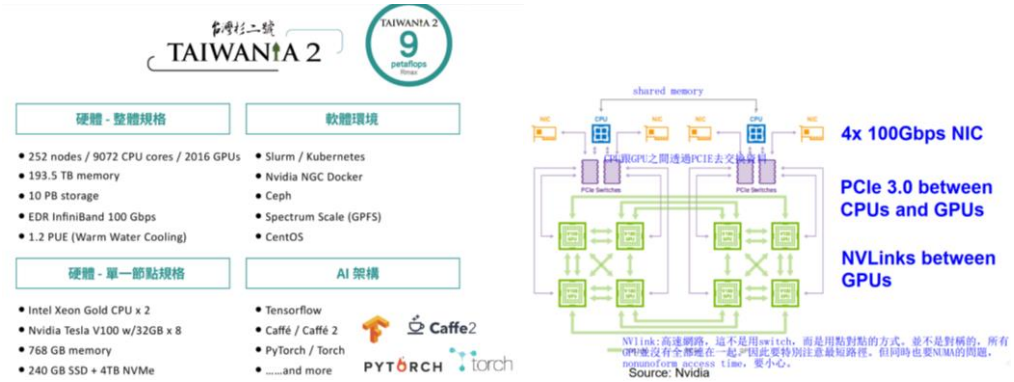
高效能巨量資料與人工智慧系統 — 期中考

B05611047 化學四 鍾宜樺

I have not cheated nor have I received any help from other people in this exam.

1.

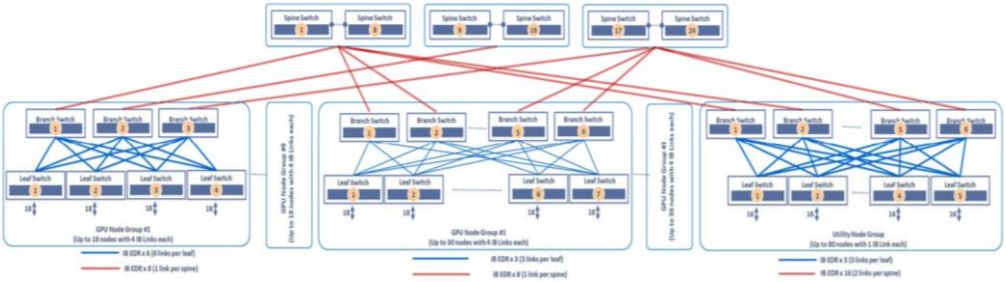
第一部分，簡述台灣杉二號的硬體規格、系統架構…等：



「台灣杉二號」超級電腦由 252 個節點組成，每個節點包含 2 顆 CPU 及 8 顆最先進 GPU，其主機架構設計與國際趨勢同步。在科學家運用大數據進行深度學習時，可表現出更優質的性能；此外，在節能方面，「台灣杉二號」的能源效率達 11.285 GF/W，計算量在 9 PFLOPS 時，用電 798 KW，亦為台灣史上最節能的高速計算主機。(此段落取自 NCHC 官網)

接著對於硬體介紹分成兩部分敘述作簡介。首先放上從 NCHC 以及老師 PPT 上擷取出的兩張圖如上所示。上圖左圖列出 Taiwania-2 的硬體配備規格、軟體環境。又 Taiwania-2 共有 252 個 nodes，而上右圖，則顯示每一個 node 裡面 CPU 與 GPU 與網卡的連接方式。每一個 node 裡面有 2 張 CPU 跟 8 張 GPU，且 CPU 跟 GPU 的擺法如上圖，會被分成兩部分，因此在計算 coast time 或者是 data transfer time 的時候可能要特別注意。再來是 CPU 與 GPU 之間適用 PCIe 3.0(網速是 x16 為 15.8 GB/s)做連接，另外每個 node 有配備 4 張 NIC(因此傳輸的網速為 4*100Gbps)，且 GPU 跟 GPU 之間適用 NVlinks 做連接。且除此之外，在同一個 node 裡面的 CPU 是可以 shared memory 的。

接著第二個部分，討論 Taiwania-2 的系統架構圖如下：



從 Taiwania-2 的系統架構圖可以看出，node 有分成兩種，分別為：utility node 跟 compute node。首先提及 utility node，它是位於上圖中的

最右邊的 node，負責 compute 以外的其他工作，例如：處理登入、監控或是儲存資料…等等項目。

接著討論 compute node。compute node 有兩種 cluster 的方式，如上圖中最左邊的 cluster 以及中間的 cluster。

第一種是上圖中最左邊的 cluster group。一個 cluster 含有三個 branch switch 和 4 個 leaf switch，從圖中可看出彼此以 bipartite graph 形式互相以 EDR Infiniband*6 相互連接，而 4 個 leaf switch 分別連接到 18 個 compute node。向外的連線則由三個 branch switch 負責，各有一條 EDR Infiniband*8 連向 spine switch，共有 9 組；

第二種含有 6 個 branch switch 和 7 個 leaf switch，其下接有 30 個 compute node，共有三組。統計一下，總共有 9*18 + 30*3 = 252 個 compute nodes，符合最上面的左圖台灣杉二號的硬體規格中所敘述。

第二部分，key features may be related to the resulted performance：

可能影響效能的部分列點如下：

1. EDR InfiniBand 100Gbps：EDR InfiniBand 從系統架構圖可看出，它是所有的(node 與 node)、(node 與 switch)以及(switch 與 switch)之間彼此的聯繫通道，因此 EDR InfiniBand 的傳輸速率，將會決定整個運算即 data transfer 的過程中資料傳輸速率。因此猜測，EDR InfiniBand 的網速有很大的機率會大大的影響效能。
2. 768GB memory：假若 memory 不夠大的時候，則 node 需要將多出來的資料放到自己的 storage(NVMe、SSD)。如此一來，取得想要的資料的時間就會增加(包含 seek time、data transfer time…等)，這也有可能會造成 overhead 影響效能
3. 4TB NVMe: 呈上面第二點，假若更不幸的，今天每個 node 所被分配到欲處理的 data 非常非常大，連 node 個別的 storage 都不夠放，則在計算大資料的時候，可能需要分批傳輸、計算，造成額外的 cost time。

2.

傳輸 data 的方式分成兩種下去討論：

A. sending over the Internet :

使用網路去傳輸的方式有很多種，例如：可以透過 scp、rsync 或 sftp 等協定去做到 send over the internet。

首先討論 send over the internet 的 detailed steps：

1. The data is broken up into bits of same size pieces called packets.

補：Data resides and converted to TCP/IP packets：

Data resides on computers in form of bits. These bits (when needed to be transferred to another computer) are converted to TCP/IP packets which are relayed through ethernet & optical fibres to the other computer. All computers share common protocol for send/recieve so they can understand each other.

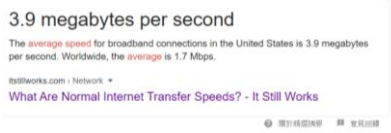
2. Header is added to each packet explaining where the data has come from, where it should end up and where it fits in with the rest of the packets. Each packet is sent from computer to computer until it find its destination.

3. Each computer on the way decides where next to send the packet. All packets may not take the same route.

4. At the destination, the packets are examined. If any packets are missing or damaged, a message is sent asking for them to be resent. This continues until all packets have been received intact.

5. The packets are now assembled into their original form.

從上面的步驟可以發現，由於第一步跟第二步，都是在原本的電腦上進行，並不會花太多時間。故 bottleneck 是產生在第三步驟，即**主要的傳輸瓶頸會在網路傳輸上**。然而，題目並沒有規定網路傳輸的速率，因此 google normal internet transfer speed：3.9MB/s(如下圖截圖所示)：



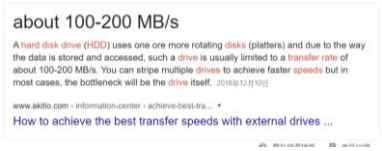
故 estimate the time (省略太小的時間，看 bottleneck)，假設網路傳輸速度為 3.9MB/s，總共有 1PB 的資料，花費時間：$\frac{1(\mathrm{PB})}{3.9(\frac{\mathrm{MB}}{\mathrm{s}})} = \frac{10^9}{3.9}(\mathrm{s}) = 2.56 * 10^8(\mathrm{s})$

B. shipping the data via hard drives.

這個做法可以是很厲害的 programmer，能夠做到自己去安排它的每個不同的 data 要怎麼擺放，配合他要處裡的問題，考慮到 data locality 的問題去自己安排每個資料的擺放位置。此步驟為人工 insert hard drive 的時間，假設這個人在人工放資料花了 10mins (= 3600sec)。另外，或許有些比較高級的超級電腦在擺放資料的時候，並不會用人工的，它可能會是由機械手臂下去做這件事情，但

不管怎麼樣，第一步都是先將 hard drive 插上去。

而在 hard drive 插上去之後會再去做 data transfer (第二步驟)。然而，題目並沒有假設 hard drive transfer speed，因此 google 的到 Normal hard drive transfer speed：100-200MB/s(如下截圖)：



透過外接硬碟的方式，將資料傳進 taiwania2 的 storage。從步驟中我們可以發現，主要瓶頸會受限於硬碟本身的傳輸速率(第二步驟)。假設硬碟傳輸速率為 150MB/s，共有 1PB 資料，共需要$\frac{1(\mathrm{PB})}{150(\frac{\mathrm{MB}}{\mathrm{s}})} = \frac{10^9}{150}(\mathrm{s}) = 6.67 * 10^6(\mathrm{s})$。即使加上一個人人工 insert hard drive 的 3600 秒，shipping the data via hard drives 的傳輸速率(cost time = $6.67 * 10^6(\mathrm{s}) + 3600(\mathrm{s})$)還是比 sending over the Internet(cost time = $2.56 * 10^8(\mathrm{s})$) 快很多。

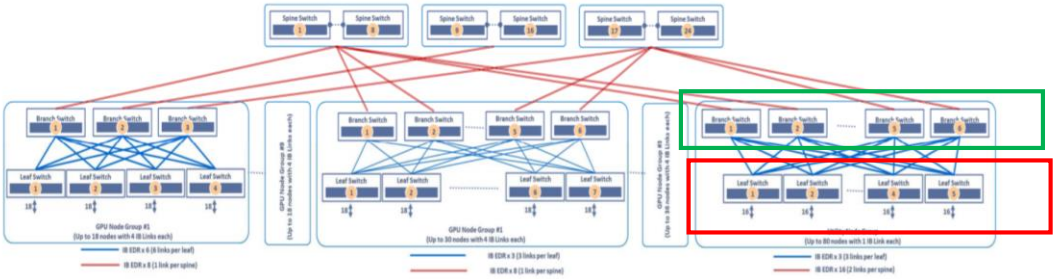
3.

欲算出整個過程的傳輸時間，先敘述流程的細節後，算出各個部分所 cost 的時間後，再做加總。並可在過程中在去找到 bottleneck。

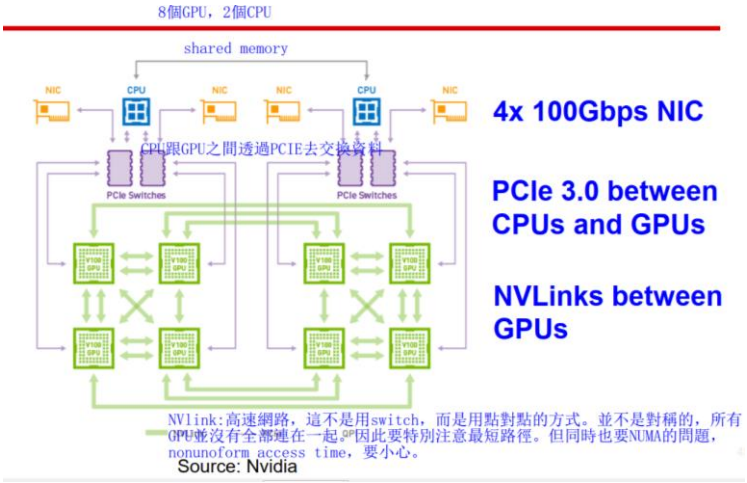
共 1PB data $\begin{cases} 99\%：9900\text{ 個 } 100GB\text{的大檔案} \\ 1\%：10^7\text{個 } 1MB\text{的小檔案} \end{cases}$，分給 252 個 compute nodes，每個 node 大約分 3.968 (TB) 的 data。由於每個 compute node 有 4TB 的 NVMe，能夠把所有 data 放到 NVMe 因為它放的下，故不須放到 SSD。

根據題目假設 dataset 已經在 Taiwania-2 的 storage 中，找檔案的速率是 100 files/sec，平均一個檔案大小是$\frac{9900}{9900+10^7} * 100GB + \frac{10^7}{9900+10^7} * 1MB =$ $99.9MB$，所以是 9990(MB/s)。

且置放於一個 storage node 中，且假設此 storage node 的快速資料存取是放在 IBM 的 LTO8 裡面 (每個 LTO8 有 12TB 的容量，此數據是自 LTO8 的 datasheet 得到的)。另外，為什麼是選擇放在 IBM 的 LTO8 磁帶中的原因，是因為根據 reference4 的 PDF 檔案中，提到 Taiwania-2 所使用的磁帶是 LTO8，故選用此磁帶。所以總共有 $1\mathrm{PB}/12\mathrm{TB} = 83.333 = 84$ 個 LTO8。假設每個 LTO8 的傳輸速率是 360MB/s(根據 LTO8 的 datasheet)，則 84 個同時傳輸資料的速率是 $84 * 360\mathrm{MB/s} = 30240\mathrm{MB/s}$。



由圖中紅色框框可以看出我們有 storage node 有五個 leaf switch，一個 EDR infiniband 的最大流量是100Gbps ＝ 12.5GB/s，總共最大流量可以到$5 * 12.5\text{GB/s} = 62.5\text{GB/s}$，接下來看綠色框框從 branch 到 spine 有四條 EDR infiniband，所以最大流量是$4 * 12.5\text{GB/s} = 50\text{GB/s}$，



當資料到達每個 compute node 的時候，資料透過 16 條 PCIe 3.0 傳到 CPU 跟 GPU 上，且 PCIe 3.0(x16)的最大傳輸速率為 15.8 GB/s。

最後比較各個步驟的最大傳輸速率，我們發現，搜尋檔案的速率是最慢的，為 bottleneck，因此傳輸所有檔案的時間大概是$\frac{1PB}{9990\frac{MB}{s}} = \mathbf{101010}(\boldsymbol{sec})$。然而，此數字是經由多次計算會有誤差值產生。故最後重新計算所求，即等同於求傳輸檔案的時間：$(10^7 + 9900) * 10(\text{ms}) = 100099(sec)$。

4.

先回答第一小題，***Is it possible for the 252 nodes of Taiwania-2 to hold the small files in the memory collectively?：***

承上題，小檔案總共有10^7個，且一個為 1MB，因此總共有10^7MB要被放進去。我們有 252 個 compute node，每個 node 的 memory 大小為 768GB，所以共有$252 * 768\text{GB} = 193536\text{GB} = 1.935 * 10^8\text{MB} > 10^7\text{MB}$。所以，可以將所有小檔案存在 252 個 node 的記憶體中。

再來討論第二小題，***Is it faster for a compute node to acquire the data of a small file from the memory of another node than from the storage?***

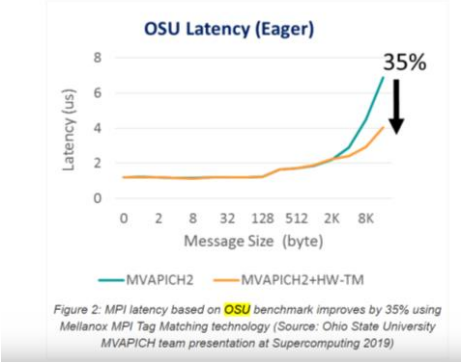


Figure 2: MPI latency based on OSU benchmark improves by 35% using Mellanox MPI Tag Matching technology (Source: Ohio State University MVAPICH team presentation at Supercomputing 2019)

先算 acquire the data of a small file from the memory of another node：

流程如下：

1. A send request to B： 從上圖可看出來大約是1.8μs
2. B search file： 題目說1μs
3. B send back file to A : 1MB/100Gbps = 80μs (EDR infiniband = 100Gbps)。
4. A receive file： 從上圖可看出來大約是1.8μs

Cost time = 1.8+1+80+1.8 = **84.6**μs

再來算 A 直接從 storage 拿資料的 cost time：

由於延續上題表示，A 的 storage 採用 LTO8，因此：

Cost time = 1MB/(360MB/s) = **2778**μs

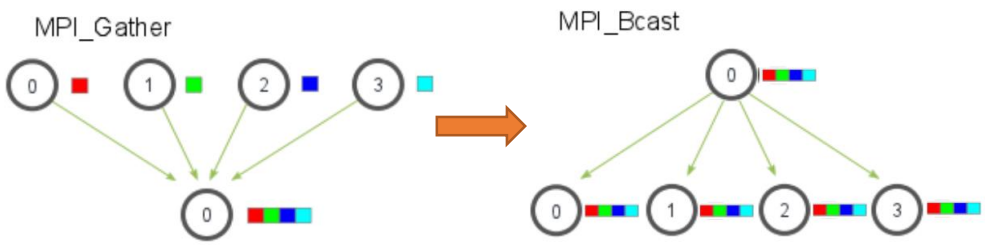
因此從兩者的 Cost time 相差很大，會發現 acquire the data of a small file from the memory of another node 的 cost time 較短，故較快。

5.

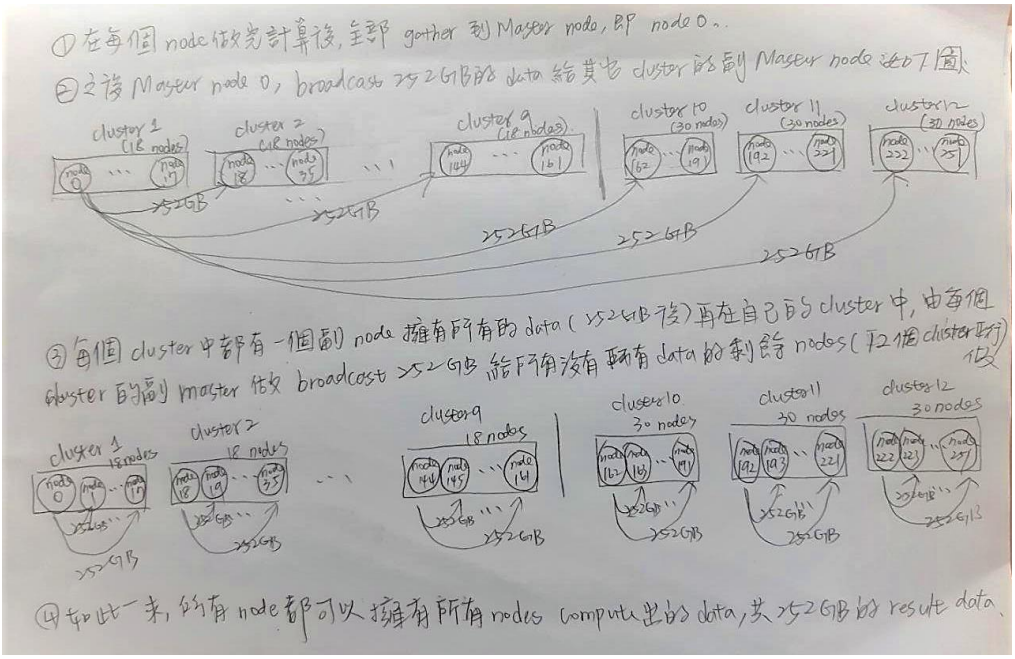
因為在每一次的 iteration 中，我們都要讓每一個人擁有所有人的資料，根據 MPI 的 function，我們有 send, receive, Scatter, Gather, and Allgather, Broadcast and Collective Communication。

最後根據題目要求，先使用 MPI_Gather 讓 Master node 去擁有所有人的資料後。再利用 MPI_Bcast，讓已經擁有所有的 data 的 Master node 將完整的 data broadcast 給剩餘的所有 nodes。另外這樣做並不會產生 dead lock。

簡易的流程如下圖：



然而，設計的演算法要搭配 Taiwania-2 的系統架構圖，在 MPI_Bcast 的時候做分組，以免造成過大的 bottleneck。那為甚麼只有用一個 Master 會造成瓶頸的因，是因為一個 node 的網卡只有 4 張，他的網速是 4000Gpbs，如果只有一個 master node 在做 gather 之後，去同時做 252GB 的 broadcast 給其餘 251 個 node 的話，會出現 252*251/(400Gbps)=1260(sec)的時間，太長。因此在設計演算法的時候，就設計在一開始 gather 給一個主要的 master 之後，此 master 會在根據 Taiwania-2 的拓譜圖去分配給所有的 cluster 中的副 masters，然後再讓這些副 masters 從主要 master 那邊得到的 252GB 的 result data 去 broadcast 給同一個 cluster 中的其他 nodes。示意圖如下圖：



high-level pseudo code for the MPI program：(由於版面，請看下頁)

C:\Users\asce0\Desktop\fat.c • - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

設定 | 20200511-report.txt | 行事曆.txt | 期中考分類 | mpi_mm1.c

```
#include <mpi.h>
int masters[12] = {0, 18, 36, 54, 72, 90, 108, 126, 144, 162, 192, 222}
for(int i = 0; i < 9; i++)
    for(int j = 0; j < 18; j++)
        group1_members[i][j] = i * 18 + j;
for(int i = 0; i < 3; i++)
    for(j = 0; j < 30; j++)
        group2_members[i + 9][j] = i * 30 + j + 180;
//MPI_part
MPI_Init(NULL, NULL);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Group world_group, group0;
MPI_Comm_group(MPI_COMM_WORLD, &world_group);
MPI_Group_incl(world_group, 12, masters, &group0);
MPI_Comm comm0;
MPI_Comm_create_group(MPI_COMM_WORLD, group0, 0, &comm0);
MPI_Group group1[9], group2[3];
MPI_Comm comm1[9], comm2[3];
for(int i = 0; i < 9; i++)
{
    MPI_Group_incl(world_group, 18, group1_members[i], &group1[i]);
    MPI_Comm_create_group(MPI_COMM_WORLD, group1[i], 0, &comm1[i]);
}

for(int i = 0; i < 3; i++)
{
    MPI_Group_incl(world_group, 30, group2_members[i], &group2[i]);
    MPI_Comm_create_group(MPI_COMM_WORLD, group2[i], 0, &comm2[i]);
}
Initialize Result to empty;

    for (int i = 0; i < iter_num; i++)
    {
        data = ReadDataFromStorage();
        result = analysis();
        add result to Result;
    }
    //做法：先全部資料收到Master
```

Line 31, Column 1

C:\Users\asce0\Desktop\fat.c • - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

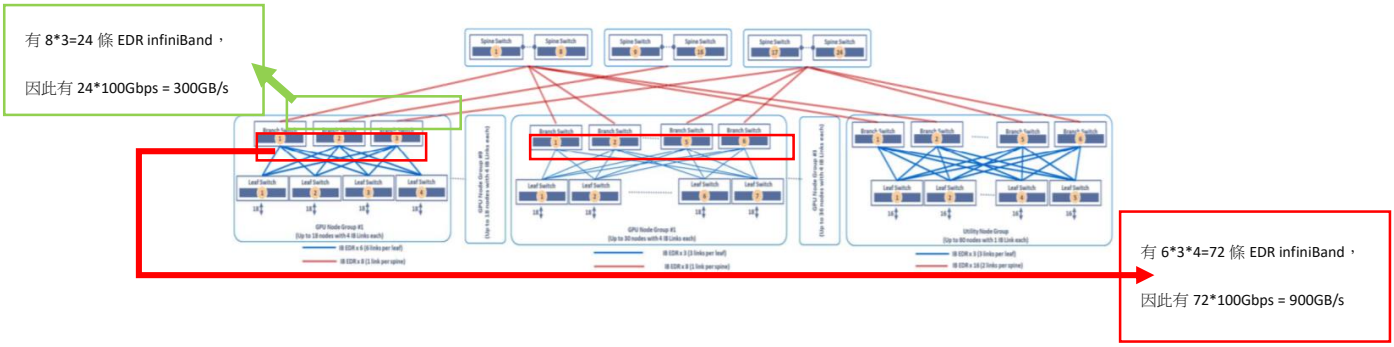
20200511-report.txt | 行事曆.txt | 期中考分類 | mpi_mm1.c | fat.c | fat2.c

```
    for (int i = 0; i < iter_num; i++)
    {
        data = ReadDataFromStorage();
        result = analysis();
        add result to Result;
    }
    //做法：先全部資料收到Master
    MPI_Gather( Result, count, datatype, Gather_Result, count, datatype, 0, MPI_COMM_WORLD);
    //b_cast，然後他再傳給全部人
    if(rank in masters)
    {
        MPI_Comm_rank(group0, &group_rank);
        MPI_Bcast( Gather_Result, count*252, Datatype, 0, comm0);
    }
    for(int i = 0; i < 12; i++)
    {
        if(masters[i] >= rank)
            break;
        group_num = i;
    }
    if(group_num < 9)
    {
        MPI_Comm_rank(comm1[group_num], &group_rank);
        MPI_Bcast( Gather_Result, count*252, Datatype, group_rank, comm1[group_num]);
    }
    else
    {
        MPI_Comm_rank(comm2[group_num - 9], &group_rank);
        MPI_Bcast( Gather_Result, count*252, Datatype, group_rank, comm2[i - 9]);
    }
MPI_finalize();
```

Line 31, Column 1

6.

在計算前 cost time，我們要先尋找 data transfer 的 bottleneck 是在哪邊，分別有：是下圖綠色框框、紅色框框處以及只有四張網卡只有 400Gbps 的網速。



從上面圖來看可以發現，bottleneck 在網卡只有四張的部分。

開始計算時間：

1. 先算 computation(analysis) time：

如題 1sec on 1CPU, can process 100MB data

已知，每個 node 有 36 cores，故 1sec on 1 node, can process 3600MB data

因此每一個 node 要先做 analysis，總共需要：

$$\text{Cost\_time1} = \frac{3.968TB}{36 * 100MB} = 1102.2(sec)$$

2. 再來計算將 data gather 到 master node 的時間，因為 bottleneck 是在四張網卡的傳輸上，因此：

$$\text{Cost\_time2} = 252 * \frac{1GB}{400Gbps} = 5.04(sec)$$

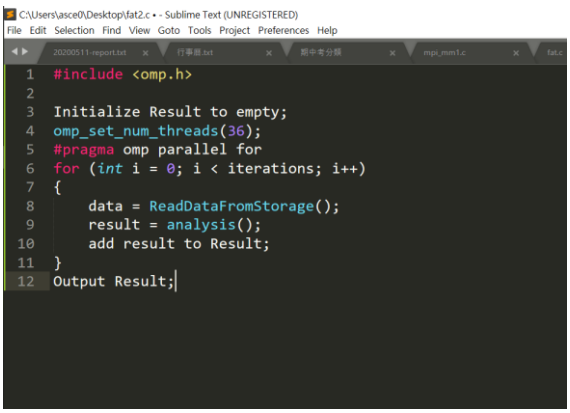
3. 最後計算 master node 將一次 iteration 中所得完整的 result 傳給所有 nodes：

此步驟分成主 master node 將 252GB 分給每個 cluster 的副 master node、和各個副 master node 將剛得到的 252GB 分給同個 cluster(有兩種 cluster)裡的其餘 node 這兩步驟，時間計算如下：

$$\text{Cost\_time3} = 11 * \frac{252GB}{400Gbps} + MAX(17 * \frac{252GB}{400Gbps} + 29 * \frac{252GB}{400Gbps}) = 201.6(\text{s})$$

因此，總 Cost time $= 1102.2 + 5.04 + 201.6 = 1308.84(\text{sec})$

high-level pseudo OpenMP code



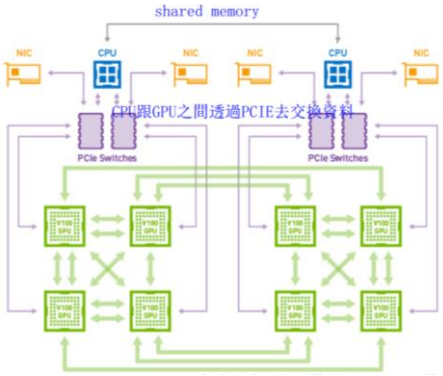
7.

總共要處理的資料量有 1PB，將 1PB 的資料量分割給 252 個 nodes，因此一個 node 大約需處理 3.968TB(= 4TB)的資料。

將 analysis time 分成三個部分，分別是：

1. CPU 將 4TB 的待處理資料傳給 GPU。
2. GPU 計算。(一個 node 有八張 GPU)
3. GPU 回傳 1GB 的 result 給 CPU。

雖然一張 GPU 的容量只有 32GB，然而將各個步驟做分割成獨立事件下去計算，因此不需考慮此問題。



其中從上圖可看出，CPU 和 GPU 互傳是用 PCIe 3.0，而 PCIe 3.0 傳輸速率是 15.8GB/s(by google)，又從上圖中可看出一個 CPU 連結四個 GPU，且每個 PCIe switch 是連出四條 PCIe，故在傳出去的時間要在除以四。

另外計算時間要除以八的原因，是因為總共有八個 GPU。最後回傳 1GB 的 result。因此 cost time 如下：

$$\text{Cost\_time1} = \frac{4TB}{4*\frac{15.8GB}{s}} + \frac{4TB}{8*100MB} * 0.01 + \frac{1GB}{4*\frac{15.8GB}{s}} = 63 + 50 + 0.0158 = 113(sec)$$

接著，承接上題的 communication time：data gather 到 master node 的時間為 5.04(s)，master node 將一次 iteration 所得完整的 result 傳給所有 nodes 需要 $1265.04(s)$。

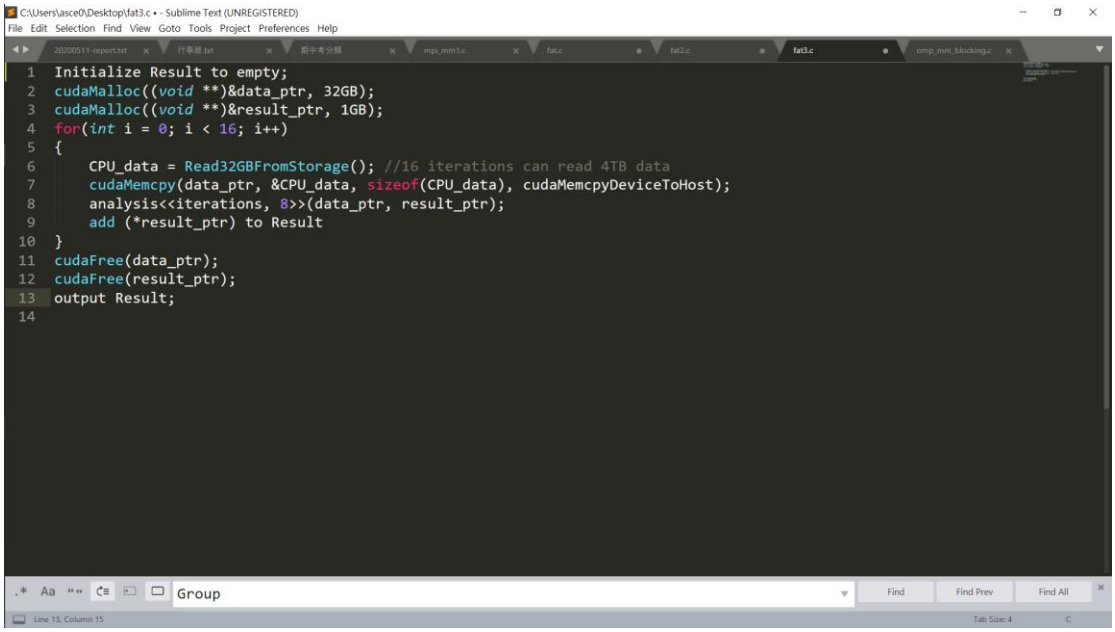
故總 totally_cost_time = 113 +206.64 = 319.64(sec)

因此所求 estimate the maximum speedup if the GPU's are used：

$$\textbf{maximum speedup} = \frac{\mathbf{1308.84}}{319.64} = 4.095$$

8.

A. high-level pseudo CUDA code to carry out the analysis function on the GPUs：



B. Slow down CUDA 的原因：

在前面幾題我們會發現，memory passing 的部分在整個 task 上是佔據最大的時間的部分，且是造成我們第七題的加速無法加速很快的原因。因此說 memory passing 是線在的 bottleneck。

因此從這個地方下去思考，去看 Slow down CUDA 的原因，也有可能是因為 memory copy。「如果要進行 data transfer 的方法可以參考 How to Optimize Data Transfers in CUDA C/C++」(https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/)。

C. Can the time for CPU-GPU data exchange be overlapped with the execution time of the CUDA kernel?

可以，pipeline 如下所示：

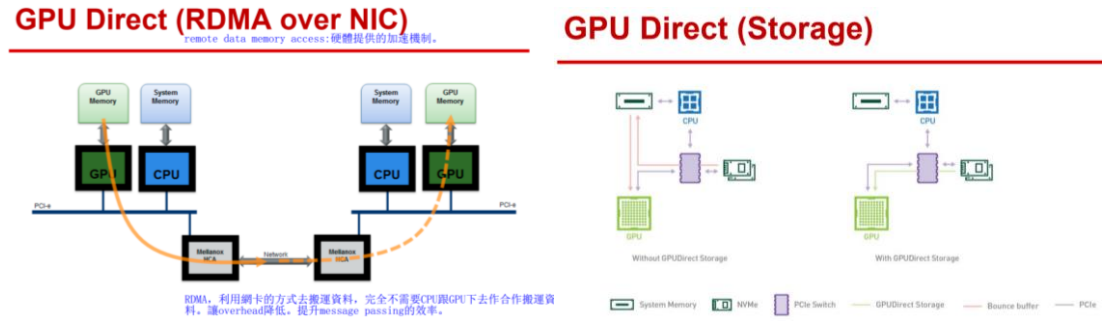
過程							Cost time for one task	Cost time totally
傳送資料 0	傳送資料 1	傳送資料 2	…	傳送資料 251			250(ms)	63(sec)
	計算資料 0	計算資料 1	…	計算資料 250	計算資料 251		198.4(ms)	50(sec)
		接收結果 0	…	接收結果 249	接收結果 250	接收結果 251	0.0627(ms)	0.0158(sec)

從上一題我們可以得到上表格，故在做 overlapped 之後的計算上的 cost time 會變短(原本的 analysis time = 113s)，如下：

新的 analysis time = 63(sec)+198.4(ms)+0.0627(ms) = 63.20(sec)

9.

從上一題可以看出來，在 analysis time 的 bottleneck 是出現在 CPU 傳送資料給 GPU 上面(需要 63(sec))。



從老師的 PPT 的圖中我們可以得知，若使用 RDMA 的技術，除了 GPU 跟 GPU 互傳資料不需要經過 CPU 之外(自上左圖)。另外，在 data transfer 的時候，同一個 GPU 欲取得所需資料就不需要經過 CPU(自上右圖)，故在此 program 中，我們再將資料傳送到 GPU 做計算的時候，並不需要經過 CPU，GPU 可以透過網路直接傳到另一個 GPU 上。

Discuss how this may or may not help improve the performance of this program.

然而，在此題目中，並沒有提到這件事情，因此以下分成兩個狀況去討論：

1. 假設在此 analysis 中，(i+1)-th iteration 會用到 i-th iteration 的結果，也就是每一次的分析會 based on 上一次的結果去做計算。如此一來，RDMA 技術在此時就會派上用場，因為原本的方式是：在 i-th 先在 GPU-1 所計算出的 result，需要回傳到 CPU，之後在從 CPU 中傳到另一顆 GPU 中進行運算。然而，若使用 GPU-direct 的技術(RDMA)，就可以在 i-th 的 iteration 結束後，直接將 GPU-1 的計算 result 直接傳到 (i+1)-th 會用到此份 result 的 GPU。如此一來，在一次的 iteration 中就可以省去很多原本 GPU 要回傳給 CPU，CPU 之後再傳給 GPU 的 data transfer 的時間。因此 RDMA 技術在此狀況下，就會能夠做到加速效能的結果。

2. 假設在此 analysis 的所有 iteration 中，GPU 跟 GPU 之間並沒有要做溝通、data transfer，如此一來 RDMA 技術就不會有加速的效果。

因此根據不同的 analysis function，可能可以加速也可能不能加速，原因如上所述。

10.

從第六題到第八題一路加速可以發現到，現在的 bottleneck 已經不是 compute time，而是 communication time，會出問題的原因是因為一開始在設計演算法的時候，是使用 MPI_Bcast，會造成一些 overhead。接下來希望能夠針對這個 bottlencek 利用硬體做改善。

現在狀況：Computation time = 63.20(s), communication time = 206.64(sec)。

現在我們希望可以利用硬體下去做加速。

在第六題的 cost_time3 我們從 **Taiwania-2 的系統架構圖中**會發現到，**Taiwania-2 的 clusters 有分成兩種，一種是 18 個 nodes 作為一個 cluster 的且共**有 9 組，另一個是 30 個 node 作為 cluster 的共有 3 組。

且，在計算第六題的：

$$\text{Cost\_time3} = \ 11 * \frac{252GB}{400Gbps} + MAX(17 * \frac{252GB}{400Gbps} + 29 * \frac{252GB}{400Gbps}) = 201.6(\text{s})$$

中我們可以發現，在所有的副 master 將 252GB 的 data 傳給同組的其餘 nodes 的時候，因為是平行在傳遞，所以因為 cluster 有不同的分配法。所以會有如上式，有取 MAX 的問題。

另外，我們會發現 cluster 包含的 node 數量越大的時候，會需要更常的 data passing 的時間，因此我們覺得如果檢討# of nodes = 30 的 cluster 的個數，多使用 # of node = 18 的 cluster，如此一來 data passing 的時間就會減小，故先分成下列幾種情況討論：

且由於總共有 252 個 node，只分成下列這兩種 cluster，可能會有沒辦法整除的問題，例如下表中的情況二，先使用了 2 個# of nodes = 30 的 cluster 後，會剩下 252-60=192 個 nodes，要分成# of node = 18 的 cluster，即 192/18=10…12 個。然而，為什麼是取 10 個# of node = 18 的原因，是因為它雖然有餘數等於 12，且這 12 個 node 組成一個 cluster，這個 cluster 在每個副 master node 拿到資料 broadcast 給同個 cluster 的 node 的時間，肯定會比其他的 node 數量較大的 cluster 的短，因此雖然它們存在，但因為並不影響 cost time 的計算，故不列在下表，然而要小心的是在計算 coast time 的第一步，就是 master node 傳給全部的副 master node 的時候的 broadcast 的組別還是要算它。

	18 node cluster 個數	30 node cluster 個數	Cost time
情況一	9	3	201.6(s) // 現在的狀況
情況二	10	2	$12 * \frac{252GB}{\frac{50GB}{s}} + \frac{29 * 252GB}{\frac{50GB}{s}} = 206.64(\text{s})$
情況三	12	1	$13 * \frac{252GB}{\frac{50GB}{s}} + \frac{29 * 252GB}{\frac{50GB}{s}} = 211.68(s)$
情況四	14	0	$13 * \frac{252GB}{\frac{50GB}{s}} + \frac{17 * 252GB}{\frac{50GB}{s}} = 151.2(s)$

算出來的到使用 14 個# of node=18 的 cluster 可以跑最少的秒數。

故將硬體全部換成這一種。且加速如下：

原本：computation time :63.20(s), communication time = 296.64(=5.04+201.6)

後來：computation time :63.20(s), communication time = 156.24(=5.04+151.2)

Speedup = $\frac{296.64}{156.24} = \mathbf{1.90}$。

Based on 這個加速結果，我們可以針對這題的答案先做一個小的結論，欲使用硬體去加速我們的 program，把全部的# of node=30 的 cluster，全部換掉都換成# of node=18 的 cluster，變成總共有 14 個 cluster，且每個 cluster 中有 18 個 nodes 的這個狀況下，會比原本的 **Taiwania-2 的系統架構，針對這題達到更快的速度。雖然這個做法的缺點就是，*會失去硬體的異質性*，因為所有的 cluster 都是用一樣的。然而，不置可否，這樣改變硬體方式的確可加速。**

從上面這個討論，我們可以發現到決定一個 cluster 的個數，的確是可以改善現在的 bottleneck：communication time 的問題。因此更深入的去討論，那如果完全為了配合我們現在的 analysis function 的狀態去修改硬體的配置。應該是可以討論出一個最佳的一個 cluster 的 # of node 應該為多少會可以得到最佳加速。因此接下來開始討論這個。

假設有兩種一樣的 cluster，藉著增多選向去看看有沒有更好的加速結果。

根據原本的條件可以假設有兩種 cluster，分別的 number of nodes = a, b，且共有 x 個 # of nodes = a 的 clusters，和 y 個 # of nodes = b 的 clusters。即：ax+by=252，且 0 < x, y, a, b < 252，並求：

$(\mathrm{x}+\mathrm{y}-1)*\frac{252GB}{\frac{50GB}{s}}+MAX((a-1)*\frac{252GB}{\frac{50GB}{s}}\ ,(b-1)*\frac{252GB}{\frac{50GB}{s}})$的 minimum

為所求。

寫一個很暴力的 C code 如下：

```
#include <stdio.h>
#include <stdlib.h>
float min_ans = 999999;
int main (){
        int l_a, l_b, l_x, l_y;
        float tmp_1, tmp_2, tmp_3, tmp_min;
        for (int x = 0; x < 252; x++){
                for (int y = 0; y < 252; y++){
                        for (int a = 0; a < 252; a++ ){
                                for (int b = 0; b < 252; b++){
                                        tmp_1 = 252/50*(a-1);
                                        tmp_2 = 252/50*(b-1);
                                        tmp_3 = tmp_1;
                                        if(tmp_2 > tmp_1)
                                                tmp_3 = tmp_2;
                                        tmp_min = (x+y-1)*252/50 + tmp_3;
                                        if(tmp_min < min_ans && a*x+b*y >= 252){
                                                min_ans = tmp_min;
                                                l_a = a;
                                                l_y = y;
                                                l_b = b;
                                                l_x = x;        }}}}}
        printf("min cost time = %f\n", min_ans);
        printf("having %d node number = %d's cluster\n",l_x, l_a);
        printf("having %d node number = %d's cluster\n",l_y, l_b);

        return 0;
}
~
"hua.c" [dos] 28L, 773C                                       18,24-66
```

最後跑出的結果如下：

```
b05611047@linux7 [~] vim hua.c
b05611047@linux7 [~] gcc hua.c
b05611047@linux7 [~] ./a.out
min cost time = 150.000000
having 0 node number = 0's cluster
having 14 node number = 18's cluster
b05611047@linux7 [~]
```

寫成 code，去考慮可以擁有兩種 clusters 的的所有狀況後，最後得到最快的時間是 cost time = 150(sec)。

從結果看出來，總共有 18 個 cluster，且每個 cluster 中有 14 個 node 是最快的速度。然而將此答案帶入公式之後，我們會發現一件很有趣的事情。在上

面證明此方法可以做加速的時候，我們是先發現全部使用一種 Cluster，且此每個 cluster 有 18 個 node，總共有 14 個 clusters 的時候，會是最快的。而在 run code 之後得到的結果是，也是全部使用一種 Cluster，且此每個 cluster 有 14 個 node，總共有 18 個 clusters 的時候，會是最快的。

然而，這個結果我們帶入上面的公式的時候會發現，這其實是一個一體兩面的結果，有點類似說這兩種狀況其實只是在講同一件事情。真的是一個很大的巧合 QQ 所以基於這種狀況，我又另外下去 run 了三層的 code，也就是上面原本只考慮可以有兩種 cluster 的情況改成可以有三種不同的 cluster 的情況，希望可以藉由增多 cluster 的種類，讓秒數降的更低。但，在讓程式跑完之後，我還是獲得了同樣概念的結果。只是以不同加乘倍率的方式分配給三種 cluster，但秒數仍然是一樣的沒有下降。因此我猜測，這個組合或許有機會已經是使用這種方式去加速的最佳解了。因此使用此方法已經沒辦法再把秒數降的更低。即使用此方法的加速已經達到極值，故不再討論使用 4 種、甚至 5 種…等更多種的 clusters 數量去討論。

因此來討論一下結。，原本的 communication time 從原本的 206.64(sec)變成 155.04(= 150+5.04(gather 的時間))。故在 communication time 加上 compute time = 63.20(s)，總時間是=218.24(sec)。

結論是，將硬體設備 cluster 的方式改成共有 18 個 cluster，每個 cluster 有 14 個 node 的時候，會是讓 communication time 最快的 cluster 的分配擺法。

最後針對每一題有加速的題目，將 cost time 整理成一個表格如下：

題目	Compute time(sec)	Communication time(sec)	Totally cost time(sec)
第六題	1102.2	206.64	1308.84
第七題(用 GPU 算)	113	206.64	319.64
第八題(overlap)	63.20	206.64	269.84
第十題(改變硬體加速)	63.20	155.04	218.24

以上都是針對 communication time 去改變硬體之後做的加速結果。

另外可以針對 compute time 去做加速的話，大概也只想到兩種方法，第一種是，因為在 GPU 在計算的時候，所有 CPU 都是閒置的，或許我們可以把原本比較高級的 CPU 賣掉，改買更多的 GPU 插上去加強算力，看看能不能去減少 compute time。這或許是一種可以減少 compute time 的做法。

最後檢討一下為甚麼加速沒有辦法在做的更快的原因，是因為一開始在設計演算法的時候，雖然有考慮到因為只有 4 張網卡的 bottleneck 了。然而，最後瓶頸仍然出現在這裡的原因是因為，在這題中，不管怎麼改變硬體結構，也就是重新設計 cluster 的放法或者分配，終究會回到一個問題，在每次的 broadcast 一定都要同時傳出很多的 data，就還是會遇到 4 張網卡的 bottleneck，假若要在加速並可以修改軟體的話，第一個想到的是改變 broadcast 的層數，現在是使用兩層的 broadcast，說不定可以再多加一層的 broadcast，達

到加速，這是第一種方法。

另外一個是在 MPI 的 function 中看到的，或許可以不要使用 gather 跟 broadcast，而是改用 allgather 的 function，並且將傳資料的方式設計成環狀的，不要 sequential 的方式下去傳，也就是改成做多次的 Iteration，但每次傳出去的較小的資料量這樣，或許也是一個可以試試看的方法。

Ref：

1.老師的 PPT

2.NCHC 官網 https://www.nchc.org.tw/

3.網路傳輸方式：https://brainly.in/question/9867137、https://www.quora.com/How-does-data-transfer-through-a-network

4. 國網中心介紹：http://esrpc.ncu.edu.tw/public/file/downloads/newer/107/1070908_01_%E5%9C%8B%E7%B6%B2%E4%B8%AD%E5%BF%83%E4%BB%8B%E7%B4%B9.pdf

5.PCIe3.0(x16) https://zh.wikipedia.org/wiki/PCI_Express

6. CUDA 相關：https://kheresy.wordpress.com/2007/11/08/nvidia-cuda-api%EF%BC%88%E4%B8%8B%EF%BC%89/

7.RDMA 技術：http://www.ipshop.xyz/3309.html

8. MPI function : https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/、https://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/

9. 台灣杉二號的一些硬體設備介紹：https://www.top500.org/system/179590