Parallel Programming Homework 4-2

Blocked All-Pairs Shortest Path (Multi-GPUs)

CS 107062546 楊仲愷

1. Implementation

(1) How do you divide your data? What's your configuration?

a. $\left\lfloor \frac{|Vertex|}{32} \right\rfloor$

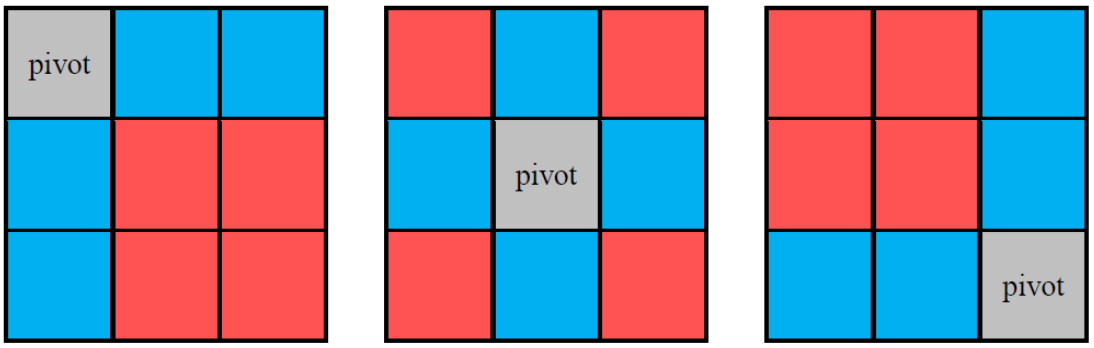
b. Blocking Factor: 32

c. Blocks: $(\frac{data\ \ amount}{32})^2$

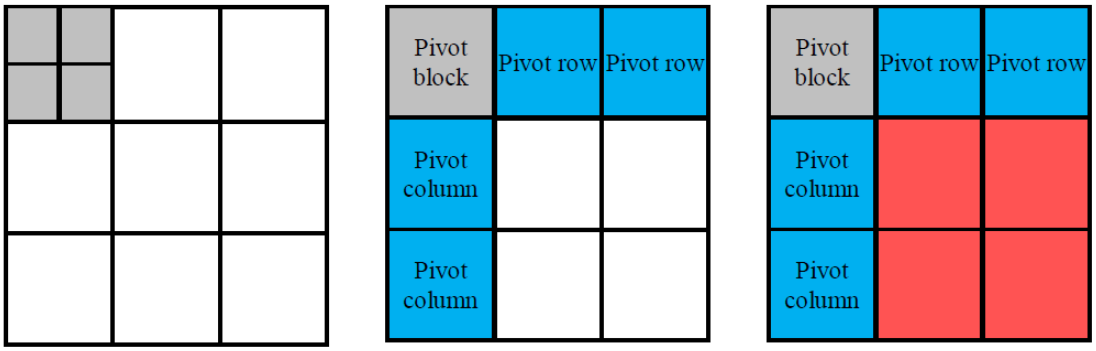
d. Threads: 32×32

(2) Briefly describe your impletion.

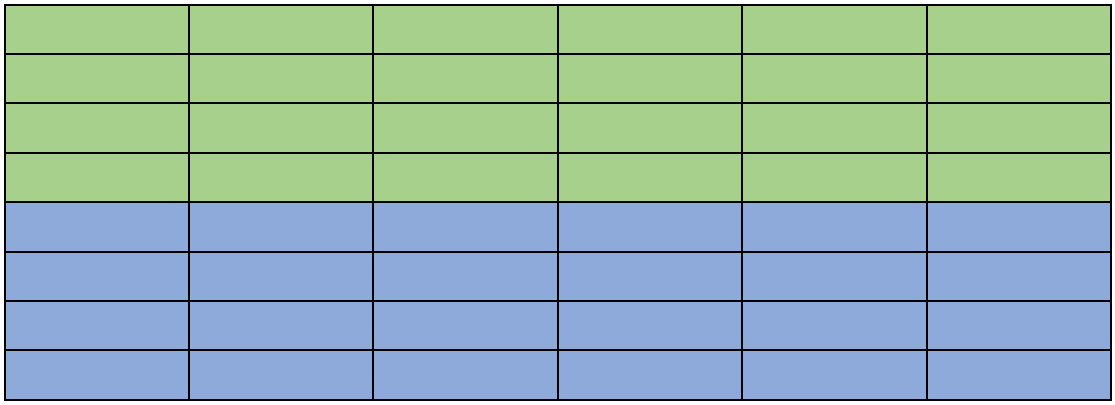
Based on the algorithm (i.e., blocked APSP algorithm),



(a) Round 1 (b) Round 2 (c) Round 3

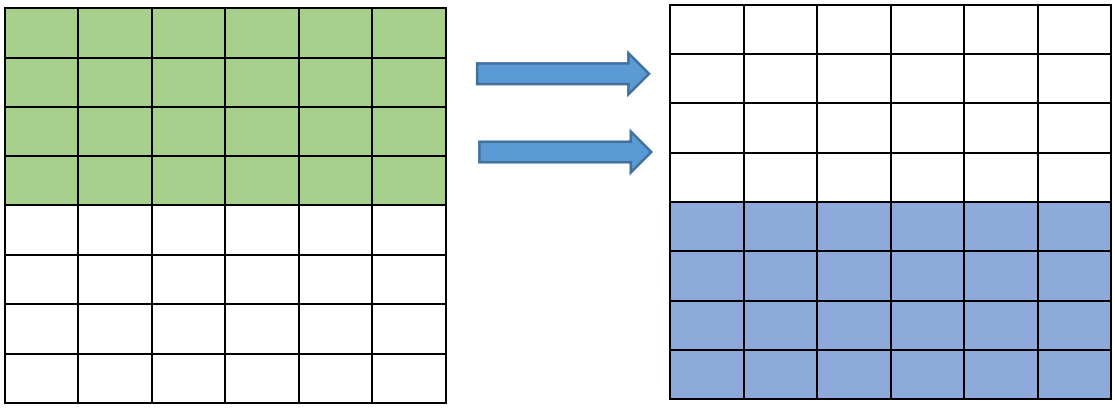


(a) Phase 1 (b) Phase 2 (c) Phase 3

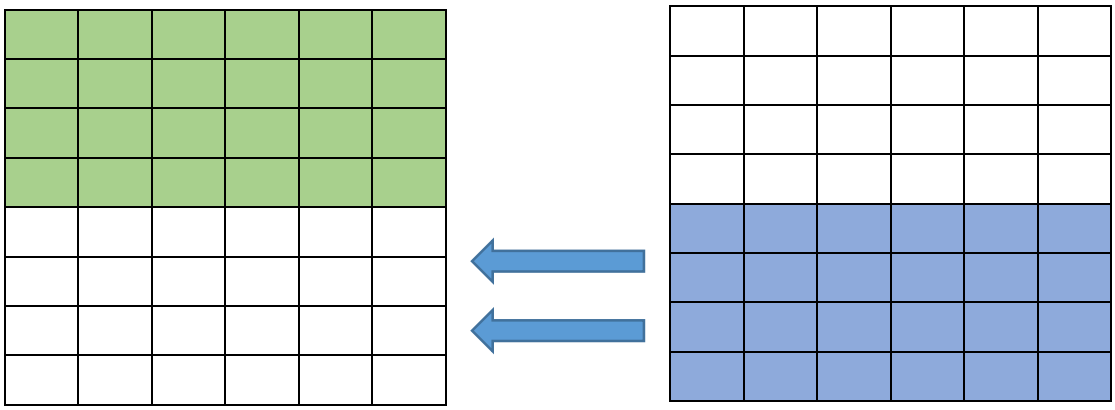


Firstly, I divide one image into two parts (i.e., green one, and blue one) for distributing them into two GPU cards. From the figure described below, thread 0 will count the green part, and thread 1 will be responsible for blue part. Also, device 0 will calculate the green part, and device 1 will be responsible for blue part.

```
if(thread_num == 0){
    phase1<<<grid1, blk, B*B*sizeof(int)>>>(r, n, d_Dist_0, B);
    phase2<<<grid2, blk, 2*B*B*sizeof(int)>>>(r, n, d_Dist_0, B);
    phase3<<<grid3, blk, 2*B*B*sizeof(int)>>>(r, n, d_Dist_0, B, round, thread_num);
}else if(thread_num == 1 && n > B){
    phase1<<<grid1, blk, B*B*sizeof(int)>>>(r, n, d_Dist_1, B);
    phase2<<<grid2, blk, 2*B*B*sizeof(int)>>>(r, n, d_Dist_1, B);
    phase3<<<grid3, blk, 2*B*B*sizeof(int)>>>(r, n, d_Dist_1, B, round, thread_num);
}
```



Before starting to execute each round, we have to transmit the data it needed which is only the pivot row. The reason for only the pivot row to be passed is the pivot column has been counted and update by itself. As a result, when we are calculating the first four rounds, device 1 needs the pivot row data from device 0, and each round will pass the pivot row of device 0 to device 1.



When it comes to the last four rounds, device 0 needs the pivot row data from device 1, so for each round, device 1 has to pass the pivot row data to device.

2. Profiling Results

I take as p20k1 as an example:

- **Single GPU**

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	96.80%	22.7385s	625	36.382ms	35.724ms	36.932ms	phase3(int, int, int, int*, int)
	1.32%	311.06ms	3	103.69ms	544ns	311.06ms	[CUDA memcpy HtoD]
	1.26%	295.86ms	1	295.86ms	295.86ms	295.86ms	[CUDA memcpy DtoH]
	0.60%	141.50ms	625	226.39us	220.71us	232.48us	phase2(int, int, int, int*, int)
	0.01%	3.3201ms	625	5.3120us	5.1200us	5.7600us	phase1(int, int, int, int*, int)
API calls:	55.41%	13.0905s	4	3.27263s	4.9370us	12.7793s	cudaMemcpy
	44.02%	10.3987s	1875	5.5460ms	3.1660us	36.931ms	cudaLaunch
	0.56%	131.70ms	3	43.901ms	4.5260us	131.57ms	cudaMalloc
	0.00%	1.1130ms	188	5.9200us	162ns	257.19us	cuDeviceGetAttribute
	0.00%	922.00us	9375	98ns	80ns	3.9310us	cudaSetupArgument
	0.00%	355.05us	2	177.52us	161.74us	193.30us	cuDeviceTotalMem
	0.00%	236.74us	1875	126ns	104ns	483ns	cudaConfigureCall
	0.00%	109.73us	2	54.863us	48.598us	61.128us	cuDeviceGetName
	0.00%	2.3690us	3	789ns	221ns	1.8450us	cuDeviceGetCount
	0.00%	1.3650us	4	341ns	199ns	699ns	cuDeviceGet

- **Multiple GPUs**

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	92.57%	21.9546s	1250	17.564ms	17.210ms	17.840ms	phase3(int, int, int*, int, int, int)
	3.79%	898.00ms	629	1.4277ms	736ns	359.95ms	[CUDA memcpy HtoD]
	2.49%	591.00ms	627	942.58us	403.49us	164.44ms	[CUDA memcpy DtoH]
	1.12%	266.03ms	1250	212.82us	206.85us	217.63us	phase2(int, int, int*, int)
	0.03%	6.3151ms	1250	5.0520us	4.8640us	5.4400us	phase1(int, int, int*, int)
API calls:	56.02%	8.81260s	625	14.100ms	18.338us	2.70658s	cudaMemcpyPeer
	40.48%	6.36787s	6	1.06131s	10.206us	2.86280s	cudaMemcpy
	3.30%	518.83ms	4	129.71ms	251.54us	257.93ms	cudaMalloc
	0.16%	24.802ms	3750	6.6130us	3.7880us	452.10us	cudaLaunch
	0.03%	4.2039ms	17500	240ns	82ns	16.979us	cudaSetupArgument
	0.01%	1.4805ms	3750	394ns	116ns	16.292us	cudaConfigureCall
	0.01%	1.1289ms	188	6.0040us	162ns	251.97us	cuDeviceGetAttribute
	0.00%	328.35us	2	164.18us	163.35us	165.00us	cuDeviceTotalMem
	0.00%	108.69us	2	54.347us	50.423us	58.271us	cuDeviceGetName
	0.00%	22.032us	2	11.016us	4.9130us	17.119us	cudaSetDevice
	0.00%	2.0080us	3	669ns	198ns	1.5710us	cuDeviceGetCount
	0.00%	1.3330us	4	333ns	174ns	759ns	cuDeviceGet

Discussion: From these two figures described above, we can see when there are in phase 3, multiple GPUs takes less time than single GPU (note: when it comes to the column of time, it has to be divided 2 because nvprof will add the time from multiple GPUs); hence, the computing time will also decrease. On the other hand, the decreased time will also make communication time (i.e., CudaMemoryPeer) longer.

3. Experiment & Analysis

(1) System Spec

Run on the hades server.

(2) Time Distribution

Test cases: <vector, edge>

pp11k1: <11000, 505586>

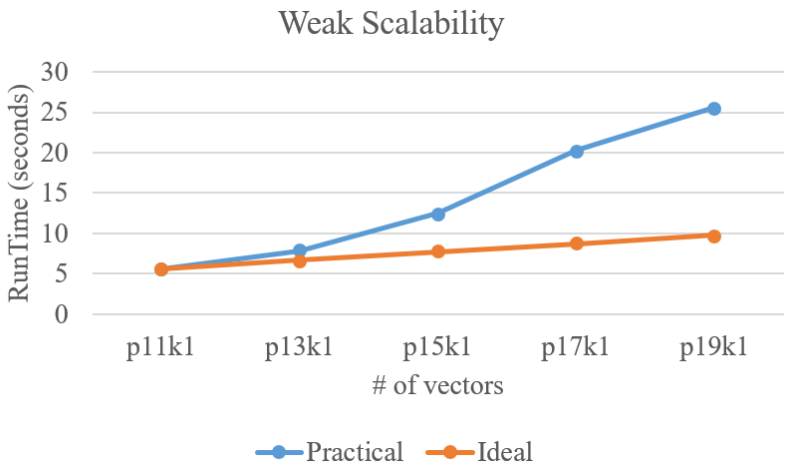
pp13k1: <13000, 1829967>

pp15k1: <15000, 5591272>

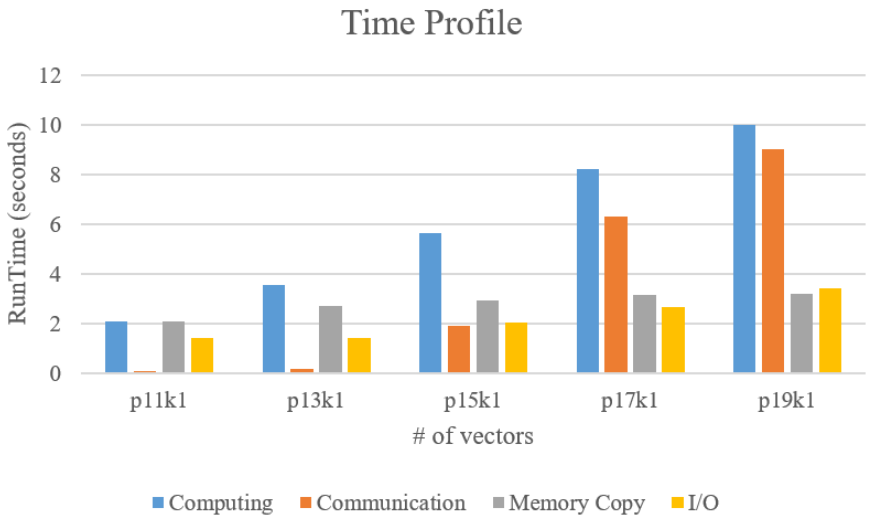
pp17k1: <17000, 4326829>

pp19k1: <19000, 333397>

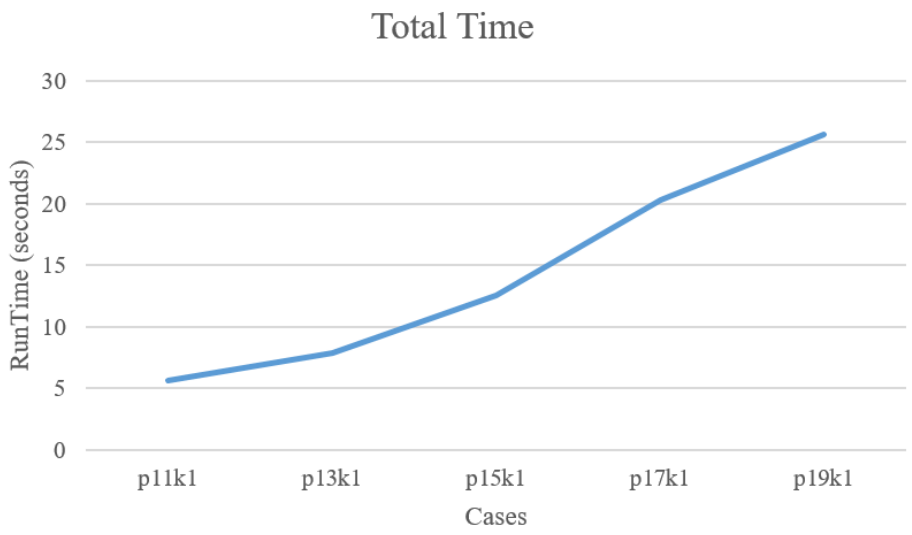
a. Weak Scalability



b. Time Distribution



c. Total Time



d. Value

	Computing	Communication	Memory Copy	I/O	Total Time
p11k1	2.071216	0.062143	2.09842	1.42345	5.655229
p13k1	3.541224	0.181143	2.690115	1.42275	7.835232
p15k1	5.645134	1.891214	2.941395	2.04131	12.519053
p17k1	8.192465	6.31012	3.143489	2.66724	20.313314
p19k1	9.984315	9.002145	3.200751	3.42165	25.608861

Discussion: From weak scalability (i.e., figure (a)), the practical time is longer than the ideal time we predict. I think the reason is the overhead of communication between two GPU cards. In figure (b) Time Distribution, we can see the computing time will grow along with the data amount, so as the time for communication and memory copy and I/O. However, computing time and communication time is affected more than the other two by the data quantity from different cases.

4. Conclusion

In this homework, based on the knowledge of single GPU (i.e., homework 4-1), I have already practiced some skills to improve the performance on single card; however, when it comes to multiple GPU cards, it is a little challenging to think about the separation of multiple cards, and I also have to consider about the communication between them. Hence, if I only pass the whole vector matrix to each other, the communication overhead will become tremendous, the cost cannot be ignored, and the performance will even be slower than single GPU. Consequently, form this experience, it is important to reduce the data quantity. Because of this, it is necessary to only pass the data (i.e., the data dependency parts) which is also needed in another GPU cards. Therefore, communication is one of the import points

of this homework.