Assignment 4, CS223 (Hardware Lab)

Dept of Comp. Sc. & Engg., Indian Institute of Technology Guwahati

Weightage=42 Marks, Partial demo on 28th March, Final demo date 18th April 2014

These options are allowed

- You can opt for not submitting this assignment but you will be graded out of 58. So maximum grade will be CC. Reduction of weightage of this assignment is impossible.
- Also you are allowed to show your demo till 11PM of 11 May 2014. *For that you and your group members need to coordinate properly.*
- All the group members are necessary to be present at the time of demo. Absent member will not get any credit.
- If some members of a group want, they can improve the assignment till 11th of may, they will get extra improved credit (Absent member will not get any credit). You are allowed to show your assignment multiple times. Change of group is not allowed.

Design and Implementation of Compute Accelerator using FPGA: As many applications/problems have best execution time complexity of order N^4 or N^3 oreven N^2 but this execution time may not acceptable for some case. If the application/problem is highly/massively parallel, some time FPGA provides an efficient way to reduce the execution time without minimizing the time complexity. FPGA based systems are one or two order faster as compared to CPU, means speed up upto 10 or 100 can be achieved. Also FPGA based system provide low cost solution and consume low power as compared to CPU or GPU based solution.

Assignment for Groups

- **Group 1, 2, 3 and 4 : Massively Parallel Multiple Pattern String Matching**
- **Group 5, 6, 7 and 8 : Integer (or Floating Point) Matrix Multiplication**
- **Group 9, 10, 11 and 12 : Map Reduced Accelerator**
- **Group 13, 14, 15 and 16 : Image Filtration**
- **Group 17, 18, 19 and 20 : K-Centriod Clustering Algorithms**

Massively Parallel Multiple Pattern String Matching

Statement

Given a text T of size n >>10^5 and m (m>500) patterns of p (p<8) size. We want to find occurrence of all the patterns in text and position in the text. A pattern may occurs multiple times, we need to report all the occurrence of the pattern. This kind of work is extremely useful in Intrusion detection (Network Intrusion Detection System NIDS), Email Spam Filtering, and etc.

C++ type Pseudo Code

```
for(i=0;i<n;i++){ // Text is T[0 to n-1]

   for(j=0;j<m;j++){  // Inner loop can be done in parallel
     //pattern P[j] matched at i of Text T
     if(Pattern_match_at(T, P[j], i))
          report_print(j,i); //pattern j occurred at position i
      }
}
//Example c++ code @
http://jatinga.iitg.ernet.in/~asahu/cs223/Assgn4/
```

Assume Text is big string (word) contain less number of spaces. All the patterns are not of same length. If number of pattern increase, the time to execute in PC take huge amount of time. As this exhibit parallelism and string matching can be done in hardware very efficiently FPGA accelerator is suitable for this kind of work.

As number and type of pattern are fixed (not changing at run time), you can think of creating efficient data structure (tries or any other) to do this efficiently in hardware.

Hint

Send m patterns to FPGA and store in block ram/Register. Send a very big Text , on character at the time to FPGA through USB. At the time of receiving character in to FPGA, it will do parallel string matching and output the report if it matches.

Design FSM and Data path for your assignment, try to utilized resources efficiently and also take advance of all the resources (CLB, Regs, BRAM, I/O, etc). Incorporated as much big m as possible. DSP slices of FPGA may not be useful for this.

Improvement

Also if you wish, you can extension this to do approximate matching with one or two errors.

Massively Parallel Integer (or Floating Point) Matrix Multiplication

Statement

Matrix Multiplication is well known operation on matrix in linear algebra and this have lots of use in machine learning, image analysis, data mining and many other field. We can do matrix multiplication in $O(n^{2.36})$ using Strassen's method, but the naive matrix multiplication take order N^3 time but it have high parallelism. In FPGA, we can take benefits of this parallelism to speed up the matrix multiplication. Matrix multiplication uses dot product heavily, so dot product operation on two vectors need to done efficiently. As in do product, we use this kind S=S+X[i]*Y[i] of operation, which can be easily pipelined and DSP(multiply and accurate unit) Slice of FPGA can be efficiently used.

C++ type Pseudocode

```
CreateTransposeOfB(); // It convert column access of B to row access
//Do matrix multiplication
  for(i=0;i<N;i++){      // Take a row of A [i]
      for(j=0;j<N;j++){
       S=0;   // Take row of B-Transpose
        for(k=0;k<N;k++) // A[i] is same in this loop
         // Dot Product of Vector A[i] and B[j]
           S=S+A[i][k]*B[j][k]; // Both A and B have row access as
           C[i][j]=S;
       }
   }
     //Example c++ code @
     http://jatinga.iitg.ernet.in/~asahu/cs223/Assgn4/
```

Hint

```
Send row A[i] FPGA,
for (j=0;j<N;j++){
      Send Row B[j] of TranposeB
      compute S=S+A[i][k]*B[i][k] efficiently
}
```

If you synchronize properly at the time of sending row B[j+1], you can perform operation on B[j]. Computation and Communication can be in parallel.

Use FPGA block memory, register and DSP slice efficiently to take benefit of FPGA. Try to multiply bigger size matrix as much as possible

Spartan 6 LX45 FPFA have 56 number of DSP48 slice which are capable to do 18 bit integer operations (add, sub, mul, mac..) and also it can do in pipeline fashion. Use `Xilinx LogiCORE™ DSP48 Macro to infere the dsp log or use 18 bit addition to infer.`

FPGA implementation of Map Reduced Accelerator

Statement

Given M text file of size n $>>10^5$ words, we want to count number of occurrence of each distinct word of the all the files. Counting occurrence of all the word in a file is Map function and summing of occurrence of words in all the file is reduce function.

Basic example of Map reduce is $S=\Sigma (x_i^2)$, where map function find out square of each x_i and reduce sum all the square. Map reduce is also highly parallel and extremely useful in data mining and analysis. Industries giant Google, Microsoft, Yahoo and Oracle, etc uses Map reduce framework (Hadoop) to mine web data and analyze for usage.

C++ like Pseudo-code

```
Read a word  till end of FILE {
      map<string,int> WordCnt;
            it = WordCnt.find(word);
            if(it==WordCnt.end()){
                        WorldCnt[word]=1;
            } else {
                        WordCnt[word]++;
            }
 //Example c++ code @
 http://jatinga.iitg.ernet.in/~asahu/cs223/Assgn4/
```

Hint

It is better to implement reading word from file in C++, but the software implementation of map word count is not efficient. We can implement/design an hardware for word count map using content addressable memory (CAM). As the CAM size will be smaller, we need to choose the word that need to be put in CAM to speed up the search operation is an critical issue. Suppose there is a 1000 words CAM is there, we can match all 1000 word in hardware very efficiently. But we need to thing about other words.

Can we be able to design a CAM to store most of the most probable occurred word in the CAM. So that effectively we can improve the performance of searching and mapping.

Further Reading

Image High Pass or Low Pass Filtration on FPGA

Statement

Given a image file of size mxn pixels, we want to do lowpass filtering of the input image to produce output image. Low pass filtering is process of averaging all the pixel points surrounding to the self. Low pass filtering of image remove random noise, periodic noise and reveal any background pattern. Also it improve smoothness of the whole image.

C++ like Pseudo-code

```
    Read_BMP_Header("test.bmp",&h,&w,bmp);
    Read_BMP_Data("test.bmp",&h,&w,bmp,R,G,B);
    for(i=1;i<h-1;i++) {
        for(j=1;j<w-1;j++) {
               R1[i][j] = (R[i-1][j-1]+R[i-1][j]+R[i-1][j+1]+
                                R[i][j-1]+R[i][j]+R[i][j+1]+
                             R[i+1][j-1]+R[i+1][j]+R[i+1][j+1])/9;
               // Same code for G and B color space
                }
        }
    write_BMP_Header("lowpass.bmp",&h,&w,bmp);
    write_BMP_Data("lowpass.bmp",&h,&w,bmp,R1,G1,B1);
//Example c++ code @
http://jatinga.iitg.ernet.in/~asahu/cs223/Assgn4/
```

Hint

You need to read the image, send data row by row. Compute the filtered image and get back the data. Construct the image 1. You may need to buffer 3 rows to calculate data for an row. We need to store all the three rows in FPGA to calculate filtering operation for an row of image.

As we can see from the code, for every pixel we require nine addition and a division operation. A division by 9 is costly, so we can do
X/9= X/8-X/72 = X/8-X/64+X/576 and may be approximated as X/8-X/64+X/512....
This approximation can be done in 3 shift and three add/sub operations.

It will better, if you do all in Integer domain. You may not require (or avoid) use of floating point operation.

Spartan 6 LX45 FPFA have 56 number of DSP48 slice which are capable to do 18 bit integer operations (add, sub, mul, mac..) and also it can do in pipeline fashion. Use Xilinx LogiCORE™ *DSP48* Macro to infere the dsp log or use 18 bit addition to infer.

Further Reading

http://www.cyanogen.com/help/maximdl/Low-Pass_Filtering.htm

K-Centriod Clustering acceleration using FPGA

Statement

Given a set a N m-dimenational data point, we want to cluster all the data points in to K cluster.

k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. A variation of k-mean algorithm is K-centroid algorithm, where cluster center get updated/calculated in each iteration of clustering.

C++ like Pseudo-code

Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function. The objective function

$$\mathbf{MD= Sum\_j=0\ to\ k\ \ Sum\_i=0\ to\ n\ \ \| X_i^j - C^j\|^2}$$

where $\| X_i^j - C^j\|^2$ s a chosen distance measure between a data point an X_i^j and the cluster centre C^j, is an indicator of the distance of the *n* data points from their respective cluster centres.

The algorithm is composed of the following steps:

1. *Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.*
2. *Assign each object to the group that has the closest centroid.*
3. *When all objects have been assigned, recalculate the positions of the K centroids.*
4. *Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.*

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centres. The k-means algorithm can be run multiple times to reduce this effect

```
//Example c++ code @
http://jatinga.iitg.ernet.in/~asahu/cs223/Assgn4/
```

Hint :

k-centriod algorithm is very parallel and it can be accelerated on FPGA. Distance function repeats for millions of times. IF we can implement as many distance functional unit in FPGA, we can efficiently speed up the algorithms. Before calculation of Dist funtion, we need to send input data for Dist function to FPGA. So array dimensional coordinate need to sent to FPGA. Also it requires a square root function to be implemented but this may be ignored as result will be same..

***Xilinx® Floating-Point* Operator *core* (LOGIcore IP) provides designers with the means to perform *floating-point* arithmetic on an FPGA. You may be able to put around n < 10 floating point IP in one spartna6 FPGA.**

Further Reading

http://stanford.edu/~cpiech/cs221/handouts/kmeans.html
http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html