HW4 Benchmark

HPL

> 目標：觀察不同變因對 Gflops 的影響，找出能最大化其值的方法。

Best Result

- **Compile:** `mpirun -np 12 ./xhpl > output.txt` (有個12個core，所以最多可用到12個process)
- **Parameters**
  - N = 12000
  - NB = 64
  - P = 2

- Q = 6

```
1            # of problems sizes (N)
12000        Ns
1            # of NBs
64           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
6            Qs
```

- **Result**: 性能提升了 **約824倍**

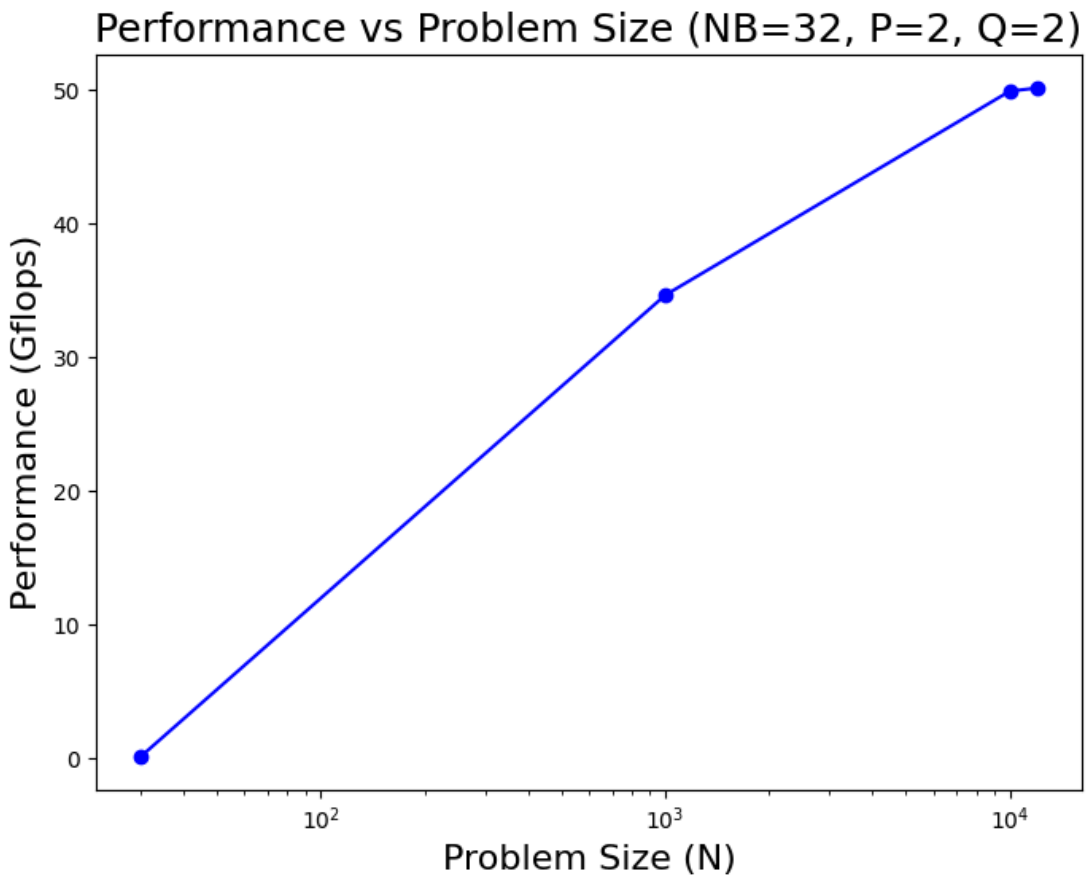
【Before】

```
----------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   1.84177500e-02 ...... PASSED
============================================================================
T/V                N    NB     P     Q               Time                 Gflops
----------------------------------------------------------------------------
WR00R2R4          30    32     2     2               0.00            1.7551e-01
HPL_pdgesv() start time Mon Apr 22 07:41:30 2024

HPL_pdgesv() end time   Mon Apr 22 07:41:30 2024

----------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   1.60609763e-02 ...... PASSED
============================================================================

Finished     18 tests with the following results:
             18 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
----------------------------------------------------------------------------

End of Tests.
============================================================================
```

【After】

```
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   3.23751010e-03 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00R2R4       12000    64     2     6               7.96             1.4466e+02
HPL_pdgesv() start time Mon Apr 22 07:02:50 2024

HPL_pdgesv() end time   Mon Apr 22 07:02:58 2024

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   3.09736947e-03 ...... PASSED
================================================================================

Finished     18 tests with the following results:
             18 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```

Problem Size (N)



- **合理problem size的重要性**：理想的problem size應能充分利用系統的primary storage，同時避免因超出記憶體容量而導致過多的swapping，否則可能會對性能產

生負面影響。

- **計算公式**

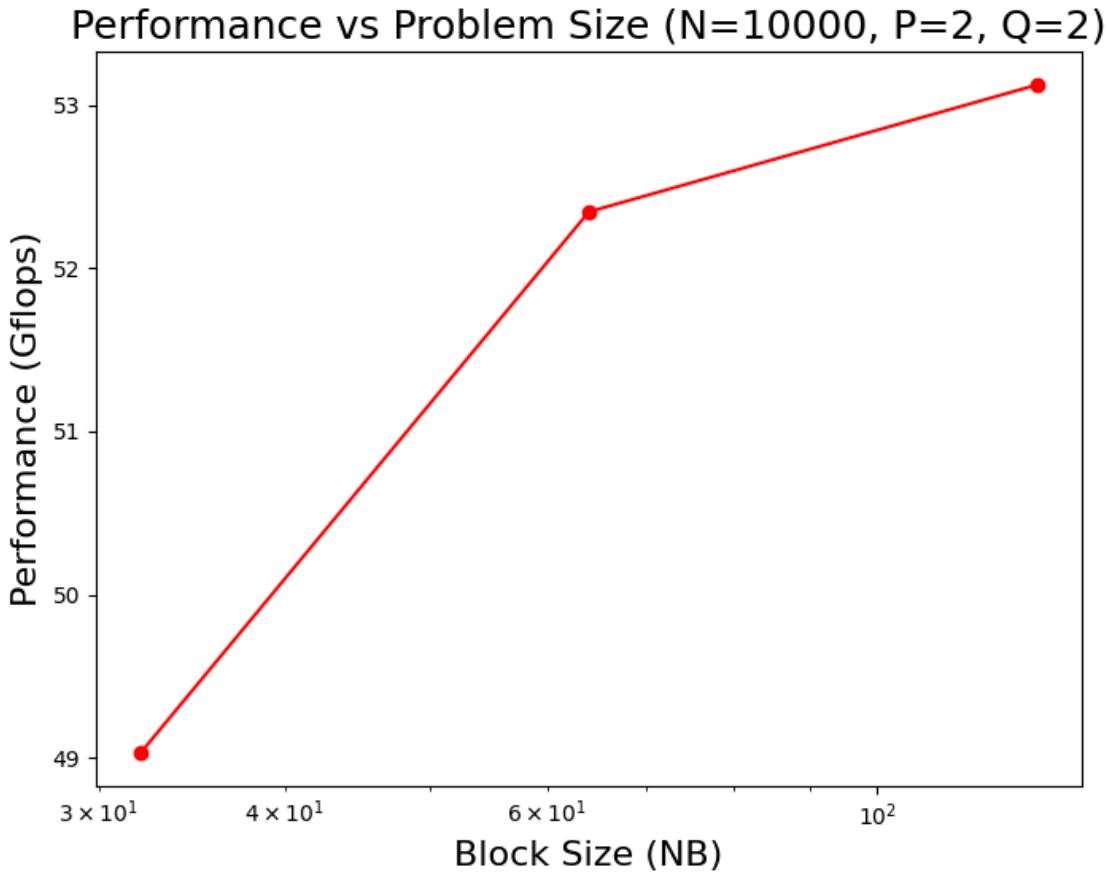
$$N = \sqrt{\frac{\text{TotalMemory}}{8}}$$

其中，TotalMemory 是**系統可用的總記憶體大小**，用 `free -b` 指令查詢可知硬體總記憶體大小為*1,905,000,448 bytes*，且實際能使用的通常是80~85%，於是我們帶入公式計算理想的problem size (N)：

$$N = \sqrt{\frac{1905000448 \times 0.8}{8}} \approx 13802$$

- **Problem Size (N) 對性能的影響**
  - 由圖表可知，當N從較小值30增加到大於10,000時， 系統性能顯著提升 。這是因為較大的problem size可以更有效利用CPU的計算能力和記憶體的bandwidth (大型問題能更充分激發出硬體的潛能)。
  - 然而，可以看到大於10,000後就沒什麼大幅提升了。因此，也必須注意加大problem size並非無限制的有效，當 N 過大超過系統的資源負載能力，會導致記憶體不足而引起效能低落 (得進行資料交換來管理記憶體需求)。
  - **結論**：N 值需要經過反覆實驗，並根據硬體配置仔細調整，以確保最大程度利用所有資源。

Block Size (NB)



- 用HPL進行計算時，N×N 的矩陣通常會被分割成多個blocks被分配到不同的nodes上進行處理，其基本單位稱為**block size(NB)**，對性能有顯著的影響。
- 較小的NB有助於實現更好的load balance，因為系統能夠將計算工作更細致地分散到多個計算節點上；但是，過小也可能限制計算效能，造成快取被有效reuse的機會減少，即資料一旦被加載快取很快就被淘汰。
- **選擇NB的方式**：通常NB會設在32~256，在load balance和計算效能之間取得平衡，所以我選擇了32、64及128進行測試 (N=10000, P=2, Q=2)。
- **實驗結果**：發現隨著NB增加，GFLOPS也有些微增加，可見在保持資料有效利用的同時，適當增加NB能夠提升計算效率；但要注意的是效果並不明顯，且後來發現還是要實際測試，不同N適用的NB仍可能有差。

Process Grid (P & Q)

- **處理器網格 (Process Grid)** 用於將計算任務有效分配到不同的處理器或節點上。以HPL為例，處理器網格通常由 P×Q 個處理器組成，其中 P 代表網格的行數，而 Q 則代表列數。
- **選擇P、Q的方式**
  - 因使用12個process，PxQ 需為12。
  - 讓 P、Q 越接近越好(近方形)，以減少communication並提高 load balance。
  - 根據實驗結果： P≤Q，且 P 為 2 的冪次時表現較好。

P*Q=12	Gflops
1*12	144.98
2*6	**149.66**
3*4	144.28
4*3	147.89
6*2	142.31
12*1	126.54

> 實際應用中，選擇更大的 P×Q 可以實現更精細的任務分配，進而提升各處理單元間的資源利用率以及GFLOPS性能。

HPCG

> 目標：觀察不同變因對 GFLOP/s rating 的影響，找出能最大化其值的方法。

Best Result

- **Compile:** `mpirun -np 12 ./xhpcg` (有個12個core，所以最多可用到12個process)
- **Parameters** (主要變因其實只有np和problem size) \
  - Problem size = `24*24*24`
  - Execution time = 200s

```
HPCG benchmark input file
Sandia National Laboratories; University of Tennessee, Knoxville
24 24 24
200
```

- **Result**: 性能提升了 **約3.94倍**
  【Before】

```
GFLOP/s Summary=
GFLOP/s Summary::Raw DDOT=0.926012
GFLOP/s Summary::Raw WAXPBY=10.9185
GFLOP/s Summary::Raw SpMV=8.35143
GFLOP/s Summary::Raw MG=6.56964
GFLOP/s Summary::Raw Total=6.17908
GFLOP/s Summary::Total with convergence overhead=6.17908
GFLOP/s Summary::Total with convergence and optimization phase overhead=6.0475
User Optimization Overheads=
User Optimization Overheads::Optimization phase time (sec)=1.71e-07
User Optimization Overheads::Optimization phase time vs reference SpMV+MG time=6.83412e-06
DDOT Timing Variations=
DDOT Timing Variations::Min DDOT MPI_Allreduce time=0.0187428
DDOT Timing Variations::Max DDOT MPI_Allreduce time=1.01285
DDOT Timing Variations::Avg DDOT MPI_Allreduce time=0.69254
Final Summary=
Final Summary::HPCG result is VALID with a GFLOP/s rating of=6.0475
Final Summary::HPCG 2.4 rating for historical reasons is=6.17908
Final Summary::Reference version of ComputeDotProduct used=Performance results are most likely suboptimal
Final Summary::Reference version of ComputeSPMV used=Performance results are most likely suboptimal
Final Summary::Reference version of ComputeMG used=Performance results are most likely suboptimal
Final Summary::Reference version of ComputeWAXPBY used=Performance results are most likely suboptimal
Final Summary::Results are valid but execution time (sec) is=10.4251
Final Summary::Official results execution time (sec) must be at least=1800
```

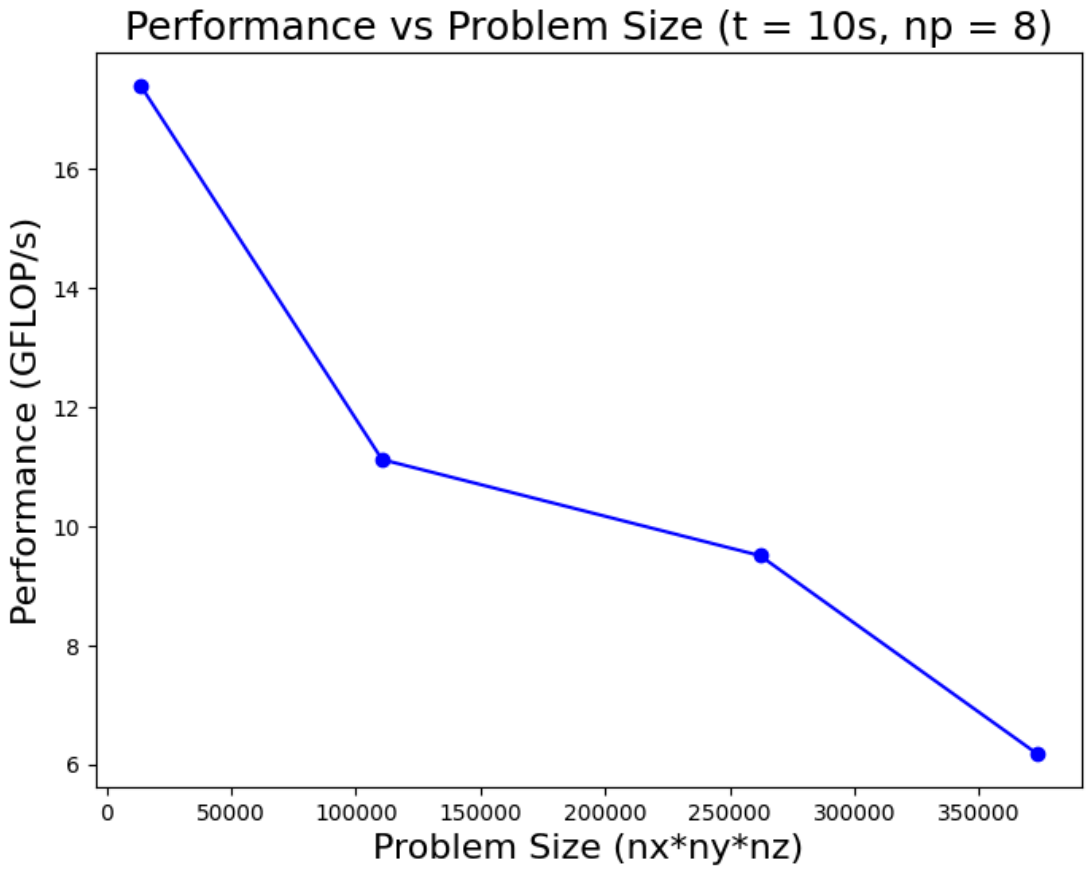
【After】

```
GFLOP/s Summary=
GFLOP/s Summary::Raw DDOT=5.94157
GFLOP/s Summary::Raw WAXPBY=84.1206
GFLOP/s Summary::Raw SpMV=32.2258
GFLOP/s Summary::Raw MG=25.4054
GFLOP/s Summary::Raw Total=25.0982
GFLOP/s Summary::Total with convergence overhead=25.0982
GFLOP/s Summary::Total with convergence and optimization phase overhead=23.8226
User Optimization Overheads=
User Optimization Overheads::Optimization phase time (sec)=1.71e-07
User Optimization Overheads::Optimization phase time vs reference SpMV+MG time=3.62598e-05
DDOT Timing Variations=
DDOT Timing Variations::Min DDOT MPI_Allreduce time=2.64939
DDOT Timing Variations::Max DDOT MPI_Allreduce time=5.52645
DDOT Timing Variations::Avg DDOT MPI_Allreduce time=4.40473
Final Summary=
Final Summary::HPCG result is VALID with a GFLOP/s rating of=23.8226
Final Summary::HPCG 2.4 rating for historical reasons is=25.0982
Final Summary::Reference version of ComputeDotProduct used=Performance results are most likely suboptimal
Final Summary::Reference version of ComputeSPMV used=Performance results are most likely suboptimal
Final Summary::Reference version of ComputeMG used=Performance results are most likely suboptimal
Final Summary::Reference version of ComputeWAXPBY used=Performance results are most likely suboptimal
Final Summary::Results are valid but execution time (sec) is=102.893
Final Summary::Official results execution time (sec) must be at least=1800
```

Problem Size（nx*ny*nz）

> 看了很多文章但好像找不到一個確切計算memory對應problem size的公式，這篇(https://community.intel.com/t5/Intel-MPI-Library/HPCG-memory-allocation/m-p/1065740)提到的hpcg.cpp當前版本也沒有所以就直接實驗了。problem size大概到 `80*80*80` 就要跑很久(也不知道跑不跑的出來)，所以只測到 `72*72*72` 。

Different sizes



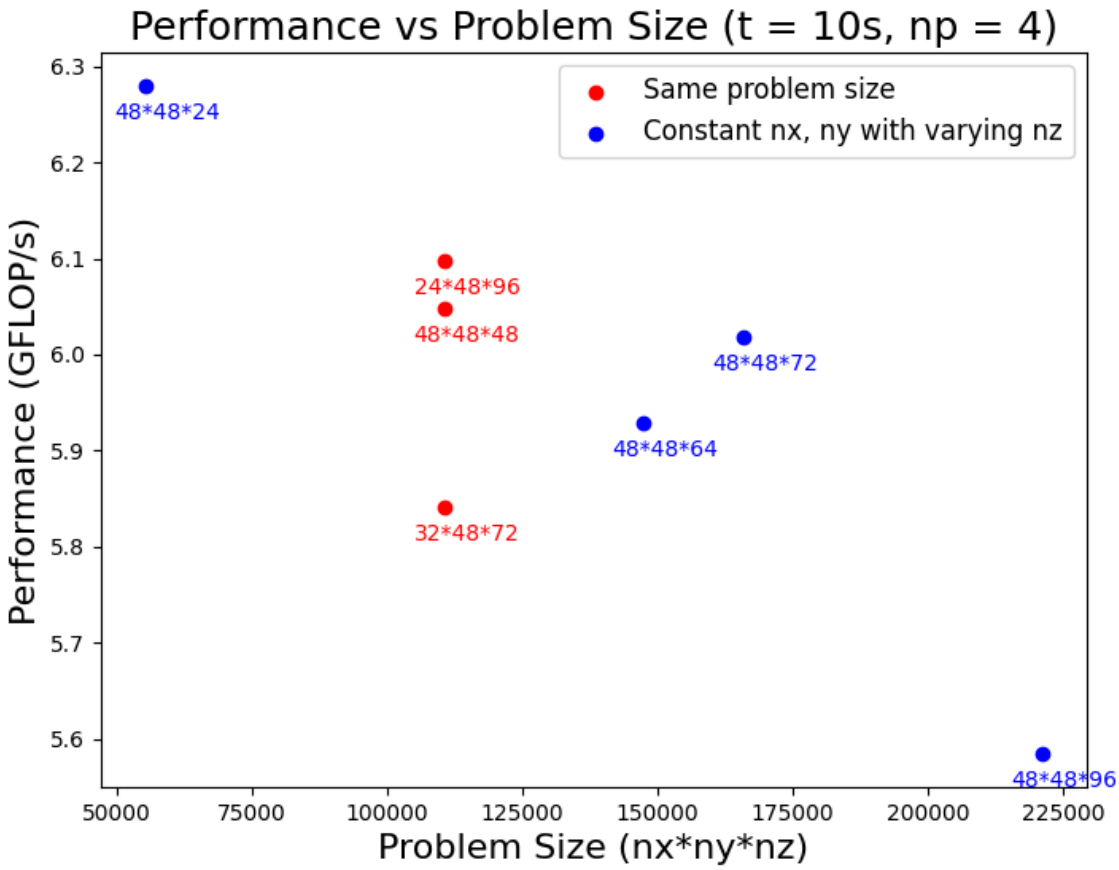
1. 各計算階段對GFLOP/s的影響

- **DDOT**：點對點乘積，數值波動較大，不同problem Size的影響不一。
- **WAXPBY**：向量操作，GFLOP/s隨著problem Size增大逐漸降低，顯示計算規模增加對性能有一定的負面影響。
- **SpMV**：疏散矩陣向量乘積，隨著problem Size增加，性能逐漸下降。
- **MG**：多重網格求解，GFLOP/s同樣隨problem Size提升而降低。

2. Problem Size與GFLOP/s的關係

- 從圖表可以看到，當problem Size增加時，總體GFLOP/s表現呈**下降趨勢**，可能是因為隨著問題尺寸的增大，每個處理步驟所需處理的數據量增加， 導致記憶體訪問延遲增加及計算密集度提高 ，是HPCG這類CPU 密集型軟體的一大瓶頸。
- 另外，實驗結果與我參考的這篇論文 (https://www.dcs.warwick.ac.uk/~sdh/pmbs14/PMBS14/Workshop_Schedule_files/10-PerformanceModelHPCG.pdf)是相符的。他們的推斷是，性能變化可以區分為problem Size「適配於CPU快取」、「過渡期」和「性能趨於穩定」三個階段。也就是說，在problem Size較小時，計算主要受益於高速快取；若problem Size增大到一定程度，過渡到主記憶體的存取，GFLOP/s表現開始下降，且因受到主記憶體bandwidth的限制，性能將趨於穩定。

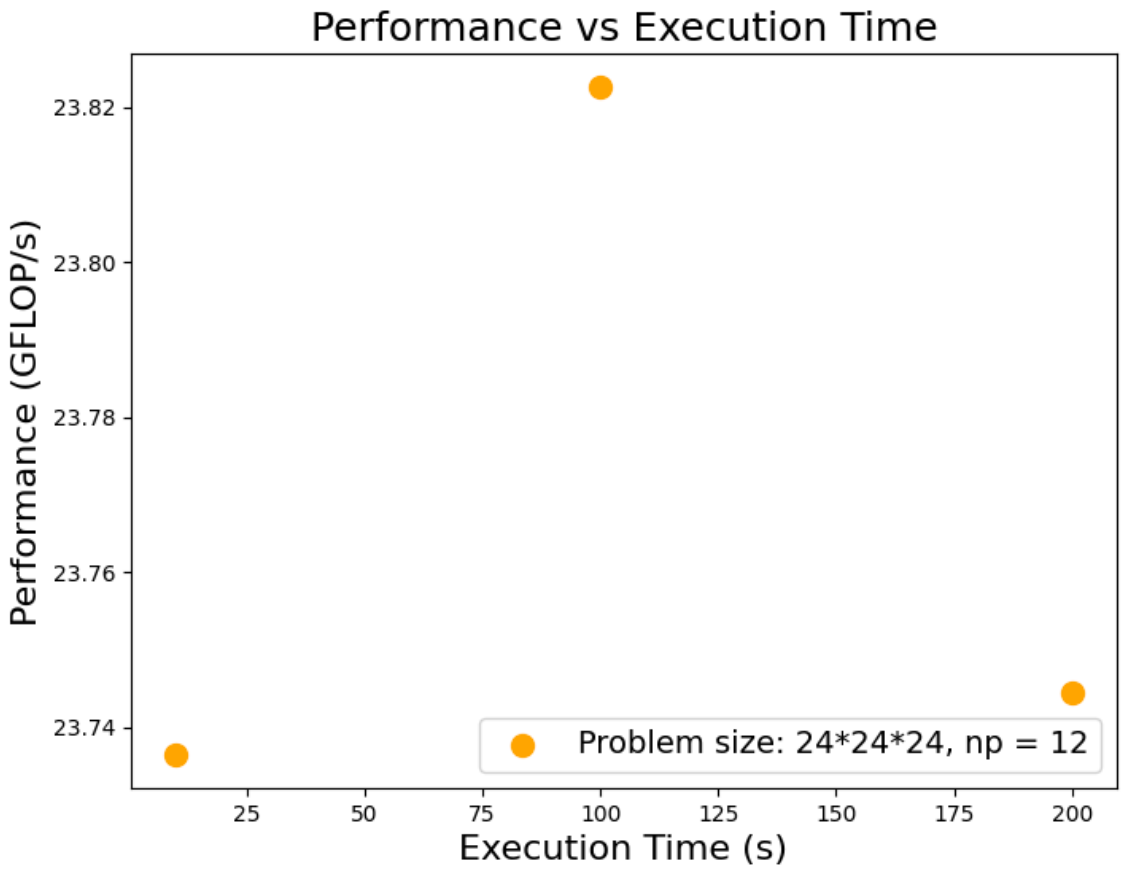
Cubic vs Noncubic



1. Same problem size (固定 `nx*ny*nz`)：三種比例中 `24*48*96` 的GFLOP/s最高，表式非立方體的資料排列方式在某些情況下仍可能稍優於立方體，與硬體架構的快取和記憶體使用方式有關，不過影響不太且規律並不明顯。

2. Constant nx, ny with varying nz：從圖表可知nz增加時，GFLOP/s整體呈現下降趨勢 (在 `48*48*72` 有略為波動)，這可能是因資料配置更符合系統快取或記憶體結構的特定效率點。

Execution Time

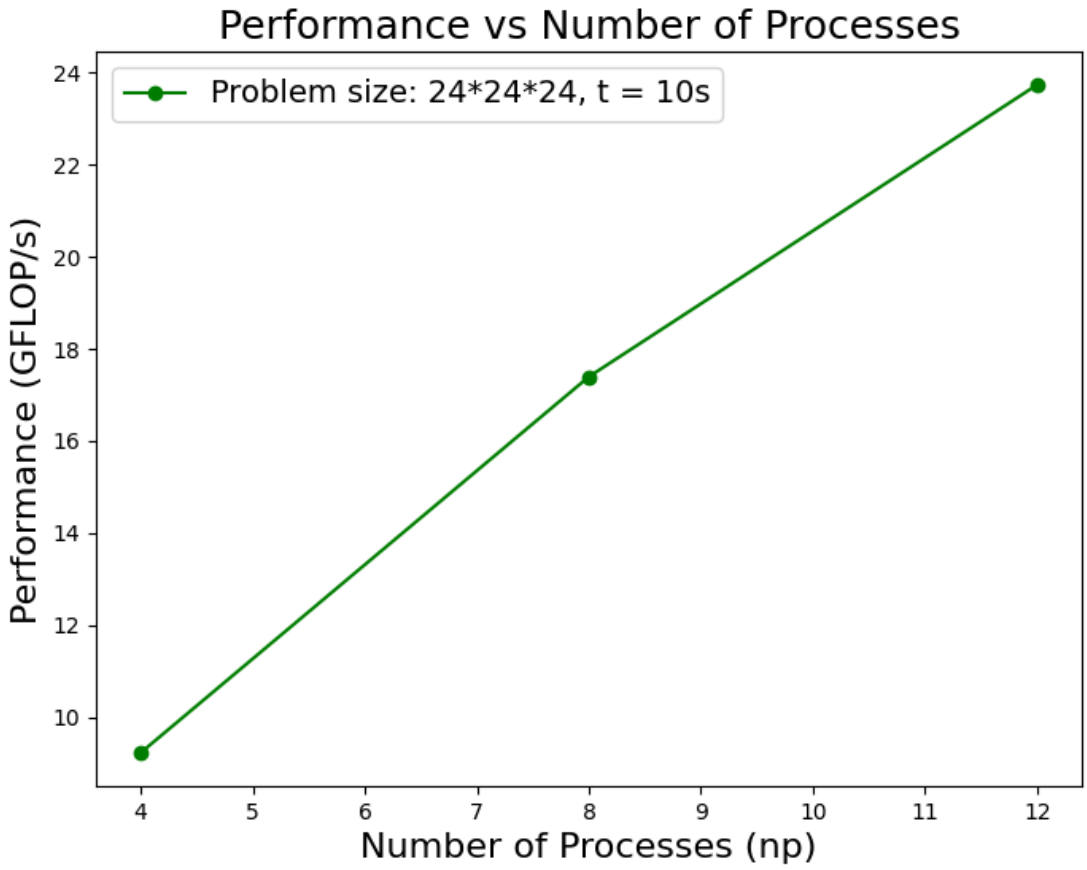


由測試結果可知，就目前的problem size，不同執行時間造成的GFLOP/s的變化幾乎可以忽略不計，這可能是因為：

1. **性能穩定性**：HPCG benchmark是用來評估高性能計算系統在處理長期、密集型計算任務時的穩定性和效率。而執行時間對GFLOP/s幾乎沒有影響表示系統在這些測試條件下表現出良好的性能穩定性；也就是說，一旦計算任務充分「熱身」，計算資源如CPU、記憶體和I/O系統等都能被有效利用。
2. **記憶體與快取使用效率**：如果problem size對於系統的快取來說夠小（如 `24*24*24` ），則資料很可能大部分時間都存放在快取，所以需要的資料都被加載到快取後性能就會保持穩定。

> 不過還無法確定更大的problem size是否受執行時間影響，若有更大的記憶體可進行測試。

Number of Processes (np)



由圖表可知，增加process數量可顯著提升性能，np = 12 的性能達到 4 的**兩倍以上**，原因如下：

1. **計算資源的平行利用**：HPCG benchmark尤其依賴計算資源的平行化。當process數量增加，更多任務可以同時進行，有助於有效利用系統的多CPU核心，減少單個核心的負擔並提高快取的效率，因此若使用OpenMP的multithreading，效能應該還會再提升。
2. **記憶體與資源分配**：當np使用適當，記憶體bandwidth和其他資源的分配也可以最佳化，避免資源競爭對性能的負面影響。

> 雖然增加process可以提高性能，但這也可能帶來更高的communication Overhead。不過就目前實驗結果而言，Communication Overhead在目前使用的process數量範圍內尚未成為限制性因素。

References

1. Marjanović, V., Gracia, J., Glass, C.W. (2015). Performance Modeling of the HPCG Benchmark. In: Jarvis, S., Wright, S., Hammond, S. (eds) High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation. PMBS 2014. Lecture Notes in Computer Science(), vol 8966. Springer, Cham.

https://doi.org/10.1007/978-3-319-17248-4_9 (https://doi.org/10.1007/978-3-319-17248-4_9)

2. Yulong Ao, Chao Yang, Fangfang Liu, Wanwang Yin, Lijuan Jiang, and Qiao Sun. 2018. Performance Optimization of the HPCG Benchmark on the Sunway TaihuLight Supercomputer. ACM Trans. Archit. Code Optim. 15, 1, Article 11 (March 2018), 20 pages. https://doi.org/10.1145/3182177 (https://doi.org/10.1145/3182177)
3. Running the Intel Distribution for LINPACK Benchmark (https://www.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/intel-oneapi-math-kernel-library-benchmarks/intel-distribution-for-linpack-benchmark-1/run-the-intel-distribution-for-linpack-benchmark.html)
4. HOW DO I TUNE MY HPL.DAT FILE? (https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/)
5. HPL - HPC NTCU (https://hackmd.io/@hpc-ntcu/S1ImlEmBa?utm_source=preview-mode&utm_medium=rec)