Assignment OpenMP Intel i7 10700 งานเดี๋ยว ส่งวันที่ 11 ตุลาคม เวลา 13.00 น คิดเป็น 10% ของเกรด

จัดตารางการทดลองว่าใครจะทดลองก่อนหลัง

Log in ด้วย secure shell ดังนี้

ssh hpc@161.246.5.86 -p 22

password = hpc948323

ตรวจสอบว่ามีผู้ใช้คนอื่นใช้เครื่องพร้อมกันหรือไม่ โดยใช้คำสั่ง top ดังนี้

หากมีเพียง 3-4 คนให้ทดลองต่อได้

copy Mattson_OMP_exercises เป็นโฟลเดอร์ของตนเอง ด้วยคำสั่งต่อไปนี้

\$ cd Documents

\$ mkdir ชื่อโฟลเดอร์

\$ cp Mattson_OMP_exercises/*.* ชื่อโฟลเดอร์ -r

ย้าย current directory ไปยัง ชื่อโฟลเดอร์ ใหม่

\$ cd ชื่อโฟลเดอร์

ทดลองรัน pi ซึ่งเป็น executable ของ pi.c จำนวน 3 รอบโดย

\$./pi

หาค่าเฉลี่ยแล้วบันทึกเวลาลงในตารางผลการทดลอง

ทดลองรัน pi_loop ซึ่งเป็น executable ของ pi_loop.c จำนวน 3 รอบโดย

\$ cd solutions

\$./pi_loop

หาค่าเฉลี่ยแล้วบันทึกเวลาลงในตารางผลการทดลอง

ทดลองรัน pi_simple ซึ่งเป็น executable ของ pi_spmd_simple.c จำนวน 3 รอบโดย

 $\ ./pi_simple$

หาค่าเฉลี่ยแล้วบันทึกเวลาลงในตารางผลการทดลอง

ทดลองรัน pi_final ซึ่งเป็น executable ของ pi_spmd_final.c จำนวน 3 รอบโดย

\$./pi_final

หาค่าเฉลี่ยแล้วบันทึกเวลาลงในตารางผลการทดลอง

ปรับแก้ num_steps เป็น 1000 ล้าน ใน source file ทุกไฟล์ คือ pi.c, pi_loop.c, pi_spmd_simple.c, pi_spmd_final.c แล้วทำการคอมไพล์และทำการทดลองใหม่อีกรอบ แล้วบันทึกผลในตาราง ต่อไปนี้ โปรดสังเกต ว่า ไฟล์อยู่กันคนละไดเรกทอรี

\$ gcc -fopenmp pi.c -o pi

\$ gcc -fopenmp pi_loop.c -o pi_loop

\$ gcc -fopenmp pi_simple.c -o pi_spmd_simple.c

\$ gcc -fopenmp pi_final.c -o pi_spmd_final.c

เวลาเฉลี่ย	pi	pi_loop	pi_simple	pi_final
1 Thread num_steps=100 ล้าน	0.19345567	0.23200267	0.30295633	0.209269
8 Thread num_steps=100 ล้าน	-	0.47550867	0.286548	0.04821033
16 Thread num_steps=100 ล้าน	-	0.35684267	0.22	0.025462
1 Thread num_steps=1000 ล้าน	1.93594933	2.31804767	3.02926867	2.09092133
8 Thread num_steps=1000 ล้าน	-	4.59195433	4.295986	0.294676
16 Thread num_steps=1000 ล้าน	-	3.59529467	2.25268933	0.253631
				หน่วยเป็นวินาที่ (sec)

วิเคราะห์และสรุปผลการทดลอง โดยไม่ลอก นศ สามารถถกเถียงกันได้ว่าเกิดอะไรขึ้น แต่ให้เขียนวิเคราะห์และสรุป ผลการทดลองด้วยตัวเอง

```
hpc@i710700:~/Documents/Aekarin_64015172/Mattson_OMP_exercises$ gcc -fopenmp pi.c -o pi hpc@i710700:~/Documents/Aekarin_64015172/Mattson_OMP_exercises$ ./pi pi with 1000000000 steps is 3.141593 in 0.193491 seconds hpc@i710700:~/Documents/Aekarin_64015172/Mattson_OMP_exercises$ ./pi pi with 1000000000 steps is 3.141593 in 0.193348 seconds hpc@i710700:~/Documents/Aekarin_64015172/Mattson_OMP_exercises$ ./pi pi with 1000000000 steps is 3.141593 in 0.193528 seconds
```

```
hpc@i710700:~/Documents/Aekarin_64015172/Mattson_OMP_exercises$ ./pi_1000

pi with 1000000000 steps is 3.141593 in 1.935028 seconds

hpc@i710700:~/Documents/Aekarin_64015172/Mattson_OMP_exercises$ ./pi_1000

pi with 1000000000 steps is 3.141593 in 1.936000 seconds

hpc@i710700:~/Documents/Aekarin_64015172/Mattson_OMP_exercises$ ./pi_1000

pi with 1000000000 steps is 3.141593 in 1.936820 seconds
```

(รูปภาพผลภารทดลอง pi 100ล้าน และ 1000ล้าน) โปรแกรมนี้คำนวณอินทิกรัลของฟังภ์ชัน 4/(1+x^2) จาก 0 ถึง 1 (ค่า pi) ภารทำงานของโปรแกรม:

1. เธรดหลักจะเริ่มต้นเวลาโดยใช้ฟังก์ชัน omp_get_wtime()

2. เธรดหลักจะวนลูปตั้งแต่ 1 ถึง num_steps และคำนวณผลรวมของฟังก์ชัน 4/(1+x^2) โดยใช้ ตัวแปร suุm

3. หลังจากที่วนลูปสิ้นสุดลง เธรดหลักจะคำนวณค่าของ pi โดยใช้สูตร pi = step * sum

4. เธรดหลักจะแสดงผลค่าของ pi และเวลาที่ใช้ในการคำนวณโดยใช้ฟังก์ชัน printf()

โปรแกรมนี้ไม่มีปัญหาอะไร แต่ประสิทธิภาพของโปรแกรมจะไม่ดีเมื่อใช้กับจำนวนเธรดที่มาก เนื่องจาก โปรแกรมนี้ไม่ได้ใช้ประโยชน์จาก openme อย่างเต็มที่

เพื่อปรับปรุงประสิทธิภาพของโปรแกรม สามารถใช้ openmp เพื่อแบ่งงานการคำนวณออกเป็นหลายๆ เธรด วิธีนี้จะช่วยให้โปรแกรมสามารถคำนวณอินทิกร์ลได้เร็วขึ้นเมื่อใช้กับจำนวนเธรดที่มาก pi_loop

threads = 16
is 3.141594 in 0.379027 seconds and 16 t

m_threads = 1 . is 3.141593 in 0.231780 seconds and 1 thread um_threads = 2 i Is 3.141593 in 0.330931 seconds and 2 threads m_threads = 11 i is 3.141594 in 0.438647 seconds and 11 thre n_threads = 15 is 3.141867 in 0.357211 seconds and 15 thr

m_threads = 12 Is 3.141807 in 0.413070 seconds and 12 threads

(รูปภาพผลภารทดลอง pi_loop 100ล้าน และ 1000ล้าน)

โปรแกรมนี้ทำงานโดยแบ่งการวนช้ำของลูป for ออกเป็นหลายๆ เธรด แต่ละเธรดจะคำนวณผลรวม ของฟังก์ชัน 4/(1+x^2) สำหรับจำนวนเต็ม i ที่แตกต่างกัน หลังจากที่เธรุดแต่ละตัวคำนวณผลรวมของ ี่ ศนเสร็จแล้ว เธรดนั้นจะรวมผลรวมของศนเข้าภับผลรวมของเธรดอื่นๆ เพื่อให้ได้ผลรวมสุดท้าย

ภารทำงานของเธรด:

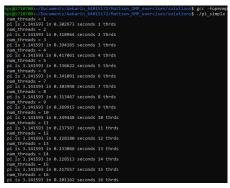
เธรดหลักจะสร้างที่มเธรดขึ้นมาโดยใช้คำสั่ง #pragma omp parallel.
 เธรดแต่ละตัวในที่มจะคำนวณผลรวมของฟังภ์ชัน 4/(1+x^2) สำหรับจำนวนเต็ม i ที่

แตกต่างกัน โดยใช้คำสั่ง #pragma omp for reduction(+:sum).
3. หลังจากที่เธรดแต่ละตัวคำนวณผลรวมของตนเสร็จแล้ว เธรดนั้นจะรวมผลรวมของตนเข้า กับผลรวมของเธรดอื่นๆ โดยใช้คำสั่ง reduction(+:sum).

4. เธรดหลักจะรวมผลรวมของเธรดทั้งหมดเข้าด้วยกันเพื่อให้ได้ผลรวมสุดท้าย

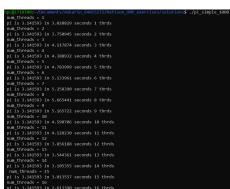
ปัญหาในโปรแกรมนี้คือเธรดแต่ละตัวจะคำนวณค่าของตัวแปร รนพ แยกจากกันโดยไม่มีการ ซึงโครไนซ์ สิ่งนี้อาจทำให้เกิดปัญหาการแข่งขันข้อมูล (race condition) ซึ่งหมายความว่าค่าของตัวแปร รนพ อาจไม่ถูกต้อง

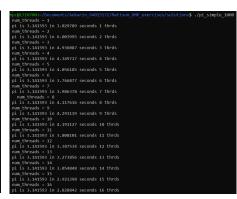
Pi_simple

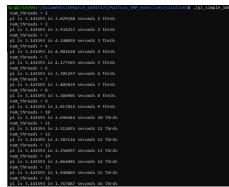


hpc@i710700: ~/Documents/Aekarin_64015172/Mattson_OMP_exercises/solutio	ns\$./pi_simple
num_threads = 1	
pi is 3.141593 in 0.302955 seconds 1 thrds	
num_threads = 2	
pi is 3.141593 in 0.230476 seconds 2 thrds	
num_threads = 3	
pi is 3.141593 in 0.318085 seconds 3 thrds	
num_threads = 4	
pi is 3.141593 in 0.275075 seconds 4 thrds	
num_threads = 5	
pi is 3.141593 in 0.325099 seconds 5 thrds	
num_threads = 6	
pi is 3.141593 in 0.344469 seconds 6 thrds	
num_threads = 7	
pi is 3.141593 in 0.298489 seconds 7 thrds	
num_threads = 8	
pi is 3.141593 in 0.275176 seconds 8 thrds	
num_threads = 9	
pi is 3.141593 in 0.254862 seconds 9 thrds	
num_threads = 10	
pi is 3.141593 in 0.233139 seconds 10 thrds	
num_threads = 11	
pi is 3.141593 in 0.265293 seconds 11 thrds	
num_threads = 12	
pi is 3.141593 in 0.260340 seconds 12 thrds	
num_threads = 13	
pi is 3.141593 in 0.305958 seconds 13 thrds num threads = 14	
ni is 3.141593 in 0.287645 seconds 14 thrds	
pi is 3.141593 in 0.28/645 seconds 14 thrds num threads = 15	
pi is 3.141593 in 0.281059 seconds 15 thrds	
num threads = 16	
pi is 3.141593 in 0.281834 seconds 16 thrds	
pr 13 3.141393 10 0.201034 Seconds 10 tillus	

hpc@i710700:~/Documents/Aekarin_64015172/Mattson_OMP_exercises/solutions\$./pi_sim
num threads = 1
pi is 3.141593 in 0.302941 seconds 1 thrds
num_threads = 2
pi is 3.141593 in 0.294353 seconds 2 thrds
num threads = 3
pi is 3.141593 in 0.289594 seconds 3 thrds
num_threads = 4
pi is 3.141593 in 0.291581 seconds 4 thrds
num threads = 5
pi is 3.141593 in 0.307029 seconds 5 thrds
num threads = 6
pi is 3.141593 in 0.348697 seconds 6 thrds
num_threads = 7
pi Īs 3.141593 in 0.304601 seconds 7 thrds
num_threads = 8
pi is 3.141593 in 0.271001 seconds 8 thrds
num_threads = 9
pi is 3.141593 in 0.251882 seconds 9 thrds
num_threads = 10
pi is 3.141593 in 0.233044 seconds 10 thrds
num_threads = 11
pi is 3.141593 in 0.223646 seconds 11 thrds
num_threads = 12
pi is 3.141593 in 0.214183 seconds 12 thrds
num_threads = 13
pi is 3.141593 in 0.212653 seconds 13 thrds
num_threads = 14
pi is 3.141593 in 0.197427 seconds 14 thrds
num_threads = 15
pi is 3.141593 in 0.186758 seconds 15 thrds
num_threads = 16
pi is 3.141593 in 0.177004 seconds 16 thrds







(รูปภาพผลภารทดลอง pi_simple 100ล้าน และ 1000ล้าน)

SPMD (Single Program Multiple Data) เป็นรูปแบบการเขียนโปรแกรมแบบคู่ขนาน ใน รูปแบบนี้ เธรดแต่ละตัวจะรันโปรแกรมเดียวกัน แต่จะมีข้อมูลที่แตกต่างกัน

ภารทำงานของเธรด:

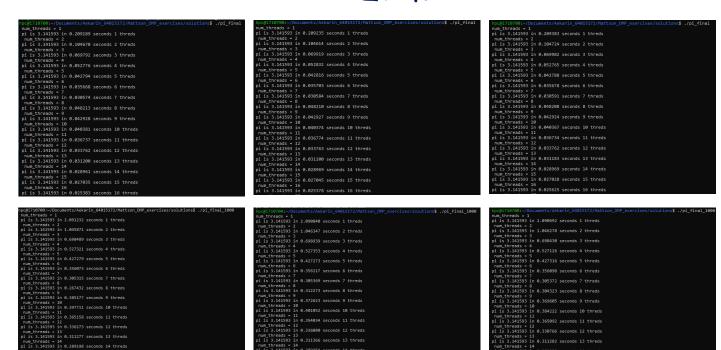
 เธรดหลักจะสร้างที่มเธรดขึ้นมาโดยใช้คำสั่ง #pragma omp parallel.
 เธรดแต่ละตัวในที่มจะคำนวณผลรวมของฟังภ์ชัน 4/(1+x^2) สำหรับจำนวนเต็ม i ที่ แตกต่างกุ้น โดยใช้คำสั่ง for.

3. หลังจากที่เธรดแต่ละตัวคำนวณผลรวมของตนเสร็จแล้ว เธรดนั้นจะรวมผลรวมของตนเข้า ภับผลรวมของเธรดอื่นๆ โดยใช้คำสั่ง sum[id].

4. เธรดหลักจะรวมผลรวมของเธรดทั้งหมดเข้าด้วยกันเพื่อให้ได้ผลรวมสุดท้าย

ปัญหาในโปรแกรมนี้คือเธรดแต่ละตัวจะคำนวณค่าของตัวแปร sum[id] แยกจากกันโดยไม่มีการ ซึ่งโครไนซ์ สิ่งนี้อาจทำให้เกิดปัญหาการแข่งขันข้อมูล (race condition) ซึ่งหมายความว่าค่าของตัวแปร sum[id] อาจไม่ถูกต้อง

Pi_final



(รูปภาพผลภารทดลอง pi_final 100ล้าน และ 1000ล้าน) โปรแกรมนี้ทำงานได้ถูกต้องและมีประสิทธิภาพมากกว่าโปรแกรมต้นฉบับ (pi.c) ภารทำงานของเธรด:

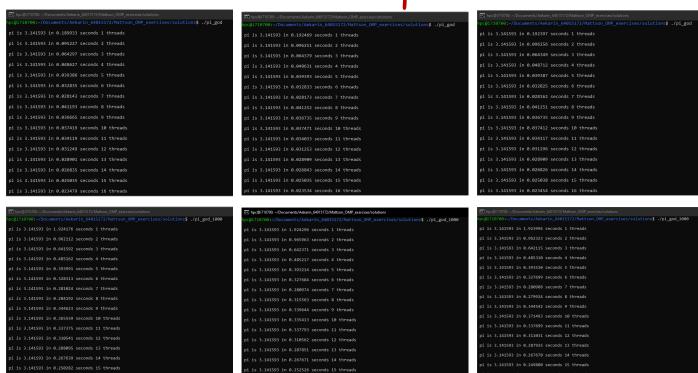
 เธรดหลักจะสร้างทีมเธรดขึ้นมาโดยใช้คำสั่ง #pragma omp parallel.
 เธรดแต่ละตัวในทีมจะคำนวณผลรวมของฟังภ์ชัน 4/(1+x^2) สำหรับจำนวนเต็ม i ที่แตกต่าง กัน โดยใช้คำสั่ง for.

3. หลังจากที่เฮรดแต่ละตัวคำนวณผลรวมของตนเสร็จแล้ว เฮรดนั้นจะรวมผลรวมของตนเข้า กับผลรวมของเธรดอื่นๆ โดยใช้คำสั่ง critical.

4. เธรดหลักจะรวมผลรวมของเธรดทั้งหมดเข้าด้วยกันเพื่อให้ได้ผลรวมสุดท้าย

โปรแกรมนี้หลีกเลี่ยงปัญหา false sharing โดยการเก็บผลรวมบางส่วนไว้ในตัวแปร private ช่วยให้ แน่ใจว่าเธรดแต่ละตัวจะไม่เขียนทับข้อมูลของเธรดอื่นๆ

Pi_God (ทดลองแก้ไข pi_final ให้เร็วขึ้น)



(รูปภาพผลภารทดลอง pi_God 100ล้าน และ 1000ล้าน)

ตารางเปรียบเทียบ pi_final กับ pi_God					
เวลาเฉลี่ย	Pi_final	Pi_God			
1 Thread 100ล้าน	0.209269	0.19159967			
8 Thread 100ล้าน	0.04821033	0.041232			
16 Thread 100ล้าน	0.025462	0.023489			
1 Thread 1000ล้าน	2.09092133	1.92415533			
8 Thread 1000ล้าน	0.294676	0.29322633			
16 Thread 1000ล้าน	0.253631	0.23430233			

ปัจจัยที่น่าจะทำให้โปรแกรมความเร็วต่างกัน (สมมติฐาน):

- 1. ภารใช้ #pragma omp for ในลูป ทำให้ OpenMP รับผิดชอบในภารแบ่งงานอัตโนมัติให้ แต่ละ Thread โดยที่ไม่ต้องระบุ id และ numthreads แต่สิ่งที่ต้องทำคือใช้ #pragma omp atomic เพื่อรวมผลลัพธ์ระหว่าง Threads โดยปลอดภัยจาภ race condition ใน ภารเขียนผลลัพธ์ full_sum.
- 2. กำหนด partial_sum เป็น 0.0 เพื่อแก้ปัญหา false sharing โดยเริ่มต้นค่า partial_sum ในแต่ละ Thread ในแต่ละรอบของการคำนวณ.
- 3. การใช้ #pragma omp for แทนการแบ่งงานด้วยตัวแปร i ทำให้เราไม่ต้องจัดการกับการ คำนวณที่จำนวน Thread หาร ทนพ_steps ไม่ลงตัว

#สรุปได้ว่า โค้ดของ pi_god เร็วภว่าเล้กน้อย เนื่องจากสามารถทำงานได้หลายขั้นตอนพร้อมกัน และไม่ต้องจัดการกับการคำนวณที่จำนวน Thread หาร ทุนm_steps ไม่ลงตัว