CS 211: High Performance Computing Project 1

Performance Optimization via Register and Cache Reuse

Name: Yuanhang Luo

Date: October 12th, 2017

1. **Register Reuse**

Part #1.

For n=1000, the time my computer spends to finish ***dgemm0*** is 5.5672(s) ; the time my computer spends to finish ***dgemm1*** is 4.2780(s) . The time wasted on accessing operands that are not in registers is 1.2892(s) .

The time spend in the triple loop for each algorithm (***dgemm0, dgemm1***) on TARDIS with n = 64, 128, 256, 512, 1024, 2048 is: (in seconds)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ­algorithm | n=64 | n=128 | n=256 | n=512 | n=1024 | n=2048 |
| dgemm0 | 0.00 | 0.04 | 0.37 | 3.95 | 34.19 | 548.45 |
| dgemm1 | 0.00 | 0.02 | 0.23 | 2.73 | 24.10 | 351.63 |

The performance of each algorithms on TARDIS with n = 64, 128, 256, 512, 1024, 2048 is: (in Gflops)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| algorithm | n=64 | n=128 | n=256 | n=512 | n=1024 | n=2048 |
| dgemm0 | NA | 0.10 | 0.09 | 0.07 | 0.06 | 0.03 |
| dgemm1 | NA | 0.21 | 0.15 | 0.10 | 0.09 | 0.05 |

Part #2.

The time spend in the algorithm ***dgemm2*** on TARDIS with n = 64, 128, 256, 512, 1024, 2048 is:

The performance of the algorithm ***dgemm2*** on TARDIS with n = 64, 128, 256, 512, 1024, 2048 is:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| algorithm | n=64 | n=128 | n=256 | n=512 | n=1024 | n=2048 |
| dgemm2 | 0.00 | 0.02 | 0.18 | 1.99 | 19.05 | 205.10 |

Part #3.

The performance comparisons of ***dgemm0, dgemm1, dgemm2, dgemm3***:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| algorithm | n=64 | n=128 | n=256 | n=512 | n=1024 | n=2048 |
| dgemm0 | NA | 0.10 | 0.09 | 0.07 | 0.06 | 0.03 |
| dgemm1 | NA | 0.21 | 0.15 | 0.10 | 0.09 | 0.05 |
| dgemm2 | NA | 1.68 | 1.49 | 1.45 | 0.90 | 0.67 |
| dgemm3 | NA | 3.36 | 3.36 | 2.08 | 1.72 | 1.13 |

1. **Cache Reuse**

Part 1.

For 10\*10 matrices with ijk, ikj, jik, jki, kij, kji algorithm, cache misses of each elements are the same:

A[1,1]: 1, A[1,2]: 0, A[1,3]: 0, …, A[1,10]: 0

A[2,1]: 1, A[2,2]: 0, A[2,3]: 0, …, A[2,10]: 0

A[3,1]: 1, A[3,2]: 0, A[3,3]: 0, …, A[3,10]: 0

…

A[10,1]: 1, A[10,2]: 0, A[10,3]: 0, …, A[10,10]: 0

B[1,1]: 1, B[1,2]: 0, B[1,3]: 0, …, B[1,10]: 0

B[2,1]: 1, B[2,2]: 0, B[3,3]: 0, …, B[3,10]: 0

…

B[10,1]: 1, B[10,2]: 0, B[10,3]: 0, …, B[10,10]: 0

C[1,1]: 1, C[1,2]: 0, C[1,3]: 0, …, C[1,10]: 0

C[2,1]: 1, C[2,2]: 0, C[2,3]: 0, …, C[2,10]: 0

…

C[10,1]: 1, C[10,2]: 0, C[10,3]: 0, …, C[10,10]: 0

The percentage of read cache misses：1.4%

For 10000\*10000 matrices with **ijk&jik** algorithm, cache misses of each elements:

A[1,1]: 10000, A[1,2]: 0, …,A[1,11]: 10000, A[1,12]: 0, …, A[1,10000]: 0

A[2,1]: 10000, A[2,2]: 0, …,A[2,11]: 10000, A[2,12]: 0, …, A[2,10000]: 0

…

A[10000,1]: 10000, A[10000,2]: 0, …,A[10000,11]: 10000, A[10000,12]: 0, …, A[10000,10000]: 0

B[1,1]: 10000, B[1,2]: 10000, B[1,3]: 10000, …, B[1,10000]: 10000

B[2,1]: 10000, B[2,2]: 10000, B[3,3]: 10000, …, B[3,10000]: 10000

…

B[10000,1]: 10000, B[10000,2]: 10000, B[10000,3]: 10000, …, B[10000,10000]: 10000

C[1,1]: 1, C[1,2]: 1, C[1,3]: 1, …, C[1,10000]: 1

C[2,1]: 1, C[2,2]: 1, C[2,3]: 1, …, C[2,10000]: 1

…

C[10000,1]: 1, C[10000,2]: 1, C[10000,3]: 1, …, C[10000,10000]: 1

The percentage of read cache misses：1.4%

For 10000\*10000 matrices with **ikj&kij** algorithm, cache misses of each elements:

A[1,1]: 1, A[1,2]: 0, …,A[1,11]: 1, A[1,12]: 0, …, A[1,10000]: 0

A[2,1]: 1, A[2,2]: 0, …,A[2,11]: 1, A[2,12]: 0, …, A[2,10000]: 0

…

A[10000,1]: 1, A[10000,2]: 0, …,A[10000,11]: 1, A[10000,12]: 0, …, A[10000,10000]: 0

B[1,1]: 10000, B[1,2]: 0, …,B[1,11]: 10000, B[1,12]: 0, …, B[1,10000]: 0

B[2,1]: 10000, B[2,2]: 0, …,B[2,11]: 10000, B[2,12]: 0, …, B[2,10000]: 0

…

B[10000,1]: 10000, B[10000,2]: 0, …,B[10000,11]: 10000, B[10000,12]: 0, …, B[10000,10000]: 0

C[1,1]: 1, C[1,2]: 0, …,C[1,11]: 1, C[1,12]: 0, …, C[1,10000]: 0

C[2,1]: 1, C[2,2]: 0, …,C[2,11]: 1, C[2,12]: 0, …, C[2,10000]: 0

…

C[10000,1]: 1, C[10000,2]: 0, …,C[10000,11]: 1, C[10000,12]: 0, …, C[10000,10000]: 0

The percentage of read cache misses：1.4%

For 10000\*10000 matrices with **jki&kji** algorithm, cache misses of each elements:

A[1,1]: 10000, A[1,2]: 10000, A[1,3]: 10000, …, A[1,10000]: 10000

A[2,1]: 10000, A[2,2]: 10000, A[3,3]: 10000, …, A[3,10000]: 10000

…

A[10000,1]: 10000, A[10000,2]: 10000, A[10000,3]: 10000, …, A[10000,10000]: 10000

B[1,1]: 1, B[1,2]: 1, B[1,3]: 1, …, B[1,10000]: 1

B[2,1]: 1, B[2,2]: 1, B[2,3]: 1, …, B[2,10000]: 1

…

B[10000,1]: 1, B[10000,2]: 1, B[10000,3]: 1, …, B[10000,10000]: 1

C[1,1]: 10000, C[1,2]: 10000, C[1,3]: 10000, …, C[1,10000]: 10000

C[2,1]: 10000, C[2,2]: 10000, C[2,3]: 10000, …, C[2,10000]: 10000

…

C[10000,1]: 10000, C[10000,2]: 10000, C[10000,3]: 10000, …, C[10000,10000]: 10000

The percentage of read cache misses：1.4%

Part 2.

For 10000\*10000 matrices with **ijk&jik** blocked version algorithm, cache misses of each elements:

A[1,1]: 1, A[1,2]: 0, A[1,3]: 0, …, A[1,11]: 1, A[1,12]: 0, A[1,13]: 0,…,A[1,10000]: 0

A[2,1]: 1, A[2,2]: 0, A[2,3]: 0, …, A[2,11]: 1, A[2,12]: 0, A[2,13]: 0,…,A[2,10000]: 0

…

A[10000,1]:1, A[10000,2]:0, A[10000,3]:0,…, A[10000,11]:1, A[10000,12]:0, A[10000,13]:0,…,A[10000,10000]: 0

B[1,1]: 1000, B[1,2]: 0, B[1,3]: 0,…, B[1,11]: 1000, B[1,12]: 0, B[1,13]: 0,…,B[1,10000]: 0

B[2,1]: 1000, B[2,2]: 0, B[2,3]: 0,…, B[2,11]: 1000, B[2,12]: 0, B[2,13]: 0,…,B[2,10000]: 0

…

B[10000,1]: 1000, B[10000,2]: 0, B[10000,3]: 0, …, B[10000,11]: 1000, B[10000,12]: 0, B[10000,13]: 0, …,B[10000,10000]: 0

C[1,1]: 1, C[1,2]: 0, C[1,3]: 0, …, C[1,11]: 1, C[1,12]: 0, C[1,13]: 0,…,C[1,10000]: 0

C[2,1]: 1, C[2,2]: 0, C[2,3]: 0, …, C[2,11]: 1, C[2,12]: 0, C[2,13]: 0, …,C[2,10000]: 0

…

C[10000,1]: 1, C[10000,2]: 0, C[10000,3]: 0, …, C[10000,11]: 1, C[10000,12]: 0, C[10000,13]: 0, …,C[10000,10000]: 0

The percentage of read cache misses：0.95%

For 10000\*10000 matrices with **ikj&kij** blocked version algorithm, cache misses of each elements:

A[1,1]: 1, A[1,2]: 0, A[1,3]: 0,…, A[1,11]: 1, A[1,12]: 0, A[1,13]: 0,…,A[1,10000]: 0

A[2,1]: 1, A[2,2]: 0, A[2,3]: 0,…, A[2,11]: 1, A[2,12]: 0, A[2,13]: 0,…,A[2,10000]: 0

…

A[10000,1]: 1, A[10000,2]: 0, A[10000,3]: 0, …, A[10000,11]: 1, A[10000,12]: 0, A[10000,13]: 0, …,A[10000,10000]: 0

B[1,1]: 1000, B[1,2]: 0, B[1,3]: 0,…, B[1,11]: 1000, B[1,12]: 0, B[1,13]: 0,…,B[1,10000]: 0

B[2,1]: 1000, B[2,2]: 0, B[2,3]: 0,…, B[2,11]: 1000, B[2,12]: 0, B[2,13]: 0,…,B[2,10000]: 0

…

B[10000,1]: 1000, B[10000,2]: 0, B[10000,3]: 0, …, B[10000,11]: 1000, B[10000,12]: 0, B[10000,13]: 0, …,B[10000,10000]: 0

C[1,1]: 1000, C[1,2]: 0, C[1,3]: 0,…, C[1,11]: 1000, C[1,12]: 0, C[1,13]: 0,…,C[1,10000]: 0

C[2,1]: 1000, C[2,2]: 0, C[2,3]: 0,…, C[2,11]: 1000, C[2,12]: 0, C[2,13]: 0,…,C[2,10000]: 0

…

C[10000,1]: 1000, C[10000,2]: 0, C[10000,3]: 0, …, C[10000,11]: 1000, C[10000,12]: 0, C[10000,13]: 0, …,C[10000,10000]: 0

The percentage of read cache misses：0.95%

For 10000\*10000 matrices with **kji&jki** blocked version algorithm, cache misses of each elements:

A[1,1]: 1000, A[1,2]: 0, A[1,3]: 0,…, A[1,11]: 1000, A[1,12]: 0, A[1,13]: 0,…,A[1,10000]: 0

A[2,1]: 1000, A[2,2]: 0, A[2,3]: 0,…, A[2,11]: 1000, A[2,12]: 0, A[2,13]: 0,…,A[2,10000]: 0

…

A[10000,1]: 1000, A[10000,2]: 0, A[10000,3]: 0, …, A[10000,11]: 1000, A[10000,12]: 0, A[10000,13]: 0, …,A[10000,10000]: 0

B[1,1]: 1000, B[1,2]: 0, B[1,3]: 0,…, B[1,11]: 1000, B[1,12]: 0, B[1,13]: 0,…,B[1,10000]: 0

B[2,1]: 1000, B[2,2]: 0, B[2,3]: 0,…, B[2,11]: 1000, B[2,12]: 0, B[2,13]: 0,…,B[2,10000]: 0

…

B[10000,1]: 1000, B[10000,2]: 0, B[10000,3]: 0, …, B[10000,11]: 1000, B[10000,12]: 0, B[10000,13]: 0, …,B[10000,10000]: 0

C[1,1]: 1, C[1,2]: 0, C[1,3]: 0,…, C[1,11]: 1, C[1,12]: 0, C[1,13]: 0,…,C[1,10000]: 0

C[2,1]: 1, C[2,2]: 0, C[2,3]: 0,…, C[2,11]: 1, C[2,12]: 0, C[2,13]: 0,…,C[2,10000]: 0

…

C[10000,1]: 1, C[10000,2]: 0, C[10000,3]: 0, …, C[10000,11]: 1, C[10000,12]: 0, C[10000,13]: 0, …,C[10000,10000]: 0

The percentage of read cache misses：0.95%

Part 3.

n=2048

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Simple Method | Block 8 | Block 16 | Block 32 | Block 64 |
|
| ijk | 480.54 | 106.57 | 98.50 | 90.92 | 87.68 |
|
| jik | 596.16 | 107.93 | 104.78 | 91.83 | 88.38 |
|
| ikj | 837.26 | 116.31 | 105.85 | 112.72 | 102.61 |
|
| kij | 829.00 | 88.73 | 97.94 | 140.71 | 129.77 |
|
| jki | 567.85 | 155.22 | 131.80 | 123.86 | 122.00 |
|
| kji | 612.17 | 148.37 | 136.96 | 129.84 | 128.67 |
|

optimal block size is 64.

Part 4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | O0 | O1 | O2 | O3 |
|
| Register&Cache Reuse | 309.41 | 80.80 | 76.37 | 76.19 |
|