FPGA Benchmark

Mohammad Hosseinabady
mohammad@hosseinabady.com

September 10, 2016

Abstract

The current version of this report is prepared to be used as a reference for the paper submitted to the IEEE Transaction on Computer. Therefore, it contains brief descriptions of tasks. The future versions explains how to implement these task in FPGA wit details.

Chapter 1

Tasks Descriptions

1.1 Matrix Mean (mm)

This task receives an $n \times m$ matrix, A, and generate a $1 \times m$ vector each of in which the i^{th} element is the average of data in the i^{th} column of matrix A.

$$mean(j) = \frac{\sum_{i=0}^{i<n} A(i,j)}{n} \tag{1.1}$$

1.2 Black Scholes (bs)[1]

The Black-Scholes model provides a partial differential equation (PDE) for the evolution of an option price under certain assumptions. For European options, a closed-form solution exists for this PDE.

$$V_{call} = S.CND(d_1) - X.e^{-rT}.CND(d_2) \tag{1.2}$$

$$V_{put} = X.e^{-rT}.CND(-d_2) - S.CND(-d_1) \tag{1.3}$$

$$d_1 = \frac{log(\frac{S}{X}) + (r + \frac{v^2}{2})T}{\nu\sqrt{T}} \tag{1.4}$$

$$d_2 = \frac{log(\frac{S}{X}) + (r - \frac{v^2}{2})T}{\nu\sqrt{T}} \tag{1.5}$$

$$CND(-d) = 1 - CND(d) \tag{1.6}$$

where V_{call} is the price for an option call, V_{put} is the price for an option put, $CND(d)$ is the Cumulative Normal Distribution function, S is the current option price, X is the strike price, T is the time to expiration. r is the continuously compounded risk free interest rate, ν is the implied volatility for the underlying stock,

1.3 Linear Regression (lr)

Linear regression algorithm tries to fit the following equation to a set of data.

$$y = a_0 + a_1 x \tag{1.7}$$

and the coefficients are as follows

$$a0 = \frac{(\sum_{i=1}^{N} y_i)(\sum_{i=1}^{N} x_i^2) - (\sum_{i=1}^{N} x_i)(\sum_{i=1}^{N} x_i y_i)}{N(\sum_{i=1}^{N} x_i^2) - ((\sum_{i=1}^{N} x_i))^2} \tag{1.8}$$

$$a1 = \frac{N(\sum_{i=1}^{N} x_i y_i) - (\sum_{i=1}^{N} x_i)(\sum_{i=1}^{N} y_i)}{N(\sum_{i=1}^{N} x_i^2) - ((\sum_{i=1}^{N} x_i))^2} \tag{1.9}$$

1.4 Parabolic Regression (pr)

Parabolic regression algorithm tries to fit the following equation to a set of data.

$$y = a_0 + a_1 x + a_2 x^2 \tag{1.10}$$

where

$$\sum_{i=1}^{N} y_i = a_0 N + a_1 \sum_{i=1}^{N} x_i + a_2 \sum_{i=1}^{x_i^2} \tag{1.11}$$

$$\sum_{i=1}^{N} x_i y_i = a_0 \sum_{i=1}^{N} x_i + a_1 \sum_{i=1}^{N} x_i^2 + a_2 \sum_{i=1}^{x_i^3} \tag{1.12}$$

$$\sum_{i=1}^{N} x_i^2 y_i = a_0 \sum_{i=1}^{N} x_i^2 + a_1 \sum_{i=1}^{N} x_i^3 + a_2 \sum_{i=1}^{x_i^4} \tag{1.13}$$

1.5 Matrix-Matrix Multiplication (mm)

This algorithm multiplies two matrix A and B and generates the matrix C as follows

$$c_{i,j} = \sum_{k=1}^{max} a_{i,k} b_{k,j} \tag{1.14}$$

1.6 Matrix-Vector Multiplication (mxv)

This algorithm multiplies a matrix A and to a vector x and generate the vector y as follows

$$y_i = \sum_{k=1}^{max} a_{i,k} b_k \tag{1.15}$$

1.7 Matrix-Vector Multiplication (saxp)

This algorithm receives a vectors A and generates the vector B as follows

$$b_i = \alpha.a_i + \beta \tag{1.16}$$

where α and β are constants.

1.8 histogram (hist)

Histogram represents the distribution of numerical data. A pseudo-code of this task that computes the histogram of pixels in an image is as follows.

```
for (int i = 0; i < DATA_LENGTH; i++) {
  u32 index = (u32)data[i];
  hist[index]++;
}
```

1.9 Covariance (cov)[2]

Computes the covariance, a measure from statistics that show how linearly related two variables are.

It receives a $N \times M$ matrix of data and generates an $M \times M$ matrix thart determines the covariance between each two columns of data using the following equations:

$$cov(i,j) = \frac{\sum_{k=0}^{N-1} (data(k,j) - mean(i))(data(k,j) - mean(j))}{N-1} \tag{1.17}$$

in which

$$mean(x) = \frac{\sum_{k=0}^{N-1} data(k,x)}{N} \tag{1.18}$$

1.10 Correlation (corr)[2]

It receives a $N \times M$ matrix of data and generates an $M \times M$ matrix thart determines the correlation between each two columns of data using the following equations:

$$corr(i,j) = \frac{\sum_{k=0}^{N-1} cov(i,j)}{stddev(i)stddev(j)} \tag{1.19}$$

in which

$$stddev(x) = \sqrt{\frac{\sum_{k=0}^{N-1} (data(k,x) - mean(x))^2}{N}} \tag{1.20}$$

1.11 nbody (nbody)

The *nbody* tasks calculate the gravitational impact of N particles on each other. The following code snippet represents the structure of this task

```
void nbody(const float *initialPositions,
                   const float *initialVelocities,
                   float *finalPositions)
{
  size_t dataSize      = NUM_BODIES*4*sizeof(float);
  float *positionsIn   = (float *)malloc(dataSize);
  float *positionsOut  = (float *)malloc(dataSize);
  float *velocities    = (float *)malloc(dataSize);

  memcpy(positionsIn, initialPositions, dataSize);
  memcpy(velocities, initialVelocities, dataSize);

  for (int itr = 0; itr < ITERATIONS; itr++) {
    for (int i = 0; i < NUM_BODIES; i++) {
      float ix = positionsIn[i*4 + 0];
      float iy = positionsIn[i*4 + 1];
      float iz = positionsIn[i*4 + 2];
      float iw = positionsIn[i*4 + 3];

      float fx = 0.f;
      float fy = 0.f;
      float fz = 0.f;

      for (int j = 0; j < NUM_BODIES; j++) {
        float jx    = positionsIn[j*4 + 0];
        float jy    = positionsIn[j*4 + 1];
```

```
        float jz    = positionsIn[j*4 + 2];
        float jw    = positionsIn[j*4 + 3];

// Compute distance between bodies
        float dx    = (jx-ix);
        float dy    = (jy-iy);
        float dz    = (jz-iz);
        float dist  = sqrt(dx*dx + dy*dy + dz*dz
                           + SOFTENING*SOFTENING);

// Compute interaction force
        float coeff = jw / (dist*dist*dist);
        fx         += coeff * dx;
        fy         += coeff * dy;
        fz         += coeff * dz;
      }

// Update velocity
      float vx = velocities[i*4 + 0] + fx * DELTA;
      float vy = velocities[i*4 + 1] + fy * DELTA;
      float vz = velocities[i*4 + 2] + fz * DELTA;
      velocities[i*4 + 0] = vx;
      velocities[i*4 + 1] = vy;
      velocities[i*4 + 2] = vz;

// Update position
      positionsOut[i*4 + 0] = ix + vx * DELTA;
      positionsOut[i*4 + 1] = iy + vy * DELTA;
      positionsOut[i*4 + 2] = iz + vz * DELTA;
      positionsOut[i*4 + 3] = iw;
  }

  // Swap buffers
  float *temp  = positionsIn;
  positionsIn  = positionsOut;
  positionsOut = temp;
}
```

1.12 Sharpen (sharpen)

This task convolves an input image with the mask f as follows:

$$f = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{1.21}$$

the implementation assumed that the mask f is not separable.

1.13 Sobel filter (sobel)

Sobel filter is one of well-known techniques for the image edge detection. It convolves two masks with an image, denoted by **A**, as follows:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \tag{1.22}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A \tag{1.23}$$

$$|G| = |G_x| + |G_y| \tag{1.24}$$

1.14 Vector add (Vadd)

This algorithm adds two vectors of data and generates the third vector where

$$c_i = a_i + b_i \tag{1.25}$$

Bibliography

[1] V. Podlozhnyuk, "Black-scholes option pricing," nvidia, Tech. Rep., 2012. [Online]. Available: http://docs.nvidia.com/cuda/samples/4_Finance/BlackScholes/doc/BlackScholes.pdf

[2] L.-N. P. Tomofumi Yuki, *PolyBench 4.1*, Inria / LIP / ENS Lyon and Ohio State University. [Online]. Available: http://web.cse.ohio-state.edu/ pouchet/software/polybench/