Ruofei Xu
11237005
Cpts483 concurrent programmming
Project 2 elrang ring

Comment file

The method I use is one process that creates all of the processes. I used same idea as the code in our lecture. The main thread create hole ring. When the ring finish it's job, it will send a message to notify main thread it's done. For each ring function, I use counter to track if all message are reached destination. If all done, then send a done to main thread and timer will stop after main thread receive the message. Detail explanation is in my code file.

Ring1 function:
The first node can send a message and wait for it to arrive before sending the next message.
I create a loop keep sending message when it received last one, if the message list is empty then the ring is finished.

Test table:
Time unit is millisecond

	0 messages	1000 messages	2000 messages	4000 messages	8000 messages
1000 processors	0 cpu 0 wall-clock	1000 cpu 1075 wall-clock	1960 cpu 2048 wall-clock	4400 cpu 5736 wall-clock	8800 cpu 11517 wall-clock
2000 processors	0 cpu 0 wall-clock	2840 cpu 3555 wall-clock	5600 cpu 6982 wall-clock	10970 cpu 13724 wall-clock	22090 cpu 27609 wall-clock
4000 processors	0 cpu 0 wall-clock	6560 cpu 8038 wall-clock	13010 cpu 15944 wall-clock	25990 cpu 31900 wall-clock	51590 cpu 63226 wall-clock
8000 processors	0 cpu 0 wall-clock	12570 cpu 13210 wall-clock	26100 cpu 31800 wall-clock	52210 cpu 63870 wall-clock	104620 cpu 144239 wall-clock

The time expense is close to linearly increasing as the number of messages or the number of processors increased.

Ring2 function:

The first node can send all of the messages then wait for them all to arrive. I create a counter function to track the number of messages that finished the loop. If that number reached number of message we sent, then it's done.

Test table:
Time unit is millisecond

	0 messages	1000 messages	2000 messages	4000 messages	8000 messages
1000 processors	0 cpu 0 wall-clock	490 cpu 446 wall-clock	940 cpu 599 wall-clock	1950 cpu 1251 wall-clock	4190 cpu 2953 wall-clock
2000 processors	0 cpu 0 wall-clock	930 cpu 554 wall-clock	1910 cpu 1197 wall-clock	3910 cpu 2305 wall-clock	8200 cpu 4850 wall-clock
4000 processors	0 cpu 0 wall-clock	1900 cpu 1268 wall-clock	3830 cpu 2214 wall-clock	8120 cpu 6057 wall-clock	16680 cpu 9959 wall-clock
8000 processors	0 cpu 0 wall-clock	3810 cpu 2749 wall-clock	7750 cpu 5120 wall-clock	16300 cpu 10758 wall-clock	34840 cpu 25058 wall-clock

The time dramatically dropped when the program send all message at once. Since head processor doesn't have to wait, multiple processor could process those message at same time. The time expense is close to linearly increasing as the number of messages or the number of processors increased.