Questions to be solved

Explain the work implementation of the local memory in OpenCL. Write a simple code

\_\_kernel squared(

\_\_global float \*input,

\_\_global float \*output,

\_\_local float \*temp,

const unsigned int count)

{

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

int lid = get\_local\_id(0);

if (gid < count)

{

temp[lid] = input[gid];

// if the threads were reading data from other threads, then we would

// want a barrier here to ensure the write completes before the read

output[gid] = temp[lid] \* temp[lid];

}

}

--------------------------------------------------------------------------------------------------------------------------------------

Explain why a parallel program might not achieve linear speedup in a multicore system? (Speedup of p with p processors = Linear)

-Synchronization

-require communications, synchronizations between parallel executions, often blocking execution

-– Adapt your code to work with local memory ( solution )

- another algorithm

- Parallel Overhead

- All forms of parallelism bring a small overhead : loading a library, launching threads

- – Monitor software and OS resources

-Remember that some parallel framework are light, designed for single computers and small task while others are very heavy, designed for large clusters

state the conditions under which a transaction must be aborted / retired in a hardware transactional memory system

--------------------------------------------------------------------------------------------------------------------------------------

OpenCL Kernel Function for Merge Sort and Binary Search

Merge Sort

**\_\_kernel** **void** ParallelMerge\_Local(**\_\_global** **const** float \*in,**\_\_global** float \*out,**\_\_local** float \*aux)

{

**int** i = **get\_local\_id**(0); ***// index in workgroup***

**int** s = **get\_local\_size**(0); ***// workgroup size = block size, power of 2***

***// Move IN, OUT to block start***

**int** offset = **get\_group\_id**(0) \* s;

in += offset; out += offset;

***// Load block in AUX[WG]***

aux[i] = in[i];

**barrier**(CLK\_LOCAL\_MEM\_FENCE); ***// make sure AUX is entirely up to date***

***// Now we will merge sub-sequences of length 1,2,...,WG/2***

**for** (**int** length=1;length<s;length<<=1)

{

Float iData = aux[i];

uint iKey = getKey(iData);

**int** ii = i & (length-1); ***// index in our sequence in 0..length-1***

**int** sibling = (i - ii) ^ length; ***// beginning of the sibling sequence***

**int** pos = 0;

**for** (**int** inc=length;inc>0;inc>>=1) ***// increment for dichotomic search***

{

**int** j = sibling+pos+inc-1;

uint jKey = getKey(aux[j]);

bool smaller = (jKey < iKey) || ( jKey == iKey && j < i );

pos += (smaller)?inc:0;

pos = min(pos,length);

}

**int** bits = 2\*length-1; ***// mask for destination***

**int** dest = ((ii + pos) & bits) | (i & ~bits); ***// destination index in merged sequence***

**barrier**(CLK\_LOCAL\_MEM\_FENCE);

aux[dest] = iData;

**barrier**(CLK\_LOCAL\_MEM\_FENCE);

}

***// Write output***

out[i] = aux[i];

}

Binary Search

\_\_global const \_int \*upper\_bound(\_\_global const \_int \*begin,

\_\_global const \_int \*end, const \_int elem)

{

while(begin != end) {

\_\_global const \_int \*mid = begin + (end - begin) / 2;

#if 0

if(!(elem < \*mid))

begin = mid + 1; // look to the right

else

end = mid; // look to the left

#else // 0

bool b\_right = !(elem < \*mid);

begin = (\_\_global const \_int \*)select((intptr\_t)begin, (intptr\_t)(mid + 1), b\_right);

end = (\_\_global const \_int \*)select((intptr\_t)mid, (intptr\_t)end, b\_right); // c ? b : a

#endif // 0

}

return begin;

}

--------------------------------------------------------------------------------------------------------------------------------------

Write a histogram estimator for data distribution. Use bins (value intervals) to count the number of values falling into that interval.

