hw1_report

1. Title, name, student ID

110590007 白宸安

2. Implementation

1. 讀取argv
2. 計算send_count[i] : 每個processor要存有幾筆資料，如果有remainder就從rank 0開始一路分下去。
3. 計算每個人讀取資料時的位移量。
4. 平行讀檔，用MPI_File_read_at_all。
5. 將每個processor自己的資料先做local sort，把資料排序好。這邊用的是qsort。
6. 計算每個人的partner，確認自己是要跟誰交換。
7. odd even sort階段，

   當even phase時，兩者使用MPI_Sendrecv交換資料，even rank進行merge_low()，odd rank進行merge_high()。

   當odd phase時，兩者使用MPI_Sendrecv交換資料，even rank進行merge_high()，odd rank進行merge_low()。

   另外有設定early termination條件，利用local data都已經sort好的特性。

   當merge_low()前，如果左邊local data最後一個 < 右邊的第一個，則其實不用交換。

   當merge_high()前，如果左邊local data第一個 > 右邊的最後一個，則其實不用交換。

8. 最後平行輸出output，每個人根據自己當初資料讀進來時的位移量去寫資料，這樣就可以平行各自寫上去。

merge_low() :

吃進自己local data跟partner data，然後進行merge，每次比最前面的元素，然後更新index。只要merge出前半部小的data就可以暫停了，最後透過指標轉移來將merge好的資料送給local data。

merge_high() :

吃進自己local data跟partner data，然後進行merge，每次比最後面的元素，然後更新index。只要merge出後半部大的data就可以暫停了，最後透過指標轉移來將merge好的資料送給local data。

3. Experiment & Analysis

i. Methodology

(a) system spec

使用的設備是Apollo機，不另外描述spec。

(b) Performance Metrics:

使用ipm來測量。

根據測量到的結果來繪製圖表。

ii. Plots: Speedup Factor & Profile

Experimental Method:

test case:

我選用的是第35筆測資，因為考量到它資料量大的特性，data size為536869888。

Parallel Configurations :

分別使用以下進行實驗：

1. 多node環境：

1 node 1 proc

1 node 12 proc

2 node 24 proc

4 node 48 proc

2. 單node環境：

1 node 1 proc

1 node 4 proc

1 node 8 proc

1 node 12 proc

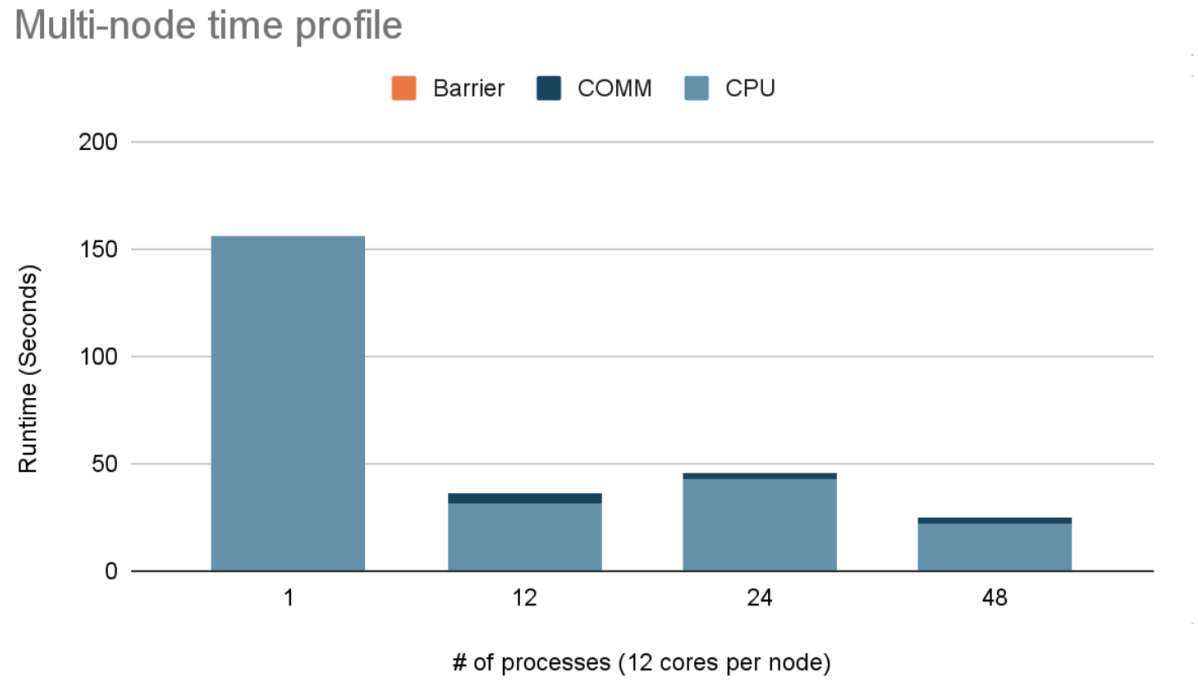
Performance Measurement:

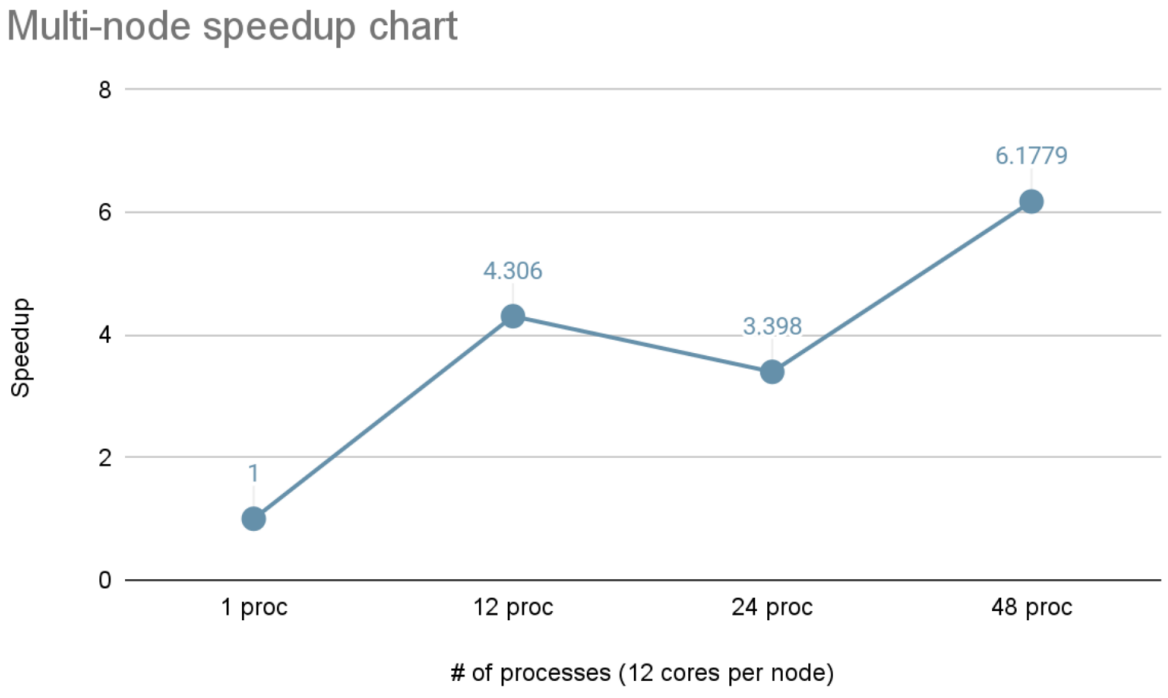
使用ipm進行。

使用CPU time（非MPI所使用的執行時間）、COMM（MPI溝通產生的時間）、Barrier（MPI_Barrier的時間）當作Metrics。

Analysis of Results:

Multi-node環境





以上為多nodes環境的time profile和Speedup圖。

從圖1中可以看到，對執行時間造成主要影響的是CPU的執行時間，而並非MPI溝通所花費之時間。我認為這應該和程式在設計上有相關，我的程式似乎在process內部自己計算時效率不好，才會導致整體時間其實是被內部計算給bound住。

從圖2中可以看見，大致上使用越多processes就能讓速度變更快。其中12 proc這個我覺得比較特別，照理來說只用12 proc不應該筆24 proc來得慢才對，目前想到的可能

原因是切割資料時剛好些到某些不好的分區，而local sort使用的是qsort，可能會因為分割不均導致時間變慢。另一個想到的原因是可能單純在profile時伺服器比較忙導致的誤差。

4634175

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

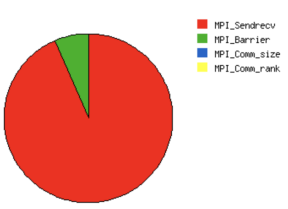
powered by IPM

command: /home/pp23/pp23s46/homework1/./hw1 536869888 /home/pp23/pp23s46/hw1/testcases/35.in 35_2node_12prc_file.log			
codename:	unknown	state:	unknown
username:	pp23s46	group:	
host:	apollo35 (x86_64_Linux)	mpi_tasks:	24 on 2 hosts
start:	10/25/23/20:36:08	wallclock:	4.57576e+01 sec
stop:	10/25/23/20:36:54	%comm:	5.67563639701383
total memory:	10.0254 gbytes	total gflop/sec:	0
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation		
Event	Count	Pop

Communication

% of MPI Time



HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)

Communication Event Statistics (100.00% detail, 1.6377e−04 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Sendrecv	16777216	576	58.319	8.478e−02	4.059e−01	93.57	5.31
MPI_Barrier	0	48	4.010	3.500e−04	4.083e−01	6.43	0.37

4634558

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

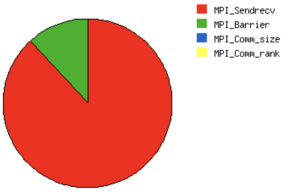
powered by IPM

command: /home/pp23/pp23s46/homework1/./hw1 536869888 /home/pp23/pp23s46/hw1/testcases/35.in 35_4node_48prc_file.log			
codename:	unknown	state:	unknown
username:	pp23s46	group:	
host:	apollo43 (x86_64_Linux)	mpi_tasks:	48 on 4 hosts
start:	10/25/23/20:47:58	wallclock:	2.51714e+01 sec
stop:	10/25/23/20:48:23	%comm:	12.1077418220679
total memory:	14.12952 gbytes	total gflop/sec:	0
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation		
Event	Count	Pop

Communication

% of MPI Time



HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)

Communication Event Statistics (100.00% detail, 6.4943e−05 error)

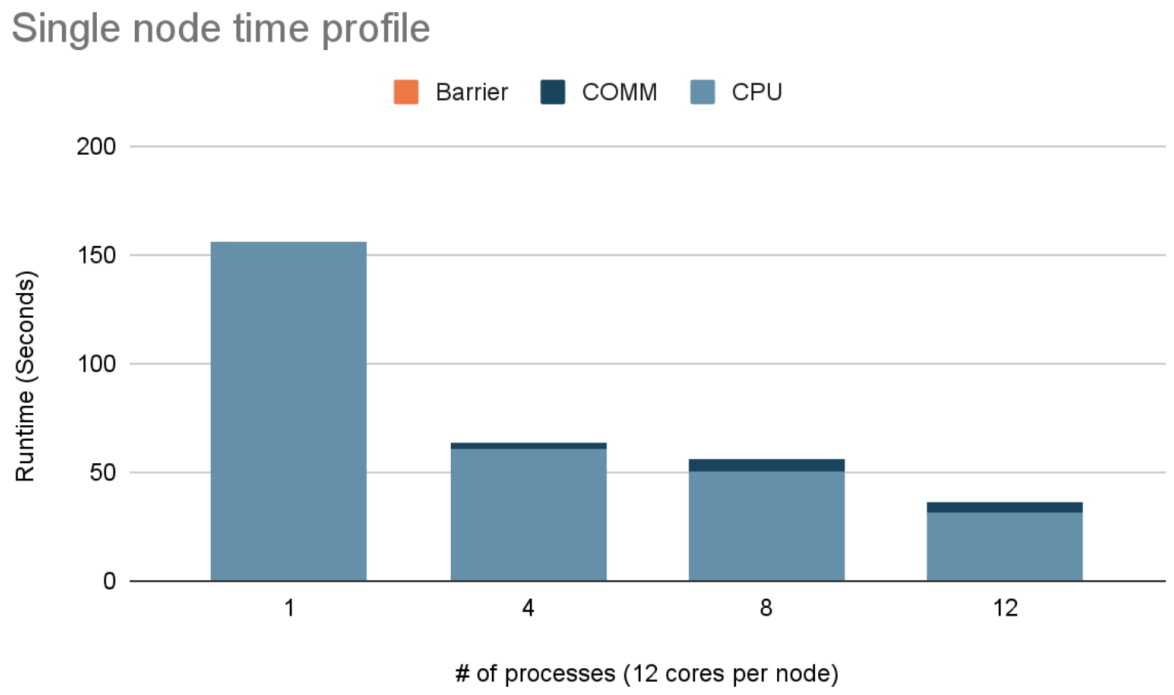
	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Sendrecv	41943040	2304	129.029	3.578e−02	6.641e−01	88.20	10.68
MPI_Barrier	0	96	17.260	4.098e−04	7.449e−01	11.80	1.43

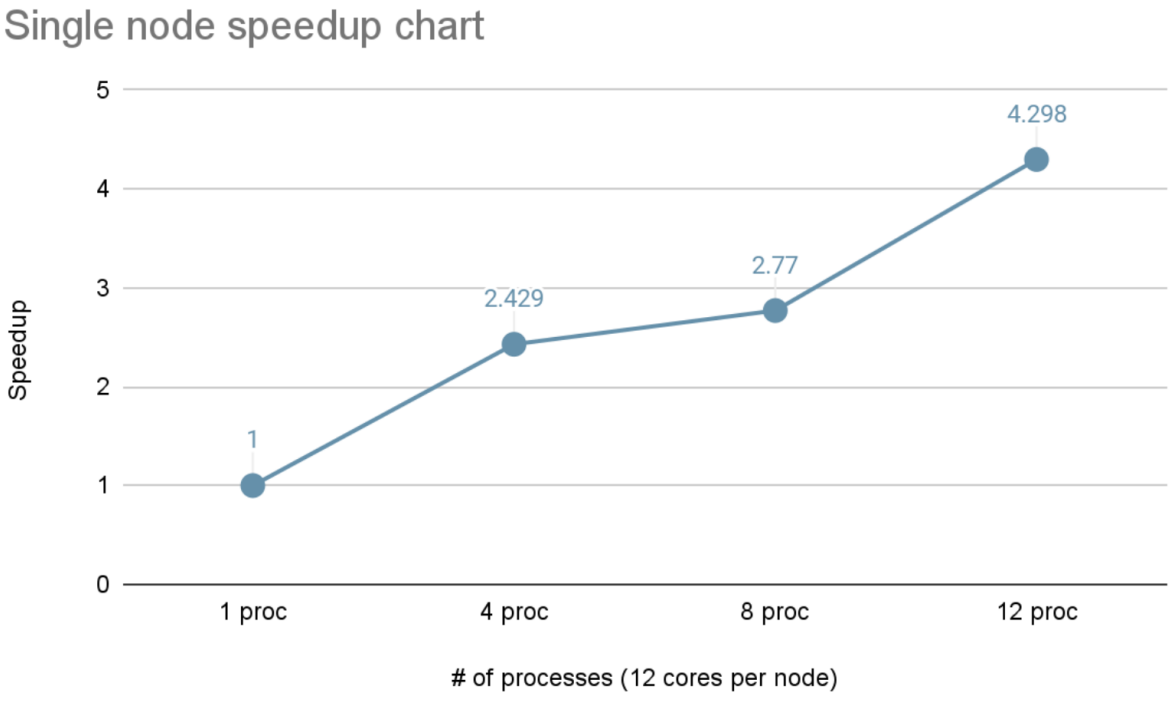
以上拿2 nodes 24 proc和4 nodes 48 porc的ipm結果為例，可以看出隨著process數量變多，MPI花費在Sendrecv和Barrier的時間佔比就變多，雖然更多processes仍然能讓程式執行得更快，但其中造成的communication overhead也不可忽視。

而從這也大致能看出這支程式在MPI方面的bottleneck在communication上，但我認為這支程式真正的bottleneck並不是在MPI溝通上，而是在local端。

另外，IO的部分不知道為啥ipm沒有看到它profile IO的時間（例如MPI_File_read_at_all等），所以無法得知確切資訊，但在實驗過程中我認為，IO似乎並不是一個bottleneck點，似乎對執行時間沒有太大的影響。

Single node環境





從上面圖片發現，single node情況下它平行的特徵比較明顯，比如scalability、speedup等。

4635280

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

powered by IPM

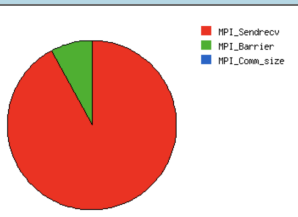
command: /home/pp23/pp23s46/homework1/./hw1 536869888 /home/pp23/pp23s46/hw1/testcases/35.in 35_1node_4prc_file.log			
codename:	unknown	state:	unknown
username:	pp23s46	group:	
host:	apollo36 (x86_64_Linux)	mpi_tasks:	4 on 1 hosts
start:	10/25/23/21:28:34	wallclock:	6.40088e+01 sec
stop:	10/25/23/21:29:38	%comm:	5.11774162302684
total memory:	6.47496 gbytes	total gflop/sec:	0
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation

Event	Count	Pop

Communication

% of MPI Time



HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)

Communication Event Statistics (100.00% detail, 3.1772e-05 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Sendrecv	469762048	16	12.065	1.150e+00	1.553e+00	92.07	4.71
MPI_Barrier	0	8	1.039	1.407e-05	4.594e-01	7.93	0.41

4635557

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

powered by IPM

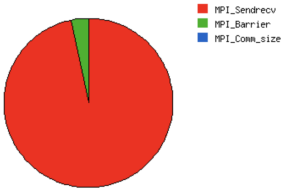
command: /home/pp23/pp23s46/homework1/./hw1 536869888 /home/pp23/pp23s46/hw1/testcases/35.in out_file.log			
codename:	unknown	state:	unknown
username:	pp23s46	group:	
host:	apollo41 (x86_64_Linux)	mpi_tasks:	8 on 1 hosts
start:	10/25/23/21:34:16	wallclock:	5.61277e+01 sec
stop:	10/25/23/21:35:12	%comm:	9.42188081820563
total memory:	7.080016 gbytes	total gflop/sec:	0
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation

Event	Count	Pop

Communication

% of MPI Time



HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)

Communication Event Statistics (100.00% detail, 1.8706e-04 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Sendrecv	234881024	64	40.950	6.453e-01	3.700e+00	96.79	9.12
MPI_Barrier	0	16	1.356	2.542e-04	3.701e-01	3.21	0.30

另外，以1 node 4 proc和1 node 8 proc的ipm profile來看，可以發現在proc數增加時，MPI_Sendrecv有使用掉更多的時間比例，代表了隨者process的數量變多，溝通的成本也變得更大。

Optimization Strategies:

我這邊優化的metric使用hw1-judge的時間當作基準。

{40 249.69} --> {40 235.46}
從MPI_File_write_at()改成 MPI_File_write_at_all()

{40 235.46} --> {40 230.85}

for(j=0; j<local_n_of_a; j++) local_data[j] = merge[j];
memcpy(local_data, merge, (local_n_of_a) * sizeof(float));

改成用memcpy的方式來複製資料給local_data

{40 221.56} --> {40 193.99}
改成merge_low和merge_high
拔掉MPI_Barrier(MPI_COMM_WORLD);

{40 193.99} --> {40 188.64}
merge_low和merge_high中改用指標交換的方式

iii. Discussion (Must base on the results in your plots)

根據以上圖片來看，有發現大致上使用越多processes，程式會執行的越快，但變快的效率並沒有特別高，例如使用48 proc才變快約6倍而已，這其中包含了MPI溝通、初始化、Barrier的時間等，確實是會有一些overhead的產生，但我認為真正的bottleneck應該在這支程式處理local端的時候，或許是local sort選用的不好，也可能是merge時其實有更快的方法，或是有更好的early termination條件等，這些都能夠解決這支程式的bottleneck。

程式的scale效果並沒有預期的好，原因我想也是因為local端的bottleneck，因為Sendrecv是blocking的傳輸，所以只要有一個process執行較慢其他人就必須等它，這樣會造成很多時間的浪費。

4. Experiences / Conclusion

總結

這次的作業很有趣，讓我更了解MPI的用途、實作細節，也知道了更多平行程式實作上的經驗，

此外，我也覺得做實驗測試的部分也十分有趣，讓我學會了程式profile的方法和製圖、分析等。

開發上遇到的最大困難是不知道怎麼讓程式再變快，最終結果的排名有點後面，不知道其他人是用什麼方式來寫的。過程中嘗試了很多方法，雖然有讓程式從240多秒降到188秒左右，但就整體而言還是希望能做得再更好一些。

期待下次作業能做得更好。