الجمهوريـــــة العربيـــــة الســـــورية
المعهد العالي للعلوم التطبيقية والتكنولوجيا
قســـــم المعلوميـــــات
العام الدراسي 2025/2024

مقرر الحوسبة المتوازية

الوظيفة الأولى

البرمجة المتوازية – النياسب المتعددة

Parallel programming - multithreading

تقديم الطالب

المهنّد ياسر حافظ

01/10/2024

العتاد المستخدم

تم تطبيق اختبارات الأداء على عتاد له المواصفات التالية:

الجدول 1 مواصفات العتاد المستخدم

CPU	Intel® Core™ i5-6200U @ 2.30GHz
Cores	2
Logical processors	4
RAM	12 GB

السؤال الأول

إيجاد الأعداد الأوليّة

تم تطبيق خوارزميتين لإيجاد الأعداد الأولية ضمن مجال ما واختبار أدائها في حالات مختلفة.

1.1- خوارزمية التقسيم إلى مجالات Chunks method

1.1.1- فكرة الخوارزمية

يتم في هذه الخوارزمية العبور على جميع الأعداد الصحيحة في المجال $[2, N]$ والتحقق من كون كل عدد k أولياً أم لا من خلال اختبار قابلية قسمته على جميع الأعداد الصحيحة في المجال $[2, \sqrt{k}]$.

إن تعقيد هذه الخوارزمية في الحالة الأسوأ Worst case هو $O(N\sqrt{N})$.

2.1.1- التحويل إلى خوارزمية متوازية

تقوم الفكرة على تقسيم المجال $[2, N]$ إلى مجموعة من المجالات الجزئية تبعاً لعدد النياسب، ويقوم كل نيسب بإيجاد الأعداد الأولية ضمن مجاله.

من الجدير بالذكر أن الأعداد الأوليّة يتم تخزينها في لائحة List موجودة ضمن الصف ChunksMethod، وبالتالي يقوم كل نيسب بالتعديل على هذه اللائحة؛ ولضمان تحقيق التعديل بشكل متزامن ودون حدوث أخطاء أو تضاربات Conflicts، تم استخدام التعليمة synchronized.

3.1.1- اختبار الخرج

تم إجراء اختبار وحدة Unit test للتحقق من صحة تطبيق هذه الخوارزمية على قيم مختلفة للعدد N وتم تجاوز الاختبار بشكل صحيح.

4.1.1- اختبارات الأداء

تم تطبيق عدد من الاختبارات لتقييم أداء الخوارزمية على قيم مختلفة للعدد N وعدد النياسب.

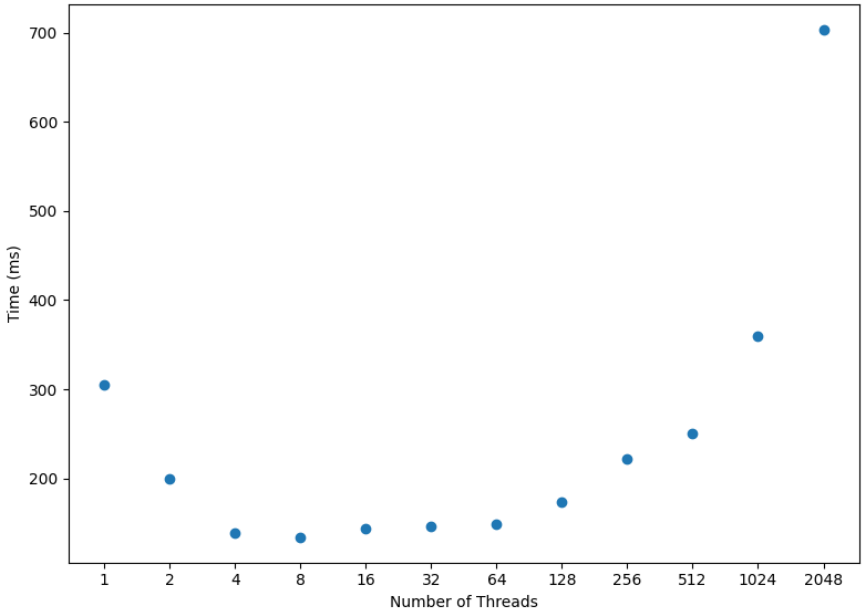
يتم في كل اختبار تطبيق الخوارزمية عدد من المرات وقياس الزمن اللازم للتنفيذ وحساب الزمن الوسطي المستغرق.

ملاحظة: يوجد المزيد من اختبارات الأداء ضمن الرماز لم يتم عرضها جميعها.

1.4.1.1- حالة $N = 10^6$

الجدول 2 نتائج الأداء بحالة $N = 10^6$

Threads	Average time(ms)	Threads	Average time(ms)
1	305	64	149
2	199	128	173
4	139	256	222
8	134	512	250
16	144	1024	359
32	146	2048	703

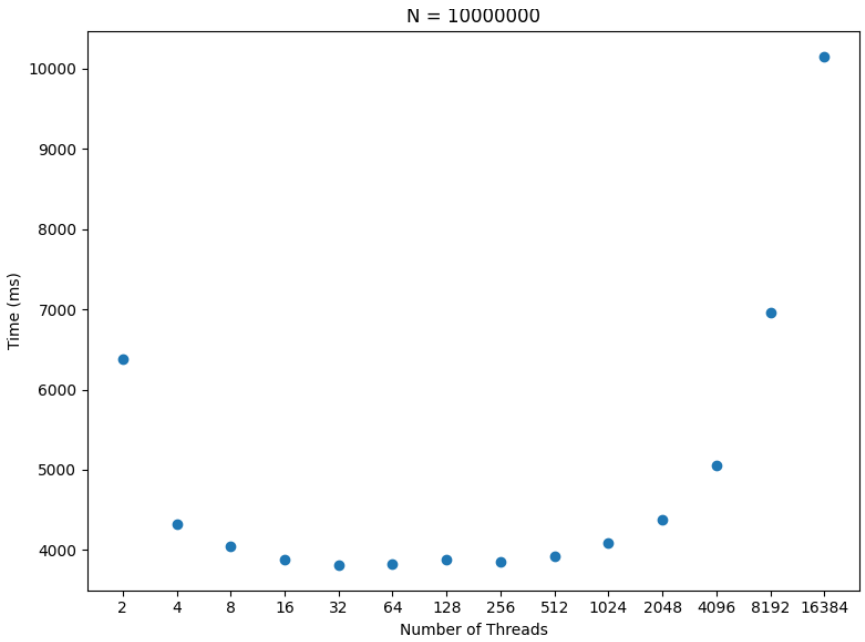


الشكل 1 مخطط الأداء بحالة $N = 10^6$

2.4.1.1- حالة $N = 10^7$

الجدول 3 نتائج الأداء بحالة $N = 10^7$

Threads	Average time(ms)	Threads	Average time(ms)
2	6381	256	3855
4	4329	512	3928
8	4045	1024	4094
16	3884	2048	4380
32	3815	4096	5061
64	3824	8192	6956
128	3885	16384	10148

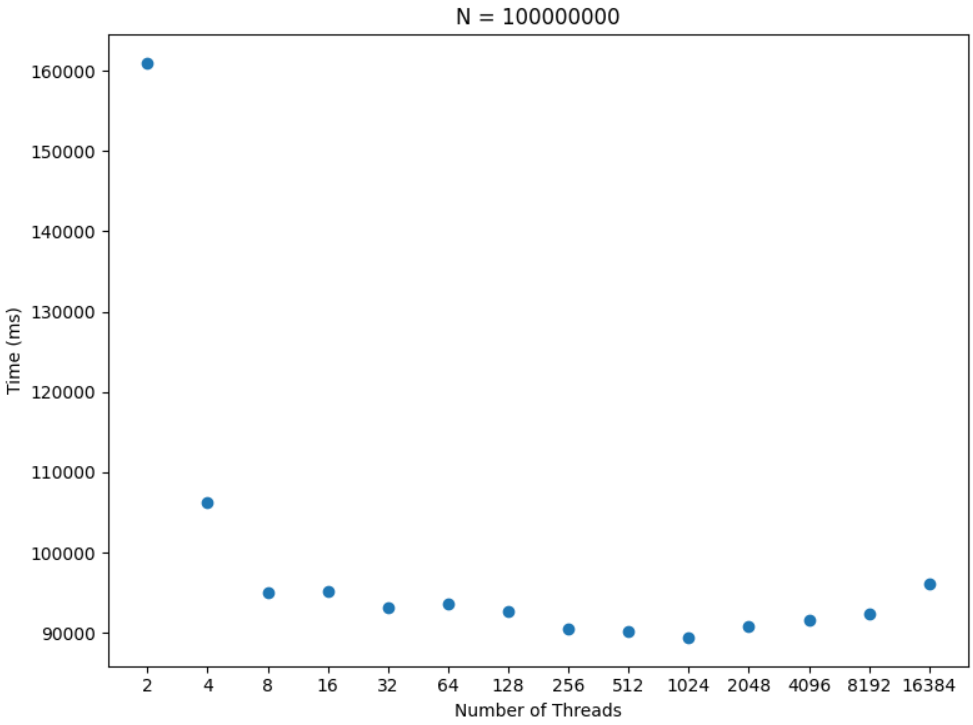


الشكل 2 مخطط الأداء بحالة $N = 10^7$

3.4.1.1- حالة $N = 10^8$

الجدول 4 نتائج الأداء بحالة $N = 10^8$

Threads	Average time(ms)	Threads	Average time(ms)
2	160905	256	90520
4	106176	512	90171
8	94980	1024	89448
16	95150	2048	90859
32	93232	4096	91669
64	93612	8192	92434
128	92616	16384	96146

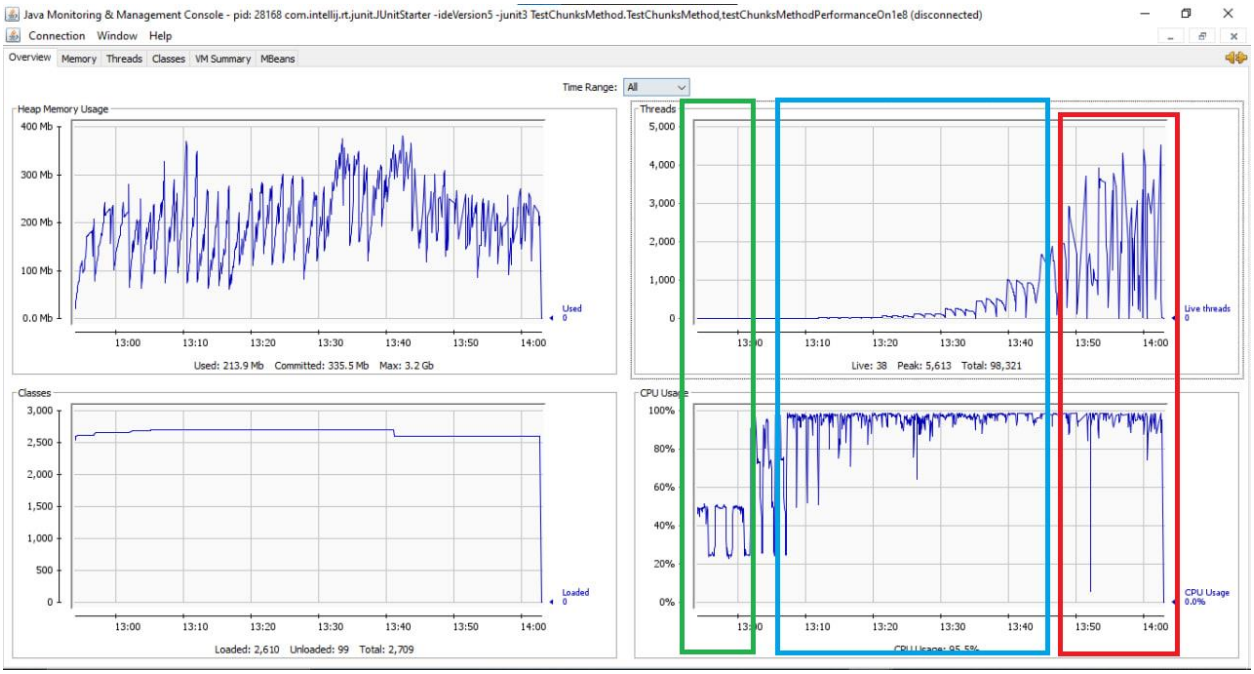


الشكل 3 مخطط الأداء بحالة $N = 10^8$

4.4.1.1- مراقبة الأداء بحالة $N = 10^8$

تم مراقبة أداء إجرائية الاختبار في هذه الحالة باستخدام الأداة JConsole، وتم تدقيق استخدام وحدة المعالجة والذاكرة بالإضافة إلى عدد النياسب المفعّلة Live threads.

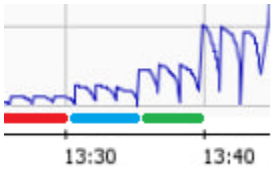
تم ضمن إجرائية الاختبار إيجاد الأعداد الأولية في المجال $[2, 10^8]$ باستخدام عدد مختلف من النياسب في كل مرة، وفي كل مرة كان يتم إيجاد الأعداد الأولية 3 مرات وقياس الزمن اللازم ومن ثم يتم حساب المتوسط.



الشكل 4 مراقبة الأداء في حالة $N = 10^8$

من الجدير بالملاحظة في الشكل السابق هو أنه في بداية تنفيذ الاختبار (باللون الأخضر) كان عدد النياسب قليلاً وبالتالي لم يتم استخدام كامل موارد وحدة المعالجة من قبل الإجرائية، أما مع نهاية الاختبار (باللون الأحمر) نلاحظ تخبطاً في عدد النياسب التي تعمل وذلك لأنه في هذه المرحلة أصبح عدد النياسب كبيراً (8192 و16384) وبالتالي لم يتمكن نظام التشغيل من جدولة هذه النياسب للعمل معاً وذلك بسبب محدودية العتاد (أقصى عدد من النياسب المفعلّة هو 5614 رغم أنه يوجد اختبارات تطلب استخدام عدد أكبر من النياسب) وبالتالي أصبحت جدولة عدد كبير من النياسب تشكل عبء إضافي على عملية التنفيذ ولم يتم تفعيلها معاً (أي أنه تم تأجيل جدولة بعض النياسب) وهذا ما يفسر التخبط في عدد النياسب في مرحلة نهاية الاختبار.

أما في المرحلة الوسطى (باللون الأزرق) فيمكن بوضوح تتبع عدد النياسب التي تعمل في كل مرحلة اختبار (أي في كل مرة نحدد فيها عدد النياسب)، كما يوضح الشكل التالي:



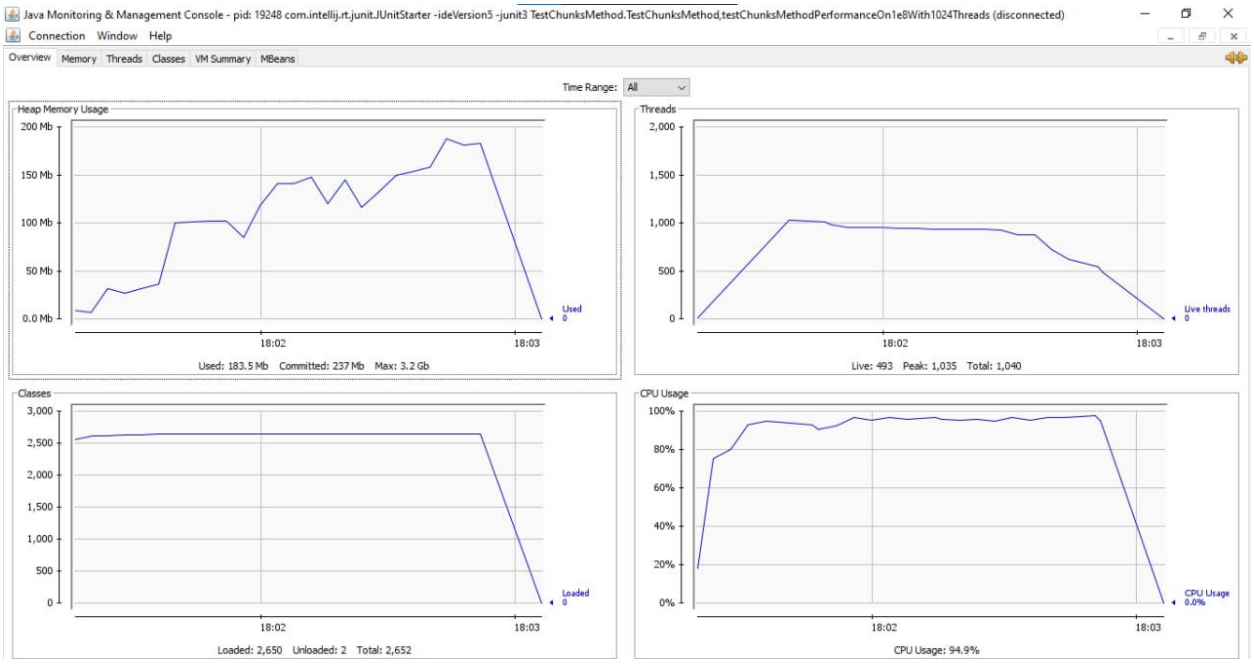
الشكل 5 النياسب المفعلة في بعض مراحل الاختبار

نلاحظ وجود النياسب 3 مرات وهو عدد المرات التي نقوم فيها بتطبيق الخوارزمية.

نلاحظ كذلك انخفاض عدد النياسب تدريجياً نحو الصفر وذلك لانتهاء عملها.

5.4.1.1- مراقبة الأداء بحالة $N = 10^8$ مع 1024 نيسب

تم مراقبة أداء إجرائية الاختبار في الحالة التي حصلنا فيها على أفضل زمن وسطي، وهي حالة استخدام 1024 نيسب.



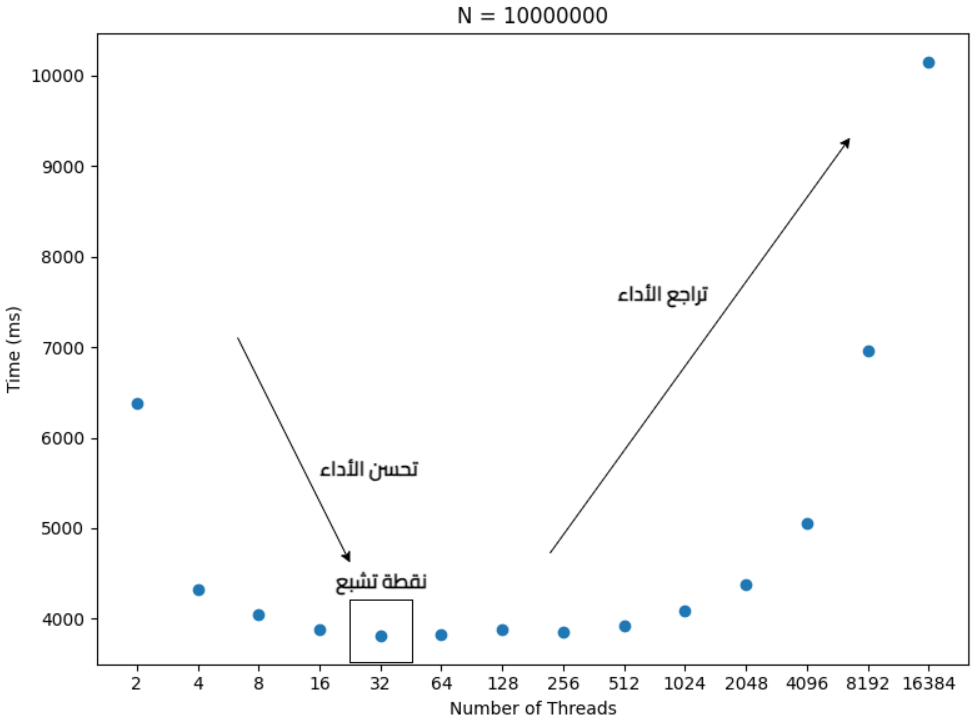
الشكل 6 مراقبة الأداء بحالة $N = 10^8$ مع استخدام 1024 نيسب

نلاحظ من الشكل السابق أنه قد تم استخدام 1024 نيسب وكانت مفعّلة live ومجدولة جميعها (لا تعمل جميعها معاً لأن العتاد لا يسمح بذلك، ولكن يتم جدولتها للعمل معاً)، وأن استهلاك وحدة المعالجة في جميع مراحل التنفيذ كان تقريباً 95% أي أنه يتم استخدام كامل موارد وحدة المعالجة.

بالنسبة لاستخدام الذاكرة نلاحظ تزايده مع تقدم التنفيذ، وذلك بسبب عملية تخزين الأعداد الأولية ضمن لائحة List يتم تخزينها ضمن الذاكرة؛ حيث أنه مع تقدم التنفيذ يتم تخزين المزيد من الأعداد الأولية في اللائحة.

6.4.1.1- تحليل نتائج اختبارات الأداء

نلاحظ من النتائج السابقة أن زيادة عدد النياسب له تأثير إيجابي على الأداء؛ حيث نلاحظ انخفاض زمن التنفيذ مع زيادة عدد النياسب إلى حد معين (نقطة تشبّع)، ولكن عند تجاوز هذا الحد يصبح زيادة عدد النياسب ذو أثر سلبي على الأداء وذلك لأنه يشكل حملاً إضافياً على العتاد بسبب ما يستلزم من عمليات جدولة بالإضافة إلى الاستهلاك الزائد للذاكرة بسبب الحاجة إلى تخصيص بعض الموارد المستقلة لكل نيسب (كالمكدس Stack).



الشكل 7 تغير الأداء تبعاً لعدد النياسب

ضمن هذه الخوارزمية؛ تم إسناد أطوال مجالات متساوية لكل نيسب، وفي رأيي هذه الطريقة هي الأنسب وذلك لأنه لا فرق بين نيسب وآخر ولا يمكننا التحكم بآلية جدولة النياسب ضمن وحدة المعالجة، ويمكننا أن نلاحظ أن استخدام نياسب مع أطوال مجالات كبيرة نسبياً (الجزء الأول من المخطط السابق) يعيدنا إلى حالة مشابهة لحالة البرمجة التسلسلية (مع عبء إضافي ناتج عن تعريف النياسب وجدولتها وانتظارها)، وكذلك استخدام نياسب مع أطوال مجالات صغيرة نسبياً (الجزء الأخير من

المخطط السابق) يؤدي إلى أداء غير مرضٍ بسبب الزيادة الكبيرة في عدد النياسب، لذا من الأفضل استخدام أطوال مجالات متوسطة الطول نسبياً.

كذلك، عند مراقبة الأداء تمت ملاحظة تأثير مواصفات العتاد على أداء البرنامج، وخصوصاً عند ملاحظة عجز نظام التشغيل عن جدولة عدد كبير من النياسب معاً، وهذا الأمر متعلق ببنية وحدة المعالجة وعدد النوى Cores والمعالجات الافتراضية Logical processors ضمنها.

لذا؛ يجب دراسة مواصفات العتاد جيداً قبل التفكير باستخدام البرمجة المتوازية لأن تحقيق أداء جيد بحاجة إلى دراسة جيدة لعدد النياسب الواجب استخدامها بالإضافة إلى حجم العمل المسند إلى كل نيسب بما يتوافق مع مواصفات العتاد من حيث وحدة المعالجة والذواكر.

2.1- خوارزمية غربال إيراتوستين Sieve of Eratosthenes

تم تطبيق خوارزمية غربال إيراتوستين وتحويلها إلى شكل متوازي من خلال تقسيم المجال $[2, N]$ إلى مجموعة من المجالات الجزئية، ويكون كل نيسب مسؤول عن تحديد الأعداد الأولية ضمن مجاله.

تعقيد هذه الخوارزمية في الحالة الأسوأ هو $O(N \log(log(N))$ $time\ complexity$، ولكنها بحاجة إلى حجم ذاكرة إضافي من أجل تخزين كون كل عدد صحيح في المجال $[2, N]$ أولياً أم لا $O(N)$ $memory\ complexity$.

تم اجراء اختبار للتحقق من صحة خرج الخوارزمية وتم تجاوز الاختبار بشكل صحيح.

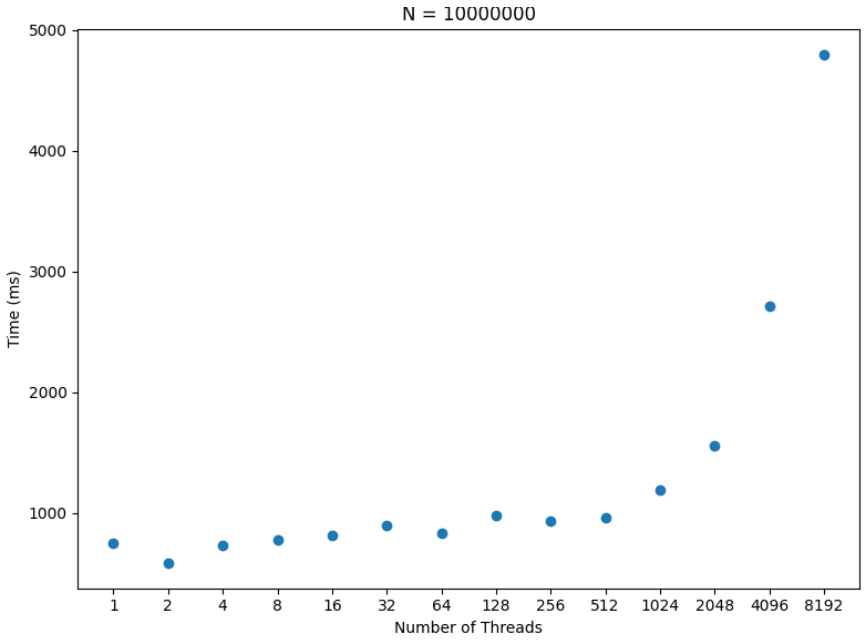
1.2.1- اختبارات الأداء

تم تطبيق اختباري أداء على هذه الخوارزمية بأسلوب مشابه للخوارزمية السابقة.

1.1.2.1- حالة $N = 10^7$

الجدول 5 نتائج أداء خوارزمية غربال إيراتوستين بحالة $N = 10^7$

Threads	Average time(ms)	Threads	Average time(ms)
1	751	128	980
2	585	256	936
4	730	512	956
8	780	1024	1192
16	815	2048	1554
32	894	4096	2710
64	831	8192	4794

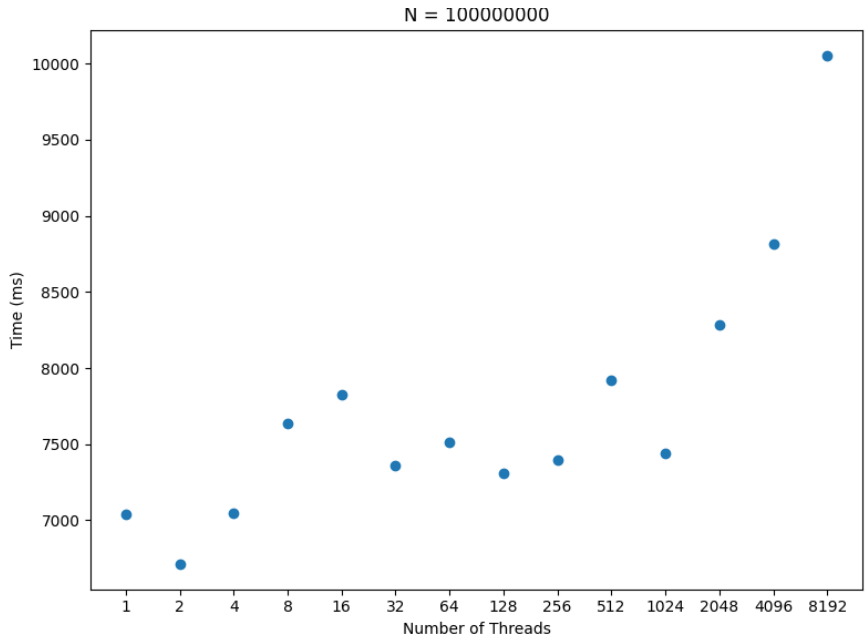


الشكل 8 مخطط أداء خوارزمية غربال إيراتوستين بحالة $N = 10^7$

2.1.2.1- حالة $N = 10^8$

الجدول 6 نتائج أداء خوارزمية غربال إيراتوستين بحالة $N = 10^8$

Threads	Average time(ms)	Threads	Average time(ms)
1	7037	128	7307
2	6714	256	7396
4	7045	512	7918
8	7639	1024	7442
16	7829	2048	8286
32	7362	4096	8818
64	7513	8192	10051



الشكل 9 مخطط أداء خوارزمية غربال إيراتوستين بحالة $N = 10^8$

3.1.2.1- تحليل نتائج اختبارات الأداء

نلاحظ من النتائج السابقة أن أداء هذه الخوارزمية أفضل من أداء الخوارزمية الأولى وهذه نتيجة متوقعة بسبب كون تعقيد هذه الخوارزمية أقل، ولكنها تستهلك حجماً أكبر من الذواكر (فمثلاً لم أتمكن من تطبيق هذه الخوارزمية على حالة $N = 10^9$ بسبب امتلاء الذواكر وتوقف البرنامج عن العمل).

كذلك، نلاحظ تحسن الأداء مع زيادة عدد النياسب، والوصول إلى نقطة تشبع (في حالة استخدام نيسبين)، وعند استخدام عدد نياسب أكبر فإن الأداء يتراجع.

3.1- خلاصة

بعد اختبار الخوارزميتين السابقتين لإيجاد الأعداد الأولية، تبين أن استخدام خوارزمية غربال إيراتوستين أفضل من حيث الأداء ولكنها تستهلك موارد كبيرة نسبياً بسبب الحاجة إلى حجز ذاكرة إضافية.

كذلك، نلاحظ أن الأداء بين الخوارزميتين متقارب عندما تكون قيمة N صغيرة نسبياً (حوالي 10^5)، كما أنه في بعض حالات القيم الصغيرة لـN لا يحقق استخدام البرمجة المتوازية تأثيراً إيجابياً ملحوظاً وفي هذه الحالة يكفي استخدام البرمجة التسلسلية.

من الأفضل التفكير في المعيار الأهم لقياس الأداء (سرعة التنفيذ أم استهلاك الذواكر) عند اختيار أحد الطرق السابقة لإيجاد الأعداد الأولية في مجال كبير نسبياً، ويجب دراسة إمكانيات العتاد بشكل جيد لتحديد العدد الأنسب من النياسب التي يتم استخدامها ضمن كل طريقة وآلية توزيع العمل على كل نيسب.

السؤال الثاني

تعدد المهام في معالجة الصور

تم تطبيق خوارزميتين لتلوين صورة وقياس أداء كل منها في حالات مختلفة.

1.2- طريقة التقسيم الأفقية Slices method

تم تنجيز هذه الطريقة واجراء اختبار Unit test للتحقق من صحة الخرج من خلال مقارنة الصورة التي تم توليدها بالصورة الصحيحة (التي يجب توليدها) من خلال تعريف تابع areImagesEqual يقوم بالمقارنة بين صورتين، وتم تجاوز الاختبار بشكل صحيح.

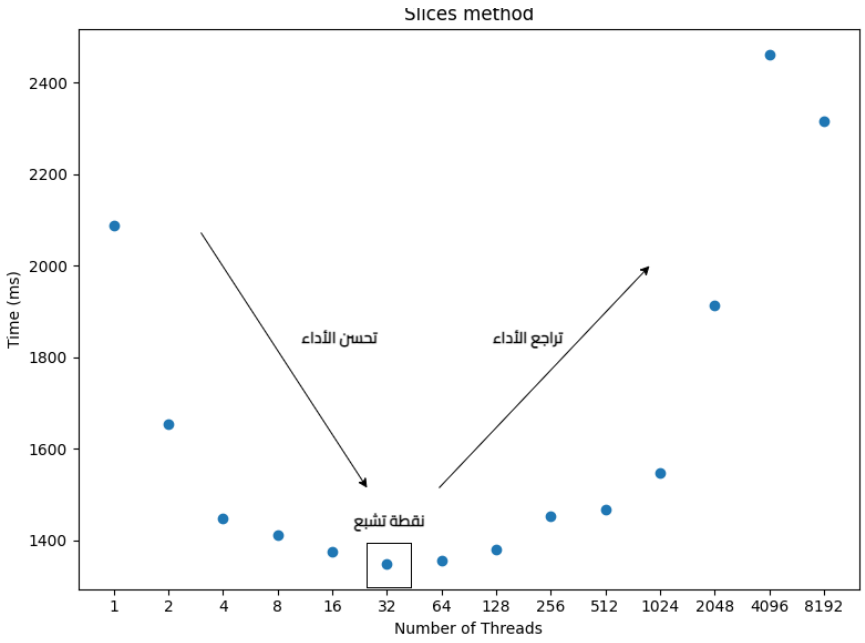
ملاحظة: تم اعتبار الصورة الموجودة ضمن الرماز المرفق بالوظيفة كصورة مرجعية للاختبار.

1.1.2- اختبارات الأداء

تم تطبيق هذه الخوارزمية مع تغيير عدد النياسب المستخدمة وقياس الزمن اللازم للتنفيذ، وفي كل مرة كان يتم تطبيق الخوارزمية عدد من المرات ويتم حساب الزمن الوسطي للتنفيذ.

الجدول 7 نتائج أداء طريقة التقسيم الأفقية

Threads	Average time(ms)	Threads	Average time(ms)
1	2087	128	1379
2	1653	256	1452
4	1449	512	1468
8	1412	1024	1548
16	1376	2048	1914
32	1349	4096	2461
64	1356	8192	2316



الشكل 10 مخطط أداء طريقة التقسيم الأفقية

2.2- طريقة التقسيم إلى كتل Blocks method

تم تنجيز هذه الطريقة بأسلوب مشابه للطريقة السابقة، ولكن بدلاً من أن يقوم كل نيسب بمعالجة شريحة أفقية من الشكل $(0, yStart) \rightarrow (width, yEnd)$ يتم معالجة كتلة من الشكل $(xStart, yStart) \rightarrow (xEnd, yEnd)$.

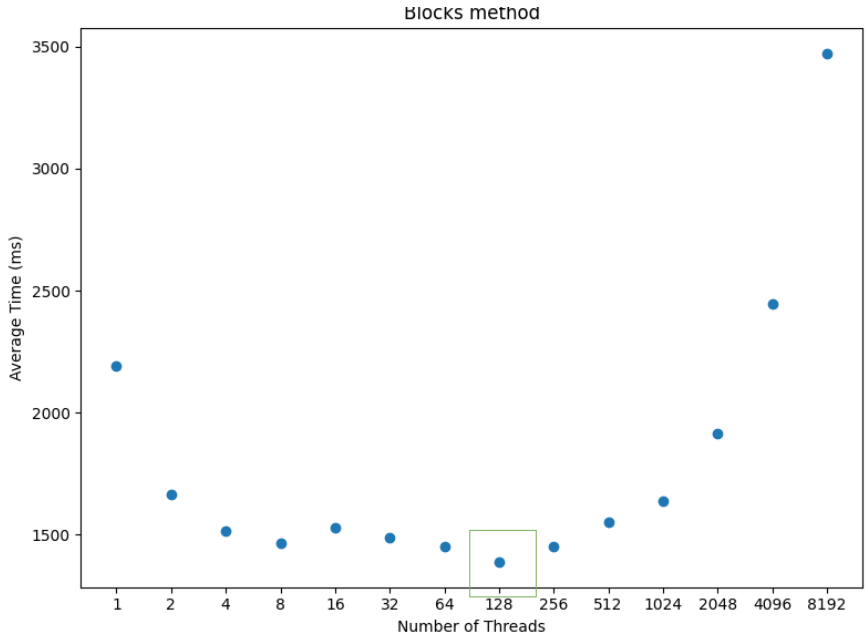
تم اجراء اختبار Unit test للتحقق من صحة الخرج وتم تجاوزه بشكل صحيح.

1.2.2– اختبارات الأداء

تم إجراء اختبارات أداء لهذه الطريقة بشكل مماثل للطريقة السابقة، وكانت النتائج كما يلي:

الجدول 8 نتائج أداء طريقة التقسيم إلى كتل

Threads	Average time(ms)	Threads	Average time(ms)
1	2192	128	1388
2	1663	256	1449
4	1515	512	1552
8	1466	1024	1636
16	1528	2048	1914
32	1485	4096	2446
64	1452	8192	3471



الشكل 11 مخطط أداء طريقة التقسيم إلى كتل

3.2- مناقشة النتائج والمقارنة بين الطريقتين

نلاحظ من النتائج السابقة تحسن الأداء مع زيادة عدد النياسب إلى حد معين (نقطة التشبع، 32 نيسب في طريقة التقسيم الأفقية و128 نيسب في طريقة التقسيم إلى كتل)، ومن ثم يحصل تراجع في الأداء (كما في حالة السؤال السابق).

أما بالنسبة للمقارنة بين الطريقتين فنلاحظ أن نتائج الأداء متقاربة لكل من الطريقتين، ويعود هذا إلى أننا نستخدم صورة واحدة فقط لاختبار أداء الطريقتين ولم يتم اختبار الأداء في حالة صور لها طبيعة مختلفة من حيث توزع العناصر.

بشكل عام، لا يمكن اختيار الطريقة الأفضل بين الطريقتين لأن هذا الأمر يعتمد على طبيعة الصور التي يتم معالجتها بالإضافة إلى نوع المهمة المطلوبة.

أحد عيوب طريقة التقسيم الأفقية، قد نواجه مشكلة في اختلال توزع الحمل بين النياسب؛ حيث أن بعض قد النياسب تعالج جزء أكثر تعقيداً من الصورة (مثلاً أن تحتوي الصورة في الجزء العلوي على سماء وشمس ولا تحتوي على أزهار وبالتالي لن يكون للنيسب المسؤول عن هذا الجزء أي عمل)، أما بحال استخدام طريقة التقسيم إلى كتل مع إجراء دراسة جيدة لآلية توزيع الكتل ضمن الصورة، ستساعد هذه الطريقة في تحقيق توزيع حمل أكثر توازناً بحيث يعالج كل نيسب مزيجاً من المناطق المعقدة والبسيطة ضمن الصورة.

من وجهة نظر أخرى، إن تخزين الصورة ضمن الذاكرة يتم على شكل مصفوفة $2D$، وبالتالي فإن كل نيسب ضمن طريقة التقسيم الأفقية سيتعامل مع مناطق متتالية من الذاكرة، أما عند استخدام طريقة التقسيم إلى كتل، فإن أجزاء الصورة التي يتعامل معها كل نيسب ليست بالضرورة متتالية، وسيؤدي هذا إلى زيادة عمليات تبديل الصفحات وبالتالي قد يسبب مشاكلاً في الأداء واستهلاك الذاكرة.