الجمهوريـــــــة العربيـــــــة الســـــــورية
المعهد العالي للعلوم التطبيقية والتكنولوجيا
قســـــــم المعلوميـــــــات
العام الدراسي 2025/2024

مقرر الحوسبة المتوازية

الوظيفة الأولى

البرمجة المتوازية – النياسب المتعددة

Parallel programming - multithreading

تقديم الطالب

المهنّد ياسر حافظ

03/10/2024

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

ملاحظات

- إن نتائج اختبارات الأداء تقريبية (ليست دقيقة 100%) وذلك لأننا لا نستخدم طريقة دقيقة تماماً لقياس الزمن المستغرق للتنفيذ بالإضافة إلى تأثر اختبارات الأداء بالعديد من العوامل مثل ارتفاع حرارة الجهاز، ووجود برامج أخرى قيد التنفيذ.
- نلاحظ من المواصفات السابقة وجود نواتين ضمن المعالج و4 معالجات افتراضية، وسنلاحظ ضمن اختبارات الأداء استخدام عدد كبير من النياسب (1000 نيسب مثلاً) ونلاحظ عند مراقبة الأداء باستخدام Jconsole أن جميعها live threads أي أنها تعمل معاً على التوازي (في الواقع لا تعمل معاً ولكن سرعة التبديل توحي بذلك)، وهذا الأمر متعلق بآلية تعامل JVM مع تعدد النياسب ومفهوم Virtual threads المستخدم ضمنها؛ حيث أن مفهوم النياسب لا يرتبط بشكل مباشر بعدد نوى وحدة المعالجة عند التعامل مع JVM التي تعمل بالتنسيق مع نظام التشغيل لتتولى مهمة جدولة النياسب للعمل ضمن وحدة المعالجة ويتم التبديل بين النياسب بسرعة بحيث يسمح لعدد من النياسب بالتقدم في عملها حتى لو لم تتمكن جميعها من العمل في نفس الوقت. تفاصيل إضافية عن جدولة النياسب في JVM.

السؤال الأول

إيجاد الأعداد الأوليّة

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

1.1– طريقة 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 الممكن التحكم بأطوال المجالات لكل نيسب، وفي هذا الحل تم تقسيم المجال $[2, N]$ إلى مجموعة من المجالات متساوية الطول.

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 وعدد النياسب 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.

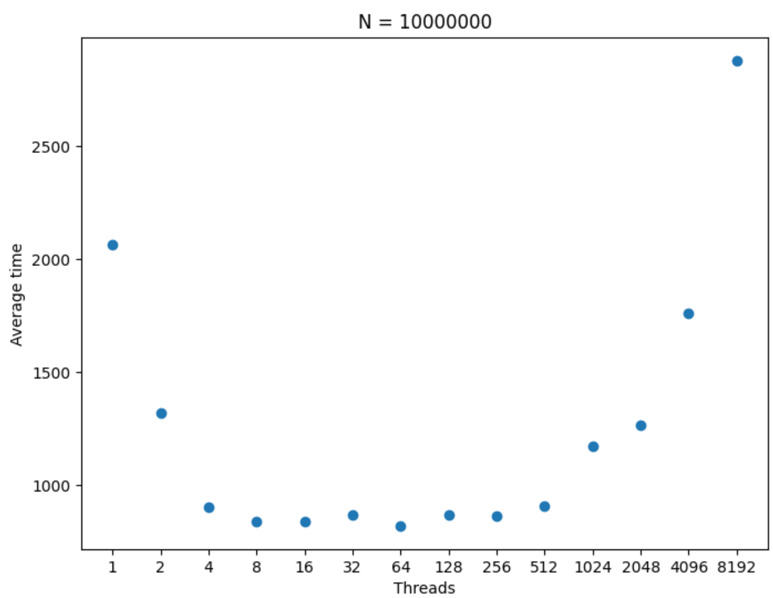
يتم في كل اختبار تطبيق الخوارزمية عدد من المرات وقياس الزمن اللازم للتنفيذ وحساب الزمن الوسطي المستغرق.

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

1.4.1.1– حالة $N = 10^7$

الجدول 2 نتائج أداء طريقة التقسيم إلى مجالات بحالة $N = 10^7$

Threads	Average time(ms)	Threads	Average time(ms)
1	2062	128	870
2	1318	256	865
4	904	512	909
8	841	1024	1175
16	841	2048	1267
32	868	4096	1763
64	821	8192	2877

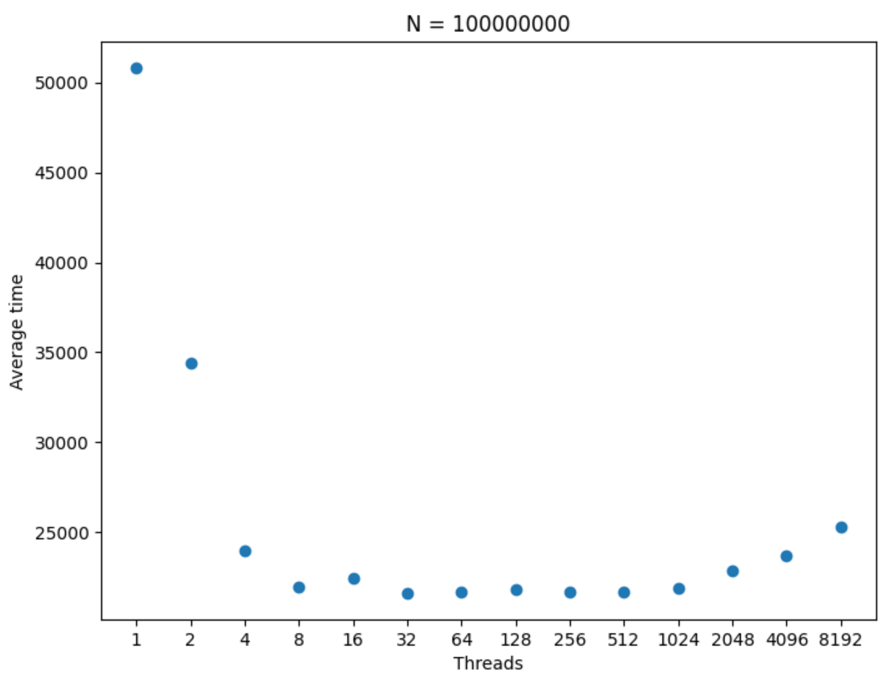


الشكل 1 مخطط أداء طريقة التقسيم إلى مجالات بحالة $N = 10^7$

2.4.1.1– حالة $N = 10^8$

الجدول 3 نتائج أداء طريقة التقسيم إلى مجالات بحالة $N = 10^8$

Threads	Average time(ms)	Threads	Average time(ms)
1	50818	128	21857
2	34432	256	21677
4	23999	512	21718
8	21960	1024	21890
16	22443	2048	22893
32	21614	4096	23671
64	21703	8192	25286

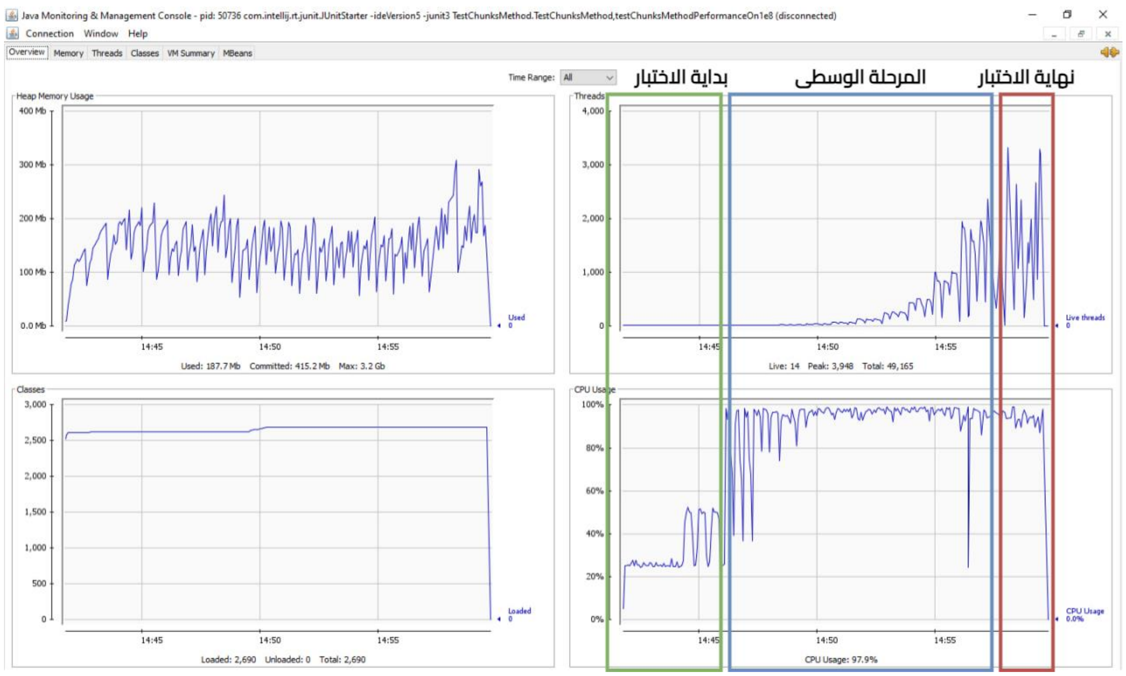


الشكل 2 مخطط أداء طريقة التقسيم إلى مجالات بحالة $N = 10^8$

3.4.1.1– 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.



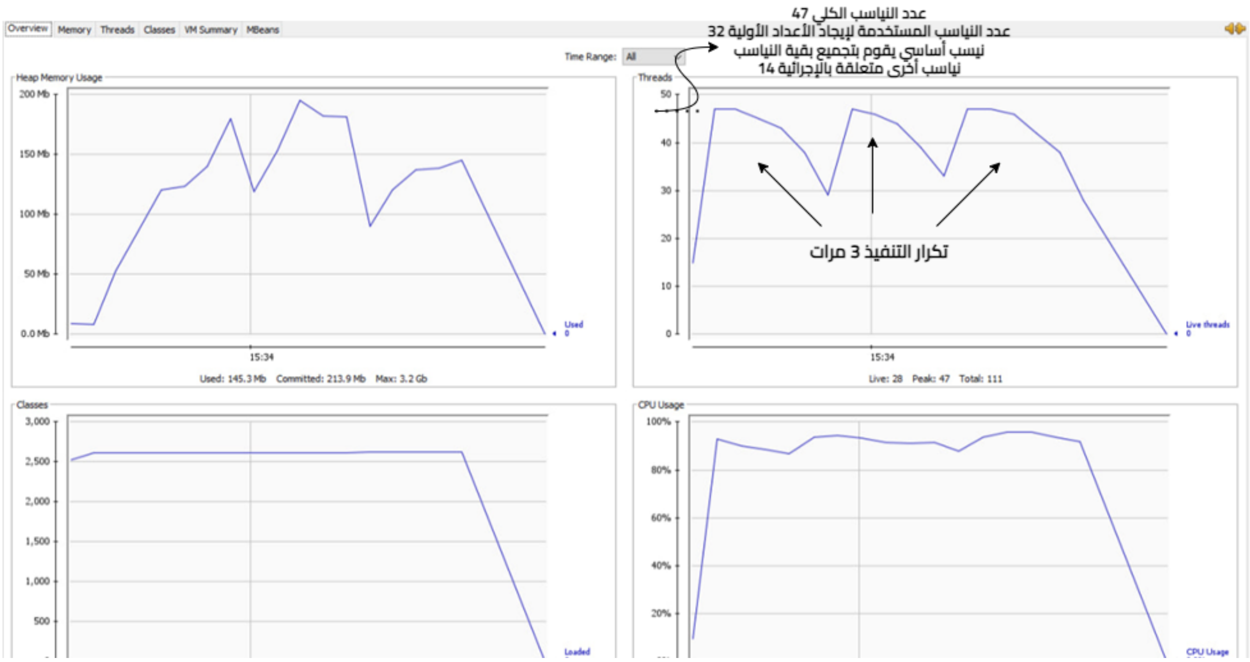
الشكل 3 مراقبة أداء طريقة التقسيم إلى مجالات في حالة $N = 10^8$

من الجدير بالملاحظة في الشكل السابق هو أنه في بداية تنفيذ الاختبار كان عدد النياسب قليلاً وبالتالي لم يتم استخدام كامل موارد وحدة المعالجة من قبل الإجرائية، أما مع نهاية الاختبار نلاحظ تخبطاً في عدد النياسب التي تعمل وذلك لأنه في هذه المرحلة أصبح عدد النياسب كبيراً (8192 و4096) ولم يتمكن نظام التشغيل من جدولة هذه النياسب للعمل معاً وذلك بسبب محدودية العتاد (أقصى عدد من النياسب المفعلّة هو 3948 رغم أنه يوجد اختبارات تطلب استخدام عدد أكبر من النياسب) وبالتالي أصبحت عملية التبديل بين النياسب تأخذ وقتاً أكبر (وقت ملحوظ) ولم تعد تعمل بشكل يوحي بأنها تعمل على التوازي 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)، حيث نلاحظ بداية وانتهاء النياسب 3 مرات وهو عدد المرات التي نقوم فيها بتطبيق الخوارزمية. نلاحظ كذلك انخفاض عدد النياسب تدريجياً نحو الصفر وذلك لانتهاء عملها.

4.4.1.1– مراقبة الأداء بحالة $N = 10^8$ مع 32 نيسب

تم مراقبة أداء إجرائية الاختبار في الحالة التي حصلنا فيها على أفضل زمن وسطي، وهي حالة استخدام 32 نيسب.



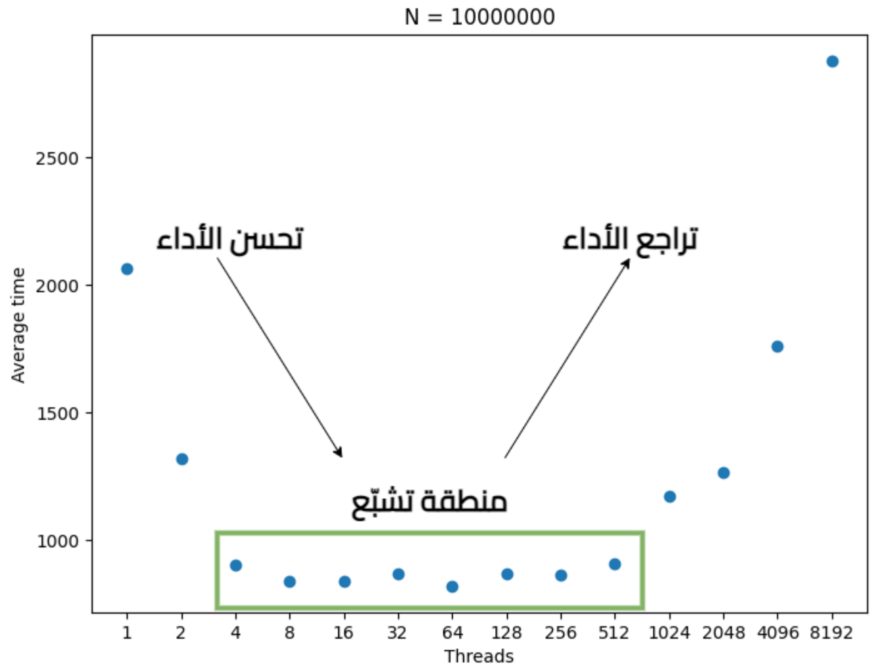
الشكل 4 مراقبة أداء طريقة التقسيم إلى مجالات بحالة $N = 10^8$ مع استخدام 32 نيسب

نلاحظ من الشكل السابق أنه قد تم استخدام 32 نيسب وكانت مفعّلة live ومجدولة جميعها، وأن استهلاك وحدة المعالجة في معظم مراحل التنفيذ كان تقريباً 92% أي أنه يتم استخدام كامل موارد وحدة المعالجة.

بالنسبة لاستخدام الذاكرة نلاحظ تغير استهلاك الذاكرة حسب مرحلة التنفيذ، حيث أنه مع تقدم التنفيذ يتم تخزين المزيد من الأعداد ضمن لائحة الأعداد الأولية، وكذلك يتم تحرير الموارد المحجوزة للنياسب التي انهت عملها وعند الانتهاء يتم تحرير الذواكر المحجوزة.

5.4.1.1– 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\نقاط 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 5 تغير أداء طريقة التقسيم إلى مجالات 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 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– طريقة 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 اختبارات أداء على هذه الخوارزمية بأسلوب مشابه للخوارزمية السابقة.

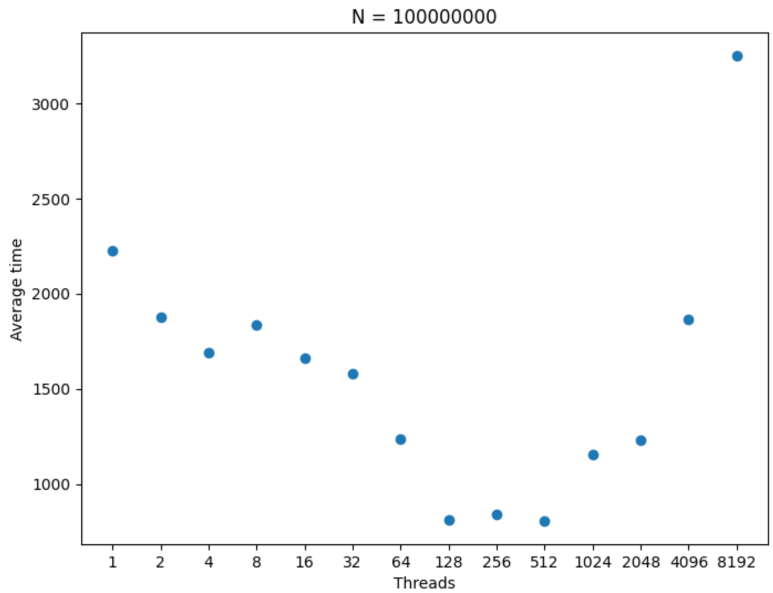
سمحت لنا هذه الخوارزمية بتطبيقها على قيم أكبر (بليون 10^9) وذلك لأنها كانت تأخذ وقتاً معقولاً للتنفيذ، عكس الخوارزمية السابقة.

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

1.1.2.1– حالة $N = 10^8$

الجدول 4 نتائج أداء طريقة غربال إيراتوستين بحالة $N = 10^8$

Threads	Average time(ms)	Threads	Average time(ms)
1	2226	128	809
2	1878	256	841
4	1693	512	806
8	1837	1024	1154
16	1663	2048	1231
32	1582	4096	1868
64	1237	8192	3251

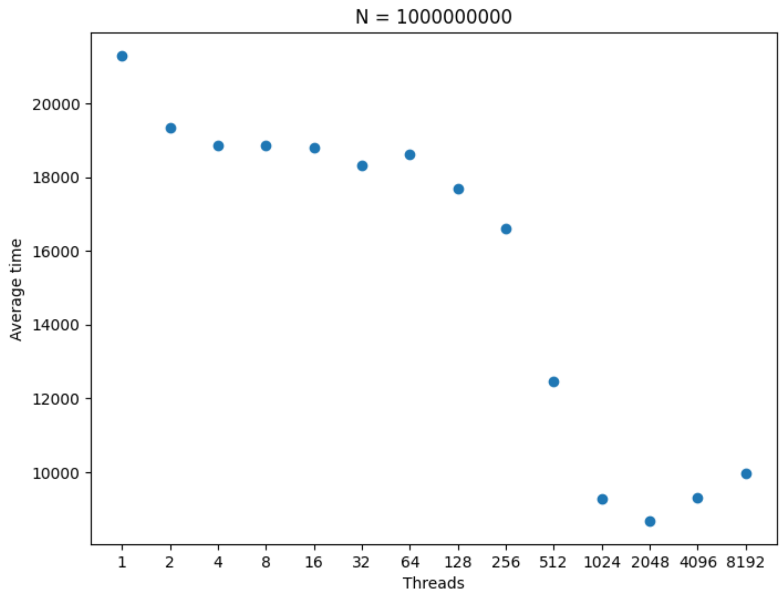


الشكل 6 مخطط أداء طريقة غربال إيراتوستين بحالة $N = 10^8$

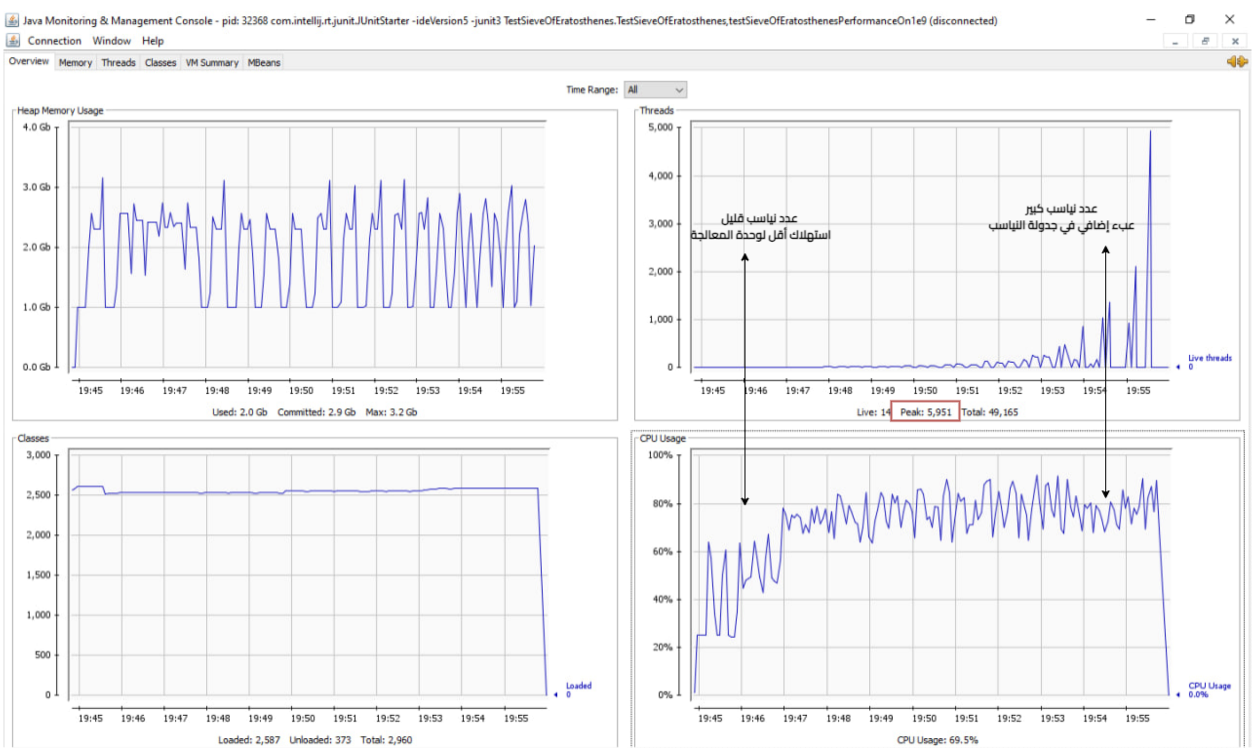
2.1.2.1− حالة $N = 10^9$

الجدول 5 نتائج أداء طريقة غربال إيراتوستين بحالة $N = 10^9$

Threads	Average time(ms)	Threads	Average time(ms)
1	21302	128	17696
2	19362	256	16622
4	18868	512	12474
8	18857	1024	9268
16	18809	2048	8678
32	18334	4096	9318
64	18615	8192	9960



الشكل 7 مخطط أداء طريقة غربال إيراتوستين بحالة $N = 10^9$



الشكل 8 مراقبة أداء طريقة غربال إيراتوستين بحالة $N = 10^9$

3.1.2.1– تحليل نتائج اختبارات الأداء

نلاحظ من النتائج السابقة ومراقبة الأداء أن هذه الخوارزمية أفضل من الخوارزمية الأولى وهذه نتيجة متوقعة بسبب كون تعقيد هذه الخوارزمية أقل، ولكنها تستهلك حجماً أكبر من الذواكر.

كذلك، نلاحظ تحسن تدريجي للأداء مع زيادة عدد النياسب، والوصول إلى نقطة تشبع، وعند استخدام عدد نياسب أكبر فإن الأداء يتراجع كما في حالة الخوارزمية السابقة.

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 أم معايير أخرى) 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 لتغيير الألوان في 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 المولدة من الرماز المرفق بالوظيفة كصورة مرجعية للاختبار.

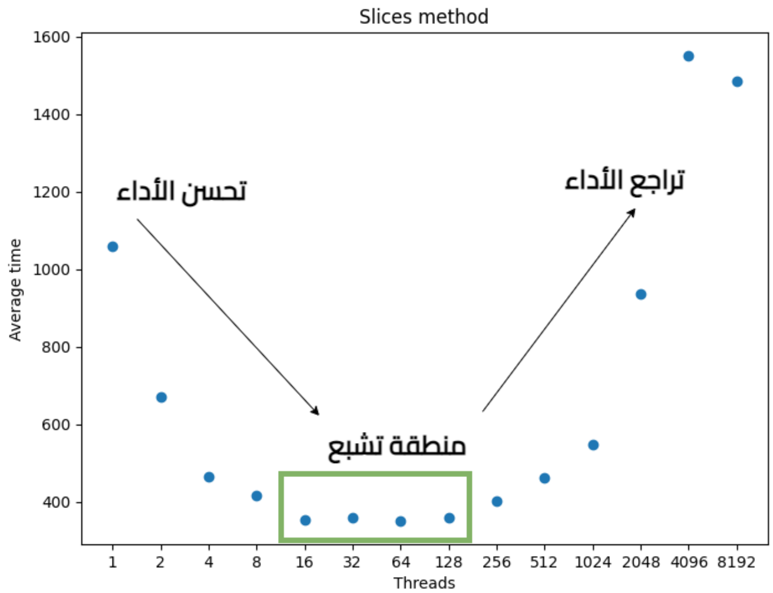
1.1.2– اختبارات الأداء

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.

ملاحظة: الزمن المقاس لا يشمل الزمن اللازم لفتح الصورة وحفظ الصورة الناتجة.

الجدول 6 نتائج أداء طريقة التقسيم الأفقية

Threads	Average time(ms)	Threads	Average time(ms)
1	1060	128	359
2	670	256	403
4	465	512	463
8	417	1024	548
16	354	2048	937
32	360	4096	1551
64	351	8192	1485



الشكل 9 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.

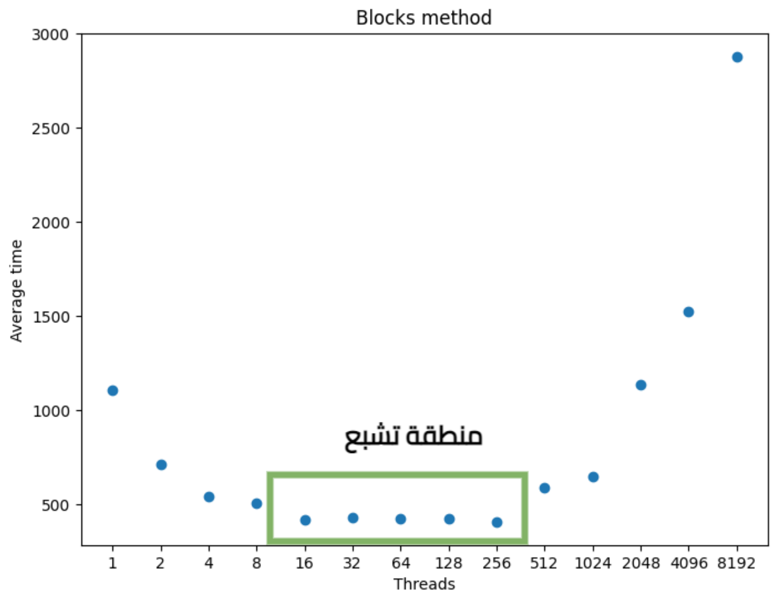
1.2.2– اختبارات الأداء

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

الجدول 7 نتائج أداء طريقة التقسيم إلى كتل

Threads	Average time(ms)	Threads	Average time(ms)
1	1106	128	427
2	713	256	407
4	540	512	588
8	510	1024	651
16	416	2048	1135
32	431	4096	1523
64	425	8192	2879

ملاحظة: الزمن المقاس لا يشمل الزمن اللازم لفتح الصورة وحفظ الصورة الناتجة.



الشكل 10 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)، ومن ثم يحصل تراجع في الأداء (كما في حالة السؤال السابق).

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

نلاحظ كذلك فروقاً في زمن التأخير بين الطريقتين مع استخدام عدد مختلف من النياسب، ويعود ذلك إلى اختلاف آلية التقسيم وتوزيع العمل (الحمل) Load balancing بين الطريقتين.

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

- أحد عيوب طريقة التقسيم الأفقية، هي أنها قد تسبب 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 استهلاك الذواكر مما يؤثر سلباً على الأداء.