استراتيجية ذاتية التكيف للتطبيقات المتوازية

نها جبير وسمير جعفر

الملخص

سنقدم في هذا البحث استراتيجية ذاتية التكيف تمكننا من كتابة خوارزمية متوازية تتكيف مع عدد الموارد المتوفرة على البيئة التفرعية المخصصة لتنفيذ البرنامج المتوازي. إن التطبيقات المتوازية المدروسة والمعنية بالبحث هي تطبيقات ممثلة بمخطط تدفق البيانات المبني ديناميكياً خلال التنفيذ. تقوم الطريقة المقترحة هنا على المزاوجة بين خوارزمية تسلسلية وأخرى متوازية معتمدين على مبدأ سرقة العمل في جدولة المهام. ونقدم دراسة لتعقيد هذه الخوارزمية المتكيفة وتحليل لأدائها على p معالج ومقارنته مع خوارزمية تفرعية تقليدية.

الكلمات المفتاحية: الخوارزمية المتوازية، الخوارزمية المتكيفة، مخطط تدفق البيانات، سرقة العمل، المهام، جدولة العمل

Auto Adaptive Strategy for Parallel Applications

N. Jobeir And S. Jafar

Abstract

We introduce an auto adaptive strategy enables to write a parallel algorithm adapts to the number of available resources at allocated parallel environment to execute the parallel program. The parallel applications we are studying which are represented by data-flow graph which built dynamically during the execution. The new suggested strategy is based on coupling of a sequential algorithm and a parallel one and relies on the principle of work stealing in the tasks scheduling. We offer a study of the complexity of the adaptive algorithm and analyze its performance on p processors and compare it with a performance of a classic parallel algorithm.

Keywords: $parallel\ algorithm, Adaptive\ Algorithm, data-flow\ graph, work\ stealing, tasks, work\ scheduling.$

1. المقدمة:

سهّل دخول الحاسوب لكافة مجالات العمل حياة الإنسان وساعد على حل الكثير من المشكلات، ورغم كل التطور الذي طرأ على بنية وسرعة الحواسيب إلا أن بعض المشكلات الضخمة التي تحتاج إلى حسابات كثيرة لم يقوَ على حلها بزمن مقبول كمسائل النمذجة الرياضية والتنبؤ الجوي على سبيل المثال لا الحصر. في نهاية خمسينيات القرن العشرين ظهرت البنى التفرعية ورافقها مفهوم البرمجة المتوازية وكانت بداية عبارة عن بنى بذاكرة مشتركة أي معالجات متعددة تشترك بذاكرة واحدة [5]. في منتصف العقد الثامن منه ظهرت الحواسيب الضخمة وتلاها ظهور عناقيد الحواسيب في أواخره وهي مجموعة من الحواسيب المتماثلة موجودة في بقعة جغرافية واحدة وتتصل مع بعضها بشبكة محلية ثم نشأت الحوسبة الشبكية (Grid) [6] في أوائل التسعينيات وهي عبارة عن مجموعة من الحواسيب أو عناقيد الحواسيب المتصلة مع بعضها بشبكة واسعة عن طريق تمرير الرسائل وتتصف بأنها ديناميكية ومكوناتها غير متجانسة وتعتمد في مجملها على نموذج الذاكرة الموزعة. نتج عن استخدام الحوسبة الشبكية مشكلات عدة منها عدم تجانس المكونات وتغير عدد الحواسيب المشاركة في تنفيذ التطبيقات المتوازية بالزيادة أو النقصان.

2. هدف البحث:

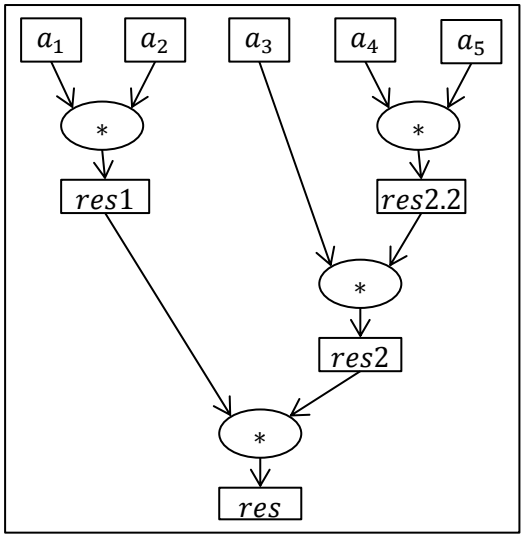
نهدف من هذا البحث إلى تحسين أداء البرامج المتوازية وتقليل زمن تنفيذها على بنية تفرعية ديناميكية من خلال إيجاد طريقة ذاتية التكيف تمكننا من بناء خوارزمية متكيفة تعمل ضمن بيئة تفرعية ديناميكية وتستغل الموارد المتاحة كافة وتَغيُّر هذه الموارد وذلك بكلفة مقبولة معتمدين على مبدأ سرقة العمل في جدولة المهام و مخطط تدفق البيانات لتمثيل حالة التطبيق المتوازي بشكل مجرد وتقديم نموذج رياضي لحساب زمن تنفيذ هذه الخوارزمية المقترحة وبالتالي الحصول على تقدير تقريبي للزمن المتوقع لتنفيذ التطبيق المتوازي باستخدام هذه الاستراتيجية.

3. المفاهيم الأساسية:

تعريف 1.3:

مخطط تدفق البيانات (Data−flow graph) [3]:

هو بيان موجه $G = (V, \Sigma)$ حيث V مجموعة منتهية من الرؤوس وتمثل المهام الحسابية، و Σ مجموعة الأضلاع تمثل اعتمادية البيانات بين المهام. على سبيل المثال يوضح الشكل(1) مخطط تدفق البيانات لتطبيق متوازي يقوم بإيجاد حاصل جداء عناصر متجهة A:



الشكل(1)

تعريف 2.3:

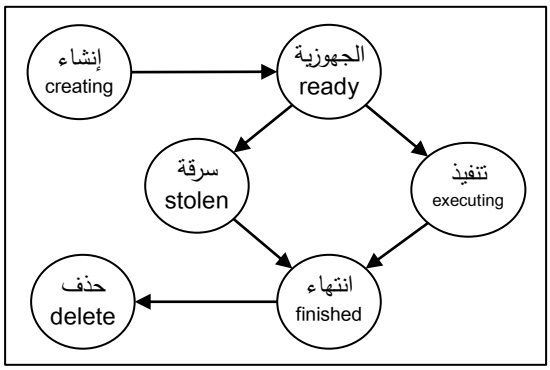
سرقة العمل (Work stealing) [3]: هي خوارزمية جدولة ديناميكية تقوم بتوزيع عبء العمل على المعالجات ومبدؤها أنه عندما ينتهي معالج خفيف (process) من مهامه فإنه يحاول سرقة عمل من معالج خفيف آخر نطلق عليه اسم الضحية ونطلق على الأول اسم السارق.

تعريف 3.3:

دورة تطور المهمة في مخطط تدفق البيانات [3]:

في البداية تدخل المهمة حالة الإنشاء حيث يتم إنشاؤها وتدفع إلى قمة المكدس وعندما تتوفر جميع مدخلاتها تدخل حالة الجهوزية. يمكن للمهمة الجاهزة الموجودة في قمة

المكدس أن تُنفَّذ (أي أنها تبرز من المكدس وتدخل حالة التنفيذ)، كما ويمكن للمهمة الجاهزة أن تُسرَق من معالج خفيف آخر وعندها ستدخل حالة السرقة في المعالج الخفيف المحلي والذي يمثل الضحية حالياً.



الشكل(2) دورة حياة المهمة

وعندما ينتهي تنفيذ المهمة سواءً على الضحية أو السارق فإنها ستدخل مرحلة الانتهاء ومن ثم تنتقل إلى حالة الحذف. ويمثل الشكل(2) دورة حياة مهمة:

تعريف 4.3:

الخوارزمية المتكيفة (Adaptive algorithm) [2]: هي خوارزمية قادرة على تغيير سلوكها تلقائياً وفقاً لسياق تنفيذها وذلك لتحقيق الأداء المطلوب.

5.3 تقنيات التكييف:

يوجد العديد من تقنيات التكييف ولكن في سياقات مختلفة وذلك بالاعتماد على:

- **البيئات المستخدمة:** حيث تدرس تقنيات التسامح مع الأخطاء (fault-tolerance) غالباً تكييف التطبيقات المتوازية على بيئات ديناميكية (يمكن للموارد أن تنضم وتغادر في أي وقت من التنفيذ). وقد تم اقتراح في [4] نموذجاً يستند على المخططات وبآلية تدفق البيانات في التنفيذ المتوازي على بيئات ديناميكية غير متجانسة [2].
- **المسائل المدروسة:** تركز على التكيف على المستوى الحسابي مع افتراض أن البيئة متسامحة مع الأخطاء. هناك تقنيتين من التقنيات غير الحصرية التي يشيع استخدامها في التكييف الحسابي تعتمد على اختيار ومزج الخوارزميات وعلى الجدولة الديناميكية باستخدام سرقة العمل [2].

قد يتواجد عدة خوارزميات لحل مسألة ما ولكل من هذه الخوارزميات قيود (كنوع البيانات وبيئة التنفيذ) وذلك لحل المسألة بفاعلية. اقتُرحت تقنيات اختيار ومزج الخوارزميات لحل المسألة بغض النظر عن نوع البيانات أو بيئة التنفيذ. يسمى النموذج القائم على هذه التقنيات نموذج متعدد الخوارزميات [2].

تعريف 6.3:

نقول عن الخوارزمية أنها متعددة الخوارزميات عندما تستند استراتيجيتها في القرار على الاختيار بين خوارزميتين أو أكثر وكل منها قادرة على حل المسألة نفسها [2].

7.3 تقنيات اختيار ومزج الخوارزميات:

يمكن لاستراتيجية القرار أن تكون:

- ديناميكية [2]: وفيها يتم اتخاذ القرار أثناء التنفيذ.
- ثابتة [2]: وفيها يتم اتخاذ القرار قبل البدء بالتنفيذ.

كما يمكن لاستراتيجية القرار أن تكون:

- خوارزمية اختيار [2]: وذلك عندما يكون الخيار وحيد ولايتغير أثناء التنفيذ وفيها يتم اختيار واحدة من خوارزميتين متوازيتين أو من أكثر من خوارزميتين متوازيتين. ويكون الهدف منها تأدية معيار أحادي.
- خوارزمية مزج [2]: وفيها يتم الاختيار بمزج حلول خوارزميتين أو أكثر لحل المسألة. ويكون الهدف تأدية معايير مختلفة. والفكرة هي أن تعمل عدة خوارزميات لحل المسألة نفسها ولكن كل منها تقوم بتحسين الأداء لمعيار معين. مثلاً لو كان المراد هو تقليل زمن التنفيذ T_p وذلك بهدف تقليل العمل W والعمق D فإن النهج العام يقوم على المزاوجة بين خوارزمية متسلسلة مثلى

لتقليل العمل W وخوارزمية متوازية ذات حجم تقسيمات صغير لتقليل العمق D. وهناك نهجين لتحقيق هذه المزاوجة [2]:

- النهج العودي: وهو تنفيذ خوارزمية متوازية لبعض التقسيمات ومن ثم تنفيذ خوارزمية متسلسلة. وعيب هذه التقنية أنه للوصول إلى عمل أمثل يجب زيادة التقسيمات وبالتالي زيادة العمق بما أنه هو عدد المراحل.
- النهج المتدرج: يجعل عمل الخوارزمية المتوازية W قريب من عمل الخوارزمية المتسلسلة W_seq . ويتم فيه أولاً تنفيذ خوارزمية متسلسلة مثلى لتقليل عدد العمليات ومن ثم تستخدم خوارزمية متوازية لتقليل العمق D. وعيب هذه التقنية هو أن الحجم الكبير للتقسيمات قد يؤدي إلى أن العمل لا يوزع على كل المعالجات الموجودة وبالتالي قد لايقوم بعضها بأي عمل.

لتحقيق تكيف كلي في تقنيات الاختيار والمزج فإنه من الضروري تكييف حجم تقسيمات الخوارزمية المختارة وذلك لأنه من جهة أولى حجم التقسيمات الكبير غير مناسب للمعالجات ذات السرعات المختلفة أو المتغيرة أو من أجل البيانات المختلفة، ومن جهة أخرى حجم التقسيمات الصغير يمكن أن يؤدي إلى تكاليف إضافية للتقسيم. ولذلك سنستخدم تكنولوجيا تأخذ تكييف حجم هذه التقسيمات بعين الاعتبار [2].

4. آلية التكيف المقترحة:

1.4. فرضيات المسألة:

سنعتمد في دراستنا على مبدأ سرقة العمل في جدولة المهام للبرامج الممثلة بمخطط تدفق البيانات باستخدام استراتيجية متكيفة ديناميكياً وتعتمد على مزج خوارزميتين إحداهما تسلسلية والأخرى متوازية آخذين بعين الاعتبار الفرضيات التالية:

- W العمل الذي نريد انجازه بالخوارزمية.
- $Alg\_seq(W)$ خوارزمية تسلسلية تنجز العمل W بكلفة قدرها T_S.

- $Alg\_par(W)$ خوارزمية تفرعية تنجز العمل W على p معالج بكلفة قدرها T_p.

2.4. استراتيجية التكيف المقترحة:

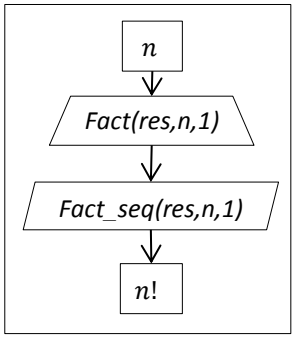
تمثل الخطوات التالية آلية عمل الاستراتيجية المقترحة وذلك في بيئة تفرعية وباستخدام سرقة العمل في جدولة المهام:

- يبدأ تنفيذ العمل W بشكل تسلسلي باستخدام الخوارزمية $Alg\_seq(W)$ على معالج واحد.
- عندما يتوفر معالج آخر خامل عن العمل ويحاول سرقة عمل معالج آخر سيقوم المعالج الضحية بتقسيم العمل المتوفر لديه إلى جزأين باستخدام الخوارزمية $Alg\_par(W)$.
- يرسل المعالج الضحية أحد الجزأين للمعالج السارق.
- يتابع كل منهما الحساب باستخدام الخوارزمية التسلسلية إلى أن يتوفر معالج جديد آخر عندئذ تكرر الخطوات من جديد اعتباراً من الخطوة الثانية.
- ويتكرر نفس السيناريو كلما توفر معالج خامل جديد.

سنقوم بشرح هذه الآلية على مثال حساب العاملي لعدد معطى وليكن n ولنفرض أن الخوارزمية التسلسلية لحساب العاملي هي $Fact\_seq(res,n,s)$ (انظر إلى الملحق) والخوارزمية التفرعية لحسابه هي $Fact\_par(res,n,s)$ (انظر إلى الملحق) وخوارزمية الاختيار والمزج هي $Fact(res,n,s)$ (انظر إلى الملحق) يمثل الوسيط res القيمة التي تعيدها الدالة و n,s يمثلان مجال الأعداد التي سيتم حساب جدائها. وسنعتمد بعض الأشكال الجديدة في مخطط تدفق البيانات حيث سنستخدم متوازي الأضلاع للاستدعاء التسلسلي وشبه المنحرف لاستدعاء خوارزمية المزج. وسنناقش في دراستنا عدة سيناريوهات.

السيناريو الأول:

بفرض لدينا معالج واحد فقط P_0 في بداية التنفيذ وأثناء العمل لم يتوفر معالج آخر في هذه الحالة سيقوم المعالج باستخدام الخوارزمية $Fact\_seq(res, n, s)$ في حساب الـ $n!$ ولن يتم استخدام الخوارزمية $Fact\_par(res, n, s)$ أي أن العمل كاملاً سيتم بشكل تسلسلي على معالج واحد. ويمثل الشكل(3) المخطط التدفقي لهذه الخوارزمية.

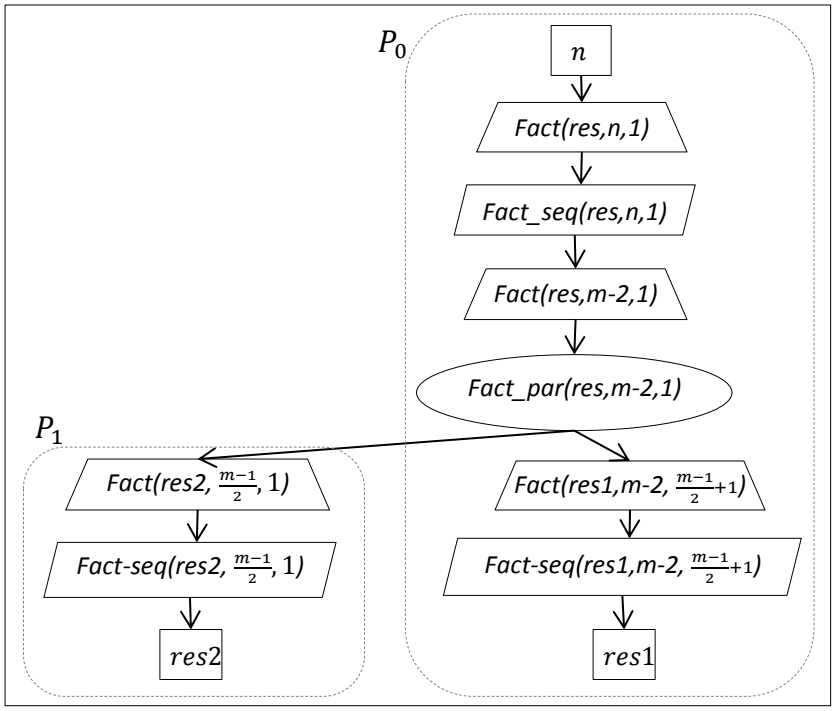


الشكل(3)

السيناريو الثاني:

بفرض عند بداية التنفيذ وجد معالج واحد فقط P_0 عندئذ يبدأ العمل بشكل تسلسلي وبفرض أنه تم حساب جداء الأعداد من n وصولاً إلى $m-1$ وفي لحظة t توفر معالج آخر P_1 وأرسل طلب سرقة للمعالج P_0 عندها سيقوم P_0 بتوليد مهمة تفرعية من العمل التسلسلي المتوفر لديه بحيث تكون مدخلات هذه المهمة متوفرة ويرسلها إلى السارق ويقوم كل منهما بتطبيق الخوارزمية التسلسلية على معطيات جديدة إلى أن ينتهيا من العمل معاً أو إلى أن ينتهي أحدهما فيرسل للآخر طلب سرقة وهكذا يتكرر السيناريو حتى ينتهي العمل ونحصل على النتيجة.

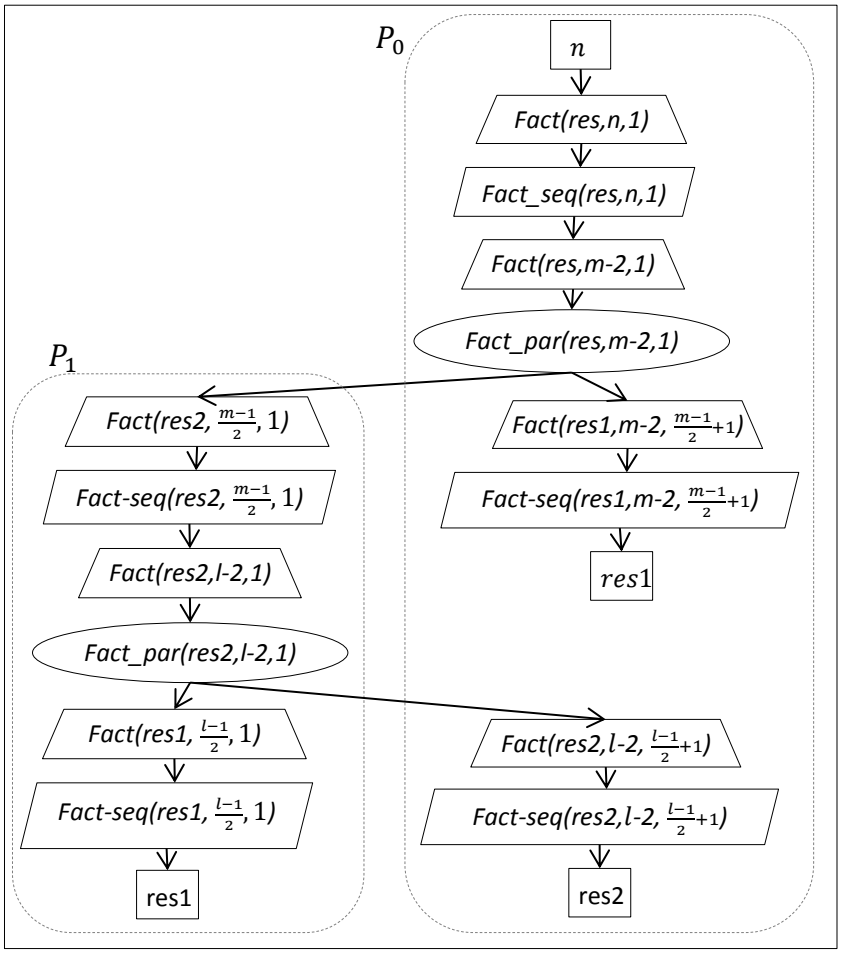
في حال حدوث السرقة مرة واحدة اي عندما يتم كِلا المعالجين عملهما معاً فيكون لدينا المخطط التدفقي الممثل بالشكل(4):



الشكل(4)

وفي حال أنهى أحد المعالجين عمله قبل الآخر في لحظة t_1 ولنفرض أنه المعالج P_0 مثلاً وأن المعالج P_1 قام بحساب ضرب الأعداد حتى $l-1$ عندئذ سيتحول P_0 إلى سارق ويرسل طلب سرقة للمعالج P_1 الذي يقوم بتوليد مهمة تفرعية من العمل التسلسلي المتوفر لديه بحيث تكون مدخلات هذه المهمة متوفرة ويرسلها إلى السارق ويقوم كل منهما بتطبيق الخوارزمية التسلسلية على معطيات جديدة وفي هذه الحالة سيتولد لدينا عمليتي سرقة ويكون المخطط التدفقي الممثل بالشكل(5):

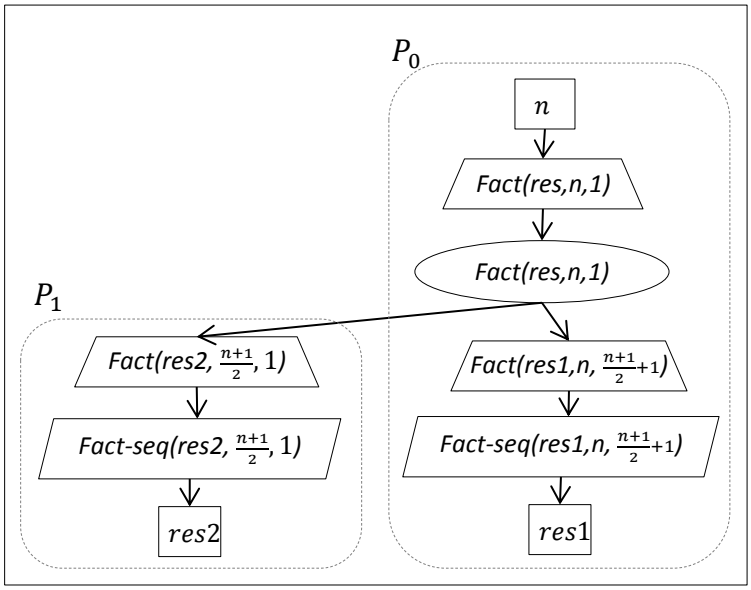
إذا تكررت السرقات سيتكرر نفس السيناريو.



الشكل(5)

السيناريو الثالث:

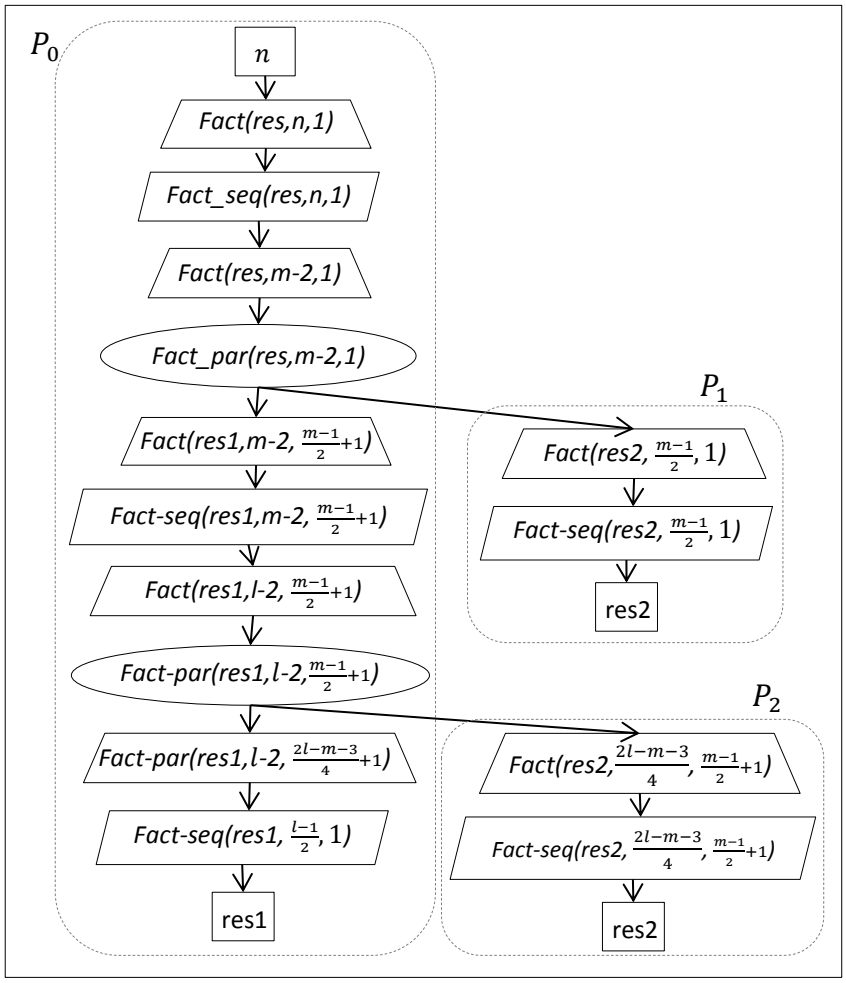
بفرض عند بداية التنفيذ وجد معالجين عندئذ يبدأ أحدهما بالعمل وليكن P_0 حيث ينشئ مهمة تفرعية جزئية من العمل ويرسلها للمعالج الآخر P_1 ويتم العمل كما في السيناريو السابق. وهذا موضح في الشكل(6).



الشكل(6)

السيناريو الرابع:

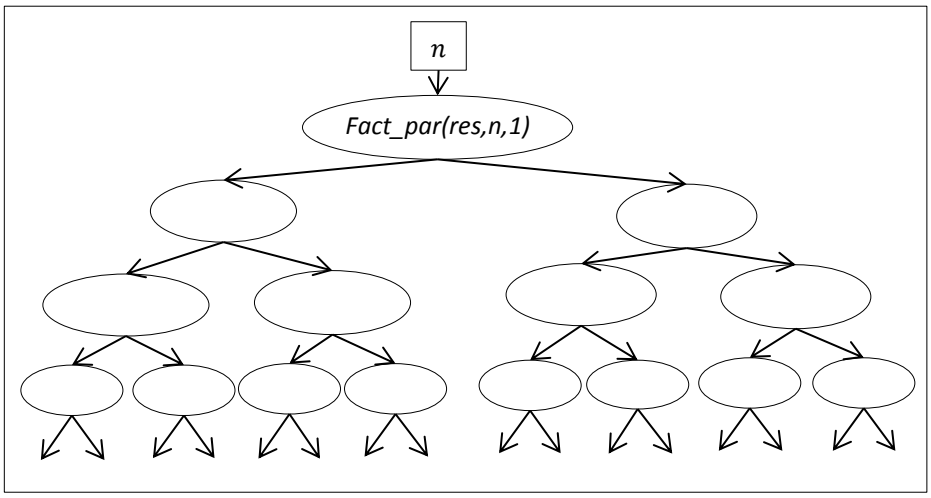
بفرض عند بداية التنفيذ وجد معالج واحد فقط P_0 عندئذ يبدأ العمل بشكل تسلسلي وبفرض أنه تم حساب جداء الأعداد من n وصولاً إلى $m-1$ وفي لحظة t توفر معالج آخر P_1 وأرسل طلب سرقة للمعالج الأول عندها سيقوم المعالج P_0 بتوليد مهمة تفرعية من العمل التسلسلي المتوفر لديه بحيث تكون مدخلات هذه المهمة متوفرة ويرسلها إلى السارق ويقوم كل منهما بتطبيق الخوارزمية التسلسلية على معطيات جديدة وفي اللحظة t_1 أرسل معالج جديد آخر P_2 طلب سرقة للمعالج P_0 الذي قد قام بحساب ضرب الأعداد وصولاً إلى $l-1$ عندئذ سيقوم P_0 بتوليد مهمة تفرعية من العمل التسلسلي المتوفر لديه بحيث تكون مدخلات هذه المهمة متوفرة ويرسلها إلى P_2 ثم يقوم كل منهما بتطبيق الخوارزمية التسلسلية على معطيات جديدة فيما يتابع المعالج P_1 عمله وتستمر المعالجات الثلاثة في عملها إلى أن تنتهي معاً كما في الشكل(7).



الشكل(7)

أو إلى أن ينتهي أحدها فيرسل طلب سرقة لمعالج آخر وهكذا يتكرر السيناريو إلى أن ينتهي العمل ونصل إلى النتيجة.

لو حاولنا أن نقارن مع خوارزمية تفرعية بشكل كامل فسيكون المخطط التدفقي كما هو موضح بالشكل(8) وذلك أياً كان عدد المعالجات:



الشكل(8)

ويستمر تجزيء المهام للوصول إلى الحد الذي وضعته الخوارزمية التفرعية ولو كان لدينا معالج واحد فقط فإنه سيقوم بتقسيم المهام كما في الشكل(8) ويقوم بعدها بالحساب مهمةً مهمة وهذا ماتجنبناه في خوارزميتنا المقترحة.

3.4 التعقيد الزمني للاستراتيجية المقترحة:

سنعتمد في تحليل الخوارزمية على نموذج تكلفة بيئة البرمجة المتوازية XKAAPI [8] والتي تمكننا من بناء تطبيقات متوازية ممثلة بمخطط تدفق البيانات وتعتمد مبدأ سرقة العمل في جدولة المهام والذي يتضمن التعاريف التالية:

T_1 : وهو زمن تنفيذ الخوارزمية التفرعية على معالج واحد.

T_p : وهو زمن تنفيذ الخوارزمية التفرعية على p معالج.

T_∞ : وهو زمن تنفيذ الخوارزمية التفرعية على عدد غير منته من المعالجات ويمثل وقت التنفيذ المرتبط بالمسار الحرج W_∞.

T_s : وهو زمن تنفيذ الخوارزمية التسلسلية.

π_{ave} : هو متوسط سرعة المعالجات.

مبرهنة (1): [2][1]

باحتمالية كبيرة عدد السرقات الناجحة هو $O(pW_\infty)$ وزمن التنفيذ T_p محدود بالمقدار:

$$T_p \leq \frac{W}{p\pi_{ave}} + O\left(\frac{W_\infty}{\pi_{ave}}\right)$$

مبرهنة (2):

باحتمالية كبيرة يكون زمن تنفيذ الخوارزمية المتكيفة المقترحة في المثال السابق محدود بالمقدار التالي:

$$T_p \leq \frac{2p^2+n}{p^2\pi_{ave}} + O\left(\frac{\log p}{\pi_{ave}}\right) \sim_{n\to\infty} \frac{2p^2+n}{p^2\pi_{ave}}$$

البرهان:

كل استدعاء للخوارزمية التسلسلية يكلف عمليتي ضرب.

وكل استدعاء للخوارزمية التفرعية يكلف عمليتين إحداهما ضرب والأخرى قسمة.

وإن $\#tasks \simeq p$ كما سنبين لاحقاً ومنه فإن عدد الاستدعاءات التفرعية هو تقريباً p وتكون تكلفتها $2p$ عملية وسيتم استدعاء الخوارزمية التسلسلية $\frac{n}{2p}$ مرة أي أن عدد العمليات فيها $\frac{n}{p}$ عملية. فيكون لدينا:

$$W = 2p + \frac{n}{p} = \frac{2p^2+n}{p} \quad , \quad W_\infty = \log p$$

ومنه بالتعويض في المبرهنة(1) نجد:

$$T_p \leq \frac{2p^2+n}{p^2\Pi_{ave}} + O\left(\frac{\log p}{\Pi_{ave}}\right) \sim_{n\to\infty} \frac{2p^2+n}{p^2\Pi_{ave}}$$

وهو المطلوب.

ولأجل خوارزمية متوازية تقليدية لحساب العاملي (انظر Fact_par في الملحق) يكون:

$$W = n-1 \ , \ W_\infty = \log n$$

وبالتعويض في المبرهنة(1) نجد:

$$T_p \leq \frac{n-1}{p\Pi_{ave}} + O\left(\frac{\log n}{\Pi_{ave}}\right) \sim_{n\to\infty} \frac{n-1}{p\Pi_{ave}}$$

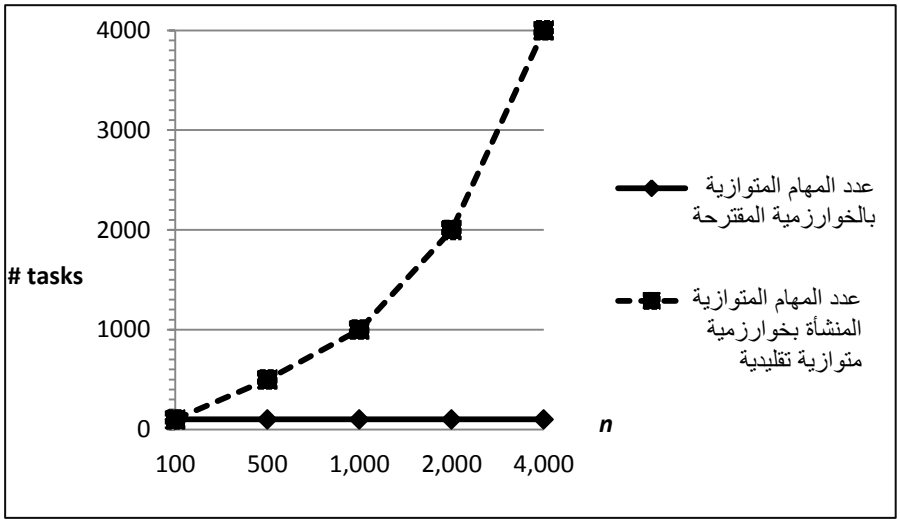
ومن الواضح أن زمن تنفيذ الخوارزمية المقترحة أصغر من زمن تنفيذ هذه الخوارزمية كون عدد المعالجات p أصغر بكثير من عدد المهام.

والآن سنقارن خورزميتنا المقترحة مع خوارزمية تسلسلية عودية (انظر الدالة factseq في الملحق) إن عدد عمليات الضرب فيها يساوي $n-1$ عملية وبالتالي فإن زمن تنفيذها $T = n-1$ ومن الملاحظ أنه أصغر من T_p زمن تنفيذ خوارزميتنا.

5. الاستنتاجات والتوصيات:

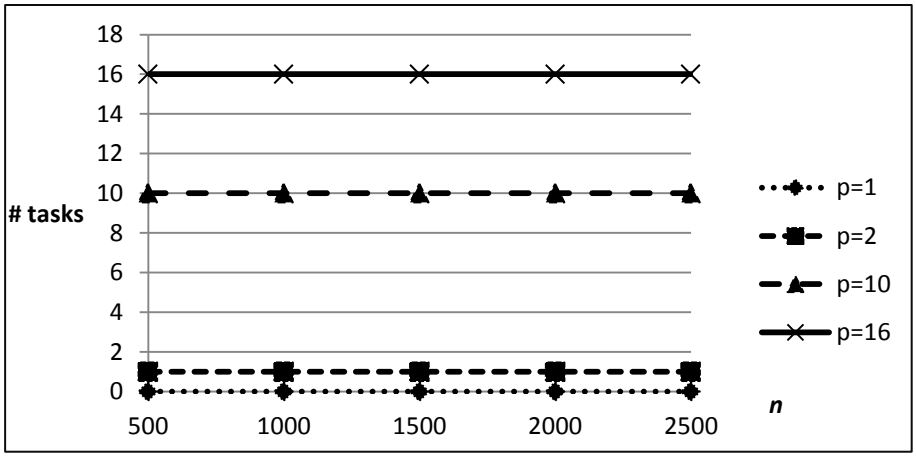
نعلم أنه في الخوارزميات المتوازية التقليدية نقوم بداية بتقسيم العمل المطلوب انجازه إلى عدد كبير من المهام المتوازية وذلك بشكل دائم حتى لو كان التنفيذ على معالج واحد ويلاحظ أن زمن إنشاء وادارة هذه المهام هو كلفة إضافية تزيد من زمن التنفيذ. إن عملنا الأساسي يقوم على عدم إنشاء مهمة تفرعية إلا عند وجود معالج خامل. يظهر الشكل(9) لنا نتائج التجربة التالية التي سنقوم فيها بمقارنة عدد المهام التي ستولدها

الخوارزمية المتكيفة المقترحة مع خوارزمية متوازية تقليدية على 100 معالج وذلك بتثبيت s وإعطاء قيم متعددة لـ n فينتج لدينا المخطط التالي:



الشكل(9)

نلاحظ الانخفاض الكبير بعدد المهام المنشأة فيها. ونستنتج من هذه التجربة أن عدد المهام المتوازية المنشأة بالخوارزمية المقترحة ثابت لايتعلق بحجم المعطيات وإنما يتعلق فقط بعدد المعالجات وهذا بعكس عدد المهام المتوازية المنشأة بخوارزمية متوازية تقليدية. ولتأكيد هذه الفكرة سنقوم بتجربة جديدة تبين أداء الخوارزمية المقترحة مع اختلاف عدد المعالجات وذلك بتثبيت s وإعطاء قيم مختلفة لـ n فنجد النتيجة الموضحة بالشكل(10):



الشكل(10)

نلاحظ في خوارزميتنا المقترحة أن: $\#tasks \cong p$ أي أن عدد المهام المتوازية المنشأة تساوي تقريباً عدد المعالجات المشاركة في التنفيذ.

6. الخاتمة:

أوردنا في هذا البحث استراتيجية متكيفة تقوم على المزاوجة العودية بين خوارزمية تسلسلية وأخرى متوازية معتمدين مبدأ سرقة العمل في جدولة المهام وعلى مخطط تدفق البيانات لتمثيل حالة التطبيق المتوازي بشكل مجرد وطبقنا هذه الاستراتيجية على مثال حساب العاملي لعدد معطى وأوجدنا الحد الأدنى لزمن تنفيذها على p معالج فوجدنا أنه أصغر مقارنة مع خوارزمية متوازية تقليدية ومع خوارزمية تسلسلية. كما وجدنا أن عدد السرقات مقارب لعدد المعالجات المشاركة في التنفيذ.

7. المراجع

[1] BENDER MA, RABIN MO, 2002, Online scheduling of parallel programs on heterogeneous systems with applications to cilk, Theory Comput, Syst. 35(3) 289–304

[2] BERNARD J, TRAORE D, ROCH JL, 2006, On-line adaptive parallel prefix computation, Euro-Par.

[3] GAUTIER T, KRINGS AW, JAFAR S, ROCH JL, 2005 Theft-Induced Checkpointing for Reconfigurable Dataflow Applications, IEEE, DOI: 10.1109/EIT.2005.1626998, 6pp.-6

[4] GAUTIER T, JAFAR S, PIGEON L, ROCH JL, 2006, Self-adaptation of parallel applications in heterogeneous and dynamic architectures, IEEE, vol.2. 3347–3352

[5] PACHECO PS, 2011_ Introduction to parallel programing, Elsevier, USA, 370p.

[6] PAL S, SINGH MK, 2011, Five Layer Security Architecture & Policies for Grid Computing System, (IJCSIT) International Journal of Computer Science and Information Technologies.

[7] GALILEE F, ROCH JL, CAVALHEIRO G, DOREILLE M, 1998, Athapascan-1: On-Line Building Data Flow Graph in a Parallel Language, Paris. PACT'98.

[8] GAUTIER T, LIMA JVF, MAILLARD N, RAFFIN B, 2013, XKaapi: A Runtime System for Data-Flow Task Programming on

Heterogeneous Architectures. In Proc. of the 27-th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Boston, USA.

ملحق:

سنعرض فيما يلي كود الخوارزمية المقترحة بلغة Athapascan-v4 [7] وهي لغة برمجة إجرائية متوازية تسمح ببناء مخطط تدفق البيانات أثناء التنفيذ مما يسمح باستخدام خوارزميات جدولة ديناميكية. وتنفذ بواسطة مكتبات الـ C++ وتستخدم مفهوم الوراثة والقوالب لتوفير واجهات محببة وسهلة للمستخدم .

```
void Fact_seq( a1:: shared_w<int> res, int n, int s ) {
    if ( n < s ) res.write( 1 );
    else if ( n==s ) res.write( n );
    else {
        int m= n*(n-1);
        a1:: shared<int> res1;
        Fact() ( res1, n-2, s );
        res.write( m*res1.read() );
    } }

struct Fact_par {
    void operator( a1:: shared_w<int> res, int n, int s ) {
        a1:: shared <int> res1;
        a1:: shared <int> res2;
        int r= (n+s)/2;
        a1:: Fork<Fact>() ( res1, n, r+1 );
        a1:: Fork<Fact>() ( res2, r, s );
        res.write( res1.read() * res2.read() );
    }
};
```

```
struct Fact {
        void operator( a1:: shared_w<int> res, int n, int s ) {
             boolean t=check();
             if ( !t || n==s)
               Fact_seq( res, n, s );
             else
               Fact_par( res, n, s );
        }
};
```

يوضح الكود التالي الخوارزمية التفرعية التقليدية بلغة Athapascan-v4 [7] :

```
int factseq(int n) {
    if ( n<=1 ) return 1;
    else return ( n*factseq(n-1));
}
struct Fact_par {
        void operator( a1:: shared_w<int> res, int n, int s ) {
             if ( n<=s ) res.write( factseq(n) );
             else {
                 a1:: shared <int> res1;
                 a1:: Fork<Fact_par>() ( res1, n-1, s );
                 res.write( n * res1.read() );
             }

        }

};
```