CS 301
High-Performance Computing

Lab 3 - Comparison in Matrix Multiplication

Problem 3: Block matrix multiplication

Jay Dobariya (202101521)
Akshar Panchani (202101522)

February 14, 2024

Contents

1 Introduction

Here in this report we would discuss about the efficient method which is the block matrix multiplication with its algorithm and complexity thorugh its graphically analyses.

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 3

3.1 Brief description of the problem

By splitting matrices into smaller blocks, block matrix multiplication lowers memory traffic and enhances cache efficiency.

3.1.1 Algorithm description

1. Divide the matrices into $k \times k$ blocks.
2. Perform conventional matrix multiplication for each block.
3. Accumulate the results to obtain the final matrix.

3.2 The complexity of the algorithm

Block size k determines the temporal complexity of block matrix multiplication. A suitable selection of k can lower the complexity to $O(n^3/k)$, which might result in performance gains over traditional techniques.

3.3 Profiling using HPC Cluster (with gprof)

The screenshots of profiling using the HPC Cluster are given below

```
Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     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
```

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

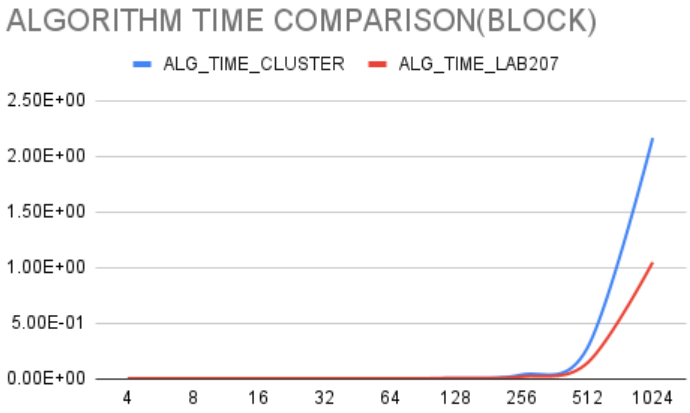

Figure 2: Total mean execution time (Algorithm time) vs Problem size plot for **problem size 2^{10} (Hardware: LAB207 PC and HPC Cluster, Problem: Block matrix multiplication)**.

3.4.2 Graph of Problem Size vs Total time

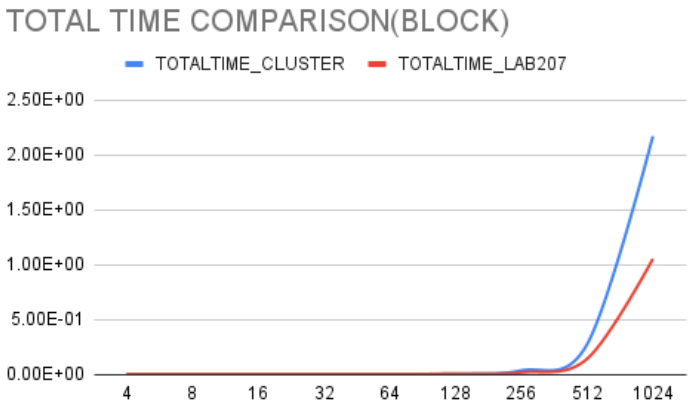

Figure 3: Total mean execution time (Algorithm time) vs Problem size plot for **problem size 2^{10} (Hardware: HPC Cluster and LAB207 PC, Problem: Block matrix multiplication)**.

4 Conclusions

- By taking advantage of data proximity, block matrix multiplication may dramatically lower memory traffic and enhance cache performance. Better scalability and efficiency may be attained by carefully selecting the block size in comparison to traditional approaches, particularly for large-scale calculations.