

# 高效能巨量資料與人工智慧系統 HW1

B05611047 化學四 鍾宜樺

1. Here is an OpenMP example of Matrix Multiply  
[https://computing.llnl.gov/tutorials/openMP/samples/C/omp\\_mm.c](https://computing.llnl.gov/tutorials/openMP/samples/C/omp_mm.c)
2. Compile and run it on a server in the CSIE Workstation Lab... Make sure it runs correctly
3. Comment out the printf statements. Run it with 1, 2, 4, 8,... threads and report the execution time

```
b05611047@linux10:~/HPBDAIS
b05611047@linux10: [~/HPBDAIS] ./test.sh
# of thread = 1
real 0m0.003s
user 0m0.000s
sys 0m0.003s
# of thread = 2
real 0m0.003s
user 0m0.000s
sys 0m0.003s
# of thread = 4
real 0m0.003s
user 0m0.000s
sys 0m0.003s
# of thread = 8
real 0m0.003s
user 0m0.000s
sys 0m0.005s
# of thread = 16
real 0m0.004s
user 0m0.000s
sys 0m0.013s
# of thread = 32
real 0m0.004s
user 0m0.000s
sys 0m0.008s
# of thread = 64
real 0m0.006s
user 0m0.006s
sys 0m0.012s
# of thread = 128
real 0m0.010s
user 0m0.010s
sys 0m0.021s
# of thread = 256
real 0m0.022s
user 0m0.003s
sys 0m0.062s
# of thread = 512
libgomp: Thread creation failed: Resource temporarily unavailable

b05611047@linux10:~/HPBDAIS
user 0m0.000s
sys 0m0.003s
# of thread = 16
real 0m0.004s
user 0m0.000s
sys 0m0.013s
# of thread = 32
real 0m0.004s
user 0m0.000s
sys 0m0.008s
# of thread = 64
real 0m0.006s
user 0m0.006s
sys 0m0.012s
# of thread = 128
real 0m0.010s
user 0m0.010s
sys 0m0.021s
# of thread = 256
real 0m0.022s
user 0m0.003s
sys 0m0.062s
# of thread = 512
libgomp: Thread creation failed: Resource temporarily unavailable

real 0m0.031s
user 0m0.000s
sys 0m0.084s
# of thread = 1024
libgomp: Thread creation failed: Resource temporarily unavailable

real 0m0.028s
user 0m0.005s
sys 0m0.084s
# of thread = 2048
libgomp: Thread creation failed: Resource temporarily unavailable

real 0m0.027s
user 0m0.000s
sys 0m0.091s
b05611047@linux10: [~/HPBDAIS]
```

report the execution time 根據上圖圖(一)：

# of threads	Real time
1	0.003(s)
2	0.003(s)
4	0.003(s)
8	0.003(s)
16	0.004(s)
32	0.004(s)
64	0.006(s)
128	0.010(s)
256	0.022(s)
512	Threads creation failed
( 超過 512 )	Threads creation failed

4. Double the values of NRA, NCA, and NCB to observe the execution time.  
(如下表格中所示)

5. Repeat Step 4 until a problem happens to the system. Report your observations.

```

b05611047@linux10:~/HPBDAIS
real 0m0.005s
user 0m0.001s
sys 0m0.005s
b05611047@linux10:~/HPBDAIS vim omp_mm_old.c
b05611047@linux10:~/HPBDAIS gcc omp_mm_old.c -fopenmp
b05611047@linux10:~/HPBDAIS time ./a.out 4

real 0m0.007s
user 0m0.011s
sys 0m0.001s
b05611047@linux10:~/HPBDAIS vim omp_mm_old.c
b05611047@linux10:~/HPBDAIS gcc omp_mm_old.c -fopenmp
b05611047@linux10:~/HPBDAIS time ./a.out 4
Multiplier = 2
real 0m0.005s
user 0m0.005s
sys 0m0.001s
b05611047@linux10:~/HPBDAIS vim omp_mm_old.c
b05611047@linux10:~/HPBDAIS gcc omp_mm_old.c -fopenmp
b05611047@linux10:~/HPBDAIS time ./a.out 4
Multiplier = 4
real 0m0.007s
user 0m0.012s
sys 0m0.000s
b05611047@linux10:~/HPBDAIS vim omp_mm_old.c
b05611047@linux10:~/HPBDAIS gcc omp_mm_old.c -fopenmp
b05611047@linux10:~/HPBDAIS time ./a.out 4
Multiplier = 8
real 0m0.017s
user 0m0.047s
sys 0m0.003s
b05611047@linux10:~/HPBDAIS vim omp_mm_old.c
b05611047@linux10:~/HPBDAIS gcc omp_mm_old.c -fopenmp
b05611047@linux10:~/HPBDAIS time ./a.out 4
Multiplier = 16
real 0m0.073s
user 0m0.272s
sys 0m0.004s
b05611047@linux10:~/HPBDAIS vim omp_mm_old.c
b05611047@linux10:~/HPBDAIS gcc omp_mm_old.c -fopenmp
b05611047@linux10:~/HPBDAIS time ./a.out 4
分割錯誤 (核心已崩潰)
real 0m0.430s
user 0m0.003s
sys 0m0.000s
b05611047@linux10:~/HPBDAIS

```

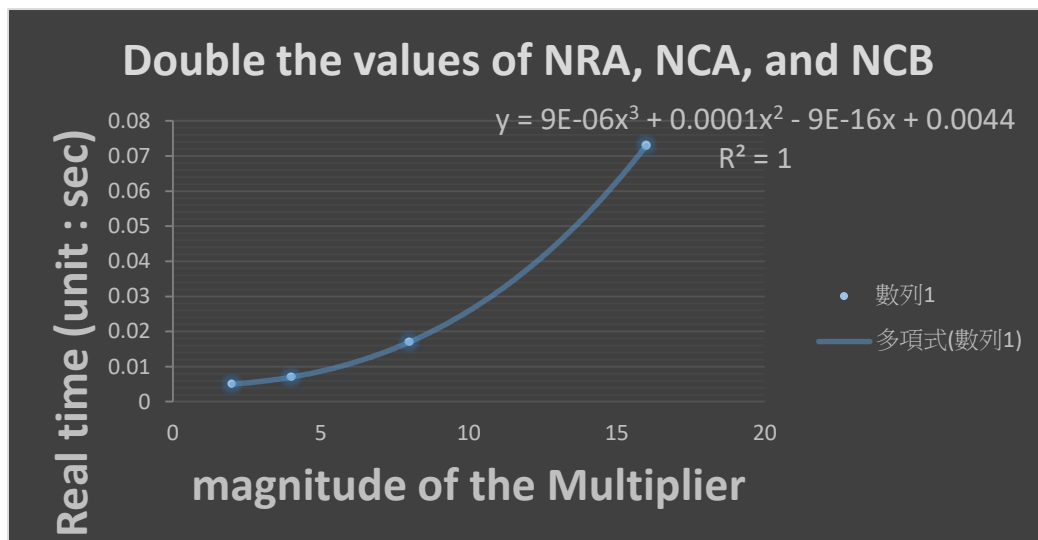
(圖二)

根據上圖圖(二)：

Multiplier value	Real time
2	0.005(s)
4	0.007(s)
8	0.017(s)
16	0.073(s)

**Report your observations :**

從上面數據看來會發現，當 multiplier (NRA, NCA, NCB 乘上的倍率) 越來越大時，run 這份 code 的 cost time 會變得越來越長。除此之外，其所花費時間的增長倍率，根據下圖圖(三) (圖(三)位於下頁) 所示：可發現其 cost time 的增長速率，正如同矩陣乘法的複雜度為  $O(n^3)$  在增長 (因為其趨勢線之  $R^2=1$ )。由此可知，當 n 的大小變得非常大的時候，optimize 便變得非常具有其重要性。



(圖三)

6. (10% Bonus) Try to optimize the code for cache when the matrices are large. Report your results.

如第 5 點所求得數據可知，工作站最多可跑到 multiplier = 16 為其極限，因此在此便設定 multiplier=16 下去 run。

※Optimize 的方法共三種：

方法(一) 如同 PPT 上寫的一樣，使用了 blocking 去做 optimize。先試出在 multiplier = 16 的狀況下，threads number 應該為多少可以讓達到 optimize 的效果。根據圖(四)可知，optimal threads number = 16。

```

b05611047@linux10:~/HPBDAIS
b05611047@linux10:~/HPBDAIS$ gcc omp_mm_old.c -fopenmp
b05611047@linux10:~/HPBDAIS$ vim test.sh
b05611047@linux10:~/HPBDAIS$ ./test.sh
# of thread = 1
Multiplier = 16
real 0m0.255s
user 0m0.246s
sys 0m0.007s
# of thread = 2
Multiplier = 16
real 0m0.112s
user 0m0.216s
sys 0m0.004s
# of thread = 4
Multiplier = 16
real 0m0.060s
user 0m0.230s
sys 0m0.004s
# of thread = 8
Multiplier = 16
real 0m0.057s
user 0m0.326s
sys 0m0.014s
# of thread = 16
Multiplier = 16
real 0m0.033s
user 0m0.436s
sys 0m0.004s
# of thread = 32
Multiplier = 16
real 0m0.044s
user 0m0.393s
sys 0m0.004s
# of thread = 64
Multiplier = 16
real 0m0.032s
user 0m0.447s
sys 0m0.008s
# of thread = 128
Multiplier = 16
real 0m0.032s
user 0m0.455s
sys 0m0.025s
# of thread = 256
Multiplier = 16
real 0m0.044s
user 0m0.421s
sys 0m0.067s
b05611047@linux10:~/HPBDAIS$

```

(圖四)

接下來試出 chunk\_size 為 64 效果最好，如圖(五)

```

b05611047@linux10:~/HPBDAIS
# of chunk = 48
Multiplier = 16
real 0m0.041s
user 0m0.435s
sys 0m0.029s
# of chunk = 52
Multiplier = 16
real 0m0.044s
user 0m0.469s
sys 0m0.007s
# of chunk = 56
Multiplier = 16
real 0m0.046s
user 0m0.483s
sys 0m0.004s
# of chunk = 60
Multiplier = 16
real 0m0.043s
user 0m0.489s
sys 0m0.004s
# of chunk = 64
Multiplier = 16
real 0m0.035s
user 0m0.468s
sys 0m0.007s
# of chunk = 68
Multiplier = 16
real 0m0.037s
user 0m0.482s
sys 0m0.007s
# of chunk = 72
Multiplier = 16
real 0m0.039s
user 0m0.493s
sys 0m0.000s
# of chunk = 76
Multiplier = 16
real 0m0.041s
user 0m0.495s
sys 0m0.007s
# of chunk = 80
Multiplier = 16
real 0m0.041s
user 0m0.505s
sys 0m0.000s
# of chunk = 84
Multiplier = 16
real 0m0.040s

```

(圖五)

因此最後得到結論，使用 blocking 的方式進行優化後。固定 multiplier = 16，調整 threads number、chunk size 得到的結果如下表。

	優化前	優化後
multiplier	16	
threads number	4	16
chunk size	10	64
Real time	0.073(s)	0.035(s)

總共花費的時間在 optimize 後減少為原本的一半。

(omp\_mm\_blocking.c Code 如副檔所示。)

方法(二) 根據網路上對 code 修改的部分為：將原本的 static 改成 dynamic。

Dynamic 和 static 時一樣，OpenMP 會將 for 迴圈的所有 iteration 依序以指定 chunk\_size 做切割成數個 chunk。但是 dynamic 時，chunk 的分配方法會是動態的；當 thread 執行完一個 chunk 後，他會在去找別的 chunk 來執行。(此段落參考資料：

[https://kheresy.wordpress.com/2006/09/15/%E7%B0%A1%E6%98%93%E7%9A%84%E7%A8%8B%E5%BC%8F%E5%B9%B3%E8%A1%8C%E5%8C%96%EF%BC%8Dopenmp%EF%BC%88%E5%9B%9B%EF%BC%89%E7%AF%84%E4%BE%8B-for/\)](https://kheresy.wordpress.com/2006/09/15/%E7%B0%A1%E6%98%93%E7%9A%84%E7%A8%8B%E5%BC%8F%E5%B9%B3%E8%A1%8C%E5%8C%96%EF%BC%8Dopenmp%EF%BC%88%E5%9B%9B%EF%BC%89%E7%AF%84%E4%BE%8B-for/))

更改前：#pragma omp for schedule (static, chunk)

更改後：#pragma omp for schedule (dynamic, chunk)

結果：

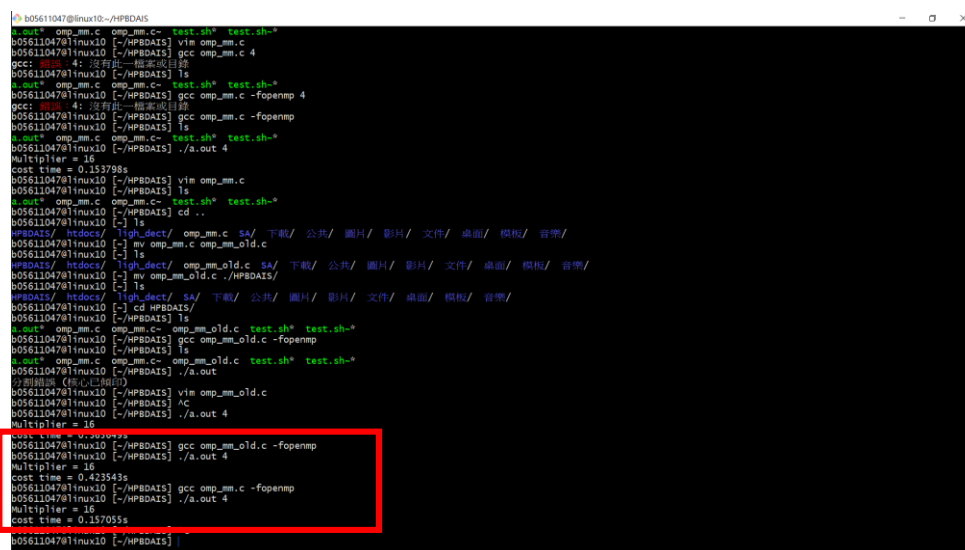
此加速方法在 cost time 上並沒有看到過多加速上的效益，因此採用了另一個方法。

方法(三) 使用計算機結構所教的 Unrolling for loops 的作法。

由於在做矩陣乘法是一共有三層的 for loop，其中 NRA=62, NCA = 15, NCB = 7。一開始我先對 NCB(=7)最 unrolled，發現有加速一點但效果並沒有很大，接著對 NCA(=15)做 unrolled。如同下圖圖(四)所示會發現速率快了近乎四倍。覺得這個方法是個很棒的 optimize 方法。

然而因為要對 NRA(=62)做 unroll 需要複製貼上 62 次，因此最後並沒有做這件事情。然而由對 NCA(=15)unrolled 後的加速結果的成效非常好。便可知，若接著對其最外層的 for loop，即對 NRA(=62)的 for 迴圈，做 unrolled 的話，速度也將在加快，達到 optimize 的效果。

且除此之外，或許在/\*\* Initialize matrices \*/的部分，也可以使用 unrolled for loop 的方式去達到加速的效益。但因為作法都一樣，所以只做了 unrolled NCA(=15)的 for loop，剩餘 for loop 做法的同理便可得知。



```
b05611047@linux10:~/HPBDAIS
$ g++ omp_mm.c -fopenmp -xOpenMP test.sh
b05611047@linux10:~/HPBDAIS$ vim omp_mm.c
b05611047@linux10:~/HPBDAIS$ gcc omp_mm.c 4
gcc: 错误: 4: 没有此选项或目录
b05611047@linux10:~/HPBDAIS$ ls
$ g++ omp_mm.c -fopenmp -xOpenMP test.sh
b05611047@linux10:~/HPBDAIS$ gcc omp_mm.c -fopenmp 4
gcc: 错误: 4: 没有此选项或目录
b05611047@linux10:~/HPBDAIS$ gcc omp_mm.c -fopenmp
b05611047@linux10:~/HPBDAIS$ ls
$ g++ omp_mm.c omp_mm.c -fopenmp test.sh
b05611047@linux10:~/HPBDAIS$ ./a.out 4
Multiplier = 16
cost time = 0.153798s
b05611047@linux10:~/HPBDAIS$ vim omp_mm.c
b05611047@linux10:~/HPBDAIS$ ls
$ g++ omp_mm.c omp_mm.c -fopenmp test.sh
b05611047@linux10:~/HPBDAIS$ cd ..
b05611047@linux10:~/HPBDAIS$ ls
HPBDAIS/ htdocs/ high_dect/ omp_mm.c SA/ 下載/ 公共/ 圖片/ 影片/ 文件/ 桌面/ 模板/ 音樂/
b05611047@linux10:~/HPBDAIS$ mv omp_mm.c omp_mm_old.c
b05611047@linux10:~/HPBDAIS$ high_dect/ omp_mm_old.c SA/ 下載/ 公共/ 圖片/ 影片/ 文件/ 桌面/ 模板/ 音樂/
b05611047@linux10:~/HPBDAIS$ mv omp_mm_old.c ./HPBDAIS/
b05611047@linux10:~/HPBDAIS$ high_dect/ SA/ 下載/ 公共/ 圖片/ 影片/ 文件/ 桌面/ 模板/ 音樂/
b05611047@linux10:~/HPBDAIS$ cd HPBDAIS/
b05611047@linux10:~/HPBDAIS$ ls
$ g++ omp_mm.c omp_mm.c omp_mm_old.c test.sh
b05611047@linux10:~/HPBDAIS$ gcc omp_mm_old.c -fopenmp
b05611047@linux10:~/HPBDAIS$ ls
$ g++ omp_mm.c omp_mm.c omp_mm_old.c test.sh
b05611047@linux10:~/HPBDAIS$ ./a.out
初始化 (核心已關閉)
b05611047@linux10:~/HPBDAIS$ vim omp_mm_old.c
b05611047@linux10:~/HPBDAIS$ AC
b05611047@linux10:~/HPBDAIS$ ./a.out 4
Multiplier = 16
cost time = 0.423543s
b05611047@linux10:~/HPBDAIS$ gcc omp_mm.c -fopenmp
b05611047@linux10:~/HPBDAIS$ ./a.out 4
Multiplier = 16
cost time = 0.157055s
b05611047@linux10:~/HPBDAIS$
```

(圖四)

	omp_mm_old.c	omp_mm.c
chunk 分配方式	static	dynamic
For loop 是否 unrolled	不做 unrolled	做 unrolled
# of threads	4	4
Magnitude of the multiplier	16	16
Cost time	0.423543(sec)	0.157055(sec)
Result	原本較慢	成功 optimized

(omp\_mm\_unroll.c Code 如副檔所示。)