PP HW2 吳岱儒 108062593

Implementation

pthread

- 以pixels為單位進行image repeat與color的計算
- thread之間會競爭share_k這個variable，並對得到的k進行image repeat與color的計算
- 利用vectorization一次進行兩個pixels的運算，離開loop後因為可能只有一個pixel達到終止條件離開，所以會再對這兩個pixel進行一次計算，以確保兩個pixels都達到最後的終止條件

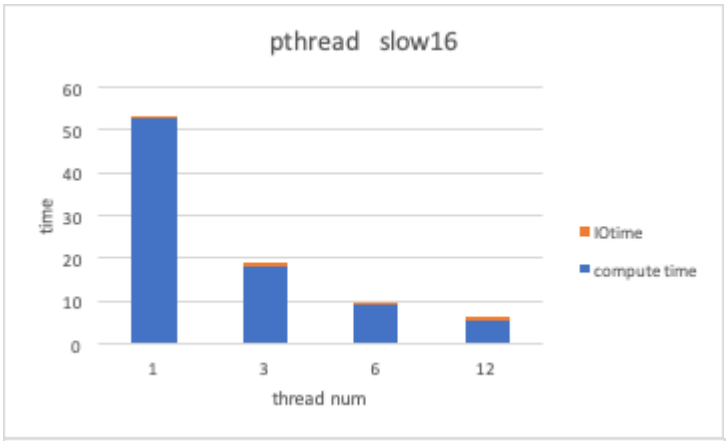
hybrid

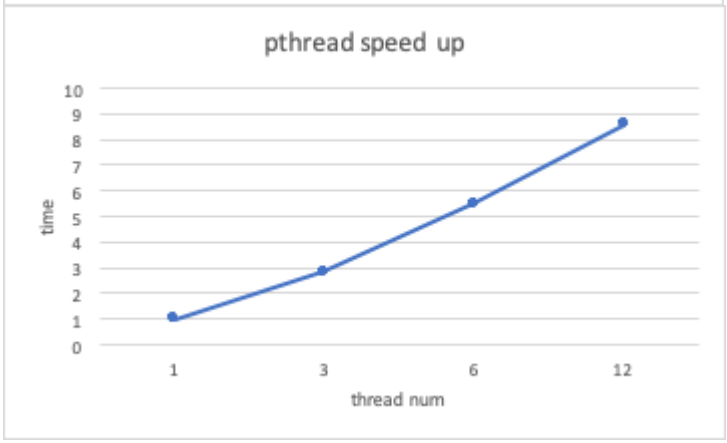
- 以pixel為單位，向每個process進行工作的分配，每個process會分配到不同的chunk，該chunk就是這個process要執行的job
- 針對process的計算，omp使用dynamic的scheduler可以有效的加速process的計算速度
- 每個process計算完自己的job以後，會將自己計算出來的結果全都送到root當中，最後再由root寫道disk當中
- 同pthread有用到vectorization

Experiment & Analysis

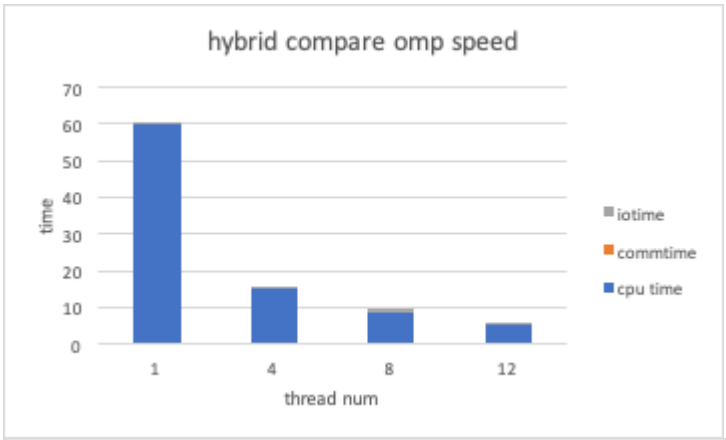
1. Methodology
    - system: apollo
    - Performance Metrics:
      pthread利用clock_gettime來測試時間
      hybrid利用MPI_Barrier與Wtime()來測試時間
2. Plots: Scalability & Load Balancing
    - pthread:

- 針對不同的thread數量做測試

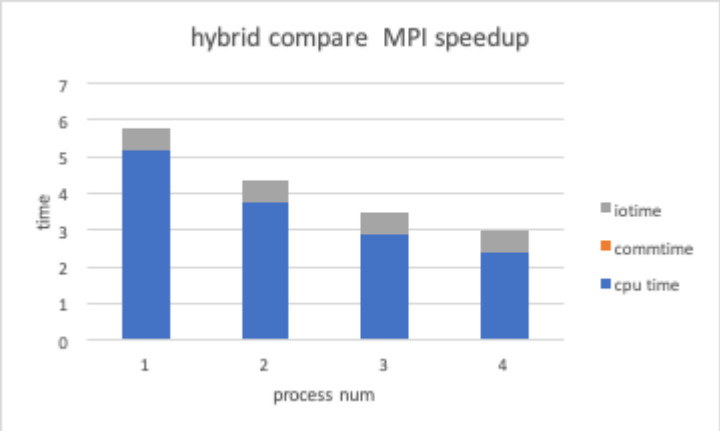


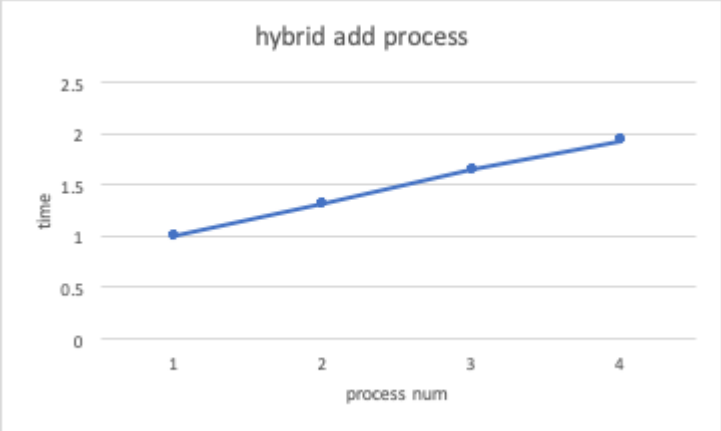


- hybrid:
  - 針對不同的thread數目做測試
    single process



- 針對不同的process數目做測試
  process的thread數量皆為12





- load balance test:
  針對hybrid和thread版本，使用相同的thread數量
  ie. pthread版本參數設-n1 -c3，則hybrid版本設為-n3 -c1來測量thread版本與process版本的load balance，並且由執行最久的process(thread)-執行最快的process(thread)

load balance test			
parameter	最慢	最快	diff
#-n1 -c3	17.7906	17.7808	0.0098
#-n3 -c1	33.7164	10.5301	23.1863
#-n1 -c6	9.12473	9.11177	0.01296
#-n6 -c1	21.7941	1.24426	20.54984
#-n1 -c8	7.58158	7.55802	0.02356
#-n8 -c1	18.1359	1.66235	16.47355
#-n1 -c12	5.58831	5.57413	0.01418
#-n12 -c1	12.7608	0.373219	12.387581

3. Discussion

1. 針對Scalability，不管是pthread或是hybrid都可以看得出來他們的sppedUp幾乎都是呈線性的狀態，雖然hybrid比較process的部分看起來speedup比較低，但這是因為每條process都使用12個thread，所以本身的執行速度就很快的關係
2. 針對hybrid的scalability，預期加入更多的process speedup不會更好，因為process之間有load balance的問題
3. 針對pthread的scalability，可以預期加入更多的thread可以讓程式的執行速度更快，而且幾乎是呈現linear的方式發展

4. 針對load balance可以看出使用hybrid各個process的load非常的不平均，而pthread版本因為有share variable可以使用，所以每個thread不會wait，只要有pixesl可以計算就直接繼續計算下去，因此相對於hybrid，使用pthread明顯有較好的load balance

4. others

    1. 針對thread版本，使用chunk會需要1500s的運行時間，用share_k讓每個thread競爭要算的variable只需要900s，最後使用vectorization只需要500s即可運行完成所有的testCase
    2. 針對process版本，因為無法像thread一樣使用share variable所以只能分為chunk進行計算使用omp的static計算時間需要900s，使用dynamic運行時間需要700秒，由此可知dynamic比static的方式可以更有效的利用CPU，加上vectorization可以將時間降低到400秒左右，
    3. 綜合上面兩個case，vectorization的效果基本上可以大約降低一半的compute time
    4. 由hybrid只調整thread num和pthread版本的比較，由此可知omp的schedule(dynamic)效果等同於pthread的share variable的方法
    5. 由4.可以推估，是否schedule(static)的方法等同於將pthread將task分為chunk? (尚未驗證)

Experience & Conclusion

1. load balance在program的時候很重要，特別是在這種計算repeat的時候，因為我們無法得知pixel要進行幾次repeat，所以直接切成chunk來進行計算的效果非常不好
2. omp和pthread的dynamic scheduler效果看起來幾乎是一樣的，不知道static的效果是否如我預期的一樣
3. 有沒有share memory在program的時候相差很多，因為在寫thread的時候沒有注意到計算image color時會將圖片翻轉，因此在找這個問題的時候找了非常久
4. vectorization是一個非常好的加速方式，如果計算量越龐大，就越可以達到理論上的速度提升