KKU Engineering Journal
http://www.en.kku.ac.th/enjournal/th/

การปรับปรุงประสิทธิภาพการทำงานของโปรแกรมที่พัฒนาด้วย WPF โดยใช้เทคนิค Multi-thread

Performance improvement of developed program by using multi-thread technique

สุรศักดิ์ จ่าบาล และ รุจชัย อึ้งอารุณยะวี*

Surasak Jabal and Rujchai Ung-arunyawee*

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น จังหวัดขอนแก่น 40002

Department of Computer Engineering, Faculty of Engineering, Khon Kaen University, Khon Kaen 40002, Thailand.

Received February 2014
Accepted May 2014

บทคัดย่อ

งานวิจัยนี้เป็นการเสนอการนำเอาเทคนิคการเขียนโปรแกรมแบบ Multi-thread มาใช้ปรับปรุงประสิทธิภาพการทำงานของโปรแกรมที่พัฒนาด้วย Windows Presentation Foundation (WPF) โดยใช้ CAI (เกม 24) เป็นกรณีศึกษา ในงานวิจัยนี้ประกอบไปด้วยส่วนสำคัญ 2 ส่วนคือ ส่วนของแก้ไขการเขียนโปรแกรมให้เป็นเชิงวัตถุ และส่วนของการเขียนโปรแกรมแบบ Multi-thread โดยจะทำการแบ่ง Thread ตามจำนวนเลข Catalan ที่คำนวณได้ ผลการวิจัยพบว่า การเขียนโปรแกรมแบบ Multi-thread ทำให้ประสิทธิภาพการทำงานของโปรแกรมได้ดีกว่าแบบ Single-thread ตั้งแต่ 44% ถึง 88% นอกจากนี้ยังพบว่าจำนวน Core ของ CPU มีผลทำให้ประสิทธิภาพการทำงานแบบ Multi-thread ดีขึ้นด้วย

คำสำคัญ : ประสิทธิภาพ WPF Multi-thread

Abstract

This research presented how to use a multi-thread programming technique to improve the performance of a program written by Windows Presentation Foundation (WPF). The Computer Assisted Instruction (CAI) software, named GAME24, was selected to use as a case study. This study composed of two main parts. The first part was about design and modification of the program structure upon the Object Oriented Programing (OOP) approach. The second part was about coding the program using the multi-thread technique which the number of threads were based on the calculated Catalan number. The result showed that the multi-thread programming technique increased the performance of the program 44%-88% compared to the single-thread technique. In addition, it has been found that the number of cores in the CPU also increase the performance of multithreaded program proportionally.

Keywords : Performance, WPF, Multi-thread

* Corresponding author.

Email address: surasakj009@hotmail.com (Jabal, S.), rujchai@gear.kku.ac.th* (Ung-arunyawee, R.)

doi: 10.14456/kkuenj.2015.13

1. บทนำ

ในการดำเนินชีวิตประจำวันการทำงานของมนุษย์จะต้องมีส่วนเกี่ยวข้องกับคณิตศาสตร์ตลอดเวลา การเรียนรู้และพัฒนาทักษะด้านคณิตศาสตร์จึงเป็นสิ่งที่สำคัญมาก ทำให้มีคนคิดที่จะนำเอาคอมพิวเตอร์มาสร้างเกมคณิตศาสตร์ เพื่อช่วยเพิ่มประสิทธิภาพในการเรียนรู้ให้มีความสะดวกและรวดเร็วขึ้น ทำให้เกิดการพัฒนาความคิดสร้างสรรค์ [1] เกม 24 เป็นอีกหนึ่งเกมที่มีคนคิดค้นขึ้นมา เพื่อเพิ่มฝึกทักษะความสามารถในการเรียนรู้คณิตศาสตร์ โดยที่ผู้เล่นจะเอาตัวเลข 4 จำนวน ซึ่งมีค่าตั้งแต่ 0 ถึง 9 มาคำนวณด้วย การบวกลบคูณหาร โดยการนำวงเล็บ มาช่วยจัดลำดับการคำนวณให้ได้ค่าเท่ากับ 24 เช่น 4788 สามารถทำได้ดังนี้ (4*(7-(8/8))) = 24 โดยสามารถสับเปลี่ยนตำแหน่งได้ทุกตำแหน่ง เป้าหมายของเกม 24 คือทำให้เกิดการพัฒนาความคิดสร้างสรรค์ทางคณิตศาสตร์

จากการศึกษางานวิจัยการเปรียบเทียบการทำงานของ CAI ที่พัฒนาโดย WPF เป็นการพัฒนาเกม 24 ในรูปของ Window Application และ Web Application กรณีศึกษา:เกม 24 [2] โดยใช้ Windows Presentation Foundation (WPF) เป็นเครื่องมือพัฒนา ใช้ภาษา Extensible Application Markup Language(XAML) ในการพัฒนาส่วนแสดงผลและใช้ภาษา C# ในการเขียนโปรแกรมทำงานแบบ Single-thread ในรูปของ Windows Application และ Web Application ของเกม 24 จะใช้ความรู้เรื่องการสับเปลี่ยนและ Catalan Number ในการค้นหาและตรวจสอบจำนวนนิพจน์ จากนั้น ทำการเปรียบเทียบผลการทำงานของ Window Application และ Web Application ที่พัฒนาด้วย WPF ซึ่งโปรแกรมยังมีส่วนที่ทำให้เกมเกิดปัญหาในส่วนต่างๆ อีกมากในการทำงานและประมวลผลของเกม 24 ได้แก่ รูปแบบสีสันของเกมยังไม่สวยงาม ความหลากหลายในการหาและตรวจสอบในการตั้งโจทย์ มีการจำกัดตัวเลขไว้ที่ 4 จำนวนในการตั้งโจทย์และคำตอบเท่ากับ 24 ขั้นตอนของการแสดงผลการทำงานของแต่ละหน้าเพจและขั้นตอนในการแสดงผลใน Level 3 ของ Window Application ทำงานช้ามาก เนื่องจากการประมวลผลของโปรแกรมเป็นแบบ Single-thread

งานวิจัยนี้จึงมีแนวคิดที่จะทำการปรับปรุงประสิทธิภาพการทำงาน CAI ของเกม 24 ที่พัฒนาโดย WPF โดยใช้ Multi-thread ในโปรแกรมให้สามารถกำหนดจำนวนตัวเลขในโจทย์ได้ตั้งแต่ 3 ถึง 6 จำนวนและค่าคำตอบใดๆ ได้ ทำให้โปรแกรมมีความหลากหลายและประสิทธิภาพในการประมวลผลโปรแกรมได้เร็วขึ้น มีลูกเล่นให้สวยงามน่าเล่น

2. วิธีการการวิจัย

งานวิจัยนี้มีวัตถุประสงค์เพื่อศึกษาและปรับปรุงประสิทธิภาพการทำงานของโปรแกรม 2 ส่วน คือ ส่วนของการเขียนโปรแกรมให้เป็นเชิงวัตถุ และส่วนของการเขียนโปรแกรมแบบ Multi-thread ใน Window Application โดยใช้ Windows Presentation Foundation (WPF) เป็นเครื่องมือในการปรับปรุงประสิทธิภาพการทำงาน [3,4] ของเกม 24 โดยการพัฒนาอัลกอริทึมที่ใช้ การออกแบบการทำงานของอัลกอริทึมที่ใช้ในการหาและตรวจสอบคำตอบของเกม 24 แบบ Single-thread กับ Multi-thread การออกแบบรูปแบบการทำงานของเกม 24 และการหาวิธีการทดลองโดยมีรายละเอียดดังต่อไปนี้

2.1 การพัฒนาอัลกอริทึมที่ใช้

การพัฒนาอัลกอริทึมการทำงานของเกม 24 ให้สามารถทำงานได้อย่างมีประสิทธิภาพได้มีการออกแบบอัลกอริทึมการทำงานของโปรแกรมเชิงวัตถุดังนี้

2.1.1 อัลกอริทึมสำหรับจัดเรียงและสับเปลี่ยนตัวเลข

สำหรับการหาผลลัพธ์ทั้งหมดของการจัดเรียงและสับเปลี่ยนชุดตัวเลขที่มีจำนวนตัวเลขใดๆ นั้น จะทำได้ยากลำบากมากถ้าหากวิธีการที่ใช้เป็นแบบการวนลูป (Iterative Approach) เพื่อหาผลลัพธ์ทั้งหมดจนครบ ดังนั้นในงานวิจัยจึงได้เลือกที่จะใช้วิธีการแบบเรียกซ้ำ (Recursive Approach) ซึ่งเป็นวิธีการที่ง่ายและตรงไปตรงมา ทำให้ชุดคำสั่งที่ใช้มีขนาดที่สั้นกว่าอีกวิธีการมาก

แนวความคิดหลักที่ใช้ในการแก้ปัญหาของการจัดเรียงและสับเปลี่ยนแบบการเรียกซ้ำก็คือ หากมีชุดตัวเลขจำนวน 4 ตัวคือ 1 2 3 และ 4 ถูกจัดเก็บในรูปสตริงได้เป็น “1234” แล้ว จะเห็นว่าผลลัพธ์ของการจัดเรียงและ

สับเปลี่ยนทั้งหมดจะประกอบด้วยกลุ่มผลลัพธ์ย่อย 4 กลุ่ม ดังนี้

1. กลุ่มที่ขึ้นต้นเลข ‘1’ ตามด้วยผลลัพธ์จากการจัดเรียงและสับเปลี่ยนชุดตัวเลข ‘234’

2. กลุ่มที่ขึ้นต้นเลข ‘2’ ตามด้วยผลลัพธ์จากการจัดเรียงและสับเปลี่ยนชุดตัวเลข ‘134’

3. กลุ่มที่ขึ้นต้นเลข ‘3’ ตามด้วยผลลัพธ์จากการจัดเรียงและสับเปลี่ยนชุดตัวเลข ‘124’

4. กลุ่มที่ขึ้นต้นเลข ‘4’ ตามด้วยผลลัพธ์จากการจัดเรียงและสับเปลี่ยนชุดตัวเลข ‘123’

ซึ่งโดยทั่วไป เราอาจจะพูดได้ว่า ในการหา หรือแสดงผลลัพธ์การจัดเรียงและสับเปลี่ยนสตริงตัวเลขจำนวน n ตัว เราสามารถกระทำได้โดยการดึงเอาตัวเลขทุกตัวออกมาสตริง แล้วจึงทำการแสดงตัวเลขแต่ละตัวตามด้วยผลลัพธ์จากการจัดเรียงและสับเปลี่ยนสตริงตัวเลขที่เหลือจำนวน n-1 ตัว

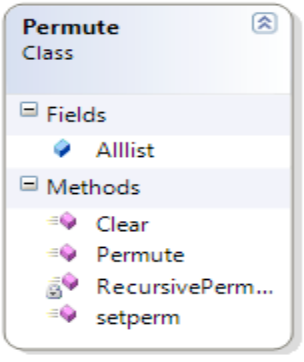
จากแนวความคิดข้างบน สามารถนำไปเขียนเป็นฟังก์ชัน Recursive Permute ที่มีการทำงานแบบเรียกซ้ำได้โดยมี Pseudo Code ดังที่แสดงในชุดคำสั่งที่ 1

ชุดคำสั่งที่ 1 รายละเอียด Pseudo Code ของการหาผลลัพธ์การจัดเรียงและสับเปลี่ยนแบบเรียกซ้ำ

```
void RecursivePermute(string prefix, string rest)
{
if (rest is empty)
{
Display the prefix string.
}
else
{
For each character in rest
{
Add the character to the end of prefix.
Remove character from rest.
Use recursion to generate permutations with the updated values for prefix and rest.
}
}}
```

โดยปกติแล้ว ตอนเริ่มต้นจะกำหนดให้ prefix เป็นสตริงว่าง และ rest เท่ากับสตริงตัวเลขที่ต้องการแสดงผลลัพธ์ทั้งหมดของการจัดเรียงและสับเปลี่ยน ในขณะที่ prefix ยาวขึ้นจะเห็นว่า rest จะมีความยาวสั้นลง จนกระทั้ง rest มีความยาวเท่ากับศูนย์ (Base Case) ก็จะแสดงค่าผลลัพธ์ออกมาหนึ่งค่า

ในงานวิจัยนี้ได้ออกแบบคลาส Permute เพื่อใช้ในโปรแกรมดังแสดงตามภาพที่ 1 โดยการใช้อัลกอริทึมสำหรับหาผลลัพธ์การจัดเรียงและสับเปลี่ยนเป็นแบบเรียกซ้ำดังที่ได้กล่าวไว้ในข้างต้น จากนั้นก็ทำการสร้างคลาส Permute ในภาษา C# ตามรายละเอียดที่แสดงในชุดคำสั่งที่ 2 ซึ่งจะสังเกตเห็นได้ว่าในคลาส Permute จะไม่มีการแสดงผลลัพธ์ทันที แต่จะเก็บผลลัพธ์ทั้งหมดจากการจัดเรียงและสับเปลี่ยนลงในอาเรย์ที่ชื่อ Alllist เพื่อให้โปรแกรมสามารถอ่านค่าผลลัพธ์มาใช้งานภายหลังได้นั่นเอง



รูปที่ 1 คลาสไดอะแกรมของ Permute class

ชุดคำสั่งที่ 2 รายละเอียดคำสั่งภาษา C# ของคลาส Permute ตามที่ได้ออกแบบ

```
class Permute
{
 public ArrayList Alllist;
 public Permute()
 {
 Alllist = new ArrayList();
 }
 public void setperm(string s)
 {
```

```
Clear();
        RecursivePermute("", s);
}
Private void RecursivePermute (string prefix,
string rest)
{
if (rest == " ")
{
Alllist.Add(prefix);
}
else
{
for (int i = 0; i < rest.Length; i++)
{
string newPrefix = prefix + rest[i];
string newRest = rest.Substring(0, i)+rest.
Substring(i + 1); RecursivePermute(newPrefix,
newRest);
}}}
public void Clear()
{
Alllist.Clear();
}}
```

2.1.2 อัลกอลิทึมสำหรับการจัดกลุ่มวงเล็บ

ในงานวิจัยนี้ยังไม่สามารถคิดอัลกอริทึมสำหรับการจัดกลุ่มวงเล็บที่สามารถทำงานอัตโนมัติด้วยตัวโปรแกรมเองได้ ดังนั้นเราจึงดำเนินการจัดกลุ่มวงเล็บด้วยวิธีการเก็บรูปแบบทั้งหมดของกลุ่มวงเล็บไว้ในอาเรย์ของสตริงแทน โดยจำนวนรูปแบบวงเล็บจะได้จากการคำนวณตัวเลขแคทาแลน C_n จำนวนแคทาแลนตัวที่ n สามารถหาได้โดยใช้สูตรสัมประสิทธิ์ทวินาม ดังนี้

$$C_n C_n = \frac{1}{(n+1)} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} \quad (1)$$

สำหรับ n ≥ 0 เมื่อ n เท่ากับจำนวนตัวเลขลบด้วยหนึ่ง ตัวอย่างเช่น สำหรับเกม 24 ที่จำนวนตัวเลข 4 จำนวน สามารถจัดรูปแบบวงเล็บได้ 5 รูปแบบ ซึ่งจะถูกเก็บในอาเรย์ Terms ได้เป็น

```
private string[] terms =
{ "((({0}{4}{1}){5}{2}){6}{3})",
 "(({0}{4}{1}){5}({2}{6}{3}))",
 "(({0}{4}({1}{5}{2})){6}{3})",
 "({0}{4}(({1}{5}{2}){6}{3}))",
 "({0}{4}({1}{5}({2}{6}{3})))"};
```

โดยที่ตำแหน่ง (Place Holder) ที่เป็น {0} {1} {2} และ {3} ในอาเรย์จะถูกแทนด้วยตัวเลขที่สุ่มมาได้สำหรับใช้เป็นโจทย์แต่ละครั้ง ส่วนที่ตำแหน่ง {4} {5} และ {6} ในอาเรย์จะถูกแทนด้วยเครื่องหมายคณิตศาสตร์ที่ถูกอนุญาติให้ใช้ได้ในโจทย์นั้น

2.1.3 อัลกอริทึมสำหรับการจัดการใช้เครื่องหมายคณิตศาสตร์ในการสร้างนิพจน์คณิตศาสตร์

เมื่อเราได้ชุดตัวเลขจากการจัดเรียงและสับเปลี่ยนทั้งหมดตามหัวข้อ 1.1 และได้รูปแบบของการจัดกลุ่มวงเล็บทั้งหมดตามหัวข้อ 1.2 แล้ว ขั้นตอนต่อไปก็คือการสร้างนิพจน์คณิตศาสตร์ขึ้นมาจากรูปแบบวงเล็บและชุดตัวเลขเหล่านั้นโดยการใส่เครื่องหมายบวก ลบ คูณ และหารให้ถูกต้องตรงตำแหน่ง เพื่อให้ได้นิพจน์คณิตศาสตร์ทั้งหมดที่มีความถูกต้องตามหลักการคณิตศาสตร์ (Valid) และจะต้องไม่ซ้ำกันกันด้วย

จากการที่เราได้ทำการจัดเก็บรูปแบบของกลุ่มวงเล็บต่างๆ ลงในอาเรย์ของสตริงและมีการนำ Place Holder มาใช้ระบุตำแหน่งที่จะแทนด้วยตัวเลขแต่ละตัวและเครื่องหมายคณิตศาสตร์ที่อนุญาตให้ใช้ได้ในโจทย์ ทำให้เราสามารถสร้างนิพจน์คณิตศาสตร์ได้โดยการใช้คำสั่ง String.Format (...) เพียงคำสั่งเดียว สำหรับการจัดการกับการเปลี่ยนใส่เครื่องหมายคณิตศาสตร์ในนิพจน์คณิตศาสตร์ที่สร้างขึ้นสำหรับชุดตัวเลขและกลุ่มวงเล็บใดๆ นั้น ก็ทำได้อย่างง่ายๆโดยการใช้โครงสร้างคำสั่งแบบวนลูปซ้อนกันตามจำนวนตำแหน่งของเครื่องหมายที่มีในรูปแบบวงเล็บนั้น

ชุดคำสั่งที่ 3 เป็นตัวอย่างคำสั่งในโปรแกรมที่เขียนขึ้นเพื่อจัดการใส่เครื่องหมายคณิตศาสตร์ของเกม 24 ที่มีจำนวนตัวเลขเท่ากับ 4 จำนวน จะเห็นว่ามีการซ้อนลูป 3 ลูปตามจำนวนตำแหน่งของเครื่องหมายคณิตศาสตร์ที่สามารถมีได้ใน terms[n] (รูปแบบวงเล็บที่ n) สำหรับ

ชุดตัวเลขที่มี 4 จำนวนซึ่งมีผลลัพธ์ของการจัดเรียงและสับเปลี่ยนทั้งหมดเก็บไว้ใน Perms.Alllist

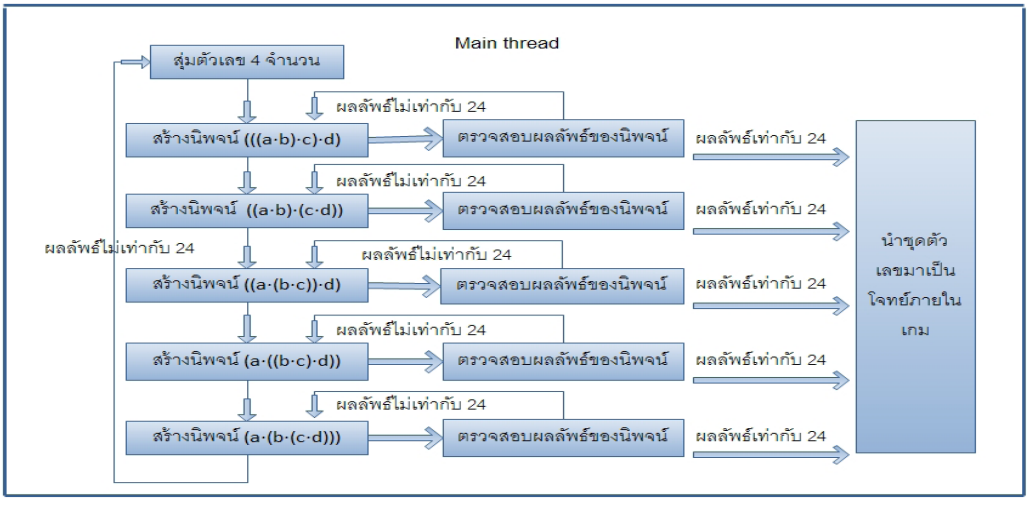
ชุดคำสั่งที่ 3 รายละเอียดคำสั่งภาษา C# ที่ใช้เปลี่ยนเครื่องหมายในการสร้างนิพจน์

```
private char[] ops = { '+', '-', '*', '/' };
foreach (object s in Perms.Alllist)
{
string expTerm = "";
for (int i = 0; i < 4; i++)
for (int j = 0; j < 4; j++)
for (int k = 0; k < 4; k++)
{
 // Create an expression by replacing the place holder the n'th
// member 0f terms with the numbers from the permutation list and
// operators from the character array ops.
expTerm = string.Format(terms[n], s[0], s[1], s[2], s[3], ops[i], ops[j], ops[k]);
// Evaluate the expression here
} }
```

2.2 การออกแบบการทำงานของอัลกอริทึมที่ใช้ในการหาและตรวจสอบคำตอบของเกม 24 แบบ Single-thread และ Multi-thread

2.2.1 การออกแบบการทำงานของอัลกอริทึมที่ใช้ในการหาและตรวจสอบคำตอบของเกม 24 แบบ Single-thread

การทำงานของเกม 24 แบบ Single-thread การทำงานทุกอย่างจะทำงานใน Main Thread โดยโปรแกรมจะทำการค้นหาและตรวจสอบคำตอบของแต่ละนิพจน์เรียงลำดับตามรูปแบบที่ได้ออกแบบไว้ หากไม่เจอคำตอบโปรแกรมจะสุ่มตัวเลขขึ้นมาใหม่จนกว่าจะหาค่าคำตอบได้เท่ากับ 24 แล้วจึงนำเอาชุดตัวเลขที่ได้มาตั้งเป็นโจทย์ ดังแสดงในภาพที่ 2

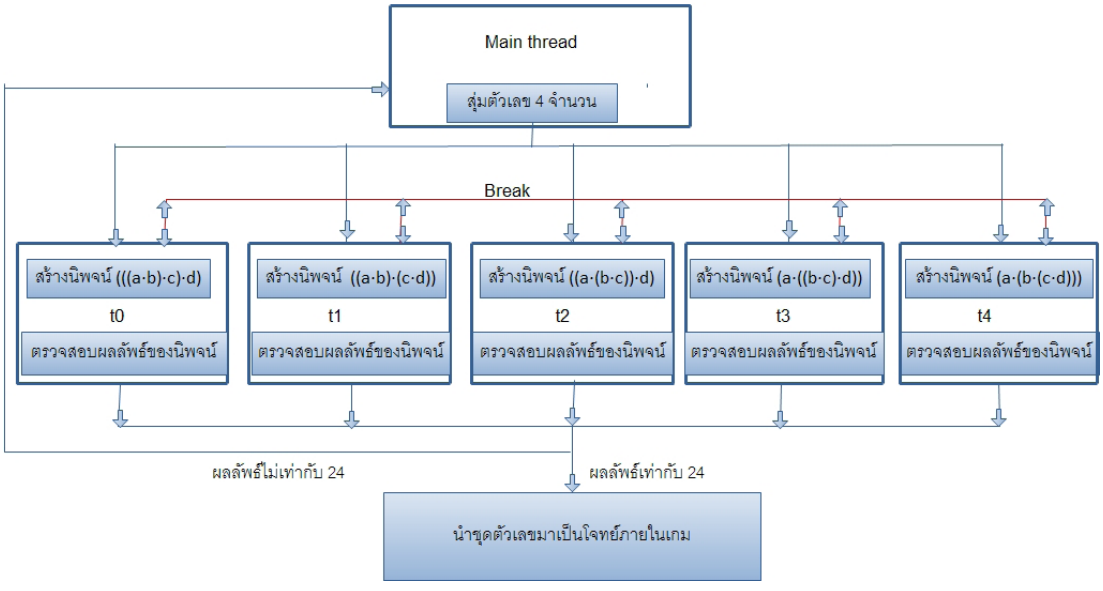


รูปที่ 2 การออกแบบการทำงานของอัลกอริทึมแบบ Single-thread

2.2.2 การออกแบบการทำงานของอัลกอริทึมที่ใช้ในการหาและตรวจสอบคำตอบของเกม 24 แบบ Multi-thread

การทำงานของเกม 24 แบบ Multi-thread การทำงานจะเริ่มต้นการทำงานและสุ่มตัวเลข 4 จำนวนที่ Main Thread โดยนำตัวเลขที่สุ่มได้มาค้นหาและตรวจสอบคำตอบของแต่ละนิพจน์ใน Thread ที่ได้ออกแบบไว้ตามหลักการแคทาแลน แต่ละ Thread เริ่มต้นทำงานไปพร้อมๆกัน โดยที่ Thread ไหนทำการค้นหาและตรวจสอบคำตอบแต่ละนิพจน์เสร็จก่อน จะเริ่มทำการค้นหาและตรวจสอบในนิพจน์ต่อไปก่อนจนกว่าจะเจอคำตอบ แล้วค่อยหยุดการทำงานของแต่ละ Thread โดยส่งสัญญาณ Break ไปให้แต่ละ Thread หยุดการ

ทำงาน หากตรวจสอบครบทุกนิพจน์ไม่เจอคำตอบ โปรแกรมจะสุ่มตัวเลขขึ้นมาใหม่ เพื่อทำการค้นหาและตรวจสอบคำตอบได้เท่ากับ 24 แล้วจึงนำเอาชุดตัวเลขที่ได้มาตั้งเป็นโจทย์ ดังแสดงในภาพที่ 3



รูปที่ 3 การออกแบบการทำงานของอัลกอริทึมแบบ Multi-thread

3. วิธีการทดลอง

วิธีการทดลองการหาและตรวจสอบคำตอบของเกม 24 แบ่งการทำงานออกเป็น 2 แบบ คือ Multi-thread กับ Single-thread ผู้วิจัยได้ทำการทดลองเพื่อเปรียบเทียบประสิทธิภาพการทำงานออกแบบเป็น 3 รูปแบบ ดังนี้

1) การเปรียบเทียบประสิทธิภาพเมื่อกำหนดให้ค่าคำตอบคงที่แต่จำนวนตัวเลขแตกต่างกัน โดยจะทำการนับจำนวนรอบที่ใช้

2) การเปรียบเทียบประสิทธิภาพเมื่อกำหนดให้จำนวนตัวเลขคงที่ ที่ค่าคำตอบแตกต่างกัน โดยการจับเวลาที่ใช้ในแต่ละค่าคำตอบ

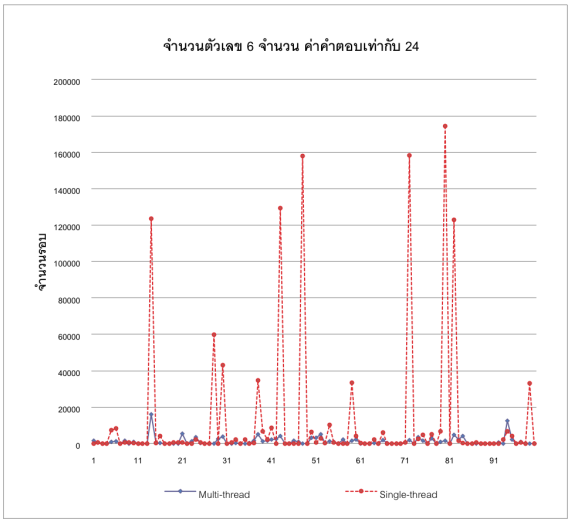
3) การเปรียบเทียบประสิทธิภาพเมื่อกำหนดให้จำนวนตัวตัวเลขและค่าคำตอบคงที่ บนเครื่องคอมพิวเตอร์ที่จำนวน Core และThread ใน CPU ต่างกัน

4. ผลการวิจัยและอภิปราย

ในส่วนนี้จะกล่าวถึงผลการวิจัยและอธิบายผลการทำงานของเกม 24 ที่พัฒนาโดย Windows Presentation Foundation (WPF) ในรูปแบบของ Window Application และการเปรียบเทียบประสิทธิภาพการทำงานแบบ Multi-thread กับ Single-thread

1) การเปรียบเทียบประสิทธิภาพเมื่อกำหนดให้ค่าคำตอบคงที่แต่จำนวนตัวเลขแตกต่างกัน โดยจะทำการนับจำนวนรอบที่ใช้ จากการทดสอบซ้ำ 100 ครั้ง การเปรียบเทียบประสิทธิภาพเมื่อกำหนดให้ค่าของจำนวนตัวเลข 6 จำนวนค่าคำตอบเท่ากับ 24 เปรียบเทียบรอบของการทำงานแบบ Multi-thread กับ Single-thread

ผู้วิจัยกำหนดค่าคำตอบเท่ากับ 24 จำนวนตัวเลข 6 จำนวน ทำการทดลอง 100 ครั้ง โดยทำการสุ่มค่าตัวเลข 6 จำนวนที่มีค่าคำตอบเท่ากับ 24 ในรูปแบบการทำงานแบบ Multi-thread ทำการบันทึกจำนวนรอบในการทำงาน แล้วนำตัวเลขที่สุ่มได้ไปทดลองจำนวนรอบการทำงานในรูปแบบ Single-thread ทำการบันทึกการทำงานและนำข้อมูลที่ได้มาเขียนกราฟเพื่อเปรียบเทียบประสิทธิภาพการทำงาน ดังแสดงในภาพที่ 4

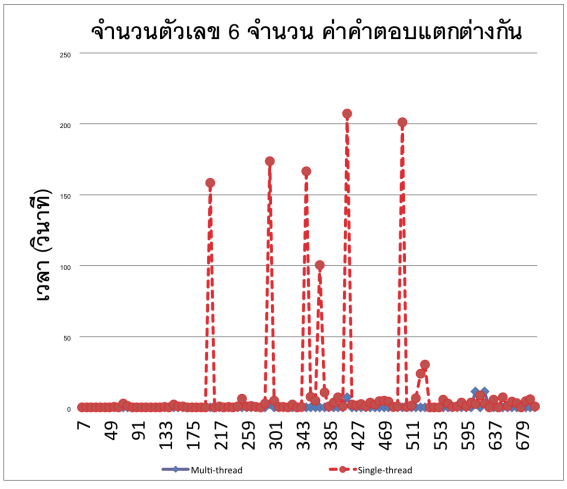


รูปที่ 4 จำนวนตัวเลข 6 จำนวน ค่าคำตอบเท่ากับ 24

ผลการทดลอง จากการทดลอง 100 ครั้ง พบว่าการทำงานแบบ Multi-thread มีจำนวนครั้งของรอบการทำงานน้อยกว่าแบบ Single-thread 39ครั้งและจำนวนรอบที่ใช้ในการทำงานเท่ากัน 22 ครั้ง จากกราฟจะพบว่ารอบการทำงานแบบ Multi-thread อยู่ในช่วง 1-20000 รอบ แต่รอบการทำงานแบบ Single-thread อยู่ในช่วง 1-180000 รอบ อัตราค่าเฉลี่ยของรอบการทำงานแบบ Multi-thread เท่ากับ 1275.75 รอบต่อครั้ง การทำงานแบบ Single-thread ค่าเฉลี่ยของรอบการทำงาน 11909.64 รอบต่อครั้ง จากผลการทดลองแสดงว่าการทำงานแบบ Multi-thread มีประสิทธิภาพดีกว่าแบบ Single-thread คิดแล้วเท่ากับ ((11909.64-1275.75)/11909.64)*100 = 98.28 เปอร์เซ็นต์

2) การเปรียบเทียบประสิทธิภาพเมื่อกำหนดให้จำนวนตัวเลขคงที่ ที่ค่าคำตอบแตกต่างกัน โดยการจับเวลาที่ใช้ในแต่ละค่าคำตอบ

การเปรียบเทียบเวลาในการทำงานเกม 24 ของจำนวนตัวเลข 6 จำนวน ที่ระบุค่าคำตอบที่แตกต่างกัน แล้วจับเวลาในการทำงานแบบ Multi-thread กับ Single-thread ผู้วิจัยกำหนดจำนวนตัวเลข 6 จำนวนค่าคำตอบเพิ่มขึ้นทีละ 7 ทำการทดลอง 100 ครั้ง โดยทำการสุ่มค่าตัวเลข 6 จำนวนที่มีค่าคำตอบเท่ากับค่าที่กำหนดในรูปแบบการทำงานแบบ Multi-thread ทำการบันทึกจำนวนรอบในการทำงาน แล้วนำตัวเลขที่สุ่มได้ไปทดลองจำนวนรอบการทำงานในรูปแบบ Single-thread ทำการบันทึกการทำงานและนำข้อมูลที่ได้มาเขียนกราฟเพื่อเปรียบเทียบประสิทธิภาพการทำงานดังแสดงในภาพที่ 5

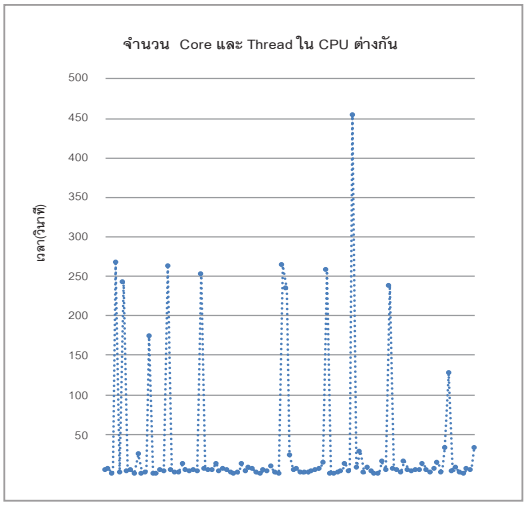


รูปที่ 5 จำนวนตัวเลข 6 จำนวนค่าคำตอบแตกต่างกัน

ผลการทดลอง จากการทดลอง 100 ครั้ง พบว่าการทำงานแบบ Multi-thread มีจำนวนครั้งในการใช้เวลาการทำงานน้อยกว่าแบบ Single-thread 98 ครั้ง จากกราฟจะพบว่าเวลาในการทำงานแบบ Multithread อยู่ในช่วง 0-50 วินาทีแต่เวลาในการทำงานแบบ Single-threadอยู่ในช่วง 0-250 วินาที อัตราค่าเฉลี่ยของเวลาการทำงานแบบ Multithread เท่ากับ 0.318 วินาทีต่อครั้ง การทำงานแบบ Single-thread ค่าเฉลี่ยของเวลาการทำงาน 12.214 วินาทีต่อครั้ง จากผลการทดลองแสดงว่าการทำงานแบบ Multi-thread มีประสิทธิภาพดีกว่าแบบ Single-thread

3) การเปรียบเทียบประสิทธิภาพเมื่อกำหนดให้จำนวนตัวตัวเลขและค่าคำตอบคงที่ บนเครื่องคอมพิวเตอร์ที่จำนวน Core และ Thread ใน CPU ต่างกัน

การเปรียบเทียบเวลาการทำงานของตัวเลข 6 จำนวน ค่าคำตอบเท่ากับ 125 ทำการทดลอง 100 ครั้ง ผู้วิจัยทำการสุ่มค่าตัวเลข 6 จำนวนที่มีค่าคำตอบเท่ากับ 125 โดยจับเวลาการทำงานของ CPU 1 Core1 Thread, 2 Core 2 Thread, 6 Core 6 Thread แล้วบันทึกเวลาการทำงานและนำข้อมูลที่ได้มาเขียนกราฟเพื่อเปรียบเทียบประสิทธิภาพการทำงานดังแสดงในภาพที่ 6



รูปที่ 6 จำนวน Core และ Thread ใน CPU ต่างกัน

ผลการทดลองเวลาในการทำงานของ CPU 1 Core 1 Thread, 2 Core 2 Thread, 6 Core 6 Thread จากกราฟพบว่าเวลาในการทำงานของ 2 Core 2 Thread, 6 Core 6 Thread เวลาทำงานจะอยู่ในช่วง 0-50 วินาที แต่เวลาในการทำงานของ CPU 1 Core 1 Thread จะอยู่ในช่วง 0-450 วินาที อัตราเฉลี่ยของเวลาการทำงานของ CPU 1 Core 1 Thread มีค่าเฉลี่ยเวลาของการทำงาน 32.358 วินาที CPU 2 Core 2 Thread มีค่าเฉลี่ยการทำงาน 3.191 วินาที CPU 6 Core 6 Thread มีค่าเฉลี่ยเวลาการทำงาน 1.903 วินาที จากการทดลองพบว่า CPU มีค่า Core และ Thread มากจะทำให้ประสิทธิภาพการทำงานของโปรแกรมดีขึ้น

5. สรุป

จากการศึกษาการปรับปรุงประสิทธิภาพการทำงานของเกม 24 โดยใช้ Window Presentation Foundation (WPF) ในการเขียนโปรแกรมแบบ Multithread ใน Window Application และทำการเปรียบเทียบการทำงานของเกม 24 ที่ทำงานแบบ Multi-thread กับ Single-thread สามารถสรุปผลการศึกษาได้ดังนี้

การปรับปรุงประสิทธิภาพเกม 24 ที่พัฒนาโดยเทคโนโลยี Window Presentation Foundation (WPF) โดยการเขียนโปรแกรมแบบ Multi-thread ในการค้นหาและตรวจสอบคำตอบจะใช้ความรู้เรื่องการเรียงสับเปลี่ยน การสลับเครื่องหมายและจำนวนแคทาแลนในการพัฒนาอัลกอริทึมการทำงานของเกม 24 โปรแกรมจะทำงานตามอัลกอริทึมที่พัฒนาขึ้นอย่างมีประสิทธิภาพ การแสดงผลการทำงานของโปรแกรมเร็วขึ้น มีความน่าเชื่อถือในขั้นตอนการการตรวจสอบและค้นหาคำตอบตรงตามรูปแบบที่ออกแบบในการพัฒนาและสามารถกำหนดจำนวนตัวเลข 3-6 จำนวนและค่าคำตอบใดๆ ได้

ในการเปรียบเทียบประสิทธิภาพการทำงานของเกม 24 ที่ทำงานแบบ Multi-thread กับ Single-thread ในการนับจำนวนรอบและเวลาในการทำงาน พบว่าการทำงานของโปรแกรมแบบ Multi-thread มีการใช้จำนวนรอบและเวลาในการทำงานน้อยกว่าการทำงานแบบ Single-thread ตั้งแต่ 44% ถึง 88% ผลของการทดลองพบว่า การทำงานของโปรแกรมแบบ Multi-thread มีประสิทธิภาพการทำงานดีกว่าแบบ Single-thread และเกม 24 ที่เขียนโปรแกรมแบบ Multithread จะมีประสิทธิภาพการทำงานดีขึ้นเมื่อนำไปทำงานบนเครื่องคอมพิวเตอร์ที่ CPU มีจำนวน Core และ Thread มาก

6. เอกสารอ้างอิง

[1] Boonsupa P. The development of creative thinking ability test in mathematics on numbers and quantity for mathayom suksal student. [M.Ed]. Ubon Ratchathani:Rajabhat Institute Ubon Ratchathani; 2003. (In Thai).

[2] Teesungnogn P. The comparison of the result of CAI developed by WPF between windows application model and web application model case study : Game 24. [M.Eng]. Khon Kaen: Khon Kean University; 2010. (In Thai).

[3] Sompanis S. Step into a whole new style of WPF programming with Visual Studio 2008 and Expression Blend.Nonthaburi:IDC;2008. (In Thai).

[4] Somsok K. 3D game development with WPF. [M.Eng]. Khon Kaen: Khon Kean University; 2010. (In Thai).

[5] Davis T. Catalan Numbers. 2011 Jan. Available from:URL:http://www. geometer.org/mathcircle /catalan.pdf.