Name : Saurabh Nitnaware
Experiment No : 8
Implement Two Vector addition using OpenCL/CUDA/ Parallel MatLab.
Date of Performance : 04/04/24
Date of Submission : 19/04/24

Aim: The aim of this task is to implement two vector addition using OpenCL/CUDA/Parallel Matlab.

Objective: The objective of this task is to implement two vector addition using parallel programming frameworks such as OpenCL, CUDA, or Parallel Matlab. Vector addition is a simple operation that can be performed in parallel, making it a good candidate for parallel computing using GPUs. By implementing two vector addition using these frameworks, we can demonstrate the benefits of parallel programming and compare the performance of different frameworks.

Theory:

Vector addition is a basic mathematical operation that adds two vectors element-wise to produce a third vector. It is a simple operation that can be easily parallelized across multiple processing elements, making it an ideal candidate for parallel programming**.**

OpenCL (Open Computing Language) and CUDA (Compute Unified Device Architecture) are two popular parallel programming frameworks used to accelerate compute-intensive applications on GPUs. These frameworks allow the programmer to write code that can be executed in parallel across multiple threads or processing elements.

In OpenCL, the programmer writes a kernel function that is executed on the GPU, and the input data is transferred from the host (CPU) to the device (GPU) memory. The kernel function is executed in parallel across multiple work-items, which are mapped to processing elements on the GPU. The output data is then transferred back to the host memory.

Similarly, in CUDA, the programmer writes a kernel function that is executed on the GPU, and the input data is transferred to the device memory. The kernel function is executed in parallel across multiple threads, which are mapped to processing elements on the GPU. The output data is then transferred back to the host memory.

Parallel Matlab is another framework that supports parallel programming using multiple cores or GPUs. The programmer can use the parfor loop construct to parallelize the computation of the vector addition operation. The parfor loop automatically distributes the computation across multiple cores or GPUs, and the output data is combined at the end of the loop.

Overall, implementing two vector addition using OpenCL, CUDA, or Parallel Matlab can significantly improve the performance of the computation, especially for large vectors. Parallel programming frameworks allow the programmer to take advantage of the parallel processing capabilities of GPUs to accelerate compute-intensive applications.

Code:

```
#include <stdio.h>
__global__ void vectorAdd(int *a, int *b, int *c, int n) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
      if (tid < n) {
       c[tid] = a[tid] + b[tid];
    }
}
int main() {
    int n = 100; // Size of the vectors
    int *h_a, *h_b, *h_c; // Host vectors
    int *d_a, *d_b, *d_c; // Device vectors
    h_a = (int*)malloc(n * sizeof(int));
    h_b = (int*)malloc(n * sizeof(int));
    h_c = (int*)malloc(n * sizeof(int));
    cudaMalloc((void**)&d_a, n * sizeof(int));
    cudaMalloc((void**)&d_b, n * sizeof(int));
```

```
    cudaMalloc((void**)&d_c, n * sizeof(int));
    for (int i = 0; i < n; ++i) {
        h_a[i] = i;
        h_b[i] = i * 2;
    }
    cudaMemcpy(d_a, h_a, n * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, n * sizeof(int), cudaMemcpyHostToDevice);
    dim3 grid(ceil((float)n / 256), 1, 1);
    dim3 block(256, 1, 1);
    vectorAdd<<<grid, block>>>(d_a, d_b, d_c, n);
    cudaMemcpy(h_c, d_c, n * sizeof(int), cudaMemcpyDeviceToHost);
    printf("Result: ");
    for (int i = 0; i < n; ++i) {
        printf("%d ", h_c[i]);
    }
    printf("\n");
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    free(h_a);
    free(h_b);
    free(h_c);
    return 0;
}
```

Output:

Result: 0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99 102 105 108 111 114 117 120 123 126 129 132 135 138 141 144 147 150 153 156 159 162 165 168 171 174 177 180 183 186 189 192 195 198 201 204 207 210 213 216 219 222 225 228 231 234 237 240 243 246 249 252 255 258 261 264 267 270 273 276 279 282 285 288 291 294 297

Conclusion: In conclusion, implementing two vector addition using parallel programming frameworks such as OpenCL, CUDA, or Parallel Matlab provides a significant performance improvement over traditional sequential programming. These frameworks allow the programmer to take advantage of the parallel processing capabilities of GPUs to accelerate compute-intensive applications.