Title : Parallel Search Algorithm- Design and implement parallel algorithm utilizing  
all resources available.for  
· Binary Search for Sorted Array  
· Depth-First Search ( tree or an undirected graph ) OR  
· Breadth-First Search ( tree or an undirected graph) OR  
· Best-First Search that ( traversal of graph to reach a target in the shortest  
possible path)

Input File: Breadth-First Search  
#include<iostream>  
#include<stdlib.h>  
#include<queue>  
using namespace std;  
class node{  
public:  
node \*left, \*right;  
int data;  
};  
class Breadthfs{  
public:  
node \*insert(node \*, int);  
void bfs(node \*);  
};  
node \*insert(node \*root, int data){  
if(!root){  
root=new node;  
root->left=NULL;  
root->right=NULL;  
root->data=data;  
return root;  
}  
queue<node \*> q;  
q.push(root);  
while(!q.empty()){  
node \*temp=q.front();  
q.pop();  
//cout<<"Enter At 1).Left\n2).Right : \n";  
//int decision = 1;

//cin>>decision;  
if(temp->left==NULL/\* && decision == 1\*/){  
temp->left=new node;  
temp->left->left=NULL;  
temp->left->right=NULL;  
temp->left->data=data;  
return root;  
}  
else{  
q.push(temp->left);  
}  
if(temp->right==NULL /\*&& decision == 2\*/){  
temp->right=new node;  
temp->right->left=NULL;  
temp->right->right=NULL;  
temp->right->data=data;  
return root;  
}  
else{  
q.push(temp->right);  
}  
}  
}

void bfs(node \*head,int goalNode){  
queue<node\*> q;  
q.push(head);  
int qSize;  
while (!q.empty()){  
qSize = q.size();  
#pragma omp parallel for  
for (int i = 0; i < qSize; i++){  
node\* currNode;  
#pragma omp critical  
{  
currNode = q.front();  
q.pop();  
cout<<"\t"<<currNode->data;  
if(currNode->data == goalNode){  
cout<<"\nGoal Node Found\n\n";  
return;  
}}  
#pragma omp critical

{  
if(currNode->left)  
q.push(currNode->left);  
if(currNode->right)  
q.push(currNode->right);  
}  
}  
}  
cout<<"\nGoal Node Not Found\n\n";  
}  
int main(){  
node \*root=NULL;  
int data, goalNode;  
char ans;  
do{  
cout<<"\n enter data=>";  
cin>>data;  
root=insert(root,data);  
cout<<"do you want insert one more node?";  
cin>>ans;  
}while(ans=='y'||ans=='Y');  
cout<<"Enter Goal Node : ";  
cin>>goalNode;  
bfs(root,goalNode);

return 0;  
}  
Output File:  
sl2-pc14@sl2pc14-HP-COMPAQ-4000-PRO-SMALL-FORM-PC:~$ cd Desktop  
sl2-pc14@sl2pc14-HP-COMPAQ-4000-PRO-SMALL-FORM-PC:~/Desktop$ g++ bt.cpp  
sl2-pc14@sl2pc14-HP-COMPAQ-4000-PRO-SMALL-FORM-PC:~/Desktop$ ./a.out  
enter data=>20  
do you want insert one more node?y  
enter data=>15  
do you want insert one more node?y  
enter data=>12  
do you want insert one more node?y

enter data=>17  
do you want insert one more node?y

enter data=>10  
do you want insert one more node?n  
Enter Goal Node : 12  
20 15 12  
Goal Node Found  
\*/

Input File : Merge Sort  
//Merge sort  
#include<iostream>  
#include<stdlib.h>  
#include<omp.h>  
using namespace std;  
void mergesort(int a[],int i,int j);  
void merge(int a[],int i1,int j1,int i2,int j2);  
void mergesort(int a[],int i,int j)  
{  
int mid;  
if(i<j)  
{  
mid=(i+j)/2;  
#pragma omp parallel sections  
{  
#pragma omp section  
{  
mergesort(a,i,mid);  
}  
#pragma omp section  
{  
mergesort(a,mid+1,j);  
}  
}  
merge(a,i,mid,mid+1,j);  
}  
}  
void merge(int a[],int i1,int j1,int i2,int j2)  
{  
int temp[100];  
int i,j,k;

i=i1;  
j=i2;  
k=0;

while(i<=j1 && j<=j2)  
{  
if(a[i]<a[j])  
{  
temp[k++]=a[i++];  
}  
else  
{  
temp[k++]=a[j++];  
}  
}  
while(i<=j1)  
{  
temp[k++]=a[i++];  
}  
while(j<=j2)  
{  
temp[k++]=a[j++];  
}  
for(i=i1,j=0;i<=j2;i++,j++)  
{  
a[i]=temp[j];

}

}

int main()  
{  
int \*a,n,i;  
cout<<"\n Enter No Of Elements:-";  
cin>>n;  
a= new int[n];  
cout<<"\n Enter Elements:-";  
for(i=0;i<n;i++)  
{  
cin>>a[i];  
}  
mergesort(a, 0, n-1);  
cout<<"\n Sorted Array:-“;

for(i=0;i<n;i++)  
{  
cout<<"\n"<<a[i];  
}  
return 0;  
}  
/Output File :  
pvgcoen-3@pvgcoen3-ThinkCentre-M700:~$ cd Desktop  
pvgcoen-3@pvgcoen3-ThinkCentre-M700:~/Desktop$ g++ mergeopen.cpp  
pvgcoen-3@pvgcoen3-ThinkCentre-M700:~/Desktop$ ./a.out  
Enter No Of Elements:-6  
Enter Elements:-  
27  
12  
32  
10  
2  
5

Sorted Array:-  
2  
5  
10  
12  
27  
32\*/

Input File : Bubble sort  
#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)  
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;  
}  
int main()  
{  
int \*a,n;  
cout<<"\n Enter 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:-";  
for(int i=0;i<n;i++)  
{  
cout<<a[i]<<endl;

}  
return 0;  
}  
/Output File:  
/\*output  
pvgcoen-3@pvgcoen3-ThinkCentre-M700:~/Desktop$ g++ bubble.cpp  
pvgcoen-3@pvgcoen3-ThinkCentre-M700:~/Desktop$ ./a.out

Enter No of Elements=>6  
Enter Elements=>12 10 15 20 16 5  
Sorted Array:-5  
10  
12  
15  
16  
20  
\*/

Implement Parallel Reduction using Min, Max, Sum and Average  
Operation

nput File:  
#include <stdio.h>  
#include <oomph>  
int main()  
{  
double arr[10];  
omp\_set\_num\_threads(4);  
double max\_val=0.0,min\_val = 0.0,sum\_val = 0.0,sub\_val = 0.0,avg\_val = 0.0;  
int i;  
for( i=0; i<=10; i++)  
arr[i]=i;  
//arr[i] = 2.0 + i;  
/\* Maximum Value \*/  
#pragma omp parallel for reduction(max : max\_val)  
for( i=0;i<=10; i++){  
printf("\n1).Thread id = %d and i = %d", omp\_get\_thread\_num(), i);  
if(arr[i] > max\_val){  
max\_val = arr[i];  
}  
}  
printf("\nmax\_val = %f\n\n", max\_val);  
/\* Minimum Value \*/  
min\_val = arr[3];  
#pragma omp parallel for reduction(min : min\_val)  
for( i=0;i<=10; i++){

Laboratory Practice I Fourth Year of Computer Engineering  
Pune Vidyarthi Griha’s College Of Engineering, NASIK-4 Prepared By: Thakare Komal  
printf("\n2).Thread id = %d and i = %d", omp\_get\_thread\_num(), i);  
if(arr[i] < min\_val){  
min\_val = arr[i];  
}  
}  
printf("\nmin\_val = %f\n\n", min\_val);  
/\* Sum Of Elements Of Array \*/  
#pragma omp parallel for reduction(+ : sum\_val)  
for( i=0;i<=10; i++){  
printf("\n3).Thread id = %d and i = %d", omp\_get\_thread\_num(), i);  
sum\_val += arr[i];  
}  
printf("\nSum\_val = %f\n\n", sum\_val);  
/\* Average Value \*/  
#pragma omp parallel for reduction(+ : avg\_val)  
for( i=0;i<=10; i++){  
printf("\n4).Thread id = %d and i = %d", omp\_get\_thread\_num(), i);  
avg\_val += arr[i];  
}  
avg\_val = avg\_val / i;  
printf("\nAvg\_val = %f\n\n", avg\_val);  
return 0;  
}  
Output File:  
pvgcoen-9@pvgcoen9-ThinkCentre-M700:~$ cd Desktop  
pvgcoen-9@pvgcoen9-ThinkCentre-M700:~/Desktop$ gcc openmp.c -fopenmp  
pvgcoen-9@pvgcoen9-ThinkCentre-M700:~/Desktop$ ./a.out

Thread id = 0 and i = 0  
1).Thread id = 0 and i = 1  
1).Thread id = 0 and i = 2  
1).Thread id = 1 and i = 3  
1).Thread id = 1 and i = 4  
1).Thread id = 1 and i = 5  
1).Thread id = 3 and i = 9  
1).Thread id = 3 and i = 10  
1).Thread id = 2 and i = 6  
1).Thread id = 2 and i = 7  
1).Thread id = 2 and i = 8  
max\_val = 10.000000

Thread id = 3 and i = 9  
2).Thread id = 3 and i = 10  
2).Thread id = 1 and i = 3  
2).Thread id = 0 and i = 0  
2).Thread id = 0 and i = 1  
2).Thread id = 0 and i = 2  
2).Thread id = 1 and i = 4  
2).Thread id = 1 and i = 5  
2).Thread id = 2 and i = 6  
2).Thread id = 2 and i = 7  
2).Thread id = 2 and i = 8  
min\_val = 0.000000

3).Thread id = 2 and i = 6  
3).Thread id = 2 and i = 7  
3).Thread id = 2 and i = 8  
3).Thread id = 0 and i = 0  
3).Thread id = 3 and i = 9  
3).Thread id = 3 and i = 10  
3).Thread id = 1 and i = 3  
3).Thread id = 1 and i = 4  
3).Thread id = 1 and i = 5  
3).Thread id = 0 and i = 1  
3).Thread id = 0 and i = 2  
Sum\_val = 55.000000

Laboratory Practice I Fourth Year of Computer Engineering  
Pune Vidyarthi Griha’s College Of Engineering, NASIK-4 Prepared By: Thakare Komal  
2).Thread id = 3 and i = 9  
2).Thread id = 3 and i = 10  
2).Thread id = 1 and i = 3  
2).Thread id = 0 and i = 0  
2).Thread id = 0 and i = 1  
2).Thread id = 0 and i = 2  
2).Thread id = 1 and i = 4  
2).Thread id = 1 and i = 5  
2).Thread id = 2 and i = 6  
2).Thread id = 2 and i = 7  
2).Thread id = 2 and i = 8  
min\_val = 0.000000  
3).Thread id = 2 and i = 6  
3).Thread id = 2 and i = 7  
3).Thread id = 2 and i = 8  
3).Thread id = 0 and i = 0  
3).Thread id = 3 and i = 9  
3).Thread id = 3 and i = 10  
3).Thread id = 1 and i = 3  
3).Thread id = 1 and i = 4  
3).Thread id = 1 and i = 5  
3).Thread id = 0 and i = 1  
3).Thread id = 0 and i = 2  
Sum\_val = 55.000000  
4).Thread id = 3 and i = 9  
4).Thread id = 3 and i = 10  
4).Thread id = 2 and i = 6  
4).Thread id = 2 and i = 7  
4).Thread id = 2 and i = 8  
4).Thread id = 1 and i = 3  
4).Thread id = 1 and i = 4  
4).Thread id = 1 and i = 5  
4).Thread id = 0 and i = 0  
4).Thread id = 0 and i = 1  
4).Thread id = 0 and i = 2  
Avg\_val = 5.000000  
\*/

Vector and Matrix Operations-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 file : Add two large vectors  
#include<stdio.h>  
#include<iostream>  
#include<cstdlib>  
#include<omp.h> //allow a programmer to use parallel paradigms  
using namespace std;  
#define MAX 100  
int main()  
{  
int a[MAX],b[MAX],c[MAX],i;  
cout<<"\n\tFIRST VECTOR :-\n\n";  
#pragma omp parallel for //Assign random values to Vector.  
for(i=0;i<MAX;i++)  
{  
a[i]=rand()%1000;  
}  
for(i=0;i<MAX;i++) //Display the vector values.

{  
cout<<"\t"<<a[i];  
}  
cout<<"\n";  
cout<<"\n\tSECOND VECTOR :-\n\n";  
#pragma omp parallel for //Assign random values to Vector.  
for(i=0;i<MAX;i++)  
{  
b[i]=rand()%1000;  
}  
for(i=0;i<MAX;i++) //Display the vector values.

Laboratory Practice I Fourth Year of Computer Engineering  
Pune Vidyarthi Griha’s College Of Engineering, NASIK-4 Prepared By: Thakare Komal  
{  
cout<<"\t"<<b[i];  
}  
cout<<"\n";  
cout<<"\n\n\tADDITION OF LARGE VECTORS(a,b,c):-\n";  
#pragma omp parallel for // Add both vector values.  
for(i=0;i<MAX;i++)  
{  
c[i]=a[i]+b[i];  
}  
cout<<"\n Vector-1 + Vector-2 = \tResult";  
cout<<"\n ---------------------------------------";  
for(i=0;i<MAX;i++) //Display the result of addition of Vector.  
{  
cout<<"\n\t"<<a[i]<<"\t+\t"<<b[i]<<"\t=\t"<<c[i];  
}  
cout<<"\n\n";  
}  
/\*  
Output:  
Copyright (C) 2009-2015 Intel Corporation. All rights reserved.  
Intel(R) VTune(TM) Amplifier XE 2016 (build 444464)  
pvgcoen-9@pvgcoen9-ThinkCentre-M700:~$ cd Desktop  
pvgcoen-9@pvgcoen9-ThinkCentre-M700:~/Desktop$ g++ add\_vectors.cpp –fopenmp

pvgcoen-9@pvgcoen9-ThinkCentre-M700:~/Desktop$ ./a.out  
FIRST VECTOR :-  
383 886 915 386 736 782 530 862 123 67 929 58 229 324  
315 370 413 526 91 980 956 873 862 170 996 777 335 649  
421 362 27 690 59 763 926 540 426 135 22 69 393 456 11  
42 373 421 919 784 537 198 367 434 364 43 750 87 808  
276 178 788 584 403 651 754 399 932 60 676 368 739 12  
226 586 94 539 793 492 172 211 368 567 429 802 167 281  
305 925 84 327 336 505 846 729 313 857 124 895 582 545  
814

SECOND VECTOR :-  
434 378 467 601 97 902 317 492 652 275 407 474 121 858  
395 29 237 235 793 818 428 143 11 928 529 301 689 444  
619 97 771 481 675 709 927 567 856 497 353 586 965 306  
683 219 624 528 871 829 503 19 795 570 756 280 286 441

440 729 31 732 270 368 708 715 340 149 796 723 618 245  
846 451 921 555 379 865 117 488 764 228 841 350 193 500  
34 764 124 914 987 856 743 491 227 365 859 936 432 551  
437 228  
ADDITION OF LARGE VECTORS(a,b,c):-  
Vector-1 + Vector-2 = Result  
---------------------------------------  
383 + 434 = 817  
886 + 378 = 1264  
915 + 467 = 1382  
386 + 601 = 987  
736 + 97 = 833  
782 + 902 = 1684  
530 + 317 = 847  
862 + 492 = 1354

123 + 652 = 775  
67 + 275 = 342  
929 + 407 = 1336  
58 + 474 = 532  
229 + 121 = 350  
324 + 858 = 1182  
315 + 395 = 710  
370 + 29 = 399  
413 + 237 = 650  
526 + 235 = 761  
91 + 793 = 884  
980 + 818 = 1798  
956 + 428 = 1384  
873 + 143 = 1016  
862 + 11 = 873  
170 + 928 = 1098  
996 + 529 = 1525  
777 + 301 = 1078  
335 + 689 = 1024  
649 + 444 = 1093  
421 + 619 = 1040  
362 + 97 = 459  
27 + 771 = 798  
690 + 481 = 1171  
59 + 675 = 734  
763 + 709 = 1472  
926 + 927 = 1853  
540 + 567 = 1107

426 + 856 = 1282  
135 + 497 = 632  
22 + 353 = 375  
69 + 586 = 655  
393 + 965 = 1358  
456 + 306 = 762  
11 + 683 = 694  
42 + 219 = 261  
373 + 624 = 997  
421 + 528 = 949  
919 + 871 = 1790  
784 + 829 = 1613  
537 + 503 = 1040  
198 + 19 = 217  
367 + 795 = 1162  
434 + 570 = 1004  
364 + 756 = 1120  
43 + 280 = 323  
750 + 286 = 1036  
87 + 441 = 528  
808 + 440 = 1248  
276 + 729 = 1005  
178 + 31 = 209  
788 + 732 = 1520  
584 + 270 = 854  
403 + 368 = 771  
651 + 708 = 1359  
754 + 715 = 1469  
399 + 340 = 739  
932 + 149 = 1081  
60 + 796 = 856  
676 + 723 = 1399  
368 + 618 = 986  
739 + 245 = 984  
12 + 846 = 858  
226 + 451 = 677  
586 + 921 = 1507  
94 + 555 = 649  
539 + 379 = 918  
793 + 865 = 1658  
492 + 117 = 609  
172 + 488 = 660  
211 + 764 = 975  
368 + 228 = 596  
567 + 841 = 1408  
429 + 350 = 779

802 + 193 = 995  
167 + 500 = 667  
281 + 34 = 315  
305 + 764 = 1069  
925 + 124 = 1049  
84 + 914 = 998  
327 + 987 = 1314  
336 + 856 = 1192  
505 + 743 = 1248  
846 + 491 = 1337  
729 + 227 = 956  
313 + 365 = 678  
857 + 859 = 1716  
124 + 936 = 1060  
895 + 432 = 1327  
582 + 551 = 1133  
545 + 437 = 982  
814 + 228 = 1042  
\*/  
Input File:  
#include<omp.h>

#include<assert.h>  
#include<stdio.h>  
#include<string.h>  
#define READ(f) freopen(f, "r", stdin)  
#define WRITE(f) freopen(f, "w", stdout)  
#define pks printf("Case %d: ",++ks);  
#define mx 1002  
int a[mx][mx];  
int b[mx][mx];  
int c[mx][mx];  
int d[mx][mx];  
void generate\_matrix(int m,int n)  
{  
int i,j;  
printf("\tPlease Enter the values for matrix A:- \t\n");  
for(i=0;i<m;i++)

{  
for(j=0;j<n;j++)  
{  
printf("\t\tEnter the value for [%d",i);printf("][");printf("%d]:-\t",j);  
scanf("%d",&a[i][j]);  
//a[i][j]=rand()%100;  
//b[i][j]=rand()%100;  
}  
}  
printf("\n");  
printf("\tPlease Enter the values for matrix B:- \t\n");  
for(i=0;i<m;i++)  
{  
for(j=0;j<n;j++)  
{  
printf("\t\tEnter the value for [%d",i);printf("][");printf("%d]:-\t",j);  
scanf("%d",&b[i][j]);  
//a[i][j]=rand()%100;  
//b[i][j]=rand()%100;  
}  
}  
}  
void check(int m,int n)

{  
int i,j;  
for(i=0;i<m;i++)  
{  
for(j=0;j<n;j++)  
assert(c[i][j]==d[i][j]);  
}  
}  
void matrix\_mult\_serial(int m,int n)  
{  
int i,j,k;  
double st=omp\_get\_wtime();  
if(m==n)  
{  
for(i=0;i<m;i++)  
{  
for(j=0;j<n;j++)  
{  
for(k=0;k<m;k++)  
{  
c[i][j]+=a[i][k]\*b[k][j];

Laboratory Practice I Fourth Year of Computer Engineering  
Pune Vidyarthi Griha’s College Of Engineering, NASIK-4 Prepared By: Thakare Komal  
}  
}  
}  
printf("\n");  
printf("\tResult of matrix multiplication is:- \t\n");  
for(i=0;i<m;i++)  
{  
for(j=0;j<n;j++)  
{  
printf("\t %d",c[i][j]);  
//a[i][j]=rand()%100;  
//b[i][j]=rand()%100;  
}  
printf("\n");  
}  
}  
else  
{  
printf("\n\tTo multiply matrices values of rows and columns should be equal...!!!");  
}  
double en=omp\_get\_wtime();  
printf("\n\tSerial: %lf\n",en-st);  
}  
void matrix\_mult\_parallel1(int m,int n)  
{  
//Static Scheduler  
memset(d,0,sizeof d);  
int i,j,k;  
double st=omp\_get\_wtime();  
if(m==n)  
{  
#pragma omp parallel for schedule(static,50) collapse(2) private(i,j,k)shared(a,b,c)  
for(i=0;i<m;i++)

for( j=0;j<n;j++)  
for(k=0;k<m;k++)  
d[i][j]+=a[i][k]\*b[k][j];  
printf("\n");  
printf("\tResult of matrix multiplication is:- \t\n");  
for(i=0;i<m;i++)  
{  
for(j=0;j<n;j++)  
{  
printf("\t %d",d[i][j]);  
//a[i][j]=rand()%100;

//b[i][j]=rand()%100;  
}  
printf("\n");  
}  
double en=omp\_get\_wtime();  
printf("\n\tParallel-1(Static Scheduler) %lf\n",en-st);  
check(m,n);  
}  
}

void matrix\_mult\_parallel2(int m,int n)  
{  
//Dynamic Scheduler  
memset(d,0,sizeof d);  
int i,j,k;  
double st=omp\_get\_wtime();  
if(m==n)  
{  
#pragma omp parallel for schedule(dynamic,50) collapse(2) private(i,j,k) shared(a,b,c)  
for(i=0;i<m;i++)  
for( j=0;j<n;j++)  
for(k=0;k<m;k++)  
d[i][j]+=a[i][k]\*b[k][j];  
printf("\n");  
printf("\tResult of matrix multiplication is:- \t\n");  
for(i=0;i<m;i++)  
{  
for(j=0;j<n;j++)

{  
printf("\t %d",d[i][j]);  
//a[i][j]=rand()%100;  
//b[i][j]=rand()%100;  
}  
printf("\n");  
}  
double en=omp\_get\_wtime();  
printf("\n\tParallel-2(Dynamic Scheduler) %lf\n",en-st);  
check(m,n);  
}  
printf("\n");  
}  
int main()  
{  
int m,n;

Laboratory Practice I Fourth Year of Computer Engineering  
Pune Vidyarthi Griha’s College Of Engineering, NASIK-4 Prepared By: Thakare Komal  
printf("\nPlease Enter the rows of the matrix:- \t");scanf("%d",&m);  
printf("\nPlease Enter the columns of the matrix:- \t");scanf("%d",&n);  
printf("\n");  
generate\_matrix(m,n);  
matrix\_mult\_serial(m,n);  
matrix\_mult\_parallel1(m,n);  
matrix\_mult\_parallel2(m,n);  
return 0;  
}  
/\*Output File :  
pvgcoen-9@pvgcoen9-ThinkCentre-M700:~$ cd Desktop  
pvgcoen-9@pvgcoen9-ThinkCentre-M700:~/Desktop$ gcc parallelmatrix.c -fopenmp  
pvgcoen-9@pvgcoen9-ThinkCentre-M700:~/Desktop$ ./a.out  
Please Enter the rows of the matrix:- 2  
Please Enter the columns of the matrix:- 2  
Please Enter the values for matrix A:-  
Enter the value for [0][0]:- 2  
Enter the value for [0][1]:- 3  
Enter the value for [1][0]:- 5  
Enter the value for [1][1]:- 6

Please Enter the values for matrix B:-  
Enter the value for [0][0]:- 9  
Enter the value for [0][1]:- 8  
Enter the value for [1][0]:- 4  
Enter the value for [1][1]:- 6  
Result of matrix multiplication is:-  
30 34  
69 76  
Serial: 0.000057  
Result of matrix multiplication is:-  
30 34  
69 76  
Parallel-1(Static Scheduler) 0.007046

Result of matrix multiplication is:-  
30 34

69 76  
Parallel-2(Dynamic Scheduler) 0.000025  
\*/