นายนคเรศ เมืองภักดี

6610505438

Homework Report

ข้อที่ 1 Intro to Parallel Programming

วิธีการที่ใช้

ใช้วิธีการ Quadratic Sieve Algorithm โดยมีหลักการคือ ถ้าเราสามารถ หา X, Y โดยที่ X != ±Y และทำให้ $X^2 \equiv Y^2 \ (mod\ n)$ เมื่อ n คือ ตัวเลขที่เราต้องการแยกตัวประกอบ เราจะได้ว่า gcd(n, Y - X) และ gcd(n, Y+X) จะเป็นตัวประกอบของ โดยจากวิธีของ [1] ได้เสนอการ ประมาณค่า X, Y ไว้ดังนี้ให้ $f(x) = Ax^2 + 2Bx + C\ mod(n)$ และคำนวณหา A, B, C โดย

$$A = D^2 \text{ เมื่อ } D \text{ คือจำนวนเฉพาะที่ไม่เป็นตัวประกอบของ } f(x)$$

$h_0 \equiv \ n^{\frac{D-3}{4}}$ (mod D), $h_1 \equiv \ n^{\frac{D+1}{4}}\ (mod\ D)$, $h_1^2 \equiv \ nn^{\frac{D-1}{2}}\ (mod\ D)$, $h_2 \equiv (2h_1)^{-1}(\frac{n-h_1^2}{D})\ (mod\ D)$

$$B \equiv \ h_1 + h_2 D\ (mod\ D)$$

$$C = \frac{B^2 - n}{A}$$

ทำให้ได้ว่า

$Af(x) = (Ax)^2 + ABx + B^2 - n \equiv (Ax + B)^2\ mod(n)$

ซึ่งทำให้ $\prod_i Af(x_i) \ \equiv \ (G(x_i))^2\ mod(n)$

ดังนั้น ถ้าเรา สามารถจัดรูป $Af(x_i)$ ให้อยู่ในรูปผลคูณของตัวประกอบจำนวนเฉพาะที่อยู่ในเซ็ตของ Factor Base ได้ เราจะหา $\prod_i Af(x_i)$ ที่เป็นกำลังสองสมบูรณ์ได้จาก การแก้สัมการ ให้ผลบวกเลขยกกำลังของตัวประกอบจำนวนเฉพาะ เป็นเลขคู่ได้ จึงให้ $X^2 = \ \prod_i Af(x_i)$ และจะได้ $Y^2 = \ \prod_i (G(x_i))^2$

จึงสามารถหา X^2, Y^2 ที่เป็นไปได้โดยใช้ Gaussian Elimination ใน GF(2)ได้ และเมื่อได้ค่า X^2, Y^2 แล้วก็สามารถหา ตัวประกอบได้จาก gcd(n, Y - X)

จากขั้นตอนข้างต้น ในขั้นตอนหาชุดของ f(x) นั้นสามารถทำแบบ parallel ได้ เพราะแต่ละ f(x) นั้นไม่ขึ้นต่อกันและใช้ข้อมูลคือ เซ็ตของ Factor Base, n และ D ซึ่ง Factor Base และ n ไม่เปลี่ยนแปลงตลอดการทำงานอยู่แล้ว จึงออกแบบให้มี process หนึ่ง(Master) หาค่า D และส่งให้ process ที่เหลือ(Slave) หาค่าของ f(x) ที่ตรงตามเงื่อนไขจาก D ที่ ได้รับและส่งค่ากลับให้ Master เป็นคนเก็บ จนกระทั่งหาชุดของ f(x) ได้ตามจำนวนที่ต้องการแล้ว Slave ทั้งหมดจะหยุดทำงานเหลือเพียง Master ที่นำชุดของ f(x) มาทำ Gaussian Elimination พร้อมสรุปค่าตัวประกอบที่หาได้ และจบการทำงาน

นายนคเรศ เมืองภักดี

6610505438

ผลลัพธ์

Test case	Number to factorize
T20	18567078082619935259
T30	350243405507562291174415825999
T40	57059795506186704463085788585426753739 83
T45	732197471686198597184965476425281169401188191

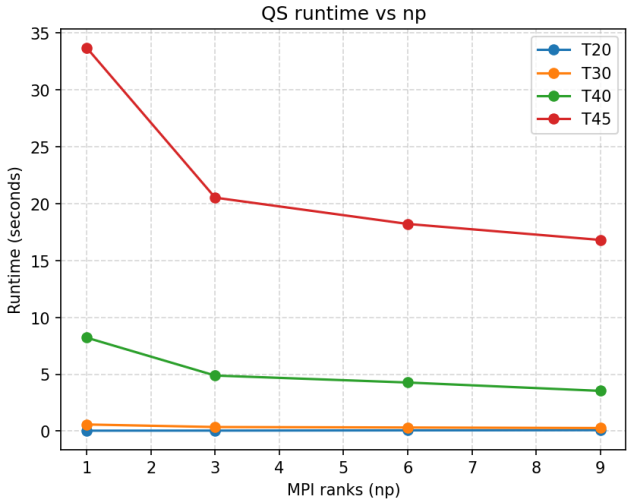
ตารางแสดงรายละเอียดของแต่ละ Test case

np	T20	T30	T40	T45
1	0.038	0.583	8.214	33.701
3	0.045	0.359	4.886	20.529
6	0.066	0.318	4.273	18.216
9	0.083	0.271	3.538	16.81

ตารางแสดงจำนวน process กับเวลาที่ใช้ในหน่วยวินาทีในแต่ละ Test case

np	T20	T30	T40	T45
1	1	1	1	1
3	0.844	1.624	1.681	1.642
6	0.576	1.833	1.922	1.850
9	0.458	2.151	2.322	2.005

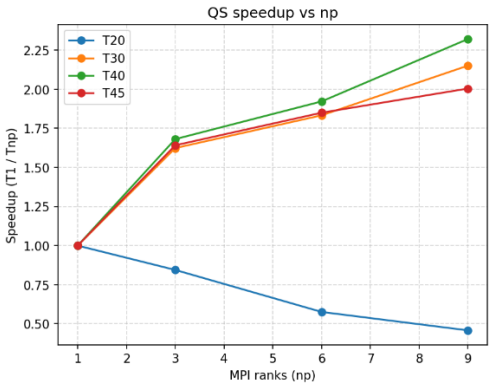
ตารางแสดงจำนวน process กับจำนวนเท่าที่เร็วกว่าเมื่อเทียบกับการใช้ 1 process



กราฟแสดงจำนวน process กับเวลาที่ใช้ในหน่วยวินาทีในแต่ละ Test case

นายนคเรศ เมืองภักดี

6610505438



กราฟแสดงจำนวน process กับจำนวนเท่าที่เร็วกว่าเมื่อเทียบกับการใช้ 1 process

วิเคราะห์ผลลัพธ์

จาก Amdahl's law ได้ว่า

$$S = \frac{1}{1 - P + \frac{P}{N}}$$

เมื่อ S คือ *Speed up*

P คือ สัดส่วนที่ *parallel*

N คือ จำนวน *process* ที่ ทำ *parallel*

จัดรูปใหม่เป็น

$$P = \frac{1 - \frac{1}{S}}{1 - \frac{1}{n}}$$

เมื่อแทน S และ N จากผลลัพธ์ที่ได้ จะได้ P ดังตาราง

N	T20	T30	T40	T45
1	0	0	0	0
3	-0.276	0.576	0.608	0.586
6	-0.884	0.545	0.576	0.551
9	-1.332	0.602	0.640	0.564

ตารางแสดง ค่า P ที่คำนวณได้จากผลลัพธ์

นายนคเรศ เมืองภักดี
6610505438

พบว่า นอกจาก ที่ T20 P มีค่าติดลบ อันเป็นผลมาจากเลขที่ใช้คำนวณมีขนาดเล็กเกินไปจน ทำให้เวลาที่ speed up น้อยกว่า over head ที่เพิ่มขึ้นของ การ parallel แล้ว test case ที่เหลือ จะมี parallel part เฉลี่ยที่ 0.583 จึงสรุปว่า โปรแกรมนี้จะมี serial part เฉลี่ย = 1 - 0.583 = 0.417

นายนคเรศ เมืองภักดี
6610505438

ข้อที่ 2 **Copy on Write**

วิธีการที่ใช้

จะจำลองการเกิด Copy on write โดยการใช้ภาษา python ด้วยการจองพื้นที่โดยใช้ numpy.array และ fork พร้อมสังเกต memory ของ child process หลังจากนั้นจึงให้ child process แก้ไขข้อมูลใน array แล้วจึงสังเกตการอีกรอบ แล้วจึงจบการทำงานของโปรแกรม โดยจะสังเกต memory ด้วยข้อมูลจากไฟล์ /proc/pid/smaps_rollup โดยอ้างอิงจาก [2] แล้ว ภายในจะ มีข้อมูลเกี่ยวกับ shared_clean และ shared_dirty อยู่ จึงจะใช้ค่าจาก ไฟล์นี้ในการสังเกตการ โดยจะสังเกตค่า shared = shared_clean + shared_dirty เป็นหลัก

ผลลัพธ์

reserve_size /edit_ratio	Rss_Before	Shared_Before	Rss_After	Shared_After	(Shared_Before-Shared_After) /reserve_size
80 MiB / 0.8	98.066	97.125	98.871	33.879	0.791
60 MiB / 0.8	77.988	77.051	78.848	29.808	0.787
90 MiB / 0.75	107.988	107.035	108.844	40.293	0.742
70 MiB / 0.75	87.984	87.039	88.785	35.27	0.740
75 MiB / 0.85	92.98	92.047	93.855	29.066	0.840

ตารางแสดง *RSS, Shared* ก่อนและหลังการแก้ไขข้อมูลใน *array* ของ *child process* โดยสังเกต *memory* ของ *child process*

```
RSS before child edit the data
[Child] child process's RSS = 92.984375 MiB parent process's RSS = 106.375 MiB
[Child] child process's shared = 92.046875 MiB parent process's shared = 99.296875 MiB
RSS after child edit the data
[Child] child process's RSS = 93.85546875 MiB parent process's RSS = 106.375 MiB
[Child] child process's shared = 29.06640625 MiB parent process's shared = 36.015625 MiB
[Child] child terminate
[Parent] child already terminate
```

รูปแสดง ตัวอย่างผลลัพธ์ที่ได้จากโปรแกรมที่มี reserve_size = 75 MiB **และ** edit_ratio = 0.85

วิเคราะห์ผลลัพธ์

จาก **ตารางแสดง *RSS, Shared* ก่อนและหลังการแก้ไขข้อมูลใน *array* ของ *child process*** พบว่า ผลต่างของ shared ก่อน และ หลังการแก้ไขพบว่า มีอัตราส่วน เท่ากับ ขนาดของข้อมูลที่ child process ได้แก้ไขจริงๆ ซึ่งตรงกับลักษณะของ การ Copy On Write ที่ child process จะแชร์ข้อมูลกับ parent process จนกระทั่ง child process มีการเปลี่ยนแปลงข้อมูล จึงสรุปได้ว่า โปรแกรมนี้สามารถจำลองการ เกิดของ Copy On Write ได้อย่างถูกต้อง

นายนคเรศ เมืองภักดี

6610505438

อ้างอิง

[1] Bytopia. (n.d.). *Quadratic Sieve (QS)*. Retrieved October 29, 2025, from https://www.bytopia.dk/qs/

[2] Linux kernel documentation. (2020, May 15). *T H E /proc F I L E S Y S T E M*. Retrieved October 29, 2025, from https://mjmwired.net/kernel/Documentation/filesystems/proc.txt