Sequential Selection Sort

0 1 2 3 index

4 2 3 4 data

I iteration Min = 4 Pos=0

4&2 2&3 2&4

Min=2

Pos=1

A[1]=4 A[0]=2

2 4 3 4

2 Iteration Min = 4 Pos=1

4&3 4&3

Min = 3

Pos = 2

A[2]=4 A[1]=3

2 3 4 4

3rd Iteration Min = 4 Pos=2

4&4

No change

i/p: Array A

0 1 2 3

4 2 3 4

Pos=2 pos=0 pos=1 pos=3

o/p: Array B

0 1 2 3

2 3 4 4

//parallel sort

\_\_kernel void sort( \_\_global int \*A, \_\_global int \*B)

{

int id = get\_global\_id(0);

int data = A[id];

int pos=0,i;

int N = get\_global\_size(0);

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

{

If((A[i] < data) || (A[i]==data && i < id))

pos++;

}

B[pos]=data;

}

Sequential

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

{

Min = A[i];

Pos = i;

for ( j=i+1; j<n;j++)

{

if( A[j] < Min)

{

Min = A[j];

Pos = j;

}

}

A[Pos] = A[i];

A[i] = Min;

}

Odd-even transposition sorting:

Index: 0 1 2 3 4

23 76 89 12 9

I o/e 23 76 89 9 12

e/o 23 76 9 89 12

II o/e 23 9 76 12 89

e/o 9 23 12 76 89

III o/e 9 12 23 76 89

e/o 9 12 23 76 89

F1.cl

\_\_kernel void odd(\_\_global int\* A)

{

int idx=get\_global\_id(0);

int N=get\_global\_size(0);

if((idx%2)!=0 && idx+1<=N-1)

{

if(A[idx]>=A[idx+1])

{

int temp=A[idx];

A[idx]=A[idx+1];

A[idx+1]=temp;

}

}

}

\_\_kernel void even(\_\_global int\* A)

{

int idx=get\_global\_id(0);

int size=get\_global\_size(0);

if((idx%2)==0 && idx+1<=size-1)

{

if(A[idx]>=A[idx+1])

{

int temp=A[idx];

A[idx]=A[idx+1];

A[idx+1]=temp;

}

}

}

Host

Buffer - BufferA

program

cl\_kernel kernel1 = clCreateKernel(program, "odd", &status);

cl\_kernel kernel2 = clCreateKernel(program, "even", &status);

globalWorkSize = N;

for(int i=0;i<N/2;i++)

{

status = clEnqueueNDRangeKernel( cmdQueue, kernel1, 1, NULL, &globalWorkSize,

NULL, 0, NULL, NULL);

status = clEnqueueNDRangeKernel( cmdQueue, kernel2, 1, NULL, &globalWorkSize,

NULL, 0, NULL, NULL);

}

clEnqueueReadBuffer( cmdQueue, bufferA, CL\_TRUE, 0, datasize, A, 0, NULL, NULL);

Input:

Istr : Hello //5 \_

N : 3

Ostr: HelloHelloHello 5\*3

HelloHelloHello

Work item : 0 1 2

Index :

//host

Istr – input string scanf

int len=strlen(Istr); //5

cl\_mem BufferIstr = clCreateBuffer(context, CL\_MEM\_READ\_ONLY, len\*sizeof(char), NULL,&status); Istr

// copy Istr to BufferIstr

cl\_mem BufferOstr = clCreateBuffer(context, CL\_MEM\_WRITE\_ONLY, len\*N\*sizeof(char), NULL,&status);

globalworksize = N ; //3 work items 0 1 2

clSetKernelArg(kernel, 0, &BufferIstr);

clSetKernelArg(kernel, 1, &BufferOstr);

clSetKernelArg(kernel,2, sizeof(cl\_int),&len); //

NDrangekernel

//copy BufferOstr to Ostr

Ostr[len\*N] = ‘\0’;

//display Ostr

//kernel

\_\_kernel void str\_cpy(\_\_global char \* Istr, \_\_global char \*Ostr, int len )

{

int id=get\_global\_id(0); 0 1 2

int i;

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

Ostr[id\*len+i] = Istr[i];

}

//2nd method

globalworksize = len; // 5work items 0 1 2 3 4

clSetKernelArg(kernel, 0, &BufferIstr);

clSetKernelArg(kernel, 1, &BufferOstr);

clSetKernelArg(kernel,2, sizeof(cl\_int),&N); //

NDrangekernel

//copy BufferOstr to Ostr

Ostr[len\*N] = ‘\0’;

//display Ostr

//kernel

\_\_kernel void str\_cpy(\_\_global char \* Istr, \_\_global char \*Ostr, int N)

{

int id=get\_global\_id(0); 0 1 2 3 4

int len = get\_global\_size(0); //5

int pos =id;

int i;

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

{

Ostr[pos] = Istr[id];

pos = pos + len;

}

}

Istr : Hello

Ostr: HelloHelloHello

Work item : 0 1 2

Ist iteration i=0

Ostr: H H H

Work item : 0 1 2

Position : 0 5 10

2nd iteration i=1

Ostr: He He He

Work item : 0 1 2

Position : 1 6 11

3rd iteration i=2

Ostr: Hel Hel Hel

Work item : 0 1 2

Position : 2 7 12

4th iteration i=3

Ostr: Hell Hell Hell

Work item : 0 1 2

Position : 3 8 13

5th iteration i=4

Ostr: HelloHelloHello

Work item : 0 1 2

Position : 4 9 14

//2nd method

Ostr: HelloHelloHello

Work item : 012340123401234

Ist iteration

Ostr: Hello

Work item : 01234

Position : 01234

2nd iteration

Ostr: HelloHello

Work item : 01234

Position : 56789

3rd iteration

Ostr: HelloHelloHello

Work item : 01234

Position : 1011121314