PCA: EXP-1 SUM ARRAY GPU

NAME: HARSHAVARDHAN

REG NO: 212222240114

EX. NO: 1

DATE: 27.02.24

SUM ARRAY ON HOST AND DEVICE

PCA-GPU-based-vector-summation.-Explore-the-differences. i) Using the program sumArraysOnGPU-timer.cu, set the block.x = 1023. Recompile and run it. Compare the result with the execution configuration of block.x = 1024. Try to explain the difference and the reason.
ii) Refer to sumArraysOnGPU-timer.cu, and let block.x = 256. Make a new kernel to let each thread handle two elements. Compare the results with other execution confi gurations.

AIM:

To perform vector addition on host and device.

EQUIPMENTS REQUIRED:

Hardware – PCs with NVIDIA GPU & CUDA NVCC Google Colab with NVCC Compiler

PROCEDURE:

1. Initialize the device and set the device properties.
2. Allocate memory on the host for input and output arrays.
3. Initialize input arrays with random values on the host.
4. Allocate memory on the device for input and output arrays, and copy input data from host to device.
5. Launch a CUDA kernel to perform vector addition on the device.
6. Copy output data from the device to the host and verify the results against the host's sequential vector addition. Free memory on the host and the device.

PROGRAM:

```
#include <cuda_runtime.h>
#include <stdio.h>

void checkResult(float *hostRef, float *gpuRef, const int N)
{
    double epsilon = 1.0E-8;
    bool match = 1;

    for (int i = 0; i < N; i++)
    {
        if (abs(hostRef[i] - gpuRef[i]) > epsilon)
        {
            match = 0;
            printf("Arrays do not match!\n");
            printf("host %5.2f gpu %5.2f at current %d\n", hostRef[i],
                   gpuRef[i], i);
            break;
        }
    }

    if (match) printf("Arrays match.\n\n");

    return;
}

void initialData(float *ip, int size)
{
    // generate different seed for random number
    time_t t;
    srand((unsigned) time(&t));

    for (int i = 0; i < size; i++)
    {
        ip[i] = (float)( rand() & 0xFF ) / 10.0f;
    }

    return;
}

void sumArraysOnHost(float *A, float *B, float *C, const int N)
{
    for (int idx = 0; idx < N; idx++)
    {
        C[idx] = A[idx] + B[idx];
    }
}


__global__ void sumArraysOnGPU(float *A, float *B, float *C, const int N){
    int i = blockIdx.x*blockDim.x+threadIdx.x;
    if (i<N) C[i] = A[i] + B[i];
}
```

```
int main(int argc, char **argv)
{
    printf("%s Starting...\n", argv[0]);

    // set up device
    int dev = 0;
    cudaDeviceProp deviceProp;
    CHECK(cudaGetDeviceProperties(&deviceProp, dev));
    printf("Using Device %d: %s\n", dev, deviceProp.name);
    CHECK(cudaSetDevice(dev));

    // set up data size of vectors
    int nElem = 1 << 24;
    printf("Vector size %d\n", nElem);

    // malloc host memory
    size_t nBytes = nElem * sizeof(float);

    float *h_A, *h_B, *hostRef, *gpuRef;
    h_A     = (float *)malloc(nBytes);
    h_B     = (float *)malloc(nBytes);
    hostRef = (float *)malloc(nBytes);
    gpuRef  = (float *)malloc(nBytes);

    double iStart, iElaps;

    // initialize data at host side
    iStart = seconds();
    initialData(h_A, nElem);
    initialData(h_B, nElem);
    iElaps = seconds() - iStart;
    printf("initialData Time elapsed %f sec\n", iElaps);
    memset(hostRef, 0, nBytes);
    memset(gpuRef,  0, nBytes);

    // add vector at host side for result checks
    iStart = seconds();
    sumArraysOnHost(h_A, h_B, hostRef, nElem);
    iElaps = seconds() - iStart;
    printf("sumArraysOnHost Time elapsed %f sec\n", iElaps);

    // malloc device global memory
    float *d_A, *d_B, *d_C;
    CHECK(cudaMalloc((float**)&d_A, nBytes));
    CHECK(cudaMalloc((float**)&d_B, nBytes));
    CHECK(cudaMalloc((float**)&d_C, nBytes));

    // transfer data from host to device
    CHECK(cudaMemcpy(d_A, h_A, nBytes, cudaMemcpyHostToDevice));
```

```
    CHECK(cudaMemcpy(d_B, h_B, nBytes, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_C, gpuRef, nBytes, cudaMemcpyHostToDevice));

    // invoke kernel at host side
    int iLen = 512;
    dim3 block (iLen);
    dim3 grid  ((nElem + block.x - 1) / block.x);

    iStart = seconds();
    sumArraysOnGPU<<<grid, block>>>(d_A, d_B, d_C, nElem);
    CHECK(cudaDeviceSynchronize());
    iElaps = seconds() - iStart;
    printf("sumArraysOnGPU <<<  %d, %d  >>>  Time elapsed %f sec\n", grid.x,
           block.x, iElaps);

    // check kernel error
    CHECK(cudaGetLastError()) ;

    // copy kernel result back to host side
    CHECK(cudaMemcpy(gpuRef, d_C, nBytes, cudaMemcpyDeviceToHost));

    // check device results
    checkResult(hostRef, gpuRef, nElem);

    // free device global memory
    CHECK(cudaFree(d_A));
    CHECK(cudaFree(d_B));
    CHECK(cudaFree(d_C));

    // free host memory
    free(h_A);
    free(h_B);
    free(hostRef);
    free(gpuRef);

    return(0);
}
```

OUTPUT:

```
/tmp/tmpilypbozq/2a5cf74b-f791-49f1-98d2-8ded16f1835e/cuda_exec.out Starting...
Using Device 0: Tesla T4
Vector size 16777216
initialData Time elapsed 0.705662 sec
sumArraysOnHost Time elapsed 0.053160 sec
sumArraysOnGPU <<<  32768, 512  >>>  Time elapsed 0.113721 sec
Arrays match.
```

BLOCK SIZE: 1203

```
/tmp/tmpz86g2f2e/f2a27769-1809-4303-9127-e7d6c90266f0/cuda_exec.out Starting...
Using Device 0: Tesla T4
Vector size 16777216
initialData Time elapsed 0.719886 sec
sumArraysOnHost Time elapsed 0.055728 sec
sumArraysOnGPU <<<  16401, 1023  >>>  Time elapsed 0.001140 sec
Arrays match.
```

BLOCK SIZE: 256

```
/tmp/tmpz86g2f2e/b6f60ed7-cad7-4669-8d9b-258dabcaad00/cuda_exec.out Starting...
Using Device 0: Tesla T4
Vector size 16777216
initialData Time elapsed 0.733954 sec
sumArraysOnHost Time elapsed 0.058169 sec
sumArraysOnGPU <<<  65536, 256  >>>  Time elapsed 0.001043 sec
Arrays match.
```

RESULT:

Thus, Implementation of sum arrays on host and device is done in nvcc cuda using random number.