CS 301
High-Performance Computing

Lab 3 - Comparison in Matrix Multiplication

Problem 2: Matrix multiplication using transpose

Jay Dobariya (202101521)
Akshar Panchani (202101522)

February 14, 2024

Contents

1 Introduction

We further extend our work from conventional method to some different method .So now we will talk about the second method which is matrix-multiplicaton through transpose.

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

- 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 2

3.1 Brief description of the problem

By taking use of matrix transpose's characteristics, matrix multiplication utilizing transpose optimizes memory access patterns.

3.1.1 Algorithm description

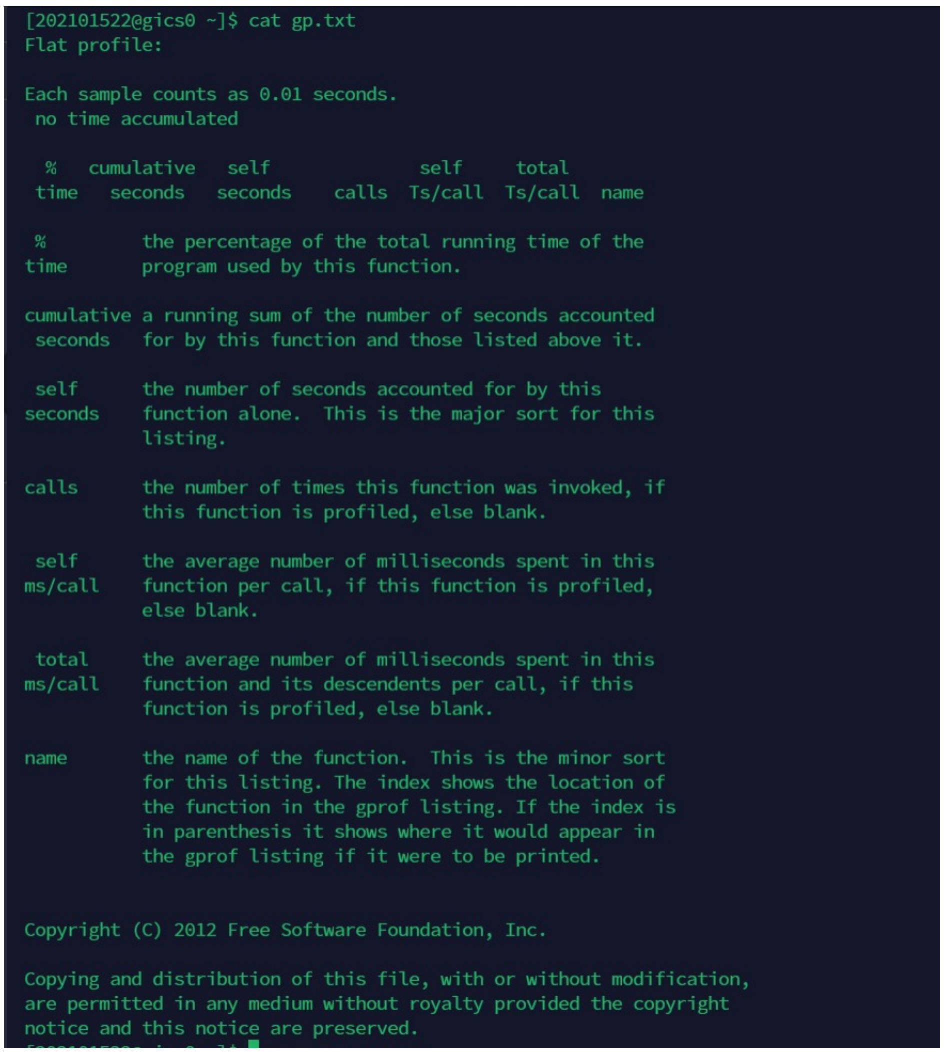
1. Compute the transpose of the second matrix.
2. Perform conventional matrix multiplication between the first matrix and the transposed second matrix.

3.2 The complexity of the algorithm

Matrix multiplication with transpose has the same temporal complexity as standard multiplication which is $O(n^3)$, but it could use cache more effectively and use less memory.

3.3 Profiling using HPC Cluster (with gprof)

The screenshots of profiling using the HPC Cluster are given below



```
[202101522@gics0 ~]$ cat gp.txt
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name

 %         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.


Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
```

Figure 1: Screenshot of the terminal from HPC Cluster

3.4 Graph of Problem Size vs Runtime

3.4.1 Graph of Problem Size vs Algorithm time

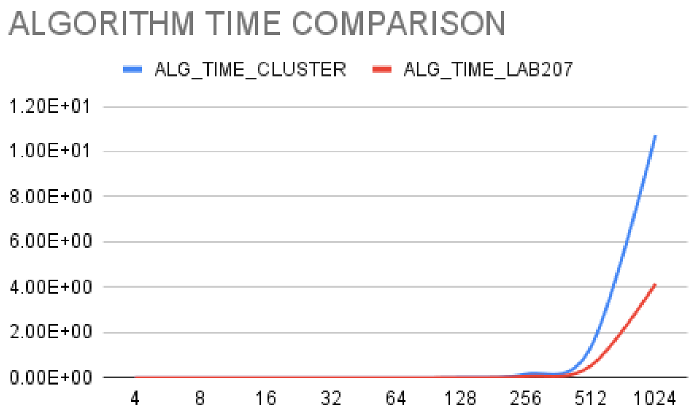


Figure 2: Total mean execution time (Algorithm time) vs Problem size plot for **problem size 2^{10} (Hardware: LAB207 PC and HPC Cluster, Problem: Transpose Matrix Multiplication)**.

3.4.2 Graph of Problem Size vs Total time

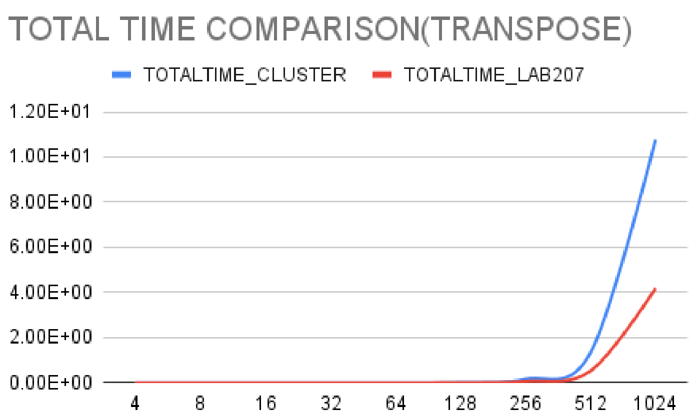


Figure 3: Total mean execution time (Algorithm time) vs Problem size plot for **problem size 2^{10} (Hardware: HPC Cluster and LAB207 PC, Problem: Transpose Matrix Multiplication)**.

4 Conclusions

- Transpose matrix multiplication can result in less memory traffic and better cache performance, particularly for big matrices. Its efficacy is nonetheless limited for some applications since its computational complexity is the same as that of traditional multiplication.