Improvements

BITES Internship @ AR-GE

Hilal Özdemir

Contents

- What is OpenCL
- BArray
- BComplex
- BSignal
- BMathEngine
    - getConnected, setKernel, setGlobalSize, setLocalSize, runKernel1D, bitReverse, deallocResources
    - add, subtract, multiply, divide, pow, conv, dft, idft
    - Incomplete: fft, chirpZTransform
    - Kernel Code

What is OpenCL

Link to the first presentation

BArray

```
template <typename T>
class BArray{
protected:
      BArray(int pSize)
   {
      //0 initialization erased
      mData = new T[pSize];
      mSize = pSize;
   }
```

```
public:
int getSize() const { … }
T& getData() const { … }
T& operator[](int pIndex) const { … }
void fill(T value) { … }
~BArray() { … }
private:
   T *mData;
   int mSize;
};
```

BComplex

```
class BComplex
{
private:
    float mRe;
    float mIm;
public:
    BComplex();
    BComplex(float pRe, float pIm);
    float re() const;
    float im() const;
void setRe(float pRe);
void setIm(float pIm);
BComplex operator+ (const BComplex& operand) const;
BComplex operator- (const BComplex& operand) const;
BComplex operator* (const BComplex& operand) const;
BComplex operator/ (const BComplex& operand) const;
BComplex& operator= (const float rhs);
BComplex& operator= (const BComplex &rhs);

};
```

BSignal

```
#include "BFloat32Array.h"
class BSignal
{
private:
public:
    BSignal();
    BSignal(BFloat32Array &arr, int arrb);
    BFloat32Array *arr;
    int arrb;
};
```

BFloat32Array class:

```
#include "BArray.h"
class BFloat32Array:public BArray<float>{
public:
    BFloat32Array(int pSize);
};
```

BMathEngine GetConnected

```
bool BMathEngine::getConnected()
{
    /* Build program */
    program = clCreateProgramWithSource(context, 1, (const char **) & KernelSource, NULL, &err);
   ...
    err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
   ...
    /* Create a command queue */
    commands = clCreateCommandQueue(context, device_id, 0, &err);
   ...
    errMessage="";
    return true;
};
```

BMathEngine SetKernel

```
bool BMathEngine::setKernel(char funcName[])
{
    kernel = clCreateKernel(program,funcName, &err);
    ...
    this->currFuncName = funcName;
    errMessage="";
    return true;
}
```

BMathEngine SetGlobalSize/SetLocalSize

```
void BMathEngine::setGlobalSize(size_t globalSize)
{
    this->global_size=globalSize;
}

void BMathEngine::setLocalSize(size_t localSize)
{
    this->local_size=localSize;
}
```

BMathEngine RunKernel1D

- bool BMathEngine::runKernel1D(float &input1, float &input2, float &output, int count)

Functions: add, subtract, multiply, divide, pow, conv

- bool BMathEngine::runKernel1D(float &input1, float &input2, float &output1, float &output2, int count)

Functions: dft, idft

- bool BMathEngine::runKernel1D(float &input, float &output, int count)

Functions: bitReverse

BMathEngine RunKernel1D

```
bool BMathEngine::runKernel1D(float &input1, float &input2, float &output, int count)
{
   /* Create data buffer */
   input_buffer1 = clCreateBuffer(context, CL_MEM_READ_ONLY |
            CL_MEM_COPY_HOST_PTR, this->input1Size * sizeof(float), &input1, &err);
   input_buffer2 = clCreateBuffer(context, CL_MEM_READ_ONLY |
            CL_MEM_COPY_HOST_PTR, this->input2Size * sizeof(float), &input2, &err);
   output_buffer = clCreateBuffer(context, CL_MEM_READ_WRITE |
            CL_MEM_COPY_HOST_PTR, this->outputSize * sizeof(float), &output, &err);
   ...
```

BMathEngine RunKernel1D

```
//Set kernel arguments
 err = 0;
 err  = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input_buffer1);
 err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &input_buffer2);
 err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &output_buffer);
 if (strcmp(this->currFuncName,"conv")==0) {
     err |= clSetKernelArg(kernel, 3, local_size * sizeof(float), NULL);
     err |= clSetKernelArg(kernel, 4, sizeof(int), &this->input2Size);
 }else{
     err |= clSetKernelArg(kernel, 3, sizeof(int), &this->input2Size);
 }
```

BMathEngine RunKernel1D

```
...
    //Run Kernel
    if (local_size == 0) {
        err = clEnqueueNDRangeKernel(commands, kernel, 1, NULL, &global_size,
                                     NULL, 0, NULL, NULL);
    }else{
        err = clEnqueueNDRangeKernel(commands, kernel, 1, NULL, &global_size,
                                     &local_size, 0, NULL, NULL);
    }
...
```

BMathEngine RunKernel1D

```
…
    //Read the kernel’s output
    err = clEnqueueReadBuffer(commands, output_buffer, CL_TRUE, 0,
                              sizeof(float)*count, &output, 0, NULL, NULL);
…
    //Deallocation
    clReleaseMemObject(output_buffer);
    clReleaseMemObject(input_buffer1);
    clReleaseMemObject(input_buffer2);
}
```

BMathEngine BitReverse

BitReverse function will be used by the fft function.

```
bool BMathEngine::bitReverse(const BFloat32Array &input, BFloat32Array &output)
{
    this->input1Size=input.getSize();
    this->outputSize=output.getSize();
    bool temp = getConnected();
    …
    int size = input1Size;
    setGlobalSize(size);
    setLocalSize(0);
```

BMathEngine BitReverse

```
    char func[] = "bitReverse";
    temp = setKernel(func);
    …
    temp = runKernel1D(input.getData(), output.getData(), size);
    …
    deallocResources();

    errMessage="";
    return true;
}
```

BMathEngine DeallocResources

```
void BMathEngine::deallocResources(){
    this->input1Size=0;
    this->input2Size=0;
    this->outputSize=0;
    this->local_size=0;
    this->global_size=0;
    this->currFuncName=NULL;
    /* Deallocate resources */
    clReleaseKernel(kernel);
    clReleaseCommandQueue(commands);
    clReleaseProgram(program);
}
```

BMathEngine Add

- bool BMathEngine::add(const BFloat32Array &input1, const BFloat32Array &input2, BFloat32Array &output)

- bool BMathEngine::add(const BFloat32Array &input1, const float input2, BFloat32Array &output)

BMathEngine Add

```
bool BMathEngine::add(const BFloat32Array &input1, const BFloat32Array &input2,
BFloat32Array &output)
{
    if (input1.getSize() != input2.getSize() || input1.getSize() != output.getSize()) {
        errMessage = "Size mismatch!";
        return false;
    }else{
   …
```

BMathEngine Add

```
...
this->input1Size=input1.getSize();
this->input2Size=input2.getSize();
this->outputSize=output.getSize();
bool temp = getConnected();
...
int size = input1.getSize();
setGlobalSize(size);
setLocalSize(0);
...
```

BMathEngine Add

```
char func[] = "add";
        temp = setKernel(func);
        ...
        temp = runKernel1D(input1.getData(), input2.getData(), output.getData(), size);
        ...
        deallocResources();
        errMessage="";
        return true;
    }
}
```

BMathEngine Subtract/Multiply/Divide/Pow

Very similar to the “add” function.

BMathEngine Conv

- bool BMathEngine::conv(const BSignal &input1, const BSignal &input2, BSignal &output)

- bool BMathEngine::conv(const BFloat32Array &input1, const BFloat32Array &input2, BFloat32Array &output)

BMathEngine Conv

```
bool BMathEngine::conv(const BFloat32Array &input1, const BFloat32Array &input2,
BFloat32Array &output)
{
    int outSignalSize = input1.getSize() + input2.getSize() - 1;
    this->input1Size=input1.getSize();
    this->input2Size=input2.getSize();
    this->outputSize=outSignalSize;

    output = *new BFloat32Array(outSignalSize);
    ...
```

BMathEngine Conv

```
    bool temp = getConnected();
    ...
    setGlobalSize(outSignalSize*input1Size);
    setLocalSize(input1Size);
    char func[] = "conv";
    temp = setKernel(func);
    …
    temp = runKernel1D(input1.getData(), input2.getData(), output.getData(), outSignalSize);
    deallocResources();
    errMessage="";
    return true;
}
```

BMathEngine Dft

- bool BMathEngine::dft(const BFloat32Array &input, BComplex32Array &output)

- bool BMathEngine::dft(const BComplex32Array &input, BComplex32Array &output)

BMathEngine Dft

```
bool BMathEngine::dft(const BFloat32Array &input, BComplex32Array &output)
{
    BFloat32Array outputRe(input.getSize());
    BFloat32Array outputIm(input.getSize());
    outputIm.fill(0);
    outputRe.fill(0);
    BFloat32Array inputIm(input.getSize());
    inputIm.fill(0);

    ...
```

BMathEngine Dft

```
// Perform dft
this->input1Size=input.getSize();
this->input2Size=input.getSize();
this->outputSize=output.getSize();
bool temp = getConnected();
…
int size = input1Size;
setGlobalSize(size);
setLocalSize(0);
...
```

BMathEngine Dft

```
    char func[] = "dft";
    temp = setKernel(func);
    …
    temp = runKernel1D(input.getData(),inputIm.getData(), outputRe.getData(),    outputIm.getData(), input1Size);
    …
    for (int i=0; i<outputSize; i++) {
        output[i].setRe(outputRe[i]);
        output[i].setIm(outputIm[i]);
    }
    deallocResources(); errMessage=""; return true;
}
```

BMathEngine Idft

Very similar to the “dft” function.

BMathEngine Kernel Code

```
#define PI 3.14159265358979323846
__kernel void add(
   __global float* input1,
   __global float* input2,
   __global float* output,
   const unsigned int count)
{
   int i = get_global_id(0);
   if(i < count)
       output[i] = input1[i] + input2[i];
}
```

BMathEngine Kernel Code

(Subtract, multiply, divide functions are similar to the add function.)

```
__kernel void powArr(
  __global float* input1,
  __global float* input2,
  __global float* output,
  const unsigned int count)
{
  int i = get_global_id(0);
  if(i < count)
     output[i] = pow(input1[i],input2[i]);
}
```

BMathEngine Kernel Code

Convolution:

$$\sum_{k=-\infty}^{\infty} x(k).h(n-k)$$

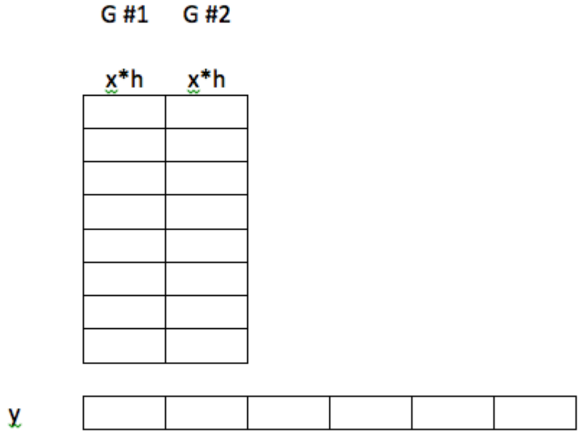
BMathEngine Kernel Code

Pseudocode (not in parallel) :

```
for n=0:y.size() begin
    float sum = 0.0f;
    for k=0:x.size() begin
        if((n-k)>=0 && (n-k)<h.size()) begin
            sum += x[k]*h[n-k];
        endif
    endfor
    global_result[n]=sum;
endfor
```

BMathEngine Kernel Code

- The bold part of the pseudocode is the part that each work item will handle.
- Variables:
  - local_size = x.size()
  - global_size = x.size()*y.size()
  - num_of_groups = y.size()



BMathEngine Kernel Code

```
__kernel void conv(
   __global float* input1,
   __global float* input2,
   __global float* output,
   __local float* local_result,
   const unsigned int count2) {
    float sum;

    int n = get_group_id(0);
    int k = get_local_id(0);
```

BMathEngine Kernel Code

```
if((n-k)>=0 && (n-k)<count2){
    local_result[k] = input1[k]*input2[n-k];
}
barrier(CLK_LOCAL_MEM_FENCE);
if(get_local_id(0) == 0) {
    sum = 0.0f;
    for(int i=0; i<get_local_size(0); i++) {
        sum += local_result[i];
    }
    output[get_group_id(0)] = sum;
} }
```

BMathEngine Kernel Code

```
__kernel void bitReverse(
   __global float* input,
   __global float* output,
   const unsigned int count)
{
   unsigned int position = get_global_id(0);
   unsigned int target = 0;
   unsigned int counter = count-1;
   unsigned int temp;
   ...
```

BMathEngine Kernel Code

```
    while(counter){
        counter = counter>>1;
        target = target<<1;
        temp = position&1;
        target+=temp;
        position = position>>1;
     }
    position = get_global_id(0);
    output[target] = input[position];
}
```

BMathEngine Kernel Code

```
__kernel void dft(
  __global float* inputRe,
  __global float* inputIm,
  __global float* outputRe,
  __global float* outputIm,
  const unsigned int count)
{
  unsigned int position = get_global_id(0);
  float realSum = 0;
  float imagSum = 0;
```

BMathEngine Kernel Code

```
  int i;
  for (i = 0; i < count; i++) {
      float angle = 2 * PI * i * position / count;
      realSum +=  inputRe[i] * cos(angle) + inputIm[i] * sin(angle);
      imagSum += -inputRe[i] * sin(angle) + inputIm[i] * cos(angle);
  }
  outputRe[position] = realSum;
  outputIm[position] = imagSum;

}
```

BMathEngine Kernel Code

- Idft function is very similar to dft function. Different three lines of code are:

```
float angle = -2 * PI * i * position / count;
...
outputRe[position] = realSum/count;
outputIm[position] = imagSum/count;
```

Questions?

Hilal Özdemir

BITES
Defence & Aerospace Technologies Inc.

July 2014

Thank you for listening.