CS 301
High-Performance Computing

Lab 2 - Performance evaluation of codes

Problem B-2: Multiplication of Two vectors followed by summation.

Jay Dobariya (202101521)
Akshar Panchani (202101522)

January 31, 2024

Contents

1 Introduction

Here in this code performance we would do the multiplication of the two vectors and find out the complexity and its detailed perspective in terms of high performance computing. The mathematical terms and its case study is given further in the report.

2 Hardware Details

2.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 6
- On-line CPU(s) list: 0-5
- Thread(s) per core: 1
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 155
- Model name: Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz
- Stepping: 10
- CPU MHz: 799.992
- CPU max MHz: 4100.0000
- CPU min MHz: 800.0000
- BogoMIPS: 6000.00
- Virtualization: VT-x
- L1d cache: 192KB
- L1i cache: 192KB
- L2 cache: 1.5MB
- L3 cache: 9MB
- NUMA node0 CPU(s): 0-5

2.2 Hardware Details for HPC Cluster (Node gics1)

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 24
- On-line CPU(s) list: 0-23
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 2
- NUMA node(s): 2
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 63
- Model name: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
- Stepping: 2
- CPU MHz: 2642.4378
- BogoMIPS: 4804.69
- Virtualization: VT-x
- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 15360K
- NUMA node0 CPU(s): 0-5,12-17
- NUMA node1 CPU(s): 6-11,18-23

3 Problem B2

3.1 Brief description with its algorithm

In this question, we add and multiply two vectors vector-wise, then total up the results. $(A(i) \cdot B(i)) + (A(i) + B(i))$ is the definition of the operation, where A and B are input vectors of size N. In the serial version, addition and multiplication are carried out element-by-element while a single master thread iteratively covers the whole issue size.

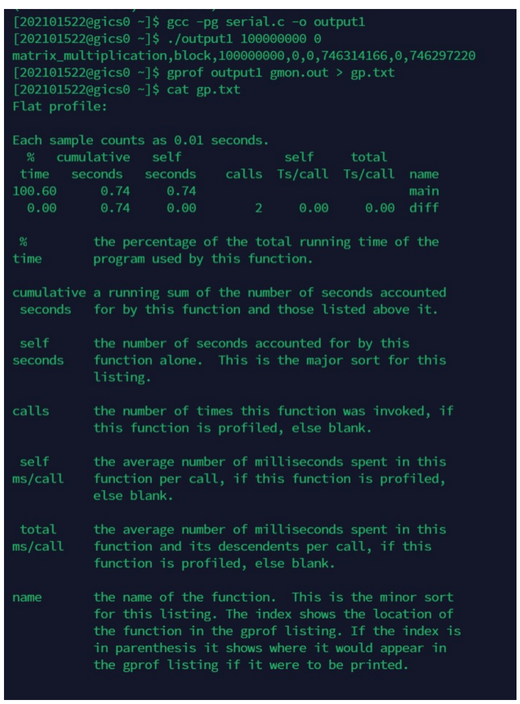
1. Initialize variables: $result = 0$.
2. For i from 1 to N (size of vectors):
   (a) Perform element-wise addition and multiplication: $temp = A(i) \cdot B(i) + A(i) + B(i)$.
   (b) Add $temp$ to $result$.
3. The final result is the summation of $(A(i) \cdot B(i)) + (A(i) + B(i))$ for all i.

3.2 The complexity of the algorithm (serial)

The time complexity of the described algorithm for vector-wise addition and multiplication followed by summation in a serial implementation is $O(N)$, where N is the size of the input vectors.

3.3 Profiling using HPC Cluster (with gprof)

The screenshots of profiling using the HPC Cluster are given below



```
[202101522@gics0 ~]$ gcc -pg serial.c -o output1
[202101522@gics0 ~]$ ./output1 100000000 0
matrix_multiplication,block,100000000,0,0,746314166,0,746297220
[202101522@gics0 ~]$ gprof output1 gmon.out > gp.txt
[202101522@gics0 ~]$ cat gp.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.60      0.74     0.74                             main
  0.00      0.74     0.00        2     0.00     0.00  diff

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
           else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
           function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
           the function in the gprof listing. If the index is
           in parenthesis it shows where it would appear in
           the gprof listing if it were to be printed.
```

Figure 1: Screenshot of terminal from HPC Cluster

3.4 Graph of Problem Size vs Runtime

3.4.1 Graph of Problem Size vs Runtime for LAB207 PCs

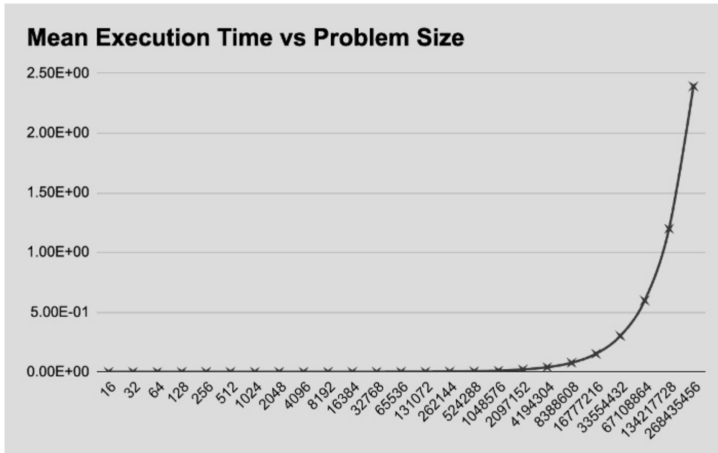


Figure 2: Total mean execution time (Algorithm time) vs Problem size plot for **problem size 10^8 (Hardware: LAB207 PC, Problem: MUL_VECTORS)**.

3.4.2 Graph of Problem Size vs Runtime for HPC Cluster

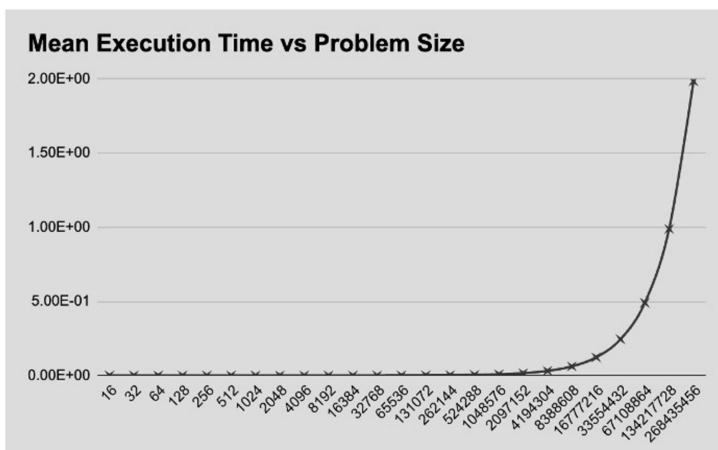


Figure 3: Total mean execution time (Algorithm time) vs Problem size plot for **problem size 10^8.(Hardware: HPC Cluster, Problem: MUL_VECTORS)**.

4 Conclusions

- Here we carried out the complex multiplication on both the lab PC and the cluster and we differentiated the result through the above resulted graphs. This shows the flavors of the computing with its time and space complexity that how a matrix multiplication and its addition is carried out.

- The cluster machine executes the given equation faster than the LAB207 machine for equivalent problem sizes.
- As the problem size increases its running time also increases. This shows its disparity.