the root process. The second and third parameters, send_count and send_datatype, dictate how many elements of a specific MPI Datatype will be sent to each process. If send_count is one and send_datatype is MPI_INT, then process zero gets the first integer of the array, process one gets the second integer, and so on. If send_count is two, then process zero gets the first and second integers, process one gets the third and fourth, and so on. In practice, send_count is often equal to the number of elements in the array divided by the number of processes. What's that you say? The number of elements isn't divisible by the number of processes? Don't worry, we will cover that in a later lesson :-)

The receiving parameters of the function prototype are nearly identical in respect to the sending parameters. The recv_data parameter is a buffer of data that can hold recv_count elements that have a datatype of recv_datatype. The last parameters, root and communicator, indicate the root process that is scattering the array of data and the communicator in which the processes reside.

An introduction to MPI_Gather

MPI_Gather is the inverse of MPI_Scatter. Instead of spreading elements from one process to many processes, MPI_Gather takes elements from many processes and gathers them to one single process. This routine is highly useful to many parallel algorithms, such as parallel sorting and searching. Below is a simple illustration of this algorithm.



Similar to MPI_Scatter, MPI_Gather takes elements from each process and gathers them to the root process. The elements are ordered by the rank of the process from which they were received. The function prototype for MPI_Gather is identical to that of MPI_Scatter.

```
MPI_Gather(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm communicator)
```

In MPI_Gather, only the root process needs to have a valid receive buffer. All other calling processes can pass NULL for recv_data. Also, don’t forget that the *recv_count* parameter is the count of elements received *per process*, not the total summation of counts from all processes. This can often confuse beginning MPI programmers.

Broadcasting with MPI_Bcast

A broadcast is one of the standard collective communication techniques. During a broadcast, one process sends the same data to all processes in a communicator. One of the main uses of broadcasting is to send out user input to a parallel program, or send out configuration parameters to all processes.

The communication pattern of a broadcast looks like this:

MPI_Bcast pattern

In this example, process zero is the root process, and it has the initial copy of data. All of the other processes receive the copy of data.

In MPI, broadcasting can be accomplished by using MPI_Bcast. The function prototype looks like this:

```
MPI_Bcast(
    void* data,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm communicator)
```

Although the root process and receiver processes do different jobs, they all call the same MPI_Bcast function. When the root process (in our example, it was process zero) calls MPI_Bcast, the data variable will be sent to all other processes. When all of the receiver processes call MPI_Bcast, the data variable will be filled in with the data from the root process.

MPI Program for Matrix-Vector Multiplication

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define N 4  // Size of the matrix and vector (assuming square matrix)
```

```
int main(int argc, char** argv) {
    int rank, size;
    double A[N][N];     // Matrix A (NxN)
    double x[N];        // Vector x (Nx1)
    double y[N];        // Result vector y (Nx1)
    double local_A[N];  // Local chunk of matrix A for each process
    double local_y;     // Local result for each process
    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (size != N) {
        if (rank == 0) {
            printf("This program requires exactly %d processes.\n", N);
        }
        MPI_Finalize();
        return -1;
    }
    // Initialize the matrix A and vector x in the root process
    if (rank == 0) {
        for (int i = 0; i < N; i++) {
            x[i] = 1.0;  // Initialize vector x (for simplicity, all elements are 1)
            for (int j = 0; j < N; j++) {
                A[i][j] = i + 1;  // Initialize matrix A (for simplicity, row-wise)
            }
        }
    }
    // Scatter rows of matrix A to all processes
    MPI_Scatter(A, N, MPI_DOUBLE, local_A, N, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
    // Broadcast vector x to all processes
    MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    // Each process performs local computation (matrix row * vector)
    local_y = 0.0;
    for (int i = 0; i < N; i++) {
        local_y += local_A[i] * x[i];
    }
    // Gather the results from all processes to form the result vector y
    MPI_Gather(&local_y, 1, MPI_DOUBLE, y, 1, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
    // Print the result vector y in the root process
    if (rank == 0) {
        printf("Result vector y:\n");
        for (int i = 0; i < N; i++) {
            printf("%f\n", y[i]);
        }
    }
    // Finalize MPI
    MPI_Finalize();
    return 0;
```

}

Explanation

1. Initialization:

   - MPI is initialized, and each process determines its rank and the total number of processes.

   - The program assumes a fixed size `N` for the matrix and vector. Each process is responsible for one row of the matrix, so the program requires exactly `N` processes.

2. Matrix and Vector Initialization:

   - The root process (rank 0) initializes the matrix `A` and vector `x`. The matrix `A` is initialized such that each row has the same values for simplicity, and vector `x` is filled with `1.0` for simplicity.

3. Scatter Operation (`MPI_Scatter`):

   - The rows of the matrix `A` are distributed to all processes. Each process receives one row (`local_A`) of the matrix.

4. Broadcast Operation (`MPI_Bcast`):

   - The vector `x` is broadcast to all processes. Each process receives the complete vector `x`.

5. Local Computation:

   - Each process computes its portion of the matrix-vector multiplication. Specifically, it computes the dot product of its local row `local_A` with vector `x` to produce a local result `local_y`.

6. Gather Operation (`MPI_Gather`):

   - The local results `local_y` from all processes are gathered back into the result vector `y` in the root process.

7. Output:

   - The root process prints the final result vector `y`.

Example Input/Output

Given the initial matrix and vector:

- Matrix `A`:

 [1, 1, 1, 1]

 [2, 2, 2, 2]

 [3, 3, 3, 3]

 [4, 4, 4, 4]

- Vector `x`: `[1, 1, 1, 1]`

The output will be the result vector `y`:

```
4.000000

8.000000

12.000000

16.000000
```

This output demonstrates that the matrix-vector multiplication has been correctly distributed across multiple processes using MPI collective communication operations.

Conclusion- Matrix vector multiplication is implemented using MPI routines.

Questions:-

1. What are MPI Gather, Scatter, and Broadcast operations, and when would you use each one?
2. How does the MPI_Gather function work, and what are the essential parameters it requires?
3. Describe how MPI_Scatter is used in a parallel program and provide an example scenario where it would be useful.
4. Explain the purpose of the MPI_Bcast function and its typical usage in parallel algorithms.
5. How can collective communication operations like Scatter and Gather be used to implement matrix-vector multiplication in MPI?

	Sanjivani Rural Education Society's SANJIVANI COLLEGE OF ENGINEERING **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:7**	**Revision : 00** **Dated : ________**
Department : Department of Computer Engineering		
Title: Program to implement Merge sort/ Graph Computation algorithm using MPI routines.		

Problem Statement

Program to implement Merge sort/ Graph Computation algorithm using MPI routines.

a. Steps in parallelizing Merge Sort/ Matrix Partitioning.

Input:- array element

Output:- sorted array using MPI

Theory:-

Parallel Merge Sort using MPI

Parallel Merge Sort using MPI involves dividing the array to be sorted among multiple processes, sorting these subarrays in parallel, and then merging the sorted subarrays to obtain the final sorted array. This approach leverages the power of distributed memory systems to handle large datasets efficiently.

Steps in Parallelizing Merge Sort

1. Initialization:

   - Initialize the MPI environment.

   - Determine the rank and size of the MPI communicator.

2. Data Distribution:

   - If the process is the root, divide the array into subarrays.

   - Use `MPI_Scatter` to distribute subarrays to all processes.

3. Local Sorting:

- Each process performs a local merge sort on its subarray.

4. Recursive Merging:

- Implement a recursive merging strategy:
  - Pair up processes.
  - Each pair of processes exchanges their sorted subarrays.
  - Merge the received subarray with the local subarray.
  - Continue merging at higher levels of the hierarchy until the root process obtains the fully merged array.

5. Collecting Results:

- Use `MPI_Gather` or custom communication to gather the final sorted array at the root process.

6. Finalization:

- Finalize the MPI environment.

Matrix Partitioning for Parallel Graph Computation

1. Matrix Representation:

- Represent the graph as an adjacency matrix or adjacency list.

2. Partitioning Strategy:

- Use row-wise or column-wise partitioning for distributing the adjacency matrix among processes.

3. Data Distribution:

- Distribute rows/columns of the matrix to different processes using `MPI_Scatter`.

4. Local Computation:

- Each process performs computation on its partition of the matrix, such as updating distances in shortest path algorithms.

5. Communication:

- Exchange intermediate results, such as distance updates, using collective communication routines like `MPI_Allgather` or `MPI_Allreduce`.

6. Aggregation:

   - Aggregate results from all processes to obtain the final solution.

7. Finalization:

   - Finalize the MPI environment.

Pseudo Code: Parallel Merge Sort with MPI

```
// Initialize MPI environment
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
// Root process: Initialize and distribute data
if (rank == 0) {
    // Input: full array `data` of size `n`
    n = get_size_of_data();
    data = allocate_array(n);
    // Divide data among processes
    chunk_size = n / size;
    remainder = n % size;
    // Send chunks to all processes
    for (i = 1; i < size; i++) {
        offset = i * chunk_size;
        if (i == size - 1) {
            chunk_size += remainder; // Last process handles remainder
        }
        MPI_Send(&chunk_size, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        MPI_Send(&data[offset], chunk_size, MPI_INT, i, 0, MPI_COMM_WORLD);
    }
    // Root process keeps its own chunk
    chunk_size = chunk_size;
    local_data = allocate_array(chunk_size);
    copy_data(local_data, data, chunk_size);
} else {
    // Non-root processes: Receive their chunk size and data
    MPI_Recv(&chunk_size, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    local_data = allocate_array(chunk_size);
    MPI_Recv(local_data, chunk_size, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
}
// Each process: Perform local merge sort on its chunk
merge_sort(local_data, 0, chunk_size - 1);
// Recursive merging phase
for (step = 1; step < size; step = 2 * step) {
    if (rank % (2 * step) == 0) {
        // Receiving process
        if (rank + step < size) {
            // Receive chunk from the process `rank + step`
            MPI_Recv(&recv_chunk_size, 1, MPI_INT, rank + step, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            recv_data = allocate_array(recv_chunk_size);
    MPI_Recv(recv_data, recv_chunk_size, MPI_INT, rank + step, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            // Merge the two chunks
            merged_data = allocate_array(chunk_size + recv_chunk_size);
            merge(local_data, chunk_size, recv_data, recv_chunk_size, merged_data);


            // Update local data
            chunk_size += recv_chunk_size;
            free(local_data);
            local_data = merged_data;
            free(recv_data);
        }
    } else {
        // Sending process
        pair_rank = rank - step;
        MPI_Send(&chunk_size, 1, MPI_INT, pair_rank, 0, MPI_COMM_WORLD);
        MPI_Send(local_data, chunk_size, MPI_INT, pair_rank, 0, MPI_COMM_WORLD);
        break; // Exit loop after sending data
    }
}
// Root process: Final result is in `local_data`
if (rank == 0) {
    // `local_data` contains the fully sorted array
    copy_data_to_final_result(local_data, n);
}
```

```
// Finalize MPI environment

MPI_Finalize();
```

Conclusion:- Parallel merge sort is implemented using MPI.

Questions:-

1. How is the input array distributed among the processes in the parallel Merge Sort using MPI?
2. What does each process do after receiving its chunk of data in the parallel Merge Sort?
3. How is the merging of sorted subarrays handled across processes in the parallel Merge Sort?
4. Where and how is the final sorted array assembled in the parallel Merge Sort using MPI?
5. What is a key challenge in ensuring efficient parallel performance in MPI-based Merge Sort?

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:8**	**Revision : 00** **Dated :** ________
Department : Department of Computer Engineering		
Title: Program to implement Map-Reduce parallelism for Warehouse -Scale Computer.		

Problem Statement

Program to implement Map-Reduce parallelism for Warehouse -Scale Computer.

a. Parallelism using Map-Reduce programming model.

b. Example of word count process with key-value pair.

Input- txt file containing sentence.

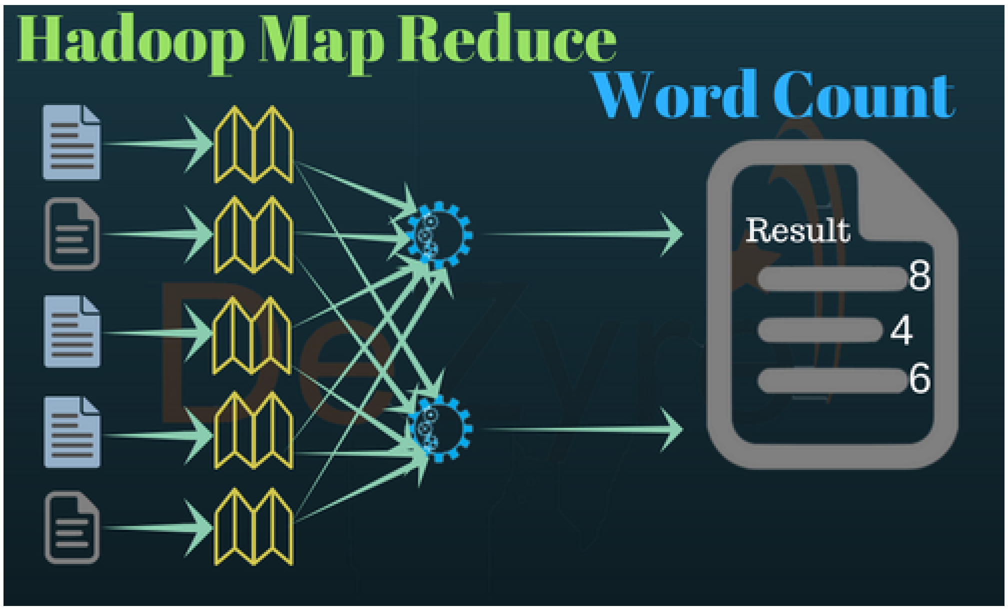
Output- Word count with frequency.

Theory-

This hadoop assignment aims to give hadoop developers a great start in the world of hadoop mapreduce programming by giving them hands-on experience in developing their first hadoop based WordCount application. Hadoop MapReduce WordCount example is a standard example where hadoop developers begin their hands-on programming with. This will help Hadoop developers learn how to implement WordCount example code in MapReduce to count the number of occurrences of a given word in the input file.

Prerequisites to follow this Hadoop WordCount Example Tutorial

i. Hadoop Installation must be completed successfully.
ii. Single node hadoop cluster must be configured and running.
iii. Eclipse must be installed as the MapReduce WordCount example will be run from eclipse IDE.



Word Count - Hadoop Map Reduce Example – How it works?

Hadoop WordCount operation occurs in 3 stages –

i. Mapper Phase
ii. Shuffle Phase
iii. Reducer Phase

Hadoop WordCount Example- Mapper Phase Execution

The text from the input text file is tokenized into words to form a key value pair with all the words present in the input text file. The key is the word from the input file and value is '1'.

For instance if you consider the sentence "An elephant is an animal". The mapper phase in the WordCount example will split the string into individual tokens i.e. words. In this case, the entire sentence will be split into 5 tokens (one for each word) with a value 1 as shown below –

Key-Value pairs from Hadoop Map Phase Execution-

```
(an,1)
(elephant,1)
(is,1)
(an,1)
(animal,1)
```

If you would like more information about Big Data and Hadoop Certification, please click the orange "Request Info" button on top of this page.

Hadoop WordCount Example- Shuffle Phase Execution

After the map phase execution is completed successfully, shuffle phase is executed automatically wherein the key-value pairs generated in the map phase are taken as input and then sorted in alphabetical order. After the shuffle phase is executed from the WordCount example code, the output will look like this -

```
(an,1)
(an,1)
(animal,1)
(elephant,1)
(is,1)
```

Hadoop WordCount Example- Reducer Phase Execution

In the reduce phase, all the keys are grouped together and the values for similar keys are added up to find the occurrences for a particular word. It is like an aggregation phase for the keys generated by the map phase. The reducer phase takes the output of shuffle phase as input and then reduces the key-value pairs to unique keys with values added up. In our example "An elephant is an animal." is the only word that appears twice in the sentence. After the execution of the reduce phase of MapReduce WordCount example program, appears as a key only once but with a count of 2 as shown below -

```
(an,2)
(animal,1)
(elephant,1)
(is,1)
```

This is how the MapReduce word count program executes and outputs the number of occurrences of a word in any given input file. An important point to note during the execution of the WordCount example is that the mapper class in the WordCount program will execute completely on the entire input file and not just a single sentence. Suppose if the input file has 15 lines then the mapper class will split the words of all the 15 lines and form initial key value pairs for the entire dataset. The reducer execution will begin only after the mapper phase is executed successfully.

Running the WordCount Example in Hadoop MapReduce using Java Project with Eclipse

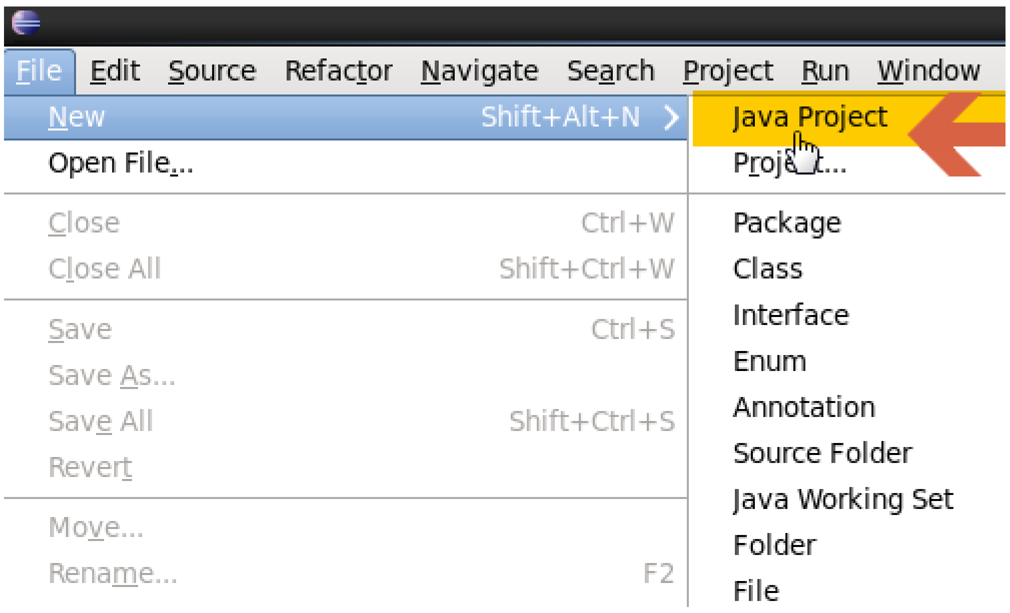
Now, let's create the WordCount java project with eclipse IDE for Hadoop. Even if you are working on Cloudera VM, creating the Java project can be applied to any environment.

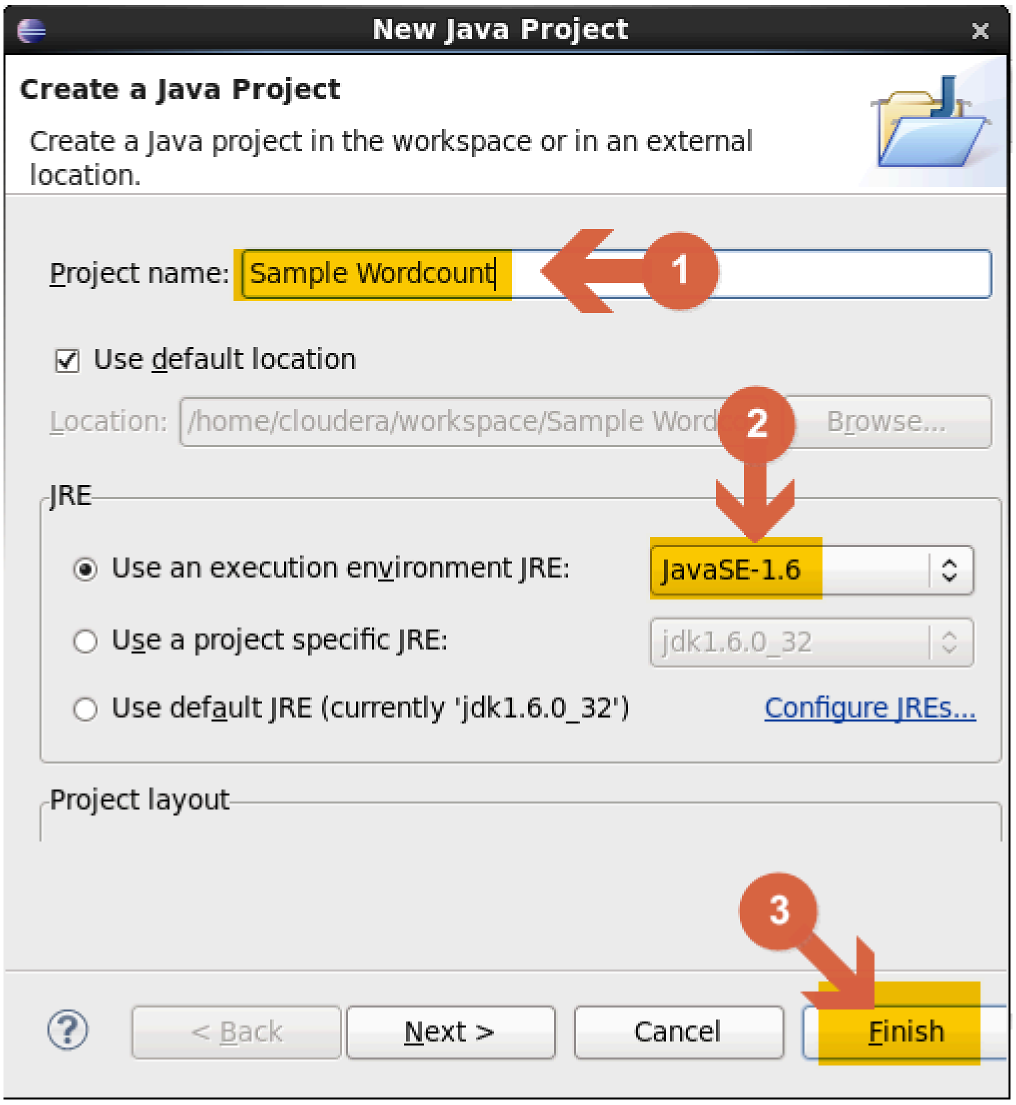
Step 1 –

Let's create the java project with the name "Sample WordCount" as shown below -

File > New > Project > Java Project > Next.

"Sample WordCount" as our project name and click "Finish":




New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: Sample Wordcount

[✓] Use default location

Location: /home/cloudera/workspace/Sample Word

JRE

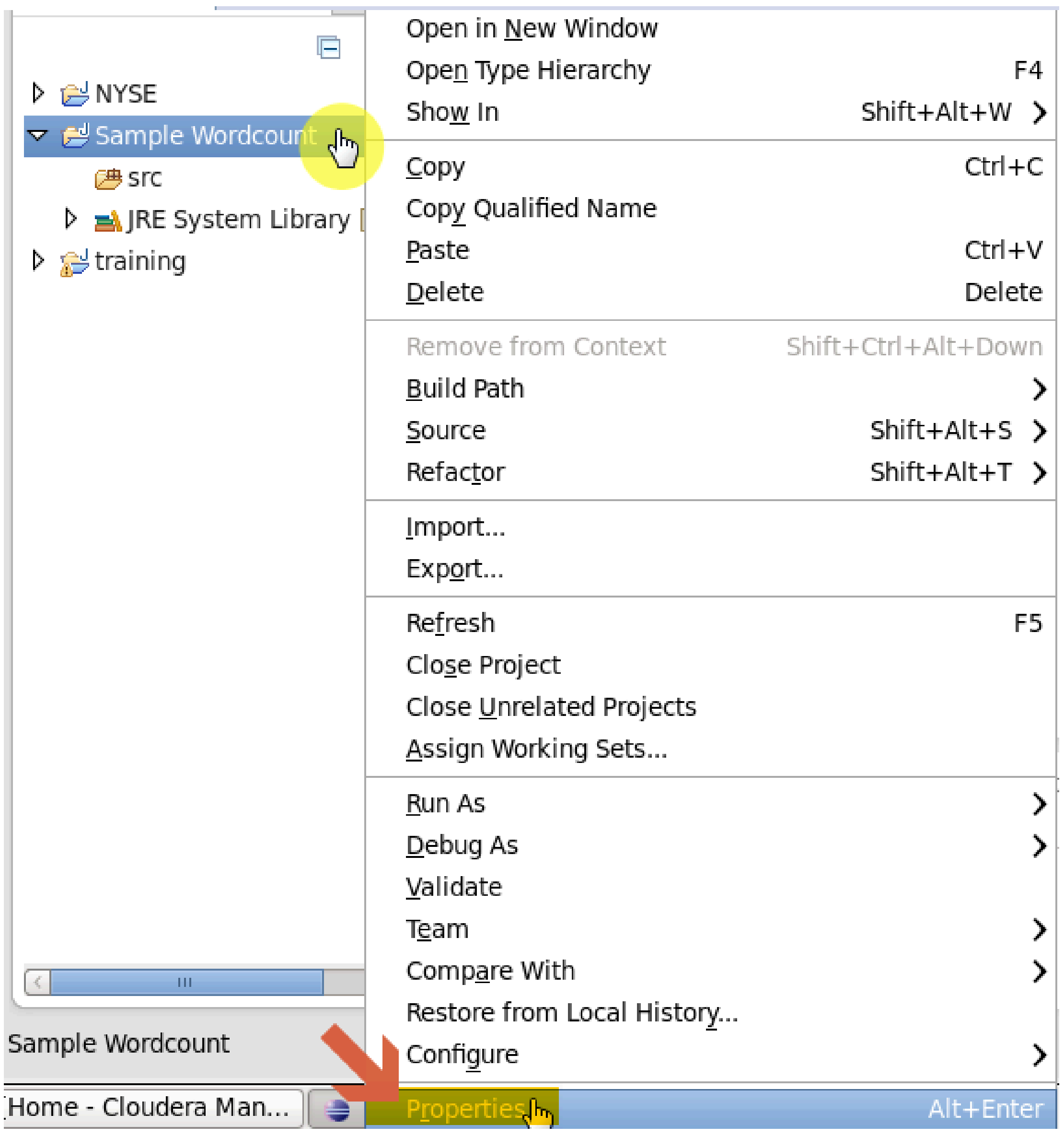
(●) Use an execution environment JRE: JavaSE-1.6

( ) Use a project specific JRE: jdk1.6.0_32

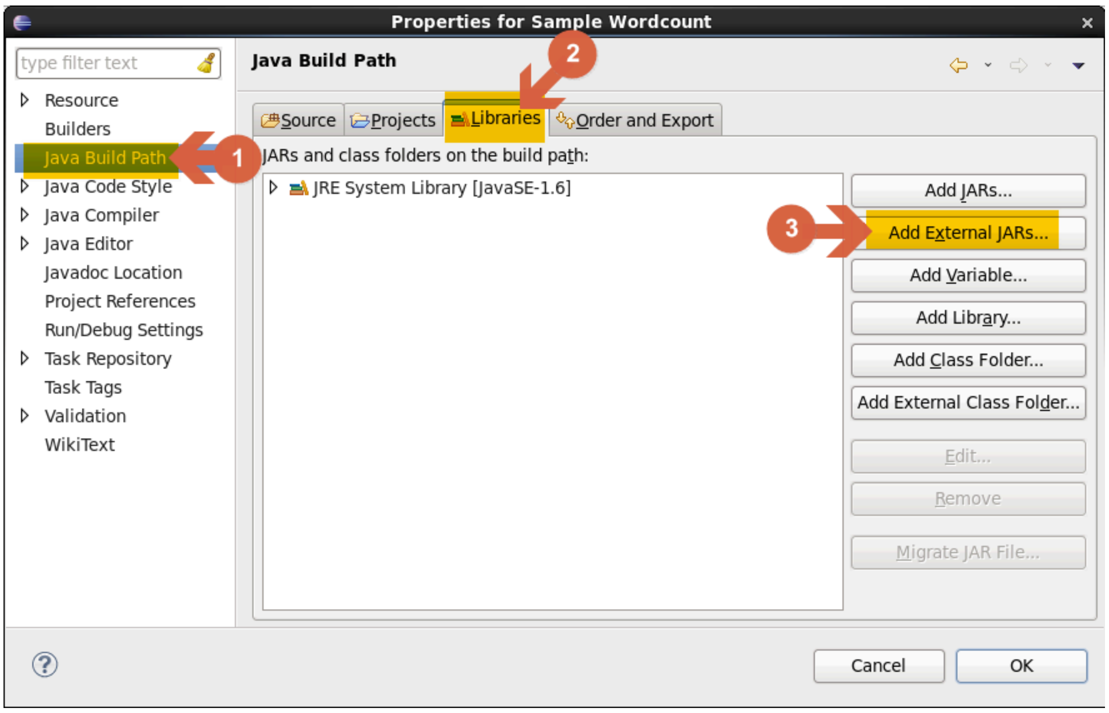
( ) Use default JRE (currently 'jdk1.6.0_32')

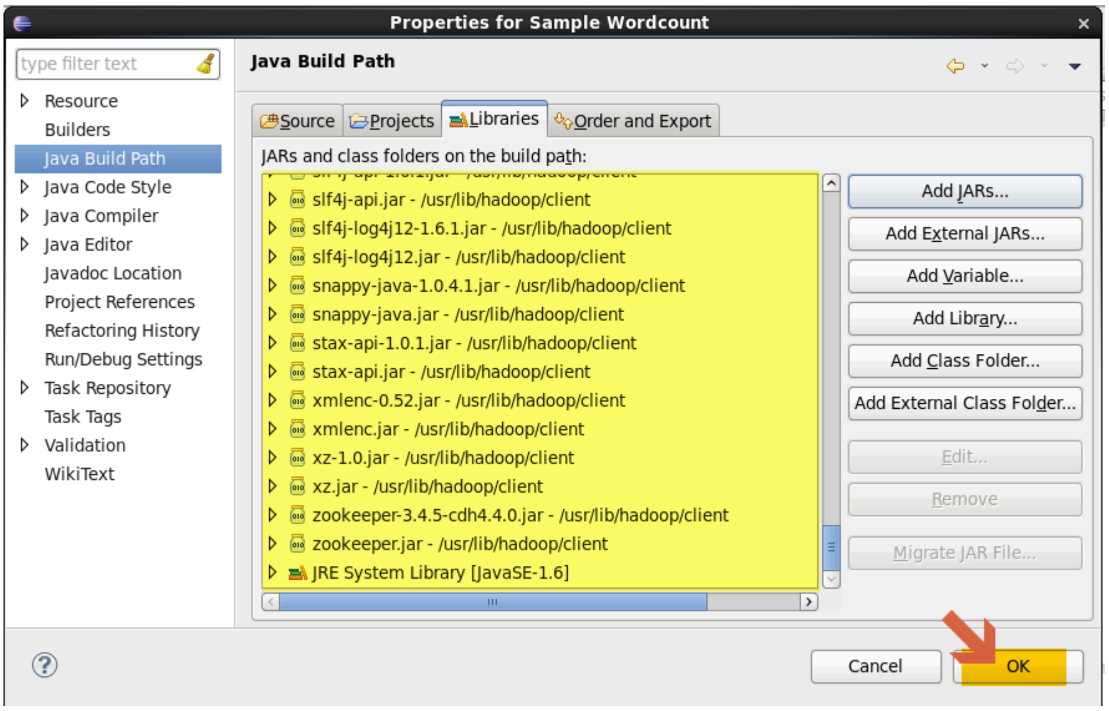
Step 2 -

The next step is to get references to hadoop libraries by clicking on Add JARS as follows –



NYSE
Sample Wordcount
src
JRE System Library
training
Sample Wordcount
Home - Cloudera Man...
Open in New Window
Open Type Hierarchy F4
Show In Shift+Alt+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Shift+Ctrl+Alt+Down
Build Path
Source Shift+Alt+S
Refactor Shift+Alt+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Validate
Team
Compare With
Restore from Local History...
Configure
Properties Alt+Enter

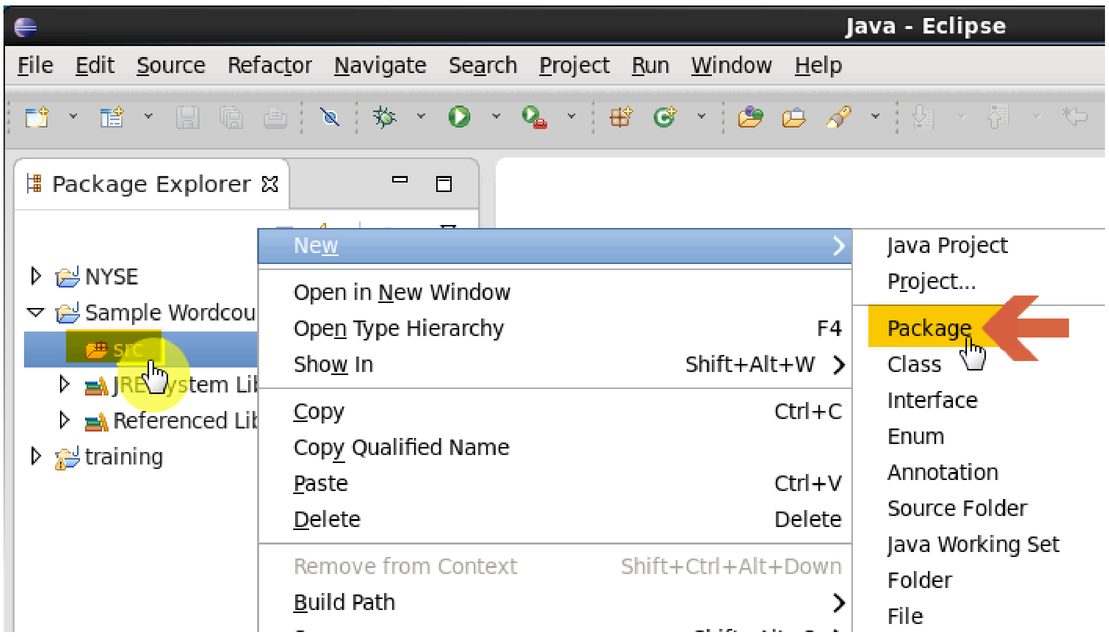


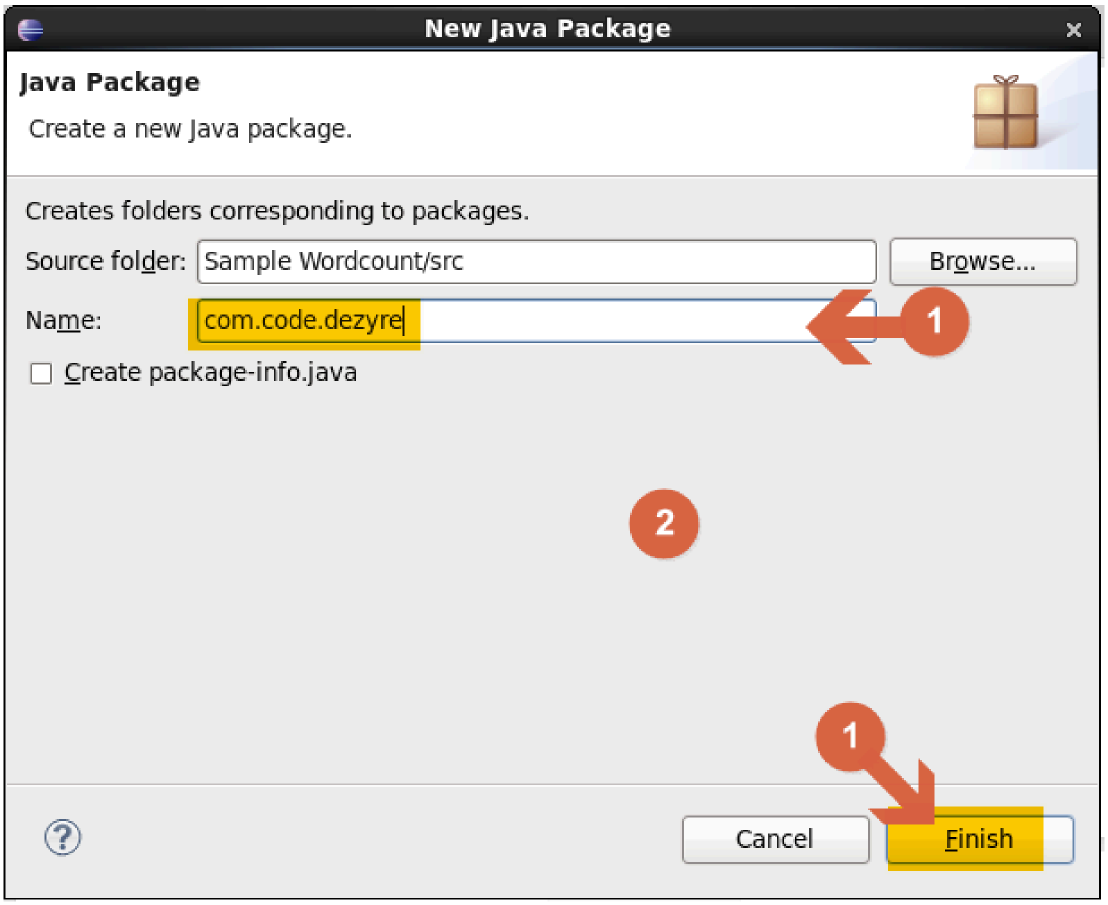


```
slf4j-api.jar - /usr/lib/hadoop/client
slf4j-log4j12-1.6.1.jar - /usr/lib/hadoop/client
slf4j-log4j12.jar - /usr/lib/hadoop/client
snappy-java-1.0.4.1.jar - /usr/lib/hadoop/client
snappy-java.jar - /usr/lib/hadoop/client
stax-api-1.0.1.jar - /usr/lib/hadoop/client
stax-api.jar - /usr/lib/hadoop/client
xmlenc-0.52.jar - /usr/lib/hadoop/client
xmlenc.jar - /usr/lib/hadoop/client
xz-1.0.jar - /usr/lib/hadoop/client
xz.jar - /usr/lib/hadoop/client
zookeeper-3.4.5-cdh4.4.0.jar - /usr/lib/hadoop/client
zookeeper.jar - /usr/lib/hadoop/client
JRE System Library [JavaSE-1.6]
```

Step 3 -

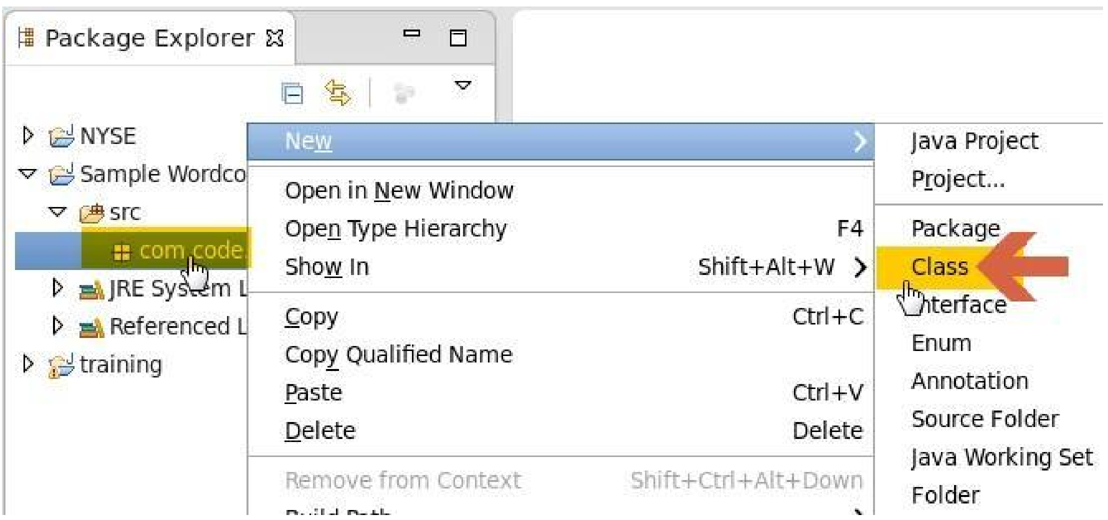
Create a new package within the project with the name com.code.dezyre-

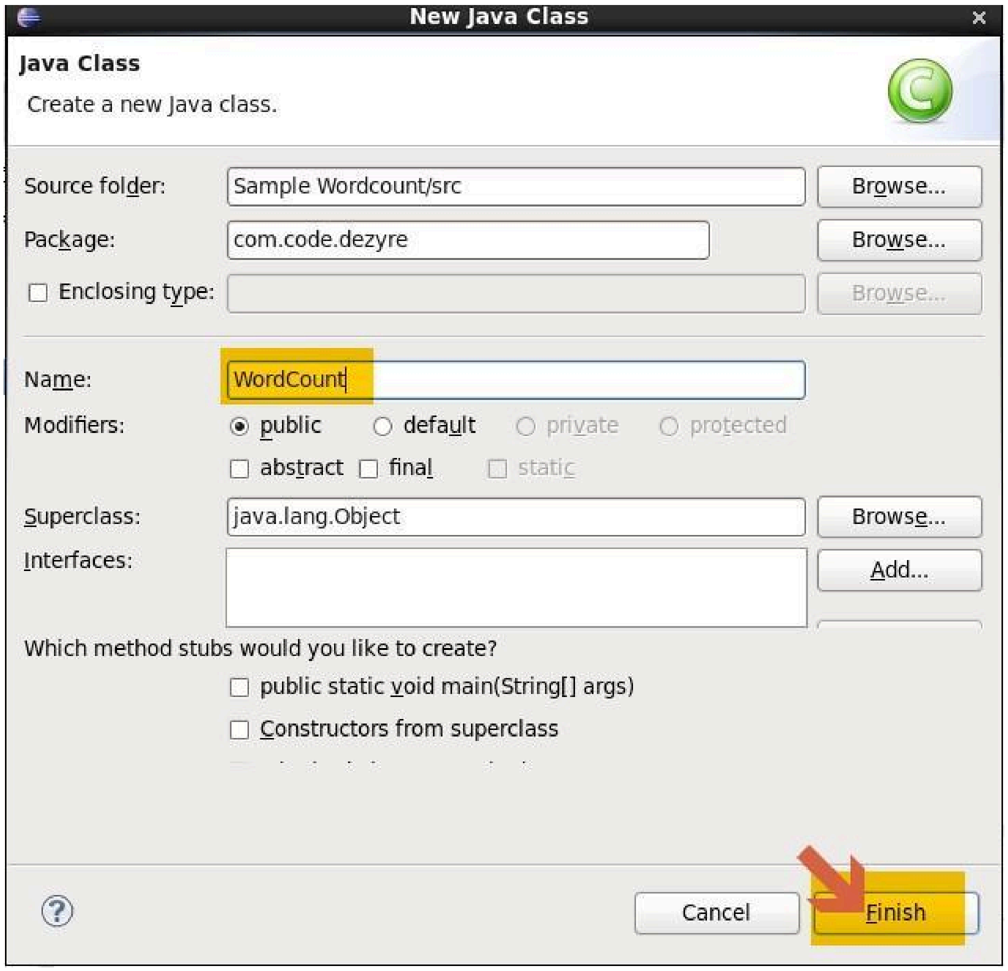



New Java Package
Java Package
Create a new Java package.
Creates folders corresponding to packages.
Source folder: Sample Wordcount/src
Browse...
Name: com.code.dezyre
1
Create package-info.java
2
1
Cancel
Finish

Step 4 –

Now let's implement the WordCount example program by creating a WordCount class under the project com.code.dezyre.




New Java Class

Java Class

Create a new Java class.

Source folder: Sample Wordcount/src

Package: com.code.dezyre

☐ Enclosing type:

Name: WordCount

Modifiers: [✓ public] default private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

Step 5 -

Create a Mapper class within the WordCount class which extends MapReduceBase Class to implement mapper interface. The mapper class will contain -

1. Code to implement the "map" method.

` 2. Code for implementing the mapper-stage business logic should be written within this method.

Mapper Class Code for WordCount Example in Hadoop MapReduce

```
public static class Map extends MapReduceBase implements Mapper {
            private final static IntWritable one = new IntWritable(1);
            private Text word = new Text();
            public void map(LongWritable key, Text value, OutputCollector output, Reporter reporter)
                        throws IOException {
                  String line = value.toString();
                  StringTokenizer tokenizer = new StringTokenizer(line);
                  while (tokenizer.hasMoreTokens()) {
                        word.set(tokenizer.nextToken());
                        output.collect(word, one);
                  }
            }
      }
```

In the mapper class code, we have used the String Tokenizer class which takes the entire line and breaks into small tokens (string/word).

Step 6 –

Create a Reducer class within the WordCount class extending MapReduceBase Class to implement reducer interface. The reducer class for the wordcount example in hadoop will contain the -

1. Code to implement "reduce" method

2. Code for implementing the reducer-stage business logic should be written within this method

Reducer Class Code for WordCount Example in Hadoop MapReduce

```
public static class Reduce extends MapReduceBase implements Reducer {

            public void reduce(Text key, Iterator values, OutputCollector output,
                        Reporter reporter) throws IOException {
                  int sum = 0;
                  while (values.hasNext()) {
                        sum += values.next().get();
                  }
                  output.collect(key, new IntWritable(sum));
            }
      }
```

Step 7 –

Create main() method within the WordCount class and set the following properties using the JobConf class -

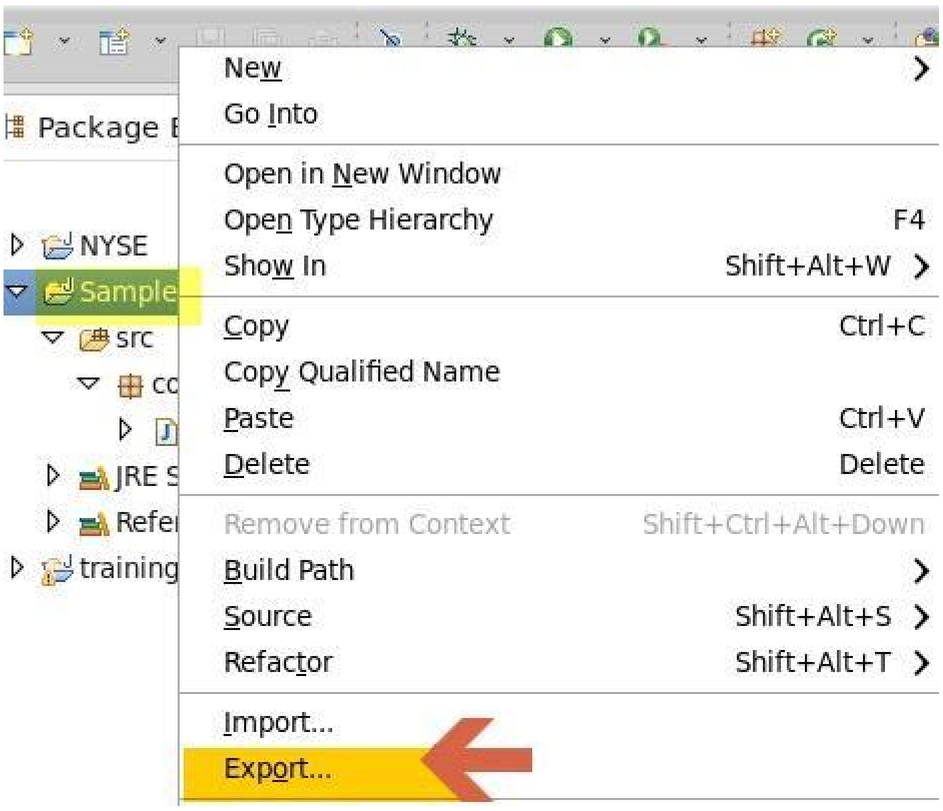
i. OutputKeyClass
ii. OutputValueClass
iii. Mapper Class
iv. Reducer Class
v. InputFormat

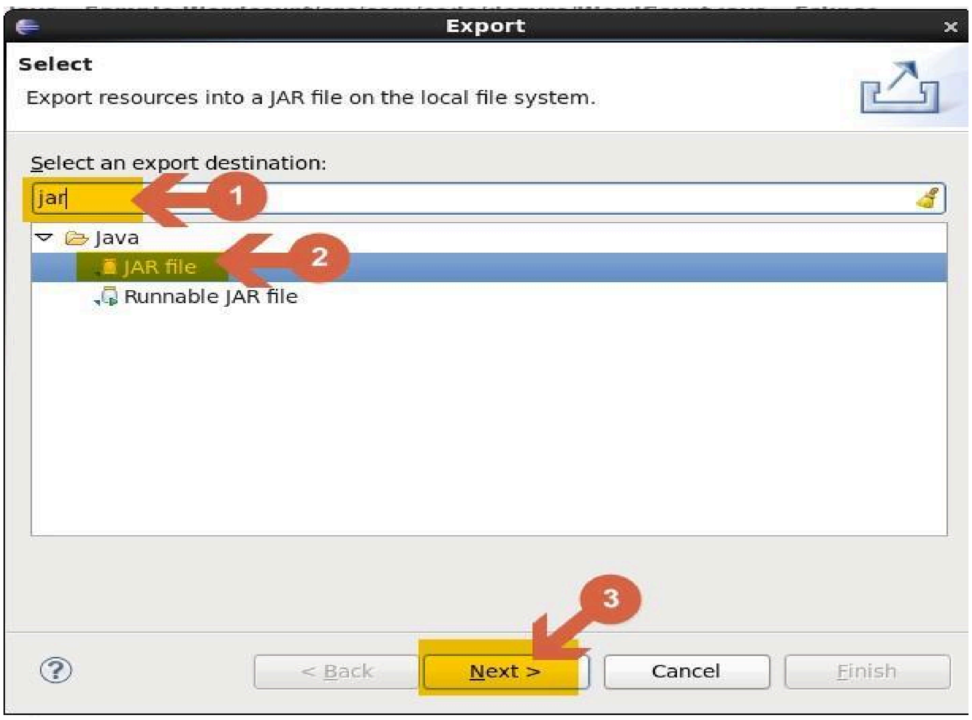
vi. OutputFormat
vii. InputFilePath
viii. OutputFolderPath

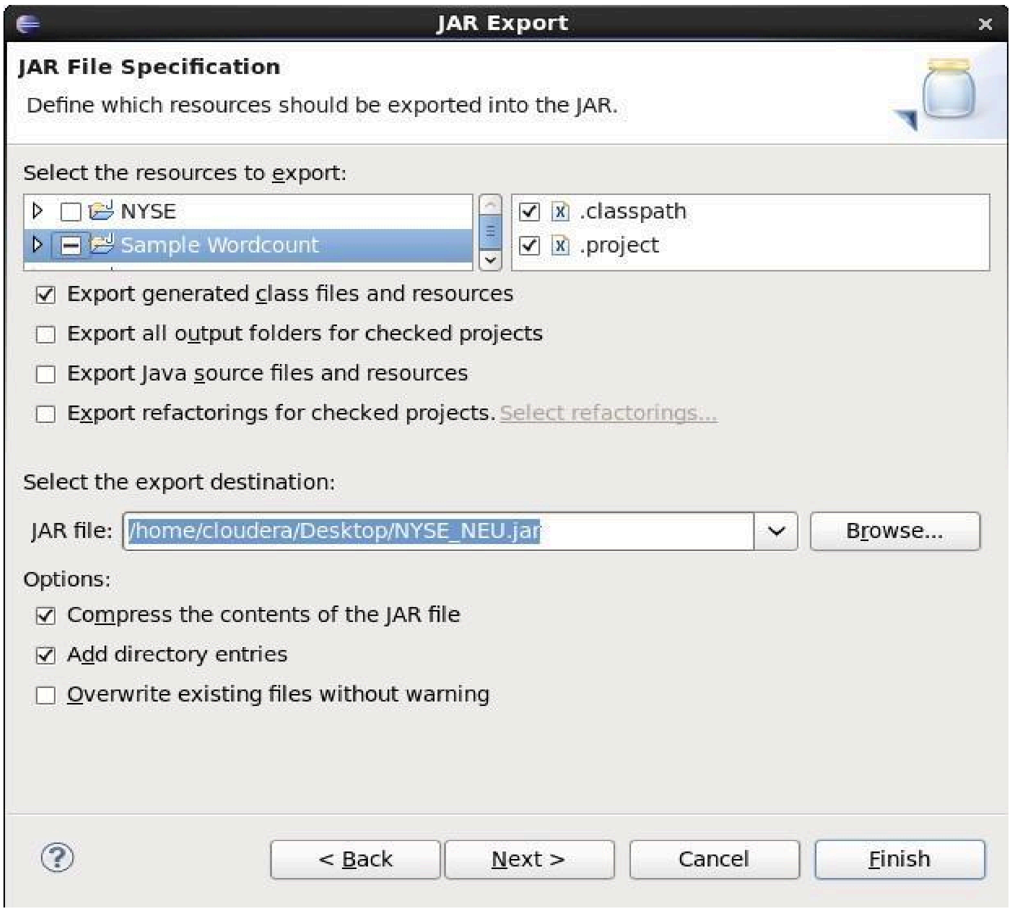
```
    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("WordCount");

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(Map.class);
        //conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
    }
}
```

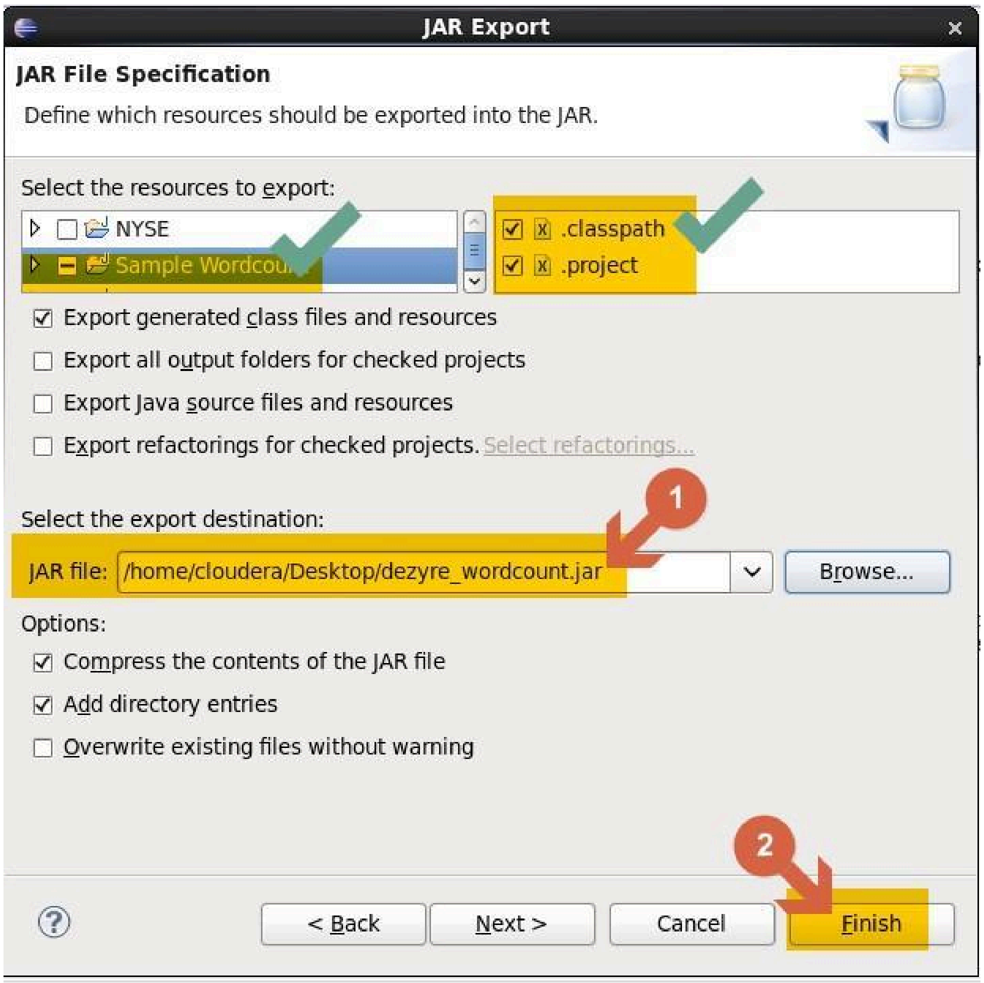
Step 8 –

Create the JAR file for the wordcount class –




Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
jar
1
Java
JAR file
2
Runnable JAR file
3
< Back
Next >
Cancel
Finish


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
NYSE
Sample Wordcount
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: /home/cloudera/Desktop/NYSE_NEU.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Cancel
Finish



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- [ ] NYSE
- [–] Sample Wordcou
- [✓] .classpath
- [✓] .project

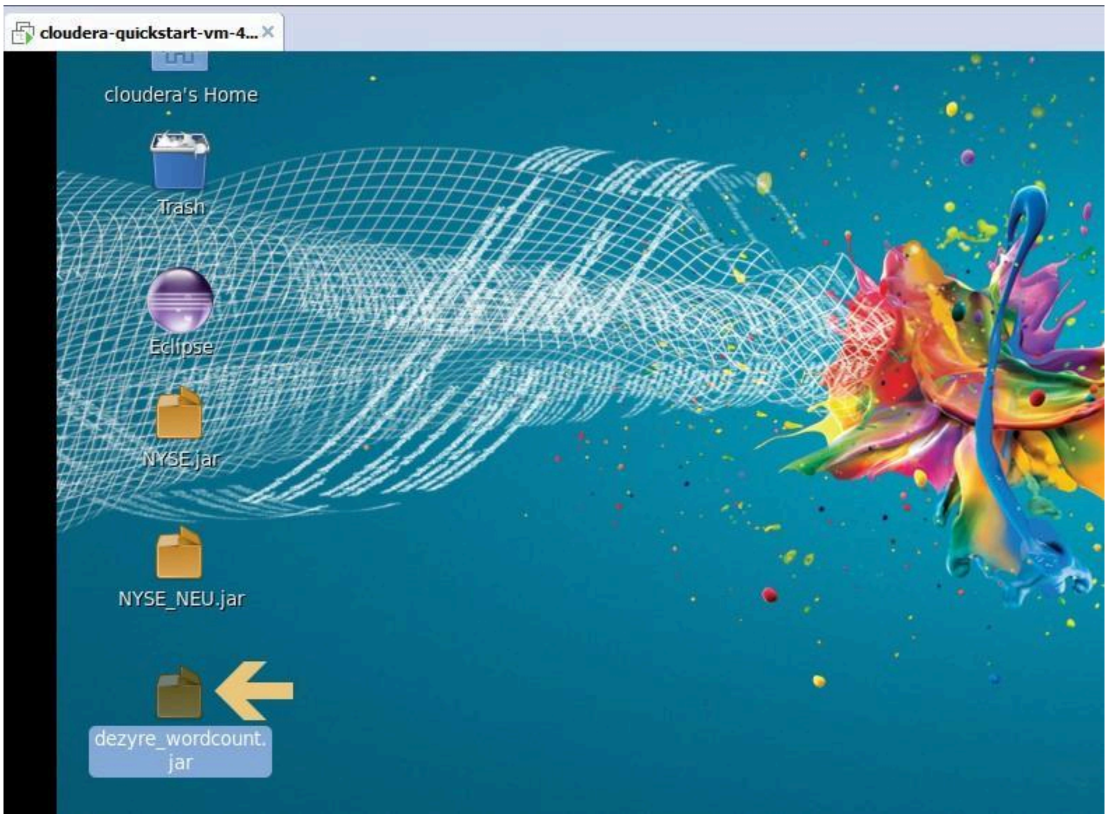
- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [ ] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

Select the export destination:

JAR file: /home/cloudera/Desktop/dezyre_wordcount.jar

Options:

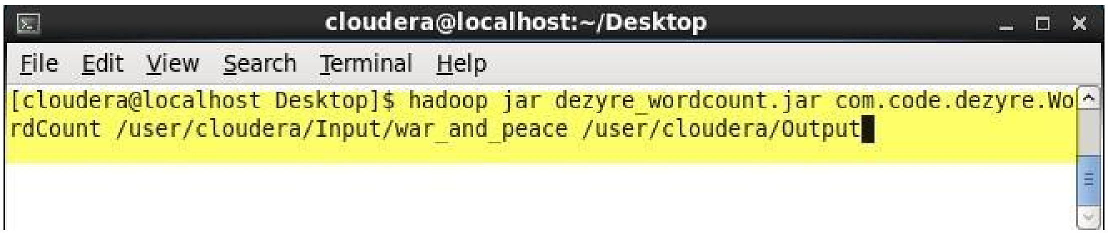
- [✓] Compress the contents of the JAR file
- [✓] Add directory entries
- [ ] Overwrite existing files without warning



How to execute the Hadoop MapReduce WordCount program ?

>> hadoop jar (jar file name) (className_along_with_packageName) (input file) (output folderpath)

hadoop jar dezyre_wordcount.jar com.code.dezyre.WordCount /user/cloudera/Input/war_and_peace /user/cloudera/Output


cloudera@localhost:~/Desktop
File Edit View Search Terminal Help

```
[cloudera@localhost Desktop]$ hadoop jar dezyre_wordcount.jar com.code.dezyre.Wo
rdCount /user/cloudera/Input/war_and_peace /user/cloudera/Output
```

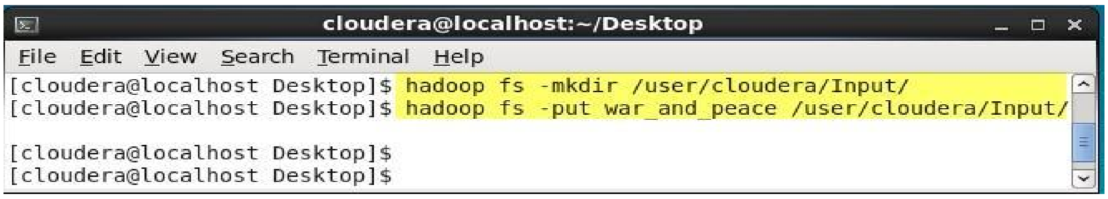
Important Note: war_and_peace**(Download link)** must be available in HDFS at /user/cloudera/Input/war_and_peace.

If not, upload the file on HDFS using the following commands -

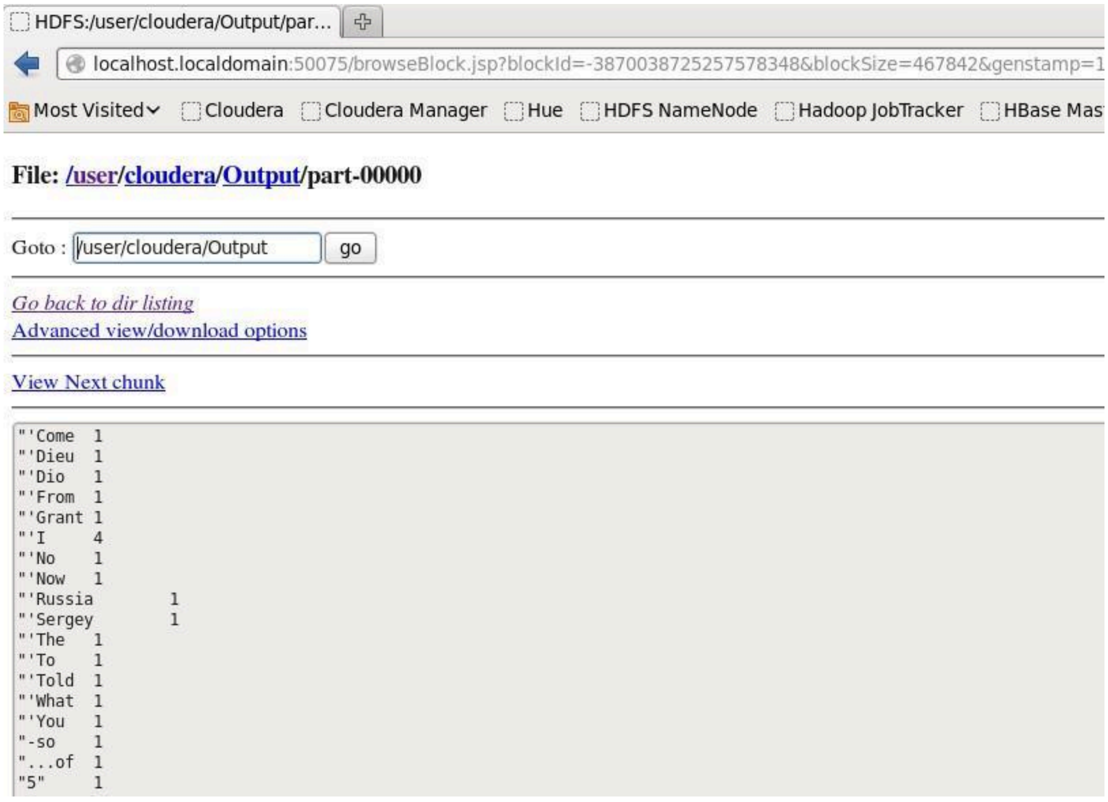
hadoop fs –mkdir /user/cloudera/Input

hadoop fs –put war_and_peace /user/cloudera/Input/war_and_peace

Output of Executing Hadoop WordCount Example –


cloudera@localhost:~/Desktop
File Edit View Search Terminal Help

```
[cloudera@localhost Desktop]$ hadoop fs -mkdir /user/cloudera/Input/
[cloudera@localhost Desktop]$ hadoop fs -put war_and_peace /user/cloudera/Input/

[cloudera@localhost Desktop]$
[cloudera@localhost Desktop]$
```


HDFS:/user/cloudera/Output/par...
localhost.localdomain:50075/browseBlock.jsp?blockId=-3870038725257578348&blockSize=467842&genstamp=1
Most Visited
Cloudera
Cloudera Manager
Hue
HDFS NameNode
Hadoop JobTracker
HBase Mas
go

File: /user/cloudera/Output/part-00000

Goto : /user/cloudera/Output

Go back to dir listing
Advanced view/download options

View Next chunk

```
"'Come  1
"'Dieu  1
"'Dio   1
"'From  1
"'Grant 1
"'I     4
"'No    1
"'Now   1
"'Russia        1
"'Sergey        1
"'The   1
"'To    1
"'Told  1
"'What  1
"'You   1
"-so    1
"...of  1
"5"     1
```

Conclusion:- Word count program is executed using map reduce and hadoop platform.

Questions-

1. What is the role of the `Map` function in the Hadoop word count program?
2. What happens during the shuffle and sort phase in Hadoop's MapReduce word count program?
3. What does the `Reduce` function do in the word count program?
4. What are the typical input and output formats for the Hadoop word count program?
5. How can a combiner function optimize the performance of the word count program in Hadoop?

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:9**	**Revision : 00** **Dated : _________**
Department : Department of Computer Engineering		
Title: Program to implement point-to-point communication using MPI routines		

Problem Statement

Program to implement point-to-point communication using MPI routines.

a. Parallelizing Trapezoidal Rule using MPI_Send and MPI_Recv

Input:- The program typically requires the following inputs:

1. Function to Integrate:

   The mathematical function \(f(x) \) to be integrated over the interval \([a, b]\).

2. Interval Boundaries:

   - a:The lower bound of the integration interval.

   - b:The upper bound of the integration interval.

3. Number of Trapezoids:

   - n: The number of trapezoids used in the approximation. A higher value of \(n \) generally increases the accuracy of the integral.

4. Number of Processes:

   - The number of MPI processes to use, typically determined by the `mpirun` or `mpiexec` command.

Sample Input

- Function: \(f(x) = x^2 \)

- Interval: \([a, b] = [0, 1]\)

- Number of trapezoids: \(n = 1000 \)

- Number of processes: 4

Output:-

The output is the approximated value of the integral calculated by the Trapezoidal Rule, combined from the contributions of all processes.

Sample Output

Given the sample input, the program might output:

The integral of f(x) from 0 to 1 is approximately 0.333333 with 1000 trapezoids.

Theory:-

Program to Implement Point-to-Point Communication Using MPI

Point-to-point communication in MPI involves direct communication between pairs of processes using routines like `MPI_Send` and `MPI_Recv`. These routines enable the explicit exchange of messages, allowing processes to send and receive data to and from specific peers. This is crucial in many parallel algorithms where processes need to coordinate or share intermediate results.

Parallelizing Trapezoidal Rule Using MPI

The Trapezoidal Rule is a numerical method for approximating the definite integral of a function. In a parallel environment, the integration interval can be divided among multiple processes, each of which computes a portion of the integral. MPI's point-to-point communication is then used to collect and combine these partial results.

Steps in Parallelizing the Trapezoidal Rule

1. Initialization:

- The MPI environment is initialized, and each process identifies its rank and the total number of processes.

2. Divide the Interval:

- The integration interval \([a, b]\) is divided into subintervals, each assigned to a different process. The subinterval size is determined by dividing the total number of trapezoids among the processes.

3. Local Computation: -

Each process computes the area under the curve for its assigned subinterval using the Trapezoidal Rule formula. This involves calculating the function values at the endpoints and summing the areas of the trapezoids.

4. Communication:

   - Each process sends its partial result (local area) to the root process using `MPI_Send`.

   - The root process collects these partial results using `MPI_Recv` and sums them to obtain the final result.

5. Final Result:

   - The root process outputs the total computed integral.

6. Finalization:

   - The MPI environment is finalized.

Key MPI Routines

- MPI_Send: Sends the local integral result from each process to the root process.

- MPI_Recv:Receives the partial integral results at the root process for aggregation.

Pseudo Code: Parallelizing Trapezoidal Rule Using MPI

```
// Initialize MPI environment

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &rank);

MPI_Comm_size(MPI_COMM_WORLD, &size);

// Define the interval and the function to integrate

a = 0.0;      // Lower bound

b = 1.0;      // Upper bound

n = 1000;     // Total number of trapezoids

// Calculate the width of each trapezoid

h = (b - a) / n;

// Determine the number of trapezoids each process will handle
```

```
local_n = n / size;

// Calculate the local integration limits for each process

local_a = a + rank * local_n * h;

local_b = local_a + local_n * h;


// Function to integrate

double f(double x) {

    return x * x;  // Example: f(x) = x^2

}

// Calculate local integral using the Trapezoidal Rule

double local_integral = 0.0;

for (int i = 1; i <= local_n - 1; i++) {

    double x = local_a + i * h;

    local_integral += f(x);

}

local_integral += (f(local_a) + f(local_b)) / 2.0;

local_integral *= h;

// Root process collects and sums up the results

if (rank == 0) {

    double total_integral = local_integral;

 // Receive partial integrals from other processes

    for (int i = 1; i < size; i++) {

        double recv_integral;

                MPI_Recv(&recv_integral, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
```

```
        total_integral += recv_integral;
    }
  // Output the final integral result
    printf("The integral from %f to %f is approximately %f\n", a, b, total_integral);
} else {
    // Non-root processes send their local integral to the root
    MPI_Send(&local_integral, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
}
// Finalize MPI environment
MPI_Finalize();
```

Explanation

1. Initialization:

   - The MPI environment is initialized. Each process obtains its rank and the total number of processes.

2. Input Parameters:

   - The interval \([a, b]\), the total number of trapezoids `n`, and the function to integrate \(f(x) \) are defined.

3. Local Interval Calculation:

   - The interval \([a, b]\) is divided among the processes. Each process computes the bounds of its local interval and the number of trapezoids it will handle (`local_n`).

4. Local Integral Calculation:

   - Each process computes the integral over its assigned subinterval using the Trapezoidal Rule. The result is stored in `local_integral`.

5. Communication:

   - The root process (rank 0) collects partial results from all other processes using `MPI_Recv`.

- Non-root processes send their local integral results to the root process using `MPI_Send`.

6. Final Result:

- The root process sums all the partial integrals to compute the final result and prints the approximate integral value.

7. Finalization:

- The MPI environment is finalized.

Conclusion:- Program implemented for point-to-point communication using MPI routines

Questions:-

1. How is the integration interval [a,b][a, b][a,b] divided among the processes in the parallel Trapezoidal Rule using MPI?
2. What does each process compute locally in the parallel Trapezoidal Rule program?
3. How do processes communicate their partial results in the parallel Trapezoidal Rule using MPI?
4. How is the final integral value obtained in the parallel Trapezoidal Rule using MPI?
5. What is the specific role of the root process in the parallel Trapezoidal Rule program?

	Sanjivani Rural Education Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 A**
Academic Year: 2023-24	**Assignment No:10**	**Revision : 00** **Dated :** ________
Department : Department of Computer Engineering		
Title: Program to execute matrix multiplication using CUDA.		

Problem Statement

Program to execute matrix multiplication using CUDA.

a. Basic CUDA host, device and memory constructs.

b. Thread- warp, block, grid usage.

Input:-

Matrix A (1024 x 1024): All elements are 1.0

Matrix B (1024 x 1024): All elements are 2.0

Output:- Matrix C (1024 x 1024): All elements are 2048.0.

Theory:-

Program to Execute Matrix Multiplication Using CUDA

Matrix multiplication is a common operation in scientific computing, and CUDA allows this operation to be parallelized on NVIDIA GPUs. By leveraging CUDA, large matrix multiplications can be performed efficiently by distributing the computation across thousands of GPU threads. The program involves managing data between the host (CPU) and device (GPU), defining the appropriate memory spaces, and organizing the computation in parallel using CUDA constructs.

Basic CUDA Host, Device, and Memory Constructs

1. Host and Device:

- Host: Refers to the CPU and its memory. The host is responsible for setting up the computation, allocating memory, and launching CUDA kernels on the device.

- Device:Refers to the GPU and its memory. The device performs the actual matrix multiplication computation in parallel across many threads.

2. Memory Constructs:

- Host Memory: Memory allocated on the CPU side using standard C/C++ functions or CUDA-specific functions like `cudaMallocHost`.

- Device Memory: Memory allocated on the GPU using `cudaMalloc`. Data must be explicitly transferred between host and device memory using functions like `cudaMemcpy`.

- Shared Memory:A small, fast memory space on the GPU that is shared among threads in the same block, used to speed up memory access and reduce latency during computation.

Thread-Warp, Block, and Grid Usage

1. Thread:

- The smallest unit of execution in CUDA. Each thread performs operations on individual elements of the matrices (e.g., computing a single element of the result matrix).

2. Warp:

- A group of 32 threads that execute the same instruction simultaneously. Efficient CUDA programs consider warp-level execution to avoid divergence, where different threads in a warp follow different execution paths.

3. Block:

- A group of threads organized in a 1D, 2D, or 3D structure. Threads in the same block can cooperate via shared memory and can be synchronized. The number of threads per block is limited, so large computations are divided among many blocks.

4. Grid:

- A collection of blocks that together cover the entire data set. The grid structure allows CUDA to scale the computation to handle large matrices by using many blocks, each processing a portion of the matrix.

Matrix Multiplication with CUDA

1. Setup:

- Allocate memory on both the host and the device for the input matrices and the output matrix.

- Transfer input matrices from host memory to device memory.

2. Kernel Launch:

- Define a CUDA kernel function that performs the matrix multiplication.

- Launch the kernel on the GPU by specifying the grid and block dimensions, which determine how the threads are organized.

3. Computation:

- Each thread computes a single element of the output matrix by performing the dot product of the corresponding row of the first matrix and the column of the second matrix.

- Use shared memory within each block to optimize access to the matrix elements and reduce global memory accesses.

4. Result Collection:

- After computation, transfer the result matrix from device memory back to host memory.

- Free the allocated memory on both the host and device.

CUDA Program for Matrix Multiplication

```
#include <stdio.h>
// Define matrix dimensions
#define N 1024  // Assuming square matrices of size N x N
// CUDA kernel for matrix multiplication
__global__ void matrixMulCUDA(float *A, float *B, float *C, int n) {
    // Calculate the row index of the C matrix element
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    // Calculate the column index of the C matrix element
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    float value = 0.0;
    // Each thread computes one element of the output matrix
    if (row < n && col < n) {
        for (int k = 0; k < n; k++) {
            value += A[row * n + k] * B[k * n + col];
        }
        C[row * n + col] = value;
    }
}
int main() {
    int size = N * N * sizeof(float);
    // Allocate host memory
    float *h_A = (float *)malloc(size);
    float *h_B = (float *)malloc(size);
    float *h_C = (float *)malloc(size);
    // Initialize input matrices
    for (int i = 0; i < N * N; i++) {
        h_A[i] = 1.0f;  // Example initialization
        h_B[i] = 2.0f;  // Example initialization
    }
    // Allocate device memory
    float *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, size);
    cudaMalloc(&d_B, size);
    cudaMalloc(&d_C, size);

    // Copy matrices from host to device memory

    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);

    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Define block and grid dimensions

    dim3 threadsPerBlock(16, 16);

    dim3 blocksPerGrid((N + threadsPerBlock.x - 1) / threadsPerBlock.x,

                       (N + threadsPerBlock.y - 1) / threadsPerBlock.y);


    // Launch the matrix multiplication kernel

    matrixMulCUDA<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);

    // Copy the result matrix from device to host memory

    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    // Example output: print the first element of the result matrix

    printf("C[0][0] = %f\n", h_C[0]);

    // Free device memory

    cudaFree(d_A);

    cudaFree(d_B);

    cudaFree(d_C);

    // Free host memory

    free(h_A);

    free(h_B);

    free(h_C);

    return 0;

}
```

Explanation

1. Matrix Dimensions and Initialization:

- The matrices are assumed to be square matrices of size `N x N`.

- The matrices `h_A` and `h_B` are initialized on the host (CPU) with values. Here, all elements of `h_A` are initialized to `1.0f` and `h_B` to `2.0f` as an example.

2. Memory Allocation:

- Memory for the matrices is allocated both on the host and the device (GPU) using `malloc` and `cudaMalloc`, respectively.

3. Data Transfer:

- The matrices `h_A` and `h_B` are copied from the host memory to the device memory using `cudaMemcpy`.

4. Kernel Execution:

- The `matrixMulCUDA` kernel is launched on the GPU. Each thread calculates a single element of the output matrix `C` by computing the dot product of a row from `A` and a column from `B`.

- The grid and block dimensions are defined to ensure that all elements of the output matrix `C` are computed.

5. Result Transfer:

- After the kernel execution, the resulting matrix `C` is copied back from the device to the host.

6. Memory Cleanup:

- Both device and host memories are freed after the computation is complete.

Conclusion:-Program executed for matrix multiplication using CUDA.

Questions:-

1. How is memory allocated on the GPU for matrices in a CUDA matrix multiplication program?
2. How do you transfer data from the CPU to the GPU before starting the computation?

3. What is the role of an individual thread in CUDA matrix multiplication?
4. How are threads organized within a block in CUDA?
5. How do you determine the grid dimensions in a CUDA matrix multiplication program?