Improve the Performance of Multi-core Processors by Accelerating the Execution of Parallel Tasks Using Socket Programming Technology

Majdoleen Husein*

(Received 9 / 11 / 2021. Accepted 23 / 12 / 2021)

□ ABSTRACT □

Socket programming is one of the most modern technologies used in the field of communications, computer networks and processor networks, and it is a method of connecting two nodes on a specific network in order to achieve communication with each other, so that one of the two nodes listens to the socket it has installed in one of its ports, while the other node creates Communication with the socket installed in the first node in order to achieve the connection between the two nodes, and in general the socket is installed on the server side that listens to this socket while the client accesses the server by communicating with this socket, and these nodes in the real world can be Terminal station (computer, smartphone..., etc.).

In this research, the socket programming technique was used in order to achieve communication between multi-core processor cores, instead of using the traditional methods used in parallel programming, which depended on the method of messages Passing between processor cores and this requires the use of other resources such as the operating memory to store these Messages passed between the processor cores, which causes a time delay resulting from accessing the operating memory by the sending core during the writing cycle and the receiving core during the reading cycle, and thus the cost of executing parallel tasks in the multi-core processor will increase, and the main reason for this delay is the difference in speed of communication between the processor cores with each other compared to the speed of communication between one of these cores with the operating memory.

A work scenario involving the implementation of a specific mathematical problem in a parallel manner distributed over the processor cores was also tested using the Riemann sum in the approximation of integration, in order to study and demonstrate the effect of using socket technology in accelerating the execution of parallel tasks and thus improving the performance of multi-core processors in this field.

The processor Intel Core i5-2430m was used to run the work scenario, it is a modern processor that has two separate physical cores, and each of these cores can run two Threads in parallel, for a total of four logical Threads, these four Threads will be used as four stand-alone logical cores that will be used to solve the supposed problem.

Keywords: Multi-core Processors, Socket Programming, Riemann Sum, Parallel Programming.

* **Master, Faculty of Electrical and Mechanical Engineering, Tishreen University, Lattakia, Syria. majdoleen.husein@gmail.com**

تحسين أداء المعالجات متعددة النوى عن طريق تسريع أسلوب تنفيذ المهام التفرعية فيها باستخدام تقنية برمجة المقابس

مجدلين حسين*

(تاريخ الإيداع 9 / 11 / 2021. قُبِل للنشر في 23/ 12 / 2021)

□ ملخّص □

تعد برمجة المقبس من أكثر التقنيات الحديثة المستخدمة في مجال الاتصالات والشبكات الحاسوبية وشبكات المعالجات، وهي عبارة عن طريقة للتوصيل بين عقدتين على شبكة معينة بهدف تحقيق التواصل مع بعضهما البعض، وبحيث تستمع إحدى العقدتين إلى المقبس المثبت لديها في أحد منافذها، بينما تقوم العقدة الأخرى بإنشاء اتصال مع المقبس المثبت في العقدة الأولى لكي يتحقق التوصيل بين العقدتين، وبشكل عام يكون المقبس مثبتاً في جهة الخادم الذي يقوم بالاستماع إلى هذا المقبس بينما يقوم العميل بالوصول إلى الخادم عن طريق الاتصال مع هذا المقبس، ويمكن أن تكون هذه العقد في العالم الحقيقي عبارة عن محطة طرفية (حاسب، هاتف ذكي...، إلخ).

تم في هذا البحث استخدام تقنية برمجة المقابس من أجل تحقيق الاتصال بين نوى المعالج متعدد النوى، بدلاً من استخدام الأساليب التقليدية المتبعة في البرمجة التفرعية، والتي كانت تعتمد على أسلوب تبادل الرسائل بين نوى المعالج الواحد وهذا ما يتطلب استخدام موارد أخرى مثل ذاكرة التشغيل لتخزين هذه الرسائل المتبادلة بين نوى المعالج مما يتسبب في تأخير زمني ناتج عن الوصول إلى ذاكرة التشغيل من قبل النواة المرسلة أثناء دورة الكتابة والنواة المستقبلة أثناء دورة القراءة، وبالتالي سوف تزداد كلفة تنفيذ المهام التفرعية في المعالج متعدد النوى، ويعود السبب الرئيسي في ذلك التأخير إلى الفارق الزمني بين سرعة تخاطب نوى المعالج مع بعضها البعض مقارنة بسرعة تخاطب إحدى هذه النوى مع ذاكرة التشغيل.

كما تم اختبار سيناريو عمل يتضمن تنفيذ مسألة رياضية معينة بأسلوب تفرعي موزع على نوى المعالج باستخدام مجموع ريمان في تقريب التكامل وذلك من أجل دراسة وبيان أثر استخدام تقنية المقابس في تسريع تنفيذ المهام التفرعية وبالتالي تحسين أداء المعالجات متعددة النوى في هذا المجال.

تم استخدام وحدة المعالجة المركزية Intel Core i5-2430m لاختبار سيناريو العمل، وهو معالج حديث يحتوي على نواتين فيزيائيتين منفصلتين، ويمكن لكل من هاتين النواتين تشغيل مسارين على التوازي، ليصبح المجموع أربعة مسارات منطقية، وستستخدم هذه المسارات الأربعة كأربعة نوى منطقية قائمة بذاتها لحل المسألة الرياضية المفترضة.

الكلمات المفتاحية: المعالج متعدد النوى، برمجة المقابس، مجموع ريمان، البرمجة التفرعية.

* **ماجستير، كلية الهندسة الكهربائية والميكانيكية، جامعة تشرين، اللاذقية، سورية.majdoleen.husein@gmail.com**

مقدمة:

تسمح المقابس بالاتصال بين عمليتين مختلفتين على نفس الجهاز أو على أجهزة مختلفة، وتعد طريقة للتواصل مع أجهزة الكمبيوتر الأخرى باستخدام التوابع الموجودة في المكتبات القياسية، وتعتبر المقابس مفيدة لكل من التطبيقات المستقلة والشبكات حيث أنها تسمح بتبادل المعلومات بين العمليات على نفس الجهاز أو عبر شبكة من الأجهزة، كما أنها تتيح الوصول بسهولة إلى البيانات المركزية.

يعد نموذج الخادم\العميل (Server/Client) أحد أكثر نماذج الاتصال استخداماً في الأنظمة المتصلة بالشبكة، ويتواصل العملاء عادة مع خادم واحد في كل مرة.

يحتاج العميل إلى معرفة وجود وعنوان الخادم، ولكن لا يحتاج الخادم إلى معرفة عنوان (أو حتى وجود) العميل قبل إنشاء الاتصال.

إن التطبيقات التي يمكن أن يتم تجزئتها إلى عدة مهام، من وجهة نظر البرمجة الموزعة، يمكن أن يتم التعامل معها على أنها عدة عملاء متصلين بخادم واحد يقوم بتوزيع المهام على هذه العملاء.[5][13]

تتمثل إحدى الخطوات الأولى في تصميم برنامج تفرعي في تقسيم المسألة إلى أجزاء منفصلة من العمل يمكن توزيعها على مهام متعددة بحيث يمكن العمل على حل المسألة في وقت واحد، ويُعرف هذا بتحليل أو تقسيم المسألة.

هناك طريقتان رئيسيتان لتحليل الخوارزمية: تحليل المجال وتحليل التابع، فبالنسبة إلى تحليل المجال، يتم تحليل البيانات المرتبطة بمسألة ما ثم تعمل كل مهمة على جزء من هذه البيانات، ويتم إسناد كل مهمة إلى معالج منفصل بحيث يحصل كل معالج على حصته المتساوية من البيانات وهنا لا يكون لإجراء اتصال بين المعالجات أي داعي، ومن التطبيقات التي يمكن استخدامها في مسألة تحليل المجال هي إيجاد التكامل الرياضي.

أما من ناحية أخرى، وبالنسبة إلى تحليل التابع يكون التركيز على الحساب الذي سيتم إجراؤه بدلاً من البيانات التي تتم معالجتها بواسطة هذا الحساب، ويتم تحليل المشكلة إلى مهام حسب العمل الذي يجب القيام به، ثم تقوم كل مهمة بعد ذلك بجزء من العمل الكلي، ومن الأمثلة على تطبيقات تحليل التابع البحث في الصور أو قواعد البيانات.

في هذا البحث سيتم اختيار طريقة تحليل المجال من أجل إيجاد حل لمسألة رياضية بسيطة تقوم على إيجاد المساحة التي تشكلها إحدى التوابع ضمن مجال محدد، لكن بدلاً من جعل كل نواة تقوم بإيجاد حل للمسألة ضمن مجال محدد (جزئي)، سيتم استخدام مفهوم مجموع ريمان لإيجاد قيمة تقريبية لهذا التكامل الجزئي.

أهمية البحث وأهدافه:

يتمثل الهدف الأساسي من هذا البحث في اختيار مفهوم برمجة المقابس لتحقيق هذا الاتصال وتوظيفه من أجل إيجاد حل لمسألة رياضية تعتمد على مفهوم مجموع ريمان في إيجاد قيمة تقريبية لتكامل بسيط ضمن مجال محدد و لذلك لما يقدمه هذا المفهوم من سهولة في تقسيم المسألة المراد حلها إلى مجموعه من المهام الصغيرة التي يسهل توزيعها.

تعتبر المقابس مفيدة في تطبيقات الشبكات وتلعب المقابس دوراً حيوياً في تطبيقات الخادم\ العميل على وجه الخصوص، يمكن للعميل والخادم التواصل مع بعضهما البعض عن طريق الكتابة إلى هذه المقابس أو القراءة منها، إلا أنه في هذا البحث سيتم العمل على الاستفادة من هذا المفهوم في توزيع المهام بين نوى تنتمي إلى نفس المعالج ويفيد ذلك في إيجاد أسلوب جديد للتخاطب وتبادل المعلومات بين هذه النوى بدلاً من الاعتماد على مفهوم تبادل الرسائل Message Passing المستخدم في البرمجة التفرعية.

طرائق البحث ومواده:

نستعرض في هذا البحث مجموعة من المفاهيم النظرية التي تم التطرق إليها في البحث بالإضافة إلى مجموعة الخطوات التي تتضمن كيفية تنفيذ المسألة الرياضية على معالج متعدد النوى باستخدام تقنية برمجة المقابس.

1. مجموع ريمان(Riemann Sum)

مجموع ريمان هو نوع معين من تقريب تكامل بمجموع محدود، وإحدى التطبيقات الشائعة جداً هي تقريب مساحة التوابع أو الخطوط على الرسم البياني، حيث يتم حساب المجموع عن طريق تقسيم المنطقة إلى أشكال (مستطيلات، شبه منحرف، قطع مكافئ، أو مكعبات) تشكل معاً منطقة مشابهة للمنطقة التي يتم قياسها، ثم حساب المساحة لكل من هذه الأشكال، وأخيراً إضافة كل هذه المناطق الصغيرة معاً، كما يمكن استخدام هذا النهج لإيجاد تقريب عددي لتكامل محدد حتى لو كانت النظرية الأساسية في التفاضل والتكامل لا تجعل من السهل إيجاد حل مغلق الشكل.[2][8][9]

ليكن f:[a,b] هو تابع معرف ضمن المجال المغلق [a,b] من الأعداد الحقيقية و :

$$\{ |X0,X1|,|X1,X2|,\ldots..|Xn-1,Xn| \}=P \quad (1)$$

حيث P هو جزء من المجال بحيث : a=X0<X1<X2<.....<Xn=b.

يعطى مجموع ريمان ضمن هذا المجال وفق العلاقة (2):

$$S = \sum_{i=1}^{n} f\,(x_i')\Delta x_i \quad (2)$$

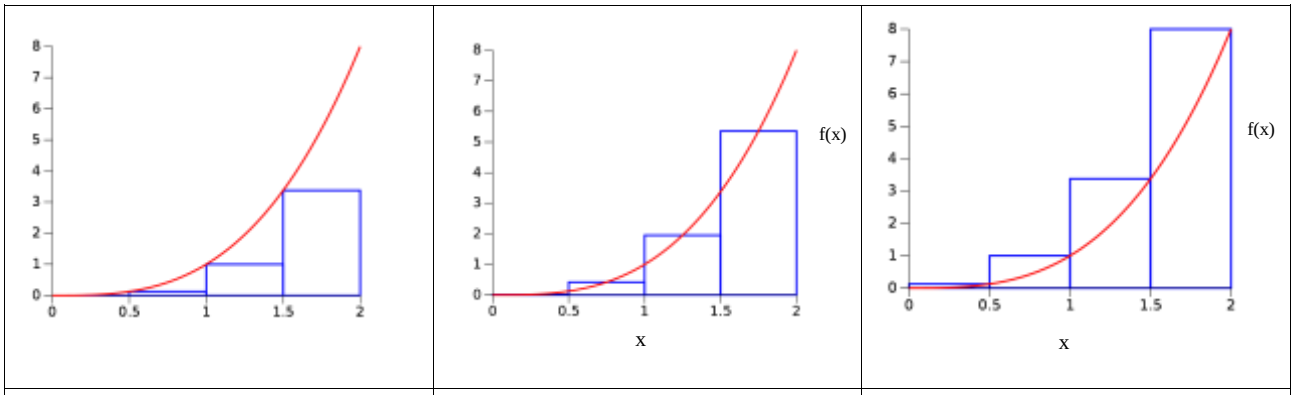
حيث:

$$x_i' \epsilon [x_{i-1}, x_i] \text{ و } \Delta x_i = x_i - x_{i-1}$$

ومن أجل اختيار قيمة x_i' التي تمثل ارتفاع المستطيل أو المعين، في الواقع هنالك عدة طرق للحصول على هذه القيمة و التي تحدد أنواع مختلفة من مجموع ريمان:[6][14]

- Left Riemman Sum: في هذه الحالة يتم اختيار قيمة x_i' مساوية للجزء الأيسر من المجال و هو $x_i' = x_{i-1}$
- Right Riemman Sum: في هذه الحالة يتم اختيار قيمة x_i' مساوية للجزء الأيمن من المجال و هو $x_i' = x_{i+1}$
- Midpoint Riemman Sum: في هذه الحالة يتم اختيار قيمة x_i' مساوية للقيمة التي تقع في وسط المجال و هي: $x_i' = (x_{i-1} + x_{i+1})/2$

يبين الشكل (1) طريقة حساب x_i' للتكامل $f(x) = \int_0^2 x^3\,dx$ في المجال [0 2] من أجل الحالات الثالثة السابقة لمجموع ريمان عبر تقسيم المجال إلى أربع مجالات جزئية.



Left Riemman Sum	Midpoint Riemman Sum	Right Riemman Sum

الشكل(1) حساب ارتفاع المستطيل x_i' حسب الحالات الثلاث لمجموع ريمان

2. توصيف المسألة

يتمثل الهدف الأساسي في إيجاد التكامل للتابع $F(x) = 4x - x^2$ في المجال [0 4] وبالتالي إيجاد المساحة الواقعة تحت المنحني الذي يرسمه التابع و ذلك بالاستفادة من مفهوم مجموع ريمان و Riemman Sum Midpoint على وجه الخصوص، مع الأخذ بالعلم أن القيمة الفعلية التي ينتجها هذا التكامل هي : $\frac{32}{3}$

تعطى العلاقة الأصليه لحساب هذه التكامل ضمن المجال المذكور أعلاه كالتالي:

$$F(x) = \int_0^4 (4x - x^2)\, dx (3)$$

سيتم استخدام مجموع ريمان، الذي يقارب المساحة الفعلية أسفل المنحنى الذي يشكله التكامل المعطى في العلاقة السابقة عن طريق تقسيمه إلى عدة مستطيلات بسيطة، والتي لها الصيغة التالية:

$$f(x) = \sum_{i=0}^{p-1} \left[\sum_{j=0}^{n-1} \left(4a_{ij} - a_{ij}^2\right) * h \right] (4)$$

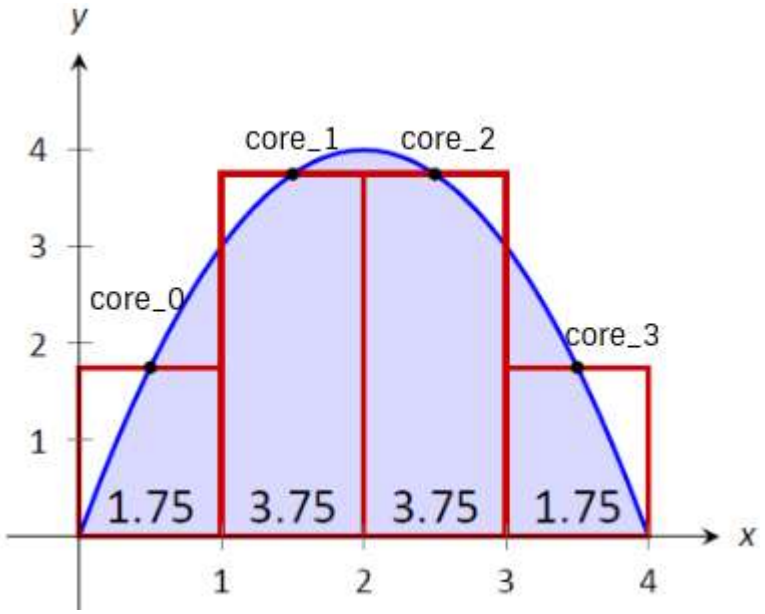
حيث:

$$h = (b - a)/pn \quad (5)$$

$$a_i = a + i * n * h \quad (6)$$

$$a_{ij} = a_i + (j + 0.5) * h \quad (7)$$

p: عدد الأجزاء (عدد النوى) ، h هو ارتفاع المستطيل الذي يشكله كل تكامل جزئي ، n عدد الأجزاء الفرعية في كل جزء والذي يتناسب دقة النتيجة النهائية مع زيادة الأجزاء الفرعية.سيتم تقسيم المنطقة إلى أربعة أجزاء و ستقوم كل نواه بحساب التكامل الجزئي الخاص بها، كما هو مبين في الشكل (2).



الشكل(2) تقريب التكامل المعطى بالعلاقة 2 باستخدام مجموع ريمان

بالنظر إلى الشكل (2) فإن كل من النواتين core_3 , core_0 سيقومان بحساب نفس القيمة، كذلك الأمر بالنسبة إلى النواتين core_2 , core_1

3. التنفيذ

سيتم في هذه الفقرة توصيف الأكواد البرمجية التي ستدخل في التنفيذ العملي للمسألة، يوجد لدينا خمسة أكواد رئيسية تم توصيفها في الملفات التالية:

core_0.py:

ويتضمن تحضير النواة الأولى لفتح المقبس، ثم إرسال التابع (function.py) مع البارامترات المناسبة عندما تتلقى أي استجابة من النوى الثلاثة الأخرى، وأخيراً، حساب النتيجة النهائية بمجرد أن يتم استلام جميع النتائج الجزئية من النوى المتبقة. والجدير بالذكر أن النواة الأولى مسؤولة عن حساب الجزء الخاص بها من مجموع ريمان وإضافته إلى النتائج الجزئية بعد أن يتم استلامها من النوى المتبقية.

core_1.py, core_2.py, core_3.py: يوجه كل ملف النواة للاستماع إلى المقبس الذي تم إنشاؤه بواسطة core_0، واستلام ملف التابع مع بارامتراته، وحساب النتيجة، وإعادتها إلى core_0.

function.py: يحتوي على تنفيذ المعادلات (2) و (3) و (4) و (5).

تم تشغيل core 0.py أولاً ، ثم core 1.py و core 2.py و core 3.py بأي ترتيب. سينتظر core_0 حتى يتلقى التكامل الجزئي النهائي قبل عرض التكامل النهائي.

النتائج والمناقشة:

4. سيناريو العمل والنتائج

عند تشغيل الملفات السابقة على التوازي تم الحصول على التكاملات الجزئية التالية من النوى الأربعة كم هو مبين في الجدول (1).

الجدول (1) التكاملات الجزئية

النواة	التكامل الجزئي المحسوب
core_0	1.6669921875
core_1	3.6669921875
core_2	3.6669921875
core_3	1.6669921875

بالنظر إلى الجدول(1) فإننا نلاحظ أن كل نواة قد قامت بحساب التكامل الجزئي الخاص بها ويمكننا أيضاً ملاحظة تباين التكاملات الجزئية الناتجة و السبب مرتبط بمجال التكامل الذي تم حسابه من أجل كل نواة.

إذا قمنا بحساب مجموع التكاملات الجزئية الظاهرة في الجدول (1) فإن الناتج هو: *10.66796875*.و بالعودة إلى العلاقة الأصلية للتكامل الموضحة في المعادلة(3)، قمنا بحساب التكامل ضمن المجال [0 4] و الناتج هو: *10.666666667*.

يمكننا ملاحظة أنه قد حصلنا على إجابة دقيقة جداً وبدقة = 99.98 % عندما نقارن بين مجموع التكاملات الجزئية والنتيجة الفعلية التي يمكن أن توفرها المعادلة (3) ضمن المجال [0 4].

الاستنتاجات والتوصيات:

تم في هذا البحث إنشاء اتصال بين النوى التي تنتمي إلى نفس المعالج باستخدام مفهوم Socket Programming بدلاً من الاعتماد على المفاهيم التقليدية للبرمجة التفرعية وذلك من أجل حل مسألة تكامل رياضي بناءً على مفهوم مجموع ريمان الذي يعطي قيمة تقريبيه لهذا التكامل، ومن خلال استعراض نتائج هذا لبحث الواردة في سيناريو العمل تبين أن الدقة التي تم الحصول عليها حيث بلغت 99.98 % مقارنة مع النتيجة الفعلية التي تقدمها الطرق التقليدية، بالإضافة إلى السرعة في تنفيذ المهام التفرعية باستخدام هذه التقنية مقارنة مع الطريقة التقليدية مما يجعل منها الحل البديل والأمثل لحل المسائل التفرعية على منصات عمل تتضمن معالجات متعددة النوى.

References:

[1] Comer, Douglas E. Internetworking with TCP/IP, volume 1, Principles, Protocols, and Architecture (fifth edition). Prentice Hall, 2005.
[2] Comer, Douglas E., and Stevens, David L. Internetworking with TCP/IP, volume 2, Design,
Implementation, and Internals (third edition). Prentice Hall, 1999.
[3] Comer, Douglas E., and Stevens, David L. Internetworking with TCP/IP, volume 3, ClientServer Programming and Applications (BSD version, second edition). Prentice Hall, 1996.
[4] Hinden, R., and Deering, S. "IP Version 6 Addressing Architecture." Internet Request for Comments 4291, February 2006.
[5] Deering, S., and Hinden, R. "Internet Protocol, Version 6 (IPv6) Specification." Internet Request for Comments 2460, December 1998.

[6] Gilligan, R., Thomson, S., Bound, J., McCann, J., and Stevens, W. “Basic Socket Interface Extensions for IPv6.” Internet Request for Comments 3493, February 2003.
[7] Srisuresh, P., and Egevang, S. “Traditional IP Network Address Translator (Traditional NAT).” Internet Request for Comments 3022, January 2001.
[8] Mockapetris, Paul. “Domain Names: Concepts and Facilities.” Internet Request for Comments 1034, November 1987.
[9] Mockapetris, Paul. “Domain Names: Implementation and Specification.” Internet Request for Comments 1035, November 1987.
[10] Peterson, Larry L., and Davie, Bruce S. Computer Networks: A Systems Approach (fourth edition). Morgan Kaufmann, 2007.
[11] M-K. Shin, et al. “Application Aspects of IPv6 Transition.” Internet Request for Comments 4038, March 2005.
[12] Postel, John. “Internet Protocol.” Internet Request for Comments 791, September 1981.
[13] Postel, John. “Transmission Control Protocol.” Internet Request for Comments 793, September1981.
[14] Postel, John. “User Datagram Protocol.” Internet Request for Comments 768, August 1980.
[15] Stevens, W. Richard. TCP/IP Illustrated, volume 1, The Protocols. Addison-Wesley, 1994.
[16] Stevens, W. Richard. UNIX Network Programming: Networking APIs: Sockets and XTI (second edition). Prentice Hall, 1997.
[17] Wright, Gary R., and Stevens, W. Richard. TCP/IP Illustrated, volume 2, The Implementation. Addison-Wesley, 1995.
[18] Axler S, "Riemann Integration. In: Measure, Integration & Real Analysis ", Graduate Texts in Mathematics, vol 282. Springer, Cham, 2021
[19] Dr. Riad Daher, "Computer Networks", Tishreen University Publications, 2014.