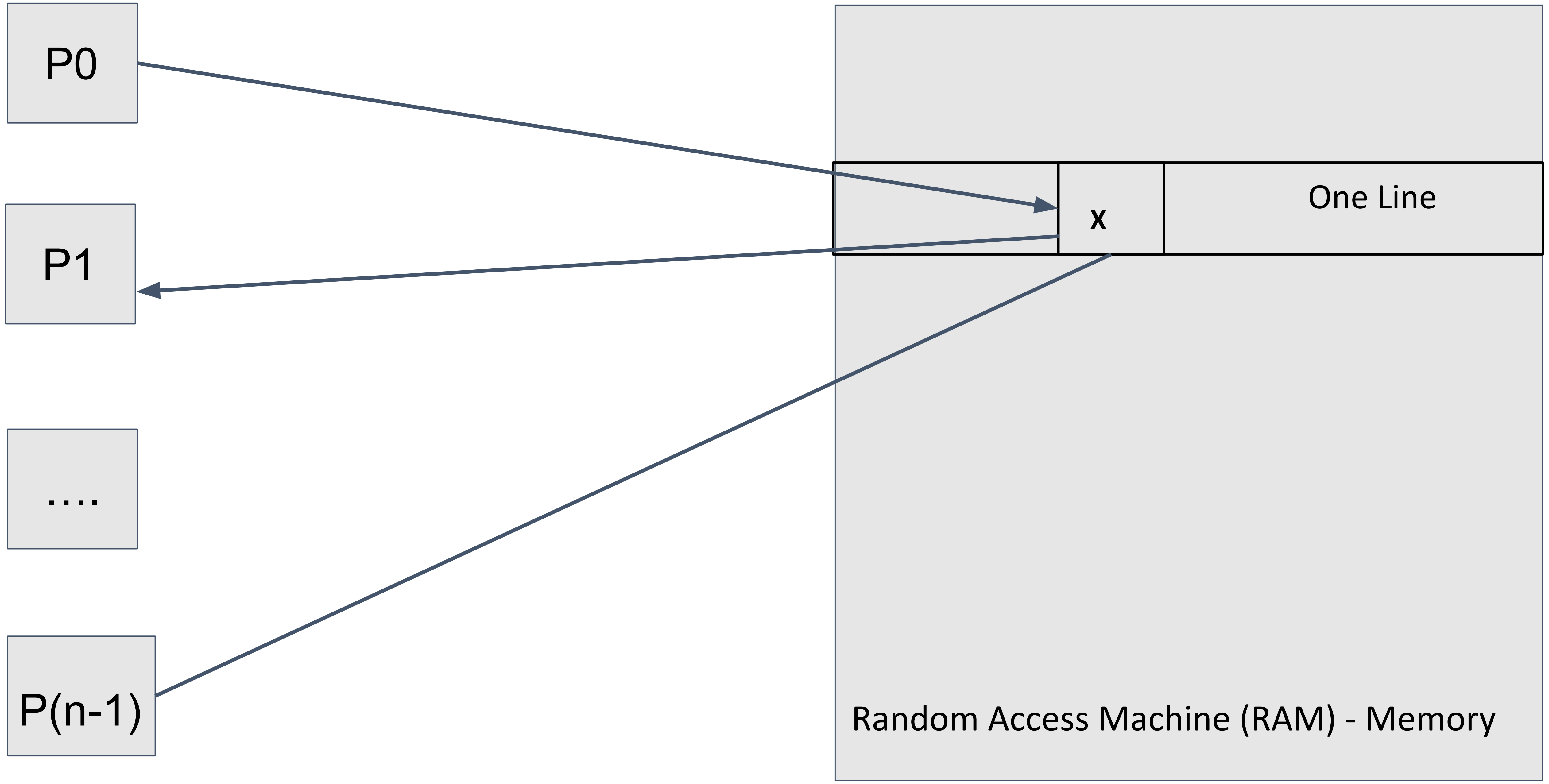
OpenMP

Parallel RAM Machine - MIMD



1. Parallel Writes or Parallel Read and Write leads to incorrect results

2. If all threads only writes, and write the same value It is okay.

Finding Maximum Element in an Array

```
#include<stdio.h>
#include<stdlib.h>
#define size 4096
int arr[size];
int max=0;
int main(int argc, char *argv[]){
    srand(atoi(argv[1]));
    for(int  i=0;i<size;i++)
        arr[i]=rand()%1048576;
for(int i=0;i<size;i++)
        if(max <arr[i])max=arr[i];
printf("max=%d\n",max);
}//end main
```

How to Parallelize this code?

Parallel Code

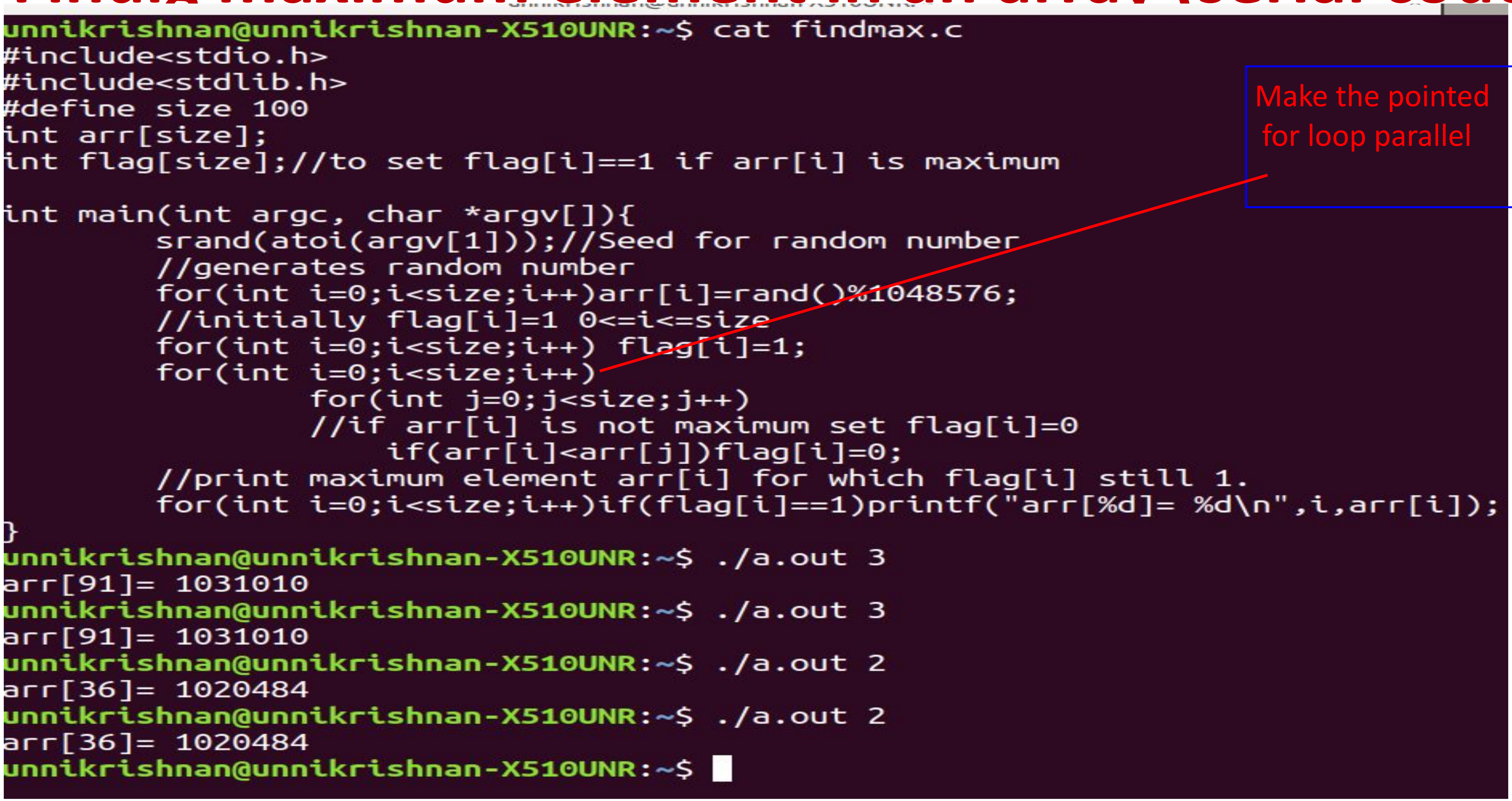
```
#include<stdio.h>
#include<stdlib.h>
#define size 4096
int arr[size];
omp_lock_t writelock;
   int main(int argc, char *argv[]){
omp_init_lock(&writelock);
     srand(atoi(argv[1]));
     for(int  i=0;i<size;i++)
           arr[i]=rand()%1048576;
```

```
#pragma omp parallel for
num_threads(12)
     for(int i=0;i<size;i++){
           omp_set_lock(&writelock);
          if(max < arr[i])   max=arr[i];

          omp_unset_lock(&writelock
          );
     }
     printf(" max=%d\n",max);
}//end main
```

Parallel Code with no atomic operations

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#define size 10000
int arr[size];
int flag[size];
int main(int argc, char *argv[]){
    srand(atoi(argv[1]));
    for(int i=0;i<size;i++)
        arr[i]=rand()%1048576;
 for(int i=0;i<size;i++) flag[i]=1;
#pragma omp parallel for num_threads(12)
    for(int i=0;i<size;i++)
        for(int j=0;j<size;j++)
          if(arr[i]<arr[j])flag[i]=0;
for(int i=0;i<size;i++)
    if(flag[i]==1)
      printf("arr[%d]= %d\n",i,arr[i]);
}//end man
```

Findig maximum element in an array (serial code)



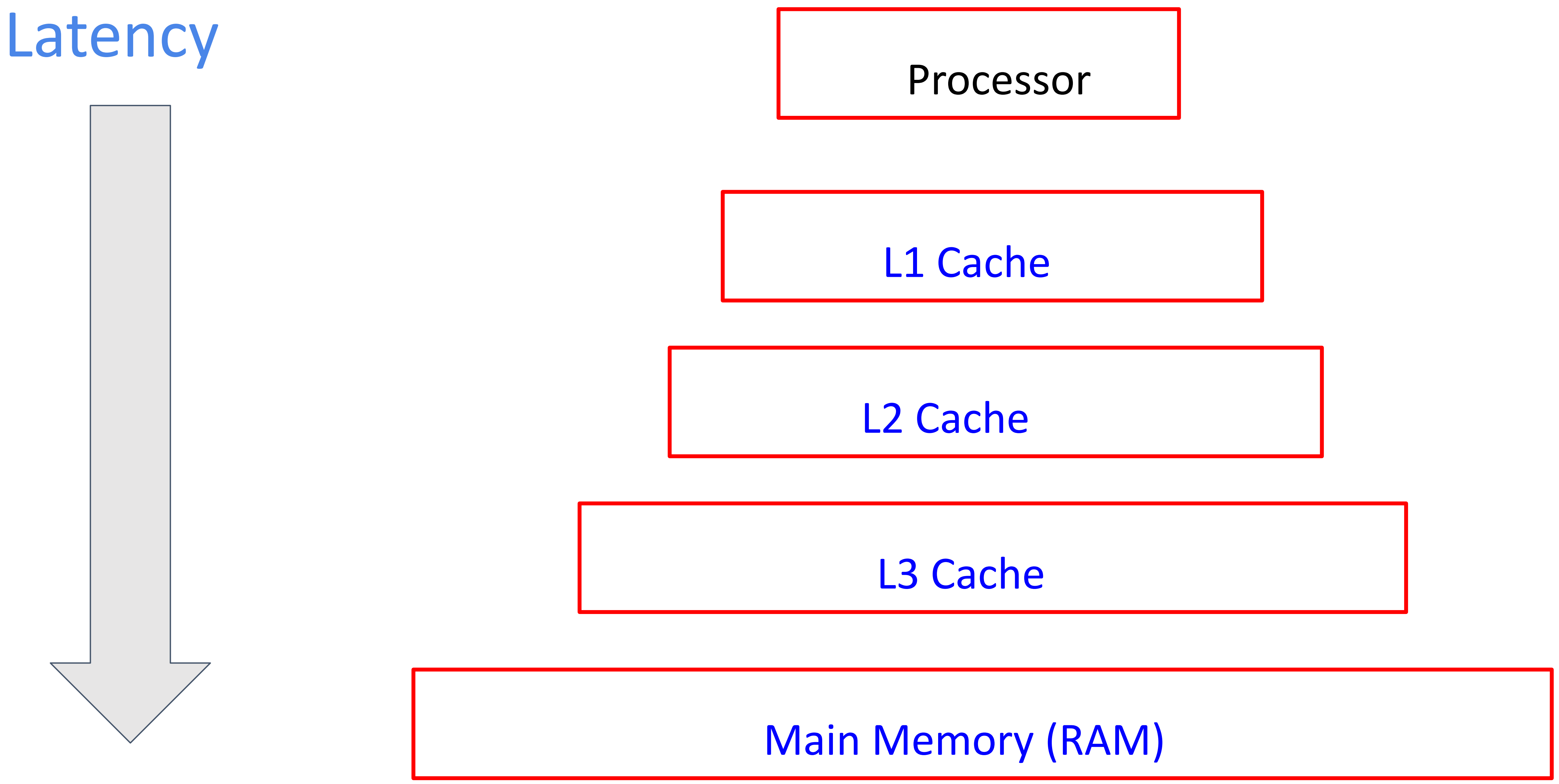
```
unnikrishnan@unnikrishnan-X510UNR:~$ cat findmax.c
#include<stdio.h>
#include<stdlib.h>
#define size 100
int arr[size];
int flag[size];//to set flag[i]==1 if arr[i] is maximum

int main(int argc, char *argv[]){
        srand(atoi(argv[1]));//Seed for random number
        //generates random number
        for(int i=0;i<size;i++)arr[i]=rand()%1048576;
        //initially flag[i]=1 0<=i<=size
        for(int i=0;i<size;i++) flag[i]=1;
        for(int i=0;i<size;i++)
                for(int j=0;j<size;j++)
                //if arr[i] is not maximum set flag[i]=0
                    if(arr[i]<arr[j])flag[i]=0;
        //print maximum element arr[i] for which flag[i] still 1.
        for(int i=0;i<size;i++)if(flag[i]==1)printf("arr[%d]= %d\n",i,arr[i]);
}
unnikrishnan@unnikrishnan-X510UNR:~$ ./a.out 3
arr[91]= 1031010
unnikrishnan@unnikrishnan-X510UNR:~$ ./a.out 3
arr[91]= 1031010
unnikrishnan@unnikrishnan-X510UNR:~$ ./a.out 2
arr[36]= 1020484
unnikrishnan@unnikrishnan-X510UNR:~$ ./a.out 2
arr[36]= 1020484
unnikrishnan@unnikrishnan-X510UNR:~$ 
```

Make the pointed for loop parallel

Matrices - How to reference

- **Temporal Locality**
    - A particular location if referenced, high chance it will be referenced again
- **Spatial Locality**
    - If a location is referenced, high chance its near by location is referenced soon.
- It is very important to consider locality of reference to reduce memory access time, for a fixed number of accesses.
- In C/C++ Programming language
    - Matrices are stored in a row major order.
    - So the pair of accesses (A[i][j], A[i][j+1]) will be faster
    - than the pair of accesses (A[i][j], A[i+1][j])
- Reason better locality of reference.

Comparison of Memory and Latency



Matrix Addition - Example

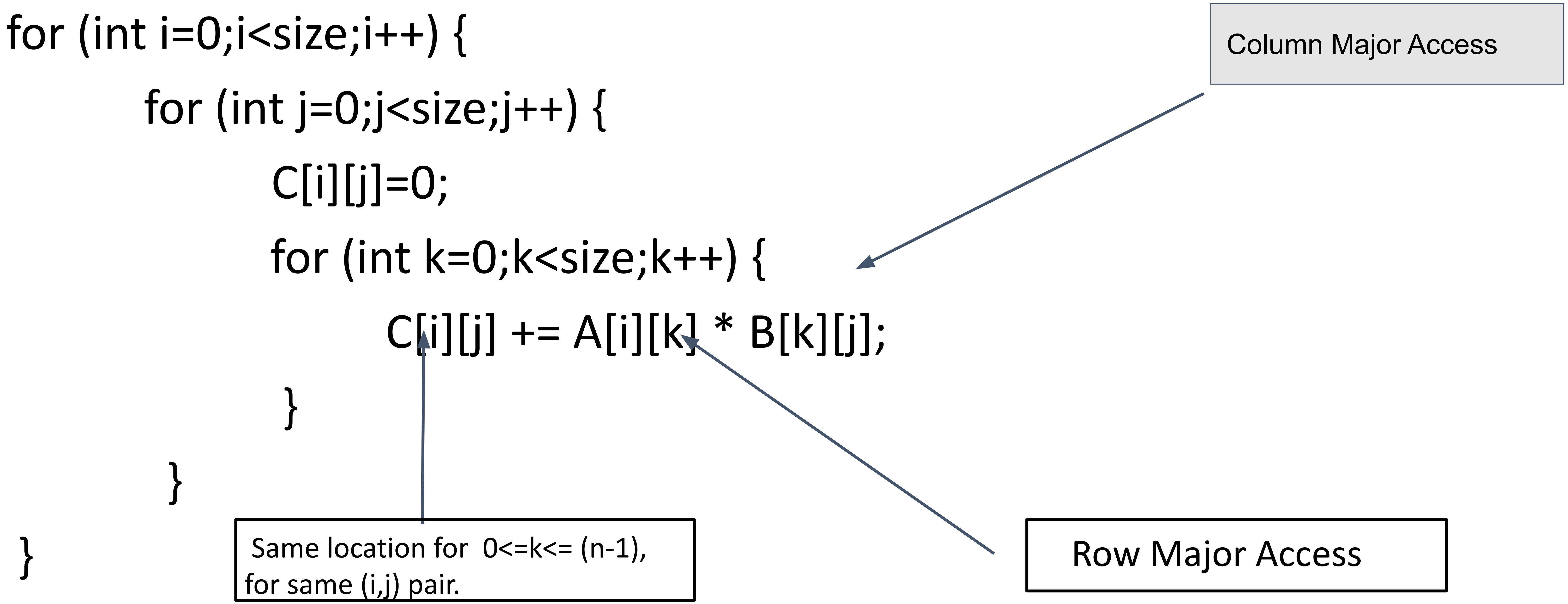
```
for (int i=0;i<N;i++)
  for(j=0;j<N;j++)
    A[j][i]=B[j][i]+C[j][i]
```

```
N=10000
Time= 2995 Milliseconds (Ms)
```

```
for (int i=0;i<N;i++)
  for(j=0;j<N;j++)
    A[i][j]=B[i][j]+C[i][j]
```

```
N=10000
Time= 328 Milliseconds (Ms)
```

Matrix Multiplication



```
for (int i=0;i<size;i++) {
        for (int j=0;j<size;j++) {
                C[i][j]=0;
                for (int k=0;k<size;k++) {
                        C[i][j] += A[i][k] * B[k][j];
                }
        }
}
```

Column Major Access

Same location for 0<=k<= (n-1),
for same (i,j) pair.

Row Major Access

Dependences

- True Dependence
  - Read after Write (**RAW**) to same memory location
- Anti Dependence
  - Write after Read (**WAR**) to same memory location
- Output Dependence
  - Write after Write (**WAW**) to same memory location.
- Parallel programs should preserve **sequential consistency**

Inter Iteration Dependence in Loops - An example

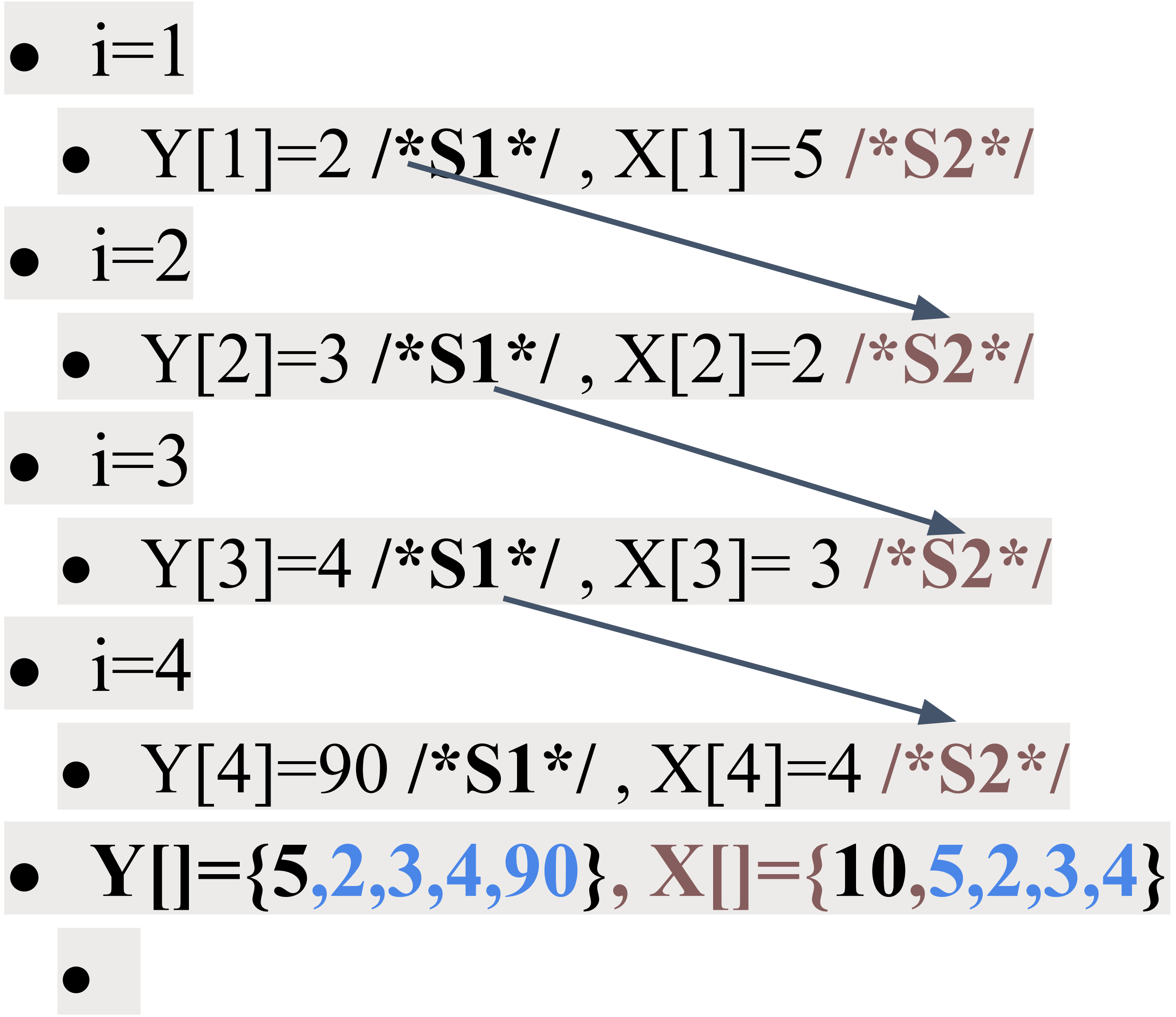
```
for (int i=1; i < 5; i++) {
        Y[i]   =  Z[i];  /*S1*/
        X[i]   = Y[i-1];  /*S2*/
}
```

SAMPLE INPUT

Z[5]= {1, 2,   3,   4, 90}

Y[5]={5, 10, 15, 20,   25}

X[5]={10, 20, 30, 40, 50}



- i=1
  - Y[1]=2 /***S1***/ , X[1]=5 /*S2*/
- i=2
  - Y[2]=3 /***S1***/ , X[2]=2 /*S2*/
- i=3
  - Y[3]=4 /***S1***/ , X[3]= 3 /*S2*/
- i=4
  - Y[4]=90 /***S1***/ , X[4]=4 /*S2*/
- **Y[]={5,2,3,4,90}, X[]={10,5,2,3,4}**
  -

Inter Iteration Dependence - An example

```
# pragma omp parallel for
for (int i=1; i < 5; i++) {
      Y[i]   =  Z[i];   /*S1*/
      X[i]   = Y[i-1];  /*S2*/
}
```

SAMPLE INPUT

Z[5]= {1, 2, 3, 4, 90}

Y[5]={5, 10, 15, 20, 25}

X[5]={10, 20, 30, 40, 50}

- i=1
  - Y[1]=2 /***S1***/ , X[1]=5 /*S2*/
- i=2
  - Y[2]=3 /***S1***/ , X[2]=10 /*S2*/
- i=3
  - Y[3]=4 /***S1***/ , X[3]= 15 /*S2*/
- i=4
  - Y[4]=90 /***S1***/ , X[4]=20 /*S2*/
- **Y[]={5,2,3,4,90}, X[]={10,5,10,15,20}**
- Sequential Consistency not preserved

Transformed code with only intra iteration dependence

```
for (int i= 1 ; i < 5; i++) {
      Y[i]   =  Z[i];  /*S1*/
      X[i]   = Y[i-1];  /*S2*/
  }
```

SAMPLE INPUT

```
Z[5]= {1,  2,   3,   4,  90}

Y[5]={5,  10,  15,  20,   25}

X[5]={10,  20,  30,  40,  50}
```

```
 X[1]=Y[0];  //L1       prologue
#pragma omp parallel for
for (I=1;I<4);I++){
  Y[I]=Z[I];//L2
  X[I+1]= Y[I];//L3
}
Y[4]= Z[4]; //L4 epilogue

  X[1]=Y[0]=5;//L1
 X[2,3,4]= Y[1,2,3]=Z[1,2,3]=[2,3,4];//L2,L3
  Y[4]=Z[4]=90 //L4
```

Nested Parallelism

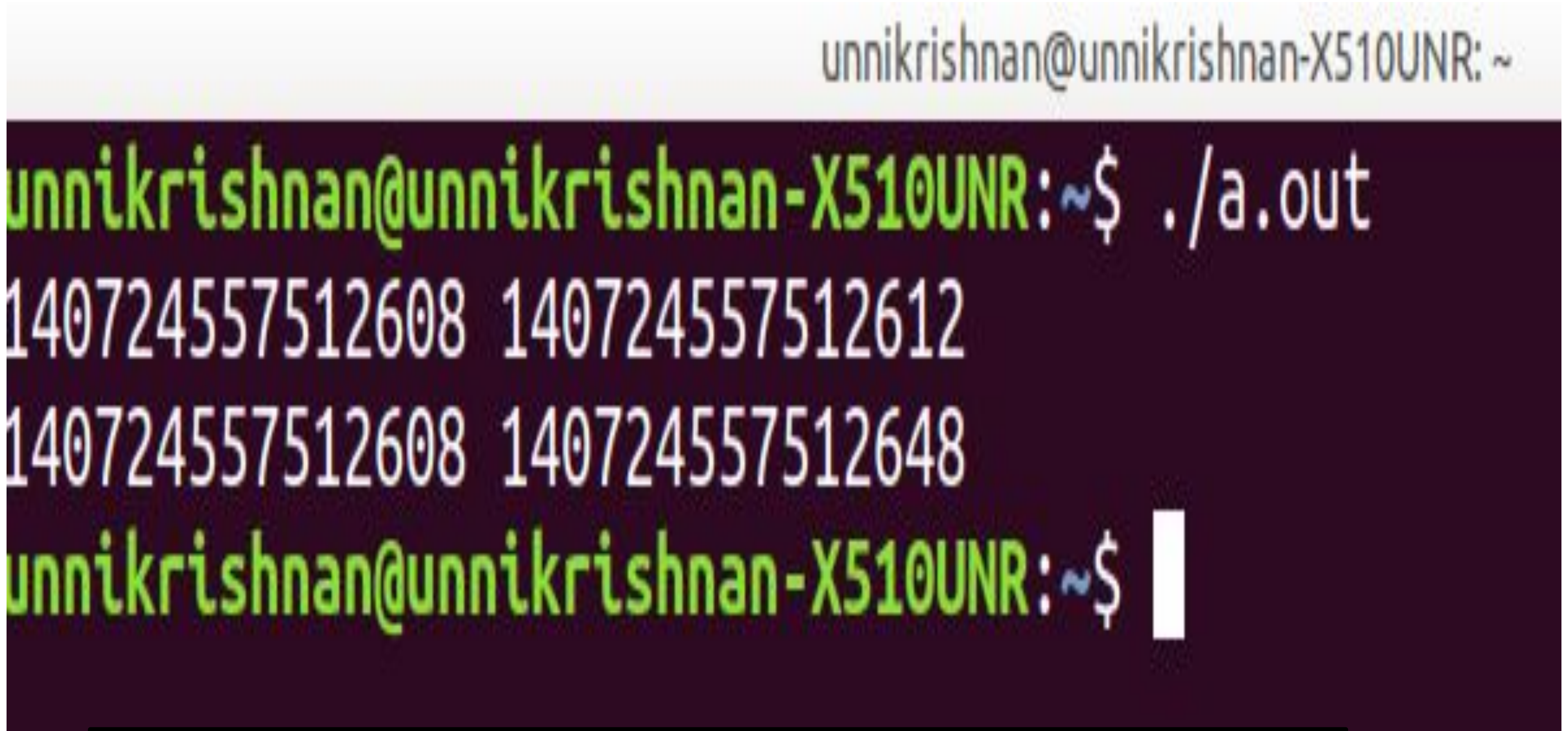
```
#pragma omp parallel for
schedule(dynamic,1) collapse(2)
 for (int i = 0; i < m; i++) {
     for (int j = 0; j < n; j++) {
     //parallel code
 }//end for

}//end for
```

Collapse supported for OpenMP Version >= 3.0

```
 #pragma omp parallel for
for(int ij=0;ij<m*n;ij++){
   int i= ij/n;
   int j=ij%n;
 //parallel code
}
```

Matrix- Storage

```
int main(){
        int arr[10][10];
printf("%ld %ld\n", &arr[2][0],
&arr[2][1]);
printf("%ld %ld\n", &arr[2][0],
&arr[3][0]);
/*printf("%ld %ld\n", (int *)arr+20,
(int *)arr+21);*/
}
```



```
unnikrishnan@unnikrishnan-X510UNR:~$ ./a.out
140724557512608 140724557512612
140724557512608 140724557512648
unnikrishnan@unnikrishnan-X510UNR:~$
```

Address of arr[[2][0]= X, sizeof(int)= 4 bytes

Output

X X+4

X X+40

Instruction Level Parallelism (ILP)

- Five stage superscalar Processor
  - IF, ID, EX, MEM, WB
- SISD Processor

```
for (int i=0;i< n;i++)
   D[i]=A[i]*B[i]+C;
```

Loop Unrolling improves instruction level parallelism.

To Measure Running Time

```
double rtclock()
{
   struct timezone Tzp;
   struct timeval Tp;
   int stat;
   stat = gettimeofday (&Tp, &Tzp);
   if (stat != 0) printf("Error return from gettimeofday: %d",stat);
   return(Tp.tv_sec + Tp.tv_usec*1.0e-6);
}
```