**CS4532 Concurrent Programming**

**Take Home Lab 3 and 4**

**Team Members:**

130 - Wickramarathna

130318K – R.A.M. Lahiruni

1. Current time was used as the seed in random number generation.
2. openMP library was used for implementing parallel constructs.
3. **Execution of serial and parallel multiplication using openMP**

Multiplication for matrices of different sizes were carried out using an initial sample size of 20.

For each matrix size, both sequential and parallel computations were executed, and the execution times were recorded. Using these execution times, the mean and standard deviation were calculated for each type of execution (serial or parallel).

Using the mean and standard deviation values from above, the number of samples was calculated for each case with an accuracy of ±5% and a confidence level of 95%.

To discover the improvement in performance, the speed up was calculated as below:

|  |  |  |  |
| --- | --- | --- | --- |
| Matrix Size (n) | Execution Type | Mean Execution Time | Speed up gained |
| 200 | Sequential |  |  |
| Parallel |  |
| 400 | Sequential |  |  |
| Parallel |  |
| 600 | Sequential |  |  |
| Parallel |  |
| 800 | Sequential |  |  |
| Parallel |  |
| 1000 | Sequential |  |  |
| Parallel |  |
| 1200 | Sequential |  |  |
| Parallel |  |
| 1400 | Sequential |  |  |
| Parallel |  |
| 1600 | Sequential |  |  |
| Parallel |  |
| 1800 | Sequential |  |  |
| Parallel |  |
| 2000 | Sequential |  |  |
| Parallel |  |

**Sequential**

Time taken to execute in n-200 : 0.0137264

Time taken to execute in n-400 : 0.0846004

Time taken to execute in n-600 : 0.364601

Time taken to execute in n-800 : 1.72761

Time taken to execute in n-1000 : 4.18141

Time taken to execute in n-1200 : 14.987

Time taken to execute in n-1400 : 24.5662

Time taken to execute in n-1600 : 38.2599

Time taken to execute in n-1800 : 58.3122

Time taken to execute in n-2000 : 82.5089

**Parallel multiplication using openMP**

Time taken to execute in n-200 : 0.00864494

Time taken to execute in n-400 : 0.0483061

Time taken to execute in n-600 : 0.28274

Time taken to execute in n-800 : 2.03893

Time taken to execute in n-1000 : 4.59359

Time taken to execute in n-1200 : 8.6835

Time taken to execute in n-1400 : 14.5553

Time taken to execute in n-1600 : 23.0712

Time taken to execute in n-1800 : 36.0491

Time taken to execute in n-2000 : 51.9351

1. **Architecture of the CPU used for execution**

|  |  |
| --- | --- |
| Architecture | x86\_64 |
| No of CPUs | 4 |
| Threads per core | 2 |
| Cores per socket | 2 |
| Sockets | 1 |
| Model name | Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz |
| CPU MHz | 1975.546 |
| L1d cache | 32K |
| L1i cache | 32K |
| L2 cache | 256K |
| L3 cache | 4096K |

Justify the gained speed up knowing the architecture of the CPU. Discuss your observations in detail while relating the observed behavior in graphs to the architecture of the CPU.

1. **Optimization techniques**
2. **Using transpose**
3. **Using Strassen Algorithm**

Strassen algorithm is faster than the standard matrix multiplication, usually for matrices with sizes larger than 1000. This method improves the *O(n3)* of standard multiplication to a *O(nlog27) = O(n2.8074).* Strassen algorithm can be applied only when the matrices are square and have a size of that is a power of two. If the matrices do not meet these constraints, they are padded with rows and columns of zeros. The algorithm partitions a given matrix into four equal sized blocks, and uses them to calculate seven matrices, which are then used to obtain the final resulting matrix. If the partitioned block matrices are still large, the algorithm recursively calculates until the matrix size reaches a defined threshold. Once the threshold is reached, the algorithm falls back to calculating using the standard method.

**Pros**

* Calculates final matrix in seven multiplication steps, whereas the standard method uses eight multiplications. Hence the time complexity is low, and the multiplication is optimized.

**Cons**

* The threshold at which the algorithm falls back to the standard method has a significant effect in optimization, particularly when matrices of varying sizes need to be calculated. Threshold measures that work for large matrices may not work for smaller matrices.
* Since most matrices usually require to be padded with zeros prior to applying the algorithm, the matrix size may become very large, hence increasing the time complexity.

**Optimizing using transpose**

|  |  |  |  |
| --- | --- | --- | --- |
| Matrix Size (n) | Execution Type | Mean Execution Time | Speed up gained |
| 200 | Sequential |  |  |
| Parallel |  |
| 400 | Sequential |  |  |
| Parallel |  |
| 600 | Sequential |  |  |
| Parallel |  |
| 800 | Sequential |  |  |
| Parallel |  |
| 1000 | Sequential |  |  |
| Parallel |  |
| 1200 | Sequential |  |  |
| Parallel |  |
| 1400 | Sequential |  |  |
| Parallel |  |
| 1600 | Sequential |  |  |
| Parallel |  |
| 1800 | Sequential |  |  |
| Parallel |  |
| 2000 | Sequential |  |  |
| Parallel |  |

**Discussion**

Performance gained by parallelizing the optimized version

Justify performance gain by architecture of the CPU utilized

Extra speed up gained = Speedup\_after\_optimization – Speedup\_before\_optimization

**Optimizing using Strassen algorithm**

|  |  |  |  |
| --- | --- | --- | --- |
| Matrix Size (n) | Execution Type | Mean Execution Time | Speed up gained |
| 200 | Sequential |  |  |
| Parallel |  |
| 400 | Sequential |  |  |
| Parallel |  |
| 600 | Sequential |  |  |
| Parallel |  |
| 800 | Sequential |  |  |
| Parallel |  |
| 1000 | Sequential |  |  |
| Parallel |  |
| 1200 | Sequential |  |  |
| Parallel |  |
| 1400 | Sequential |  |  |
| Parallel |  |
| 1600 | Sequential |  |  |
| Parallel |  |
| 1800 | Sequential |  |  |
| Parallel |  |
| 2000 | Sequential |  |  |
| Parallel |  |

**Discussion**

Performance gained by parallelizing the optimized version

Justify performance gain by architecture of the CPU utilized

Extra speed up gained = Speedup\_after\_optimization – Speedup\_before\_optimization