Documentation – PPD Laboratory 1

**Requirements analysis:**

The current project involves handling the convolution operation of a kernel C of size k x k over a large matrix F of N x M cells, resulting in a new matrix V with the same size. To evaluate efficiency, multiple test cases and scenarios were conducted using both sequential and multiple parallel distribution approaches.

The observed techniques are as follows:

* Sequential program:
  + The convolution is applied by a single thread, iterating through all indices of F
* Parallel program:
  + Using multiple threads that are assigned to execute a part of the matrix convolutions each
  + Classical approaches:
    1. Horizontal distribution (each thread is assigned multiple rows)
    2. Vertical distribution (each thread is assigned multiple columns)
    3. Block distribution (each thread is assigned blocks or sub-matrices of F)
    4. Delta function distribution (each thread is assigned indices where to apply convolution based on the Delta function rules)
  + (Almost) each of these approaches can further be split into linear or cyclic variants

Programs are tested in both Java and C++, and in C++ we test both static and dynamic memory allocations of the matrices.

**Input/output data**

The program receives the input file containing N, M, K followed by the matrices F and C, then the number of threads to create (in case it is parallel) and the technique to apply. It outputs both the result matrix and the time taken to compute all the specific operations over the matrix.

**Testing**

The following scenarios are conducted:

1) N=M=10 and n=m=3; p=4;

2) N=M=1000 and n=m=5; p=2,4,8,16

3) N=10 M=10000 and n=m=5; p=2,4,8,16

4) N=10000 M=10 and n=m=5; p=2,4,8,16

5) N=10000 M=10000 and n=m=5; p=2,4,8,16

For each of these scenarios, all the 8 techniques are applied and for all implementations (Java, C++ static and C++ dynamic).

The techniques are numbered as follows:

1. Sequential
2. Horizontal linear
3. Horizontal cyclic
4. Vertical linear
5. Vertical cyclic
6. Block
7. Delta function linear
8. Delta function cyclic

**Experiments table details:**

**Java**

|  |  |  |  |
| --- | --- | --- | --- |
| Threads | Matrix | Technique | Execution Time |
| 2 | data\_10\_10\_3.txt | 0 | 1.25695 |
| 1 | 2.30019 |
| 2 | 2.19991 |
| 3 | 2.3029 |
| 4 | 2.51697 |
| 5 | 1.10505 |
| 6 | 2.20436 |
| 7 | 2.11144 |
| data\_10\_10000\_5.txt | 0 | 13.1078 |
| 1 | 7.02254 |
| 2 | 5.9463 |
| 3 | 6.05364 |
| 4 | 5.97887 |
| 5 | 6.90029 |
| 6 | 6.88398 |
| 7 | 7.09218 |
| data\_1000\_1000\_5.txt | 0 | 50.699301 |
| 1 | 29.943289 |
| 2 | 31.27518 |
| 3 | 31.892801 |
| 4 | 32.507721 |
| 5 | 46.147198 |
| 6 | 32.855148 |
| 7 | 32.807709 |
| data\_10000\_10\_5.txt | 0 | 15.412801 |
| 1 | 6.35099 |
| 2 | 6.71733 |
| 3 | 6.47091 |
| 4 | 6.39627 |
| 5 | 7.75186 |
| 6 | 6.74483 |
| 7 | 6.89471 |
| data\_10000\_10000\_5.txt | 0 | 4168.499219 |
| 1 | 2233.741406 |
| 2 | 2214.15293 |
| 3 | 2193.586328 |
| 4 | 2222.903906 |
| 5 | 4091.316016 |
| 6 | 2535.218945 |
| 7 | 2611.583984 |
| 4 | data\_10\_10\_3.txt | 0 | 1.22494 |
| 1 | 2.31347 |
| 2 | 2.28321 |
| 3 | 2.4868 |
| 4 | 2.32842 |
| 5 | 1.32027 |
| 6 | 2.4623 |
| 7 | 2.37684 |
| data\_10\_10000\_5.txt | 0 | 11.54972 |
| 1 | 6.20414 |
| 2 | 5.17741 |
| 3 | 5.24524 |
| 4 | 5.64928 |
| 5 | 4.83875 |
| 6 | 6.26135 |
| 7 | 5.81856 |
| data\_1000\_1000\_5.txt | 0 | 54.116852 |
| 1 | 25.645721 |
| 2 | 25.13956 |
| 3 | 26.62236 |
| 4 | 26.364819 |
| 5 | 28.153769 |
| 6 | 26.842499 |
| 7 | 28.641299 |
| data\_10000\_10\_5.txt | 0 | 13.88271 |
| 1 | 6.21133 |
| 2 | 5.79311 |
| 3 | 6.44324 |
| 4 | 6.57188 |
| 5 | 5.48362 |
| 6 | 5.8987 |
| 7 | 7.3229 |
| data\_10000\_10000\_5.txt | 0 | 4290.939844 |
| 1 | 1384.739453 |
| 2 | 1463.199512 |
| 3 | 1422.237305 |
| 4 | 1314.851172 |
| 5 | 2074.194336 |
| 6 | 1848.980664 |
| 7 | 1559.363867 |
| 8 | data\_10\_10\_3.txt | 0 | 1.1976 |
| 1 | 3.10892 |
| 2 | 3.28437 |
| 3 | 3.16924 |
| 4 | 2.71305 |
| 5 | 1.30485 |
| 6 | 2.76195 |
| 7 | 2.72773 |
| data\_10\_10000\_5.txt | 0 | 12.38106 |
| 1 | 4.83043 |
| 2 | 6.07278 |
| 3 | 5.14535 |
| 4 | 5.48356 |
| 5 | 4.89967 |
| 6 | 4.73391 |
| 7 | 5.37712 |
| data\_1000\_1000\_5.txt | 0 | 53.222668 |
| 1 | 24.72052 |
| 2 | 23.455 |
| 3 | 25.070441 |
| 4 | 24.747299 |
| 5 | 28.21644 |
| 6 | 24.61916 |
| 7 | 24.991119 |
| data\_10000\_10\_5.txt | 0 | 16.36319 |
| 1 | 4.94197 |
| 2 | 4.98897 |
| 3 | 5.95842 |
| 4 | 5.96683 |
| 5 | 5.46657 |
| 6 | 5.07103 |
| 7 | 5.41761 |
| data\_10000\_10000\_5.txt | 0 | 4397.361328 |
| 1 | 944.440918 |
| 2 | 960.864063 |
| 3 | 971.645117 |
| 4 | 929.316211 |
| 5 | 1263.778223 |
| 6 | 987.922949 |
| 7 | 1127.937988 |
| 16 | data\_10\_10\_3.txt | 0 | 2.416 |
| 1 | 2.91718 |
| 2 | 4.72279 |
| 3 | 2.99813 |
| 4 | 5.14738 |
| 5 | 3.76904 |
| 6 | 5.73647 |
| 7 | 5.12847 |
| data\_10\_10000\_5.txt | 0 | 14.17663 |
| 1 | 4.69645 |
| 2 | 6.13254 |
| 3 | 6.26852 |
| 4 | 6.01807 |
| 5 | 4.63705 |
| 6 | 5.8289 |
| 7 | 8.09437 |
| data\_1000\_1000\_5.txt | 0 | 51.11366 |
| 1 | 25.616629 |
| 2 | 25.859052 |
| 3 | 28.95575 |
| 4 | 28.709698 |
| 5 | 32.61022 |
| 6 | 30.463281 |
| 7 | 30.274332 |
| data\_10000\_10\_5.txt | 0 | 16.06241 |
| 1 | 4.9885 |
| 2 | 4.9143 |
| 3 | 5.90261 |
| 4 | 7.60998 |
| 5 | 5.24562 |
| 6 | 5.1298 |
| 7 | 5.96475 |
| data\_10000\_10000\_5.txt | 0 | 4477.075391 |
| 1 | 840.229492 |
| 2 | 852.053125 |
| 3 | 846.93125 |
| 4 | 956.429297 |
| 5 | 822.858398 |
| 6 | 846.326367 |
| 7 | 986.365527 |

C++ Static

|  |  |  |  |
| --- | --- | --- | --- |
| Threads | Matrix | Technique | Execution Time |
| 2 | data\_10\_10\_3.txt | 0 | 0.035775 |
| 1 | 2.468825 |
| 2 | 2.2229 |
| 3 | 2.211325 |
| 4 | 2.2809 |
| 5 | 2.198375 |
| 6 | 2.20625 |
| 7 | 2.297775 |
| data\_10\_10000\_5.txt | 0 | 29.932301 |
| 1 | 17.4219 |
| 2 | 19.151424 |
| 3 | 17.927425 |
| 4 | 17.651251 |
| 5 | 32.727825 |
| 6 | 46.379776 |
| 7 | 57.281025 |
| data\_1000\_1000\_5.txt | 0 | 304.221008 |
| 1 | 177.945007 |
| 2 | 176.363754 |
| 3 | 180.406998 |
| 4 | 160.64325 |
| 5 | 315.553253 |
| 6 | 162.996246 |
| 7 | 167.540009 |
| data\_10000\_10\_5.txt | 0 | 47.587299 |
| 1 | 26.636051 |
| 2 | 27.656776 |
| 3 | 34.802673 |
| 4 | 36.026978 |
| 5 | 46.536926 |
| 6 | 26.5042 |
| 7 | 34.821323 |
| data\_10000\_10000\_5.txt | 0 | 30999.5 |
| 1 | 16638.125 |
| 2 | 17833 |
| 4 | data\_10\_10\_3.txt | 0 | 0.027433 |
| 1 | 2.3353 |
| 2 | 2.371733 |
| 3 | 2.4536 |
| 4 | 2.470567 |
| 5 | 2.358967 |
| 6 | 2.3919 |
| 7 | 2.488167 |
| data\_10\_10000\_5.txt | 0 | 29.837334 |
| 1 | 16.053933 |
| 2 | 12.534733 |
| 3 | 14.002266 |
| 4 | 13.8644 |
| 5 | 14.6676 |
| 6 | 10.931733 |
| 7 | 12.841067 |
| data\_1000\_1000\_5.txt | 0 | 337.899007 |
| 1 | 110.555298 |
| 2 | 107.457397 |
| 3 | 99.668467 |
| 4 | 106.433929 |
| 5 | 113.920664 |
| 6 | 117.437002 |
| 7 | 112.698629 |
| data\_10000\_10\_5.txt | 0 | 45.810699 |
| 1 | 17.824066 |
| 2 | 18.921633 |
| 3 | 28.201335 |
| 4 | 25.782867 |
| 5 | 21.204834 |
| 6 | 17.464667 |
| 7 | 30.075033 |
| data\_10000\_10000\_5.txt | 0 | 32409.30078 |
| 1 | 10032.40039 |
| 8 | data\_10\_10\_3.txt | 0 | 0.030533 |
| 1 | 2.680133 |
| 2 | 2.8551 |
| 3 | 2.826 |
| 4 | 2.702333 |
| 5 | 2.3977 |
| 6 | 2.836633 |
| 7 | 2.820667 |
| data\_10\_10000\_5.txt | 0 | 36.335668 |
| 1 | 10.780666 |
| 2 | 11.137367 |
| 3 | 9.300367 |
| 4 | 9.807266 |
| 5 | 14.9554 |
| 6 | 9.343367 |
| 7 | 9.708333 |
| data\_1000\_1000\_5.txt | 0 | 430.813314 |
| 1 | 73.61853 |
| 2 | 67.910868 |
| 3 | 72.316467 |
| 4 | 68.469233 |
| 5 | 98.823466 |
| 6 | 66.650899 |
| 7 | 70.353867 |
| data\_10000\_10\_5.txt | 0 | 54.957703 |
| 1 | 12.208 |
| 2 | 12.207733 |
| 3 | 26.7497 |
| 4 | 27.969032 |
| 5 | 21.229434 |
| 6 | 12.791567 |
| 7 | 25.904032 |
| data\_10000\_10000\_5.txt | 0 | 32124.80078 |
| 1 | 6435.330078 |
| 16 | data\_10\_10\_3.txt | 0 | 0.0323 |
| 1 | 2.8106 |
| 2 | 3.5865 |
| 3 | 2.953967 |
| 4 | 3.3251 |
| 5 | 3.590133 |
| 6 | 3.5405 |
| 7 | 3.425734 |
| data\_10\_10000\_5.txt | 0 | 29.993668 |
| 1 | 12.007033 |
| 2 | 12.0009 |
| 3 | 9.844934 |
| 4 | 10.269866 |
| 5 | 9.341667 |
| 6 | 9.354567 |
| 7 | 9.9715 |
| data\_1000\_1000\_5.txt | 0 | 335.767008 |
| 1 | 63.376933 |
| 2 | 70.152629 |
| 3 | 71.380402 |
| 4 | 70.082601 |
| 5 | 65.32663 |
| 6 | 94.394399 |
| 7 | 70.785833 |
| data\_10000\_10\_5.txt | 0 | 55.017899 |
| 1 | 12.286166 |
| 2 | 12.899667 |
| 3 | 31.123301 |
| 4 | 26.877335 |
| 5 | 17.892067 |
| 6 | 12.698433 |
| 7 | 23.669467 |
| data\_10000\_10000\_5.txt | 0 | 31523.30078 |
| 1 | 5411.600098 |

C++ Dynamic

|  |  |  |  |
| --- | --- | --- | --- |
| Threads | Matrix | Technique | Execution Time |
| 2 | data\_10\_10\_3.txt | 0 | 0.01835 |
| 1 | 1.9546 |
| 2 | 2.0897 |
| 3 | 1.9262 |
| 4 | 2.03375 |
| 5 | 2.0841 |
| 6 | 1.86135 |
| 7 | 1.88165 |
| data\_10\_10000\_5.txt | 0 | 36.7127 |
| 1 | 20.384951 |
| 2 | 17.936251 |
| 3 | 19.312302 |
| 4 | 19.052799 |
| 5 | 32.268349 |
| 6 | 23.478352 |
| 7 | 17.744801 |
| data\_1000\_1000\_5.txt | 0 | 316.156006 |
| 1 | 169.940002 |
| 2 | 182.761002 |
| 3 | 181.212997 |
| 4 | 208.867004 |
| 5 | 318.799988 |
| 6 | 198.170013 |
| 7 | 195.315491 |
| data\_10000\_10\_5.txt | 0 | 32.89135 |
| 1 | 18.018452 |
| 2 | 24.17355 |
| 3 | 17.672649 |
| 4 | 18.282101 |
| 5 | 32.505402 |
| 6 | 20.7582 |
| 7 | 18.5868 |
| data\_10000\_10000\_5.txt | 0 | 31659.65039 |
| 1 | 1866.351563 |
| 2 | 17735.80078 |
| 4 | data\_10\_10\_3.txt | 0 | 0.0154 |
| 1 | 2.306567 |
| 2 | 2.4485 |
| 3 | 2.584333 |
| 4 | 3.041633 |
| 5 | 2.351167 |
| 6 | 2.2233 |
| 7 | 2.1427 |
| data\_10\_10000\_5.txt | 0 | 34.795502 |
| 1 | 13.407567 |
| 2 | 14.630834 |
| 3 | 11.722467 |
| 4 | 14.551767 |
| 5 | 14.099734 |
| 6 | 15.323466 |
| 7 | 16.632567 |
| data\_1000\_1000\_5.txt | 0 | 330.606669 |
| 1 | 99.613729 |
| 2 | 108.473002 |
| 3 | 103.626831 |
| 4 | 110.013601 |
| 5 | 104.305664 |
| 6 | 104.45933 |
| 7 | 113.181 |
| data\_10000\_10\_5.txt | 0 | 33.531268 |
| 1 | 11.924166 |
| 2 | 15.119934 |
| 3 | 14.5198 |
| 4 | 14.547234 |
| 5 | 13.685767 |
| 6 | 13.294666 |
| 7 | 12.969367 |
| data\_10000\_10000\_5.txt | 0 | 32459.80078 |
| 1 | 10145.40039 |
| 8 | data\_10\_10\_3.txt | 0 | 0.016933 |
| 1 | 2.6843 |
| 2 | 2.636533 |
| 3 | 2.630867 |
| 4 | 2.8015 |
| 5 | 2.327233 |
| 6 | 2.697533 |
| 7 | 2.5432 |
| data\_10\_10000\_5.txt | 0 | 33.800334 |
| 1 | 11.411901 |
| 2 | 11.740267 |
| 3 | 9.917 |
| 4 | 11.754866 |
| 5 | 12.446634 |
| 6 | 10.240067 |
| 7 | 9.8989 |
| data\_1000\_1000\_5.txt | 0 | 329.529663 |
| 1 | 68.991699 |
| 2 | 76.538966 |
| 3 | 73.247365 |
| 4 | 71.016902 |
| 5 | 105.795664 |
| 6 | 70.299001 |
| 7 | 70.219666 |
| data\_10000\_10\_5.txt | 0 | 33.934532 |
| 1 | 9.917567 |
| 2 | 10.261 |
| 3 | 11.719299 |
| 4 | 11.681466 |
| 5 | 13.2912 |
| 6 | 9.5421 |
| 7 | 10.120167 |
| data\_10000\_10000\_5.txt | 0 | 32401.59961 |
| 1 | 6682.569824 |
| 16 | data\_10\_10\_3.txt | 0 | 0.017033 |
| 1 | 3.012633 |
| 2 | 3.2286 |
| 3 | 2.715033 |
| 4 | 3.355234 |
| 5 | 3.1877 |
| 6 | 3.067667 |
| 7 | 2.956667 |
| data\_10\_10000\_5.txt | 0 | 31.961833 |
| 1 | 10.788534 |
| 2 | 11.145566 |
| 3 | 10.451233 |
| 4 | 10.2521 |
| 5 | 9.832133 |
| 6 | 10.582967 |
| 7 | 9.974333 |
| data\_1000\_1000\_5.txt | 0 | 335.304667 |
| 1 | 74.095601 |
| 2 | 70.146235 |
| 3 | 72.9304 |
| 4 | 70.863597 |
| 5 | 70.791036 |
| 6 | 69.764699 |
| 7 | 74.234299 |
| data\_10000\_10\_5.txt | 0 | 32.290301 |
| 1 | 9.7076 |
| 2 | 9.589767 |
| 3 | 10.851367 |
| 4 | 12.823733 |
| 5 | 9.9653 |
| 6 | 10.530933 |
| 7 | 9.8004 |
| data\_10000\_10000\_5.txt | 0 | 3223.800781 |
| 1 | 5817.799805 |

**Analysis of techniques:**

* Sequential vs parallel

It can be observed that for the configuration of the smallest sized matrix, the sequential approach is the most efficient among all. This is most probably due to the fact that parallel programs also handle with thread creation and deletion, and also slight delays caused by scheduler’s swapping of threads, which if summed up can caused higher and unnecessary time and even computational efforts.

On the other hand, it is evident that the more the size of the matrix increases, the more the parallel programs perform faster than the sequential approach, because of the efficient parallelization mechanism which have a much more importance in this case.

Statistically, it is noticed that techniques 1, 2 and 5 tends to be the most frequent ones with the least computation time, possibly due to the fast processing of the matrix because of memory blocks being brought and used more efficiently by multiple threads at the same time.

* Java vs C++

Overall, C++ tends to take a much longer time compared to Java. This could be caused by multiple optimizations provided by Java environment, compared to C++.

* C++ static vs dynamic

Despite the somewhat closer results between the 2 implementations, the dynamic C++ program still tends to perform a little faster than the static one, which is also backed by the memory advantage brought by dynamic and fast memory allocation. Additionally, this thin gap between the two implementations persists even on the largest matrix configuration, which takes a lot of time to compute.