# 2014-02-20--CPU执行与GPU执行的区别（21：00）

我们发现，对于同样的计算规模（1000），当采用下面的方法时基于CPU的方法执行时间较长，因为我们做了一次全局内存到局部内存的拷贝。下面的代码在GPU上运行时，计算时间（6.73）较不用拷贝的方法（7.41）执行时间要短。

\_\_kernel void computemap(\_\_global uint\* startx1,\_\_global uint\* endx1,\_\_global uint\* y1,\_\_global uint\* startx2,\_\_global uint\* endx2,\_\_global uint\* y2,\_\_global uint\* map){

size\_t xId=get\_global\_id(0);

size\_t yId=get\_global\_id(1);

\_\_local volatile uint localmap[2\*ARRAY\_SIZE];

uint val=y1[xId]+y2[yId];

uint endx=endx1[xId]+endx2[yId];

uint startx=startx1[xId]+startx2[yId];

for (int i=startx;i<=endx;i++)

atomic\_min(localmap+i,val);

barrier(CLK\_LOCAL\_MEM\_FENCE);

map[2\*xId]=localmap[2\*xId];

map[2\*xId+1]=localmap[2\*xId+1];

}

当把上述的代码中的内存拷贝去掉后，代码的执行时间会更短，对于1000的规模，执行时间仅1.44ms,而同样的代码在GPU上运行时间是7.41。

\_\_kernel void computemap(\_\_global uint\* startx1,\_\_global uint\* endx1,\_\_global uint\* y1,\_\_global uint\* startx2,\_\_global uint\* endx2,\_\_global uint\* y2,\_\_global uint\* map){

size\_t xId=get\_global\_id(0);

size\_t yId=get\_global\_id(1);

uint val=y1[xId]+y2[yId];

uint endx=endx1[xId]+endx2[yId];

uint startx=startx1[xId]+startx2[yId];

for (int i=startx;i<=endx;i++)

atomic\_min(map+i,val);

}

# 2014-02-20--C++与Java的性能比较（21：30）

cvalue1=(unsigned int\*)malloc(1000\*sizeof(unsigned int));

cvalue2=(unsigned int\*)malloc(1000\*sizeof(unsigned int));

cendx1=(unsigned int\*)malloc(1000\*sizeof(unsigned int));

cendx2=(unsigned int\*)malloc(1000\*sizeof(unsigned int));

cstartx1=(unsigned int\*)malloc(1000\*sizeof(unsigned int));

cstartx2=(unsigned int\*)malloc(1000\*sizeof(unsigned int));

output=(unsigned int\*)malloc(2000\*sizeof(unsigned int));

for (int i=0;i<1000;i++){

cvalue1[i]=i;

cvalue2[i]=i;

cendx1[i]=i+1;

cendx2[i]=i+1;

cstartx1[i]=i;

cstartx2[i]=i;

output[2\*i]=10000;

output[2\*i+1]=10000;

}

time\_t start=clock();

for (int i=0;i<1000;i++){

for (int j=0;j<1000;j++){

int endx=cendx1[i]+cendx2[j];

int value=cvalue1[i]+cvalue2[j];

int startx=cstartx1[i]+cstartx2[j];

for (unsigned int k=startx;k<endx;k++){

output[k]=(output[k]<value)?output[k]:value;

}

}

}

cout<<"time="<<clock()-start<<endl;

上述代码在VS2010中运行的时间是19ms.而同样的代码在Java中的执行时间为14ms.

**int** [] endx1=**new** **int**[1000];

**int** [] endx2=**new** **int**[1000];

**int** [] value1=**new** **int**[1000];

**int** [] value2=**new** **int**[1000];

**int** [] startx1=**new** **int**[1000];

**int** [] startx2=**new** **int**[1000];

**int** [] result=**new** **int**[2000];

**for** (**int** i=0;i<1000;i++){

endx1[i]=i+1;

endx2[i]=i+1;

value1[i]=i;

value2[i]=i;

result[2\*i]=10000;

result[2\*i+1]=10000;

startx1[i]=i;

startx2[i]=i;

}

**long** starttime=System.*currentTimeMillis*();

**for** (**int** i=0;i<1000;i++){

**for** (**int** j=0;j<1000;j++){

**int** endx=endx1[i]+endx2[j];

**int** value=value1[i]+value2[j];

**int** startx=startx1[i]+startx2[j];

**for** (**int** k=startx;k<endx;k++){

result[k]=(result[k]<value)?result[k]:value;

}

}

}

System.*out*.println(System.*currentTimeMillis*()-starttime);

通过上面的实验可以发现：Java与C++的性能差异可能不大；而OpenCL的确可以提高计算性能，而且性能提升在15倍以上。