الطرائق التكرارية المتوازية

محمد واجد محمد علي

كلية التربية للبنات ، جامعة الموصل ، الموصل ، العراق

(تاريخ الاستلام: ١٧ / ٨ / ٢٠٠٨ ، تاريخ القبول: ١ / ٣ / ٢٠٠٩)

الملخص

هدف البحث هو تطوير طرائق متوازية للطريقتين التكراريتين جاكوبي وكاوس–سيدل واللتان تستعملان في البرمجة الخطية لحل منظومات من النماذج الخطية.

ان معظم هذه النماذج تتطلب وقتاً كثيراً للتنفيذ عند المعالجة باستخدام حاسبات ذات معالج تتابعي، ونحاول في هذا البحث تقليل هذا الوقت وزيادة كفاءة الخوارزميات للطريقتين من خلال اقتراح طرائق متوازية ملائمة للتنفيذ على حاسبات نوع *MIMD*.

وقد تم اقتراح ثلاث خوارزميات جديدة للتوازي، اثنان منها مطورة نسبة لطريقة جاكوبي التكرارية وخوارزمية واحدة نسبة لطريقة كاوس – سيدل كما تمت المقارنة بين هذه الخوارزميات المقترحة مع الخوارزمية الاصلية.

وعلى العموم أظهرت النتائج العملية والبرمجيات الحاسوبية المقترحة لهذه الخوارزميات الجديدة بانها افضل من مثيلاتها التي تنفذ على حاسبات ذات معالج تتابعي نسبة الى عنصري زمن التنفيذ وسرعة الخوارزمية.

١ – مقدمة:

الطرائق التكرارية *Iterative Methods* ، ذات فائدة كبيرة في حل نظام المعادلات الجبرية الخطية، اذ تكون قادرة على اعطاء حلول افضل من المعطاة بواسطة الطرائق المباشرة *(Direct Methods)*، وخصوصاً في حالة النظم الكبيرة او التي تحتوي معاملاتها على اعداد عشرية كبيرة، او التي يكون اكبر عدد من عناصرها اصفاراً *(Sparse)* [٢].

هذه الطرائق تعتمد في البداية على استخدام تقريب اولي *Initial Approximation* للحل لغرض الحصول منه على ما يسمى بالتقريب الصفري *Zero Approximation*. اذ يمكن بعد ذلك استخدام التقريب الصفري للحصول على التقريب الاول *First Approximation* وبالتكرار يمكن الحصول على التقريب الثاني فالثالث، وهكذا نستمر في هذه العملية حتى ثبات الحل، وعندها تنتهي الطريقة ونحصل على ادق تقريب يكون قريباً من الحل المضبوط [٢].

ومن المعلوم ان معظم هذه الانظمة تأخذ وقتاً طويلاً للتنفيذ عند المعالجة باستخدام حاسبات ذات معالج تتابعي، فلهذا السبب كان هدف البحث هو ايجاد طرائق متوازية لتقليل هذا الوقت وزيادة كفاءة الطريقة.

من الممكن التعرف على الشكل العام لمنظومة المعادلات الجبرية الخطية والتي تتكون من n من المعادلات الخطية و n من المجاهيل بالصيغة [8]، [١٢]:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_1 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \quad\quad \vdots \quad\quad \ddots \quad\quad \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \quad \ldots\ldots\ldots(1)$$

اذ ان المعاملات $a_{ij},\ i,j=1..n$ *(Coefficients)* وكذلك الحدود المطلقة $b_i\ ,\ i = 1..n$ *(Free terms)* تكون معطاة اما $x_i,\ i = 1..n$ فهي المجاهيل مطلوب الحصول على قيمها [٢].

وهذه المنظومة بالإمكان كتابتها بصيغة موجزة على النحو التالية [٦]،[١٢]،[١٥]:

$$AX = B \ldots\ldots(2)$$

اذ ان :

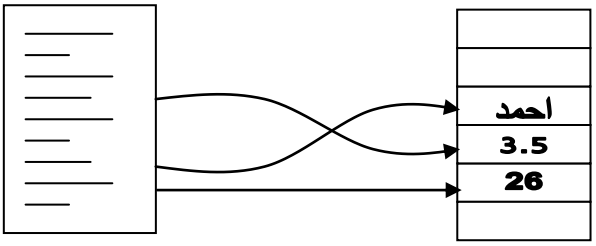
$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix},\ X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},\ B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

هذا النوع من المعادلات مهم جداً وبرز بشكل مكرر في المواد العلمية والطبية والتطبيقات الاقتصادية، العديد من طرائق التحليلات العددية تستخدم نظام المعادلات الخطية لتقريب اكثر المسائل الرياضية غير الخطية، فعلى سبيل المثال تنبؤ حالة الطقس فرع من العلم الذي يجب ان يحل انظمة معادلات كثيرة جداً لضبط احوال الطقس المستقبلية[١٥].

وبما إننا نتكلم عن الطرق المتوازية فلا بد من التطرق إلى الحاسبات المتوازية:

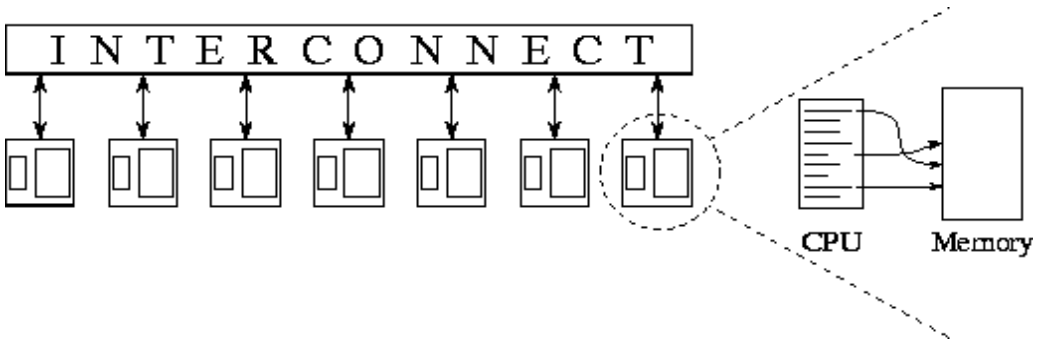
١–٢ الحاسبات المتوازية:

يعود الفضل للتقدم السريع في صناعة الحاسبات ودخولها مجالات عدة مثل التجارة والعلوم والتعليم لنموذج الحاسبة الأولى (Single Machine Model) لحاسبات Von Neuman، وكما هو معروف فان هذه الحاسبات تمتلك وحدة معالجة مركزية واحدة (CPU) مرتبطة بوحدة خزن (Memory) (شكل١)، وان المعالج يقوم بتنفيذ برنامج مخزون والذي يحدد بواسطة مجموعة متسلسلة من علميات القراءة والكتابة على الذاكرة [١١].



شكل (١) حاسبة Von Neuman، تقوم وحدة المعالجة المركزية (CPU) بانجاز البرنامج الذي ينفذ بشكل تسلسلي لعمليات القراءة والكتابة على الذاكرة المحجوزة [١١].

بينما في الحاسبات المتوازية فهو ايجاد نموذج متوازٍ يتكون من عدد من حاسباتVon Neuman (شكل ٢) يكون عاماً لكثير من التطبيقات وان تمتلك الحاسبة المتوازية ميزتين اساسيتين هما (البساطة والواقعية)، والبساطة تعني امكانية فهم الحاسبة والبرمجة لها، والواقعية هي ضمان تنفيذ النماذج البرمجية لها بكفاءة معقولة [٩].



شكل (٢) : نموذج حاسبة متوازية مثالية، تحتوي كل نقطة ارتباط على حاسبة من نوع Von Neuman ، ويمكن لكل نقطة الاتصال بالنقاط الاخرى عن طريق ارسال واستقبال الرسائل عبر الشبكة المربوطة. [١١]

١-٣ تعاريف

١-٣-١ الحاسوب المتوازي: هو مجموعة من المعالجات التي بامكانها العمل بصورة متوازية لحل مسألة حسابية ما، وهذا التعريف يشمل الحاسبات المتوازية التي تمتلك عشرات او مئات من المعالجات (Processors) [٩].

أو هو حاسوب يستخدم عدة معالجات تعمل بشكل متزامن (أي تعمل في الوقت نفسه) لحل مسألة او لأداء وظيفة معينة[٥].

١-٣-٢ الخوارزمية المتوازية:

الخوارزمية: هي مجموعة من التوجيهات لتنفيذ عمليات حسابية مصممة بشكل يؤدي الى حل المسألة المعطاة [١]، [٣].

وعليه فمن الممكن تعريف الخوارزمية المتوازية: عبارة عن تجزئة مسألة واحدة الى عدة مسائل جزئية مستقلة حتى يمكن حل كل مسألة جزئية في معالج مستقل ثم جمع الحلول لتركيب حل المسألة الاصلية [٧]، [٩].

١-٣-٣ قياس وحساب الوقت:

تقاس القدرة الحسابية للحاسوب بعدد العمليات التي ينفذها بالثانية على الأعداد الحقيقية Mega FLOPS (Floating Point Operation Per Second)، وتتطلب التطبيقات الحسابية قدرة حسابية من درجة ألف مليون عملية في الثانية، لقد وصلت الحاسبات الفائقة في عام ٢٠٠٣ إلى 35TEFLOPS وهو ما يقارب ٣٦,٥ تريليون عملية في كل ثانية، ومن اجل تحسين الأداء يجب تقصير زمن دورة المعالج للعملية الواحدة وزيادة العمليات التي تنفذ على التوازي. ولكن تسريع المعالجات لا يكفي اذا لم يصاحب القدرة على نقل المعلومات من الذاكرة الى وحدة المعالجة بالسرعة الكافية وتقاس هذه القدرة بعدد البايتات Bytes التي يمكن نقلها من الذاكرة الى وحدة المعالجة في الثانية [٥].

ملاحظة مهمة:

ان حاسبات من نوع MIMD غير متوفرة في بلادنا، فلذلك تم تنفيذ كل إجراء على حدا وحساب الوقت، ويكون الوقت الأكبر هو الوقت المعتمد في حساب تنفيذ الإجراءات المتوازية، وهذا ممكن اذ ان المعالج المستخدم في الحاسبات المتوازية هو المعالج نفسه المستخدم في الحاسبات الشخصية، كما توصل اليه الباحثY.F. Fung, et al في دراسته اذ اثبت انه بامكان الطلاب استخدام حاسباتهم الشخصية لتنفيذ ومعالجة البرامج والخورازميات المتوازية وذلك بسبب ارتفاع كلفة الحاسبات ذات المعالجات المتعددة [١٦].

١-٣-٤ التسريع (S_p): [٥]

هو حاصل قسمة الزمن التسلسلي (t_s) على الزمن المتوازي (t_p) اي ان :

$$S_p = \frac{t_s}{t_p}$$

١-٤ تصنيف الحاسبات المتوازية:

يعمل أي نوع من أنواع الحاسبات (حاسبات تسلسلية كانت أم متوازية) على تنفيذ مجموعة من الايعازات (Instructions) على مجموعة من البيانات (Data) ومن ثم يتم معالجتها وإخراجها على شكل معلومات[٩].

وهناك تصانيف عدة لمعمارية الحاسبات، من اشهر هذه التصانيف تصنيف الباحث Flynn [١٠]، فقد صنف انظمة بحسب نوع التحكم (Type of Control) والخوارزميات (Algorithms) والمعالجات المنطقية (Logical Processors) الى اربعة اصناف رئيسة [٤]، [٩]، [١٠] هي:

١-معالجات ذات ايعاز فردي وبيانات جارية مفردة

SISD: Single Instruction Stream, Single Data Stream.

٢- معالجات ذات ايعازات متعددة وبيانات جارية مفردة.

MISD: Multiple Instruction Stream, Single Data Stream.

٣- معالجات ذات ايعاز فردي وبيانات جارية متعددة.

SIMD: Single Instruction Stream, Multiple Data Stream.

٤- معالجات ذات ايعازات متعددة وبيانات جارية متعددة.

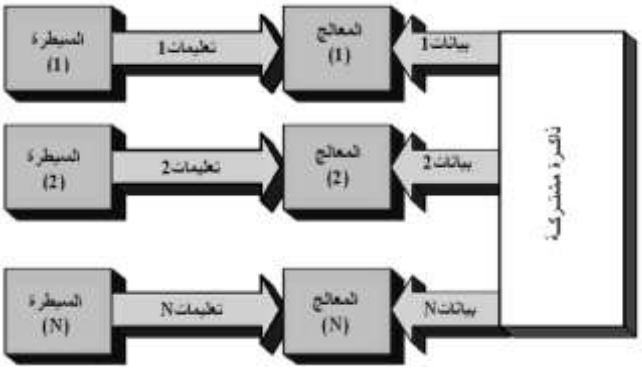
MIMD: Multiple Instruction Stream, Multiple Data Stream.

وهدفنا من البحث هو تطوير طرائق متوازية ملائمة للتنفيذ على حاسبات من نوع MIMD ولهذا سوف نقتصر الحديث عنها.

١-٥ حاسبات من نوع MIMD [٩]

يعد هذا النوع من الحاسبات الاكثر شيوعاً وكفاءة وقوة في تنفيذ التطبيقات العامة في موضوع الحاسبات المتوازية اذ تكون فيها مجموعة الايعازات ومجموعة البيانات مختلفة.

يحتوي هذا النوع من الحاسبات على (N) من المعالجات وعلى (N) من الايعازات وعلى (N) من البيانات (شكل (٣)). ويمتلك كل معالج ذاكرة خاصة به فضلاً عن وحدة الحساب والمنطق ووحدة السيطرة الخاصة به، وله القدرة على العمل بصورة ليست متزامنة ومستقلة على بياناته الجارية الخاصة، وفي اي وقت من الممكن ان تقوم معالجات مختلفة بتنفيذ ايعازات مختلفة وعلى بيانات مختلفة[4].



شكل (٣): مخطط توزيع الايعازات والبيانات في حاسبات من نوع MIMD [١٠].

٢- طريقة جاكوبي:

وهي من الطرق التكرارية الشائعة وابسطها لحل منظومة المعدلات الخطية، نبدأ مع المعادلة (2)، ونقسم المصفوفة A الى[8]:

$$A = \begin{bmatrix} & & \\ a_{21} & & \\ \vdots & \vdots & \\ a_{n1} & a_{n2} & \cdots \end{bmatrix} + \begin{bmatrix} a_{11} & & \\ & a_{22} & \\ & & \ddots \\ & & a_{nn} \end{bmatrix} + \begin{bmatrix} & a_{12} & \cdots & a_{1n} \\ & & \ddots & a_{2n} \\ & & & \\ & & & \end{bmatrix}$$

$$= L + D + U$$

اذ ان

$$L = \begin{bmatrix} & & \\ a_{21} & & \\ \vdots & \vdots & \\ a_{n1} & a_{n2} & \cdots \end{bmatrix}, D = \begin{bmatrix} a_{11} & & \\ & a_{22} & \\ & & \ddots \\ & & a_{nn} \end{bmatrix}, U = \begin{bmatrix} & a_{12} & \cdots & a_{1n} \\ & & \ddots & a_{2n} \\ & & & \\ & & & \end{bmatrix}$$

ومن خلاله يمكن اعادة ترتيب نظام المعادلات كالاتي [١٣]، [١٧]:

$$Ax = b$$

$$(L + D + U)x = b$$

$$x = [b - (L + U)x]D^{-1}$$

اذن في التكرار k سيكون النظام كالاتي:

$$x^{(k+1)} = [b - (L + U)x^{(k)}]D^{-1}$$

وتعطى قيمة ابتدائية لـ$x^{(0)}$ لبدأ احتساب القيم، وفي حالة عدم اعطاء قيمة ابتدائية، فان $x^{(0)}$ ستكون قيمة b بالنسبة للمعادلة الاولى.

وطريقة جاكوبي تعرف بالصيغة التالية[١٤]، [١٨]:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}] \ , \ i = 1..n \quad \ldots\ldots(3)$$

٢-٢ خوارزمية طريقة جاكوبي التكرارية[14]:

1) Initialize $x^{(0)} = b$
2) Set tolerance
3) Set k = 0
4) For i = 1, n
4.1) $x_i^{(k+1)} = \frac{1}{a_{ii}}[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}]$
5) if $|| x^{(k+1)} - x^{(k)}|| <$ tolerance
6.1) exit
else
6.2) increase k
6.3) goto step 4
7) the vector $x^{(k+1)}$ is the solution

٢-٣ مثال تطبيقي:

باستخدام طريقة جاكوبي التكرارية جد حلاً لمنظومة المعادلات الآتية:

$$\begin{aligned} 8x_1 + x_2 + x_3 &= 26 \\ x_1 + 5x_2 - x_3 &= 7 \\ x_1 - x_2 + 5x_3 &= 7 \end{aligned}$$

وبعد اجراء خطوات الحل التكرارية يكون الحل كالاتي:

Iteration	*0*	*1*	*2*	*3*	*4*	*5*
x_1	*3.2*	*2.90*	*2.9925*	*2.9935*	*2.998325*	*2.99934*
x_2	*1.4*	*1.03*	*1.0260*	*1.0670*	*1.02640*	*1.0683*
x_3	*1.4*	*1.03*	*1.0260*	*1.0670*	*1.02640*	*1.0683*

٢-٤ التوازي في طريقة جاكوبي التكرارية:

قدم الباحث Lori Anne [14] خوارزمية متوازية مطورة نسبة لطريقة جاكوبي التكرارية، فاقترح أن تقسم المعادلات (٣) إلى مجاميع متساوية كل مجموعة تحتوي على n/m من المعادلات إذ أن m يمثل عدد المعالجات في الحاسبة المتوازية، وللسهولة افترض الباحث أن عدد المعادلات n يقبل القسمة على m، وعلى هذا الأساس فان توزيع الأوامر على المعالجات تم على النحو الآتي:

$$x_1^{(k+1)} = b_1 - \sum_{j=2}^{n} a_{1,j} x_j^{(k)}$$

$$\vdots$$

$$x_{\frac{n}{m}}^{(k+1)} = b_{\frac{n}{m}} - \sum_{j=1}^{\frac{n}{m}-1} a_{\frac{n}{m},j} x_j^{(k)} - \sum_{j=\frac{n}{m}+1}^{n} a_{\frac{n}{m},j} x_j^{(k)}$$

تنفذ هذه العمليات على المعالج الأول..

$$x_{\frac{n}{m}+1}^{(k+1)} = b_{\frac{n}{m}+1} - \sum_{j=1}^{\frac{n}{m}} a_{\frac{n}{m}+1,j} x_j^{(k)} - \sum_{j=\frac{n}{m}+2}^{n} a_{\frac{n}{m}+1,j} x_j^{(k)}$$

$$\vdots$$

$$x_{\frac{2n}{m}}^{(k+1)} = b_{\frac{2n}{m}} - \sum_{j=1}^{\frac{2n}{m}-1} a_{\frac{2n}{m},j} x_j^{(k)} - \sum_{j=\frac{2n}{m}+1}^{n} a_{\frac{2n}{m},j} x_j^{(k)}$$

تنفذ هذه العمليات على المعالج الثاني..

وهكذا إلى المعالج m

$$x_{\frac{(m-1)n}{m}+1}^{(k+1)} = b_{\frac{(m-1)n}{m}+1} - \sum_{j=1}^{\frac{(m-1)n}{m}} a_{\frac{(m-1)n}{m}+1,j} x_j^{(k)} - \sum_{j=\frac{(m-1)n}{m}+2}^{n} a_{\frac{(m-1)n}{m}+1,j} x_j^{(k)}$$

$$\vdots$$

$$x_n^{(k+1)} = b_n - \sum_{j=1}^{n-1} a_{n,j} x_j^{(k)}$$

وفي كل تكرار تجمع النتائج في المعالج $m+1$ ويقوم باختبار تقارب الطريقة، فإذا كانت $||x^{(k+1)} - x^{(k)}|| < tolerance$ إذن الحل في هذا التكرار هو الحل الأمثل، وإلا يرجع القيم إلى المعالجات لتنفيذ التكرار التالي [14].

في هذه الطريقة اعتمد الباحث Lori Anne على تقليل الكلفة بالاعتماد على عدد من المعالجات m ، ولكن على حساب الوقت المستغرق للتنفيذ. سنناقش ألان طريقتين متوازيتين جديدتين تم التوصل إليها من خلال دراستنا لموضوع التوازي لطريقة جاكوبي التكرارية، ونأتي الان إلى شرح الطرائق التي تم التوصل اليها:

٢-٤-١ الطريقة المتوازية الأولى:

كما لاحظنا خطوات حل طريقة جاكوبي التكرارية؛ فإن هناك خطوات مستقلة فيما بينها أي أنه من الممكن ان ننجزها في الوقت نفسه؛ لأنها لا تعتمد على بعضها، وهذه الخطوات المستقلة هي ايجاد مجموع $\sum_{j=1}^{i-1} a_{ij} x_j^{(k)}$ و $\sum_{j=i+1}^{n} a_{ij} x_j^{(k)}$ ، ولكل $i=1..n$ فمن الممكن ايجاد مجموع كلاًهما كلا على حدا وتنفيذهما على معالج مستقل، وبالتالي يكون الوقت المحسوب لتنفيذ هذه الخطوات هو اعلى الوقتين لكل منهما.

فسيكون لدينا ٣ معالجات، اذ ان المعالج الاول لايجاد $\sum_{j=1}^{i-1} a_{ij} x_j^{(k)}$

وارسالها الى المعالج الثالث، اما المعالج الثاني لايجاد $\sum_{j=i+1}^{n} a_{ij} x_j^{(k)}$

وارسالها الى المعالج الثالث، ولكل $i=1..n$ ، وكما موضح في الشكل ادناه:

CPU2

١- يقوم بإيجاد

$$U_i = \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}$$

٣- يرسل القيم الى *CPU3*

CPU1

١- يقوم بإيجاد

$$L_i = \sum_{j=1}^{i-1} a_{ij} x_j^{(k)}$$

٢- يرسل القيم الى *CPU3*

↓↑

CPU3

١- يعطي قيمة ابتدائية $x_i^{(0)} = b_i \ , \ i = 1..n$ ويسلمها إلى جميع المعالجات.

٢- يستلم القيم من *CPU1*، *CPU2*.

٣- يحسب قيم $x_i^{(k+1)} = \frac{1}{a_{ii}} [b_i - L_i - U_i]$

٤- يختبر تقارب الحل $||x^{(k+1)} - x^{(k)}|| < tolerance$.

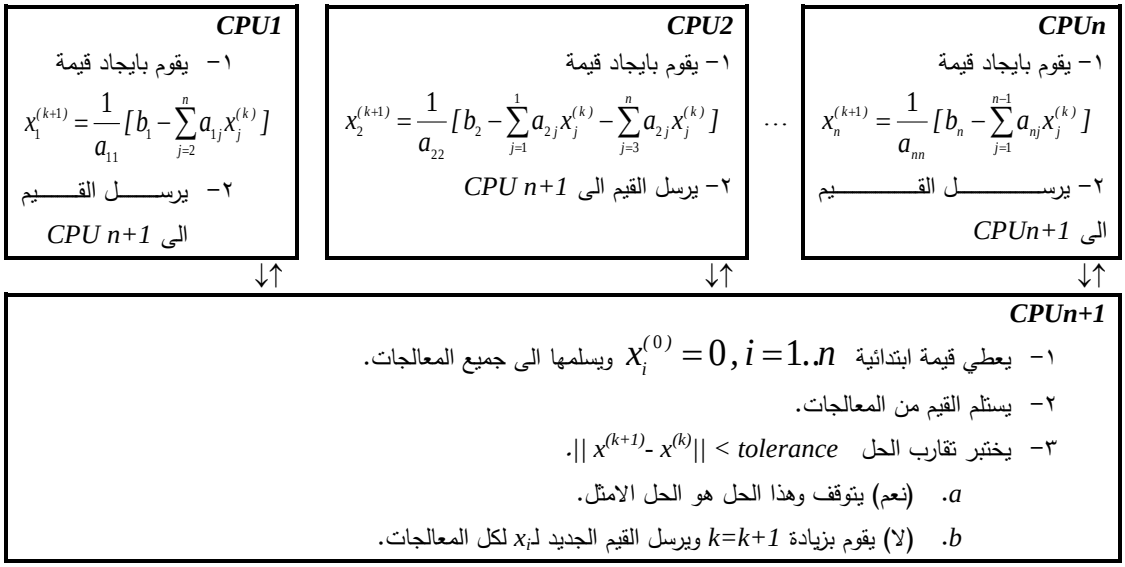
a. (نعم) يتوقف وهذا الحل هو الحل الامثل.

b. (لا) يقوم بزيادة $k=k+1$ ويرسل القيم الجديدة لـx_i لكل من *CPU1* و *CPU2*.

شكل (٤): توزيع الأوامر على المعالجات لطريقة جاكوبي المتوازية الاولى.

٢-٤-٣ الطريقة المتوازية الثانية:

بالامكان ايجاد خطوات مستقلة في طريقة جاكوبي التكرارية، فلو تأملنا المعادلة $x_i^{(k+1)} = \frac{1}{a_{ii}}[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}]$,$i,j = 1..n$ لوجدنا انه بالامكان ايجاد قيم $x_i^{(k+1)}, i = 1..n$ كلاً على حدا، وذلك لعدم اعتماد قيمة على اخرى في نفس المعادلة، وبهذا سيكون لدينا $n+1$ من المعالجات، كل معالج سيقوم بايجاد قيمة $x_i^{(k+1)}, i = 1..n$ ، اضافة الى المعالج الذي يقوم باستلام وتسلم النتائج، وكما موضح في الشكل ادناه:



شكل (٥): توزيع الأوامر على المعالجات لطريقة جاكوبي المتوازية الثانية.

٣- طريقة كاوس-سيدل *Gauss-Seidel*:

هذه الطريقة محسنة عن طريقة جاكوبي التكرارية، تستخدم الحلول المحسنة اولا بأول، بعد ان تعطى قيمة ابتدائية لـ$x^{(0)}$ لبدأ احتساب القيم، وفي حالة عدم اعطاء قيمة ابتدائية، فان $x^{(0)}$ ستكون قيمة b بالنسبة للتكرار الأول.

وطريقة كاوس-سيدل تعرف بالصيغة التالية [١٣]، [١٧]:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}] \ , \ i = 1..n \quad \text{.....(5)}$$

٣-١ خوارزمية طريقة كاوس - سيدل التكرارية[14]:

1) Initialize $x^{(0)} = b$
2) Set tolerance
3) Set k = 0
4) For i = 1, n
4.1) $x_i^{(k+1)} = \frac{1}{a_{ii}}[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}]$
5) if $||x^{(k+1)} - x^{(k)}|| < tolerance$
6.1) exit
else
6.2) increase k
6.3) goto step 4
7) the vector $x^{(k+1)}$ is the solution

٣-٢ التوازي في طريقة كاوس-سيدل التكرارية:

قدم الباحث Lori Anne [14] في دراسته خوارزمية متوازية مطورة نسبة لطريقة كاوس-سيدل التكرارية، فاقترح أن تقسم المعادلات (٥) إلى مجاميع متساوية كل مجموعة تحتوي على n/m من المعادلات إذ أن m يمثل عدد المعالجات في الحاسبة المتوازية، وللسهولة افترض الباحث أن عدد المعادلات n يقبل القسمة على m، وعلى هذا الأساس فان توزيع الأوامر على المعالجات تم على النحو الآتي:

$$\begin{aligned}
x_1^{(k+1)} &= b_1 - \sum_{j=2}^{n} a_{1,j}x_j^{(k)} \\
&\vdots \\
x_m^{(k+1)} &= b_m - \sum_{j=1}^{m-1} a_{m,j}x_j^{(k)} - \sum_{j=m+1}^{n} a_{m,j}x_j^{(k)}
\end{aligned}$$

تنفذ هذه العمليات على المعالج الأول.

$$\begin{aligned}
x_{m+1}^{(k+1)} &= b_{m+1} - \sum_{j=1}^{m} a_{m+1,j}x_j^{(k+1)} - \sum_{j=m+2}^{n} a_{m+1,j}x_j^{(k)} \\
x_{m+2}^{(k+1)} &= b_{m+2} - \sum_{j=1}^{m} a_{m+2,j}x_j^{(k+1)} - \sum_{j=m+1}^{m+1} a_{m+2,j}x_j^{(k)} - \sum_{j=m+3}^{n} a_{m+2,j}x_j^{(k)} \\
&\vdots \\
x_{2m}^{(k+1)} &= b_{2m} - \sum_{j=1}^{m} a_{2m,j}x_j^{(k+1)} - \sum_{j=m+1}^{2m-1} a_{2m,j}x_j^{(k)} - \sum_{j=2m+1}^{n} a_{2m,j}x_j^{(k)}
\end{aligned}$$

تنفذ هذه العمليات على المعالج الثاني.

$$\begin{aligned}
x_{n-m+1}^{(k+1)} &= b_{n-m+1} - \sum_{j=1}^{n-m} a_{n-m+1,j}x_j^{(k+1)} - \sum_{j=n-m+2}^{n} a_{n-m+2,j}x_j^{(k)} \\
x_{n-m+2}^{(k+1)} &= b_{n-m+2} - \sum_{j=1}^{n-m} a_{n-m+2,j}x_j^{(k+1)} - \sum_{j=n-m+1}^{n-m+1} a_{n-m+3,j}x_j^{(k)} - \sum_{j=n-m+2}^{n} a_{n-m+2,j}x_j^{(k)} \\
&\vdots \\
x_n^{(k+1)} &= b_n - \sum_{j=1}^{n-m} a_{n,j}x_j^{(k+1)} - \sum_{j=n-m+1}^{n-1} a_{n,j}x_j^{(k)}
\end{aligned}$$

تنفذ هذه العمليات على المعالج m.

وفي كل تكرار تجمع النتائج في المعالج $m+1$ ويقوم باختبار تقارب الطريقة، فإذا كانت$||x^{(k+1)} - x^{(k)}|| < tolerance$ إذن الحل في هذا التكرار هو الحل الأمثل، وإلا يرجع القيم إلى المعالجات لتنفيذ التكرار التالي [14].

في هذه الطريقة اعتمد الباحث Lori Anne على تقليل الكلفة بالاعتماد على عدد من المعالجات m، ولكن على حساب الوقت المستغرق للتنفيذ.

سنناقش في هذا البند طريقة متوازية تم التوصل إليها من خلال دراستنا لموضوع التوازي لطريقة كاوس– سيدل التكرارية، ونأتي الان إلى شرح الطريقة التي تم التوصل اليها:

٣–٢–١ طريقة كاوس – سيدل المتوازية:

كما لاحظنا خطوات حل طريقة كاوس–سيدل التكرارية؛ فإن هناك خطوات مستقلة عن بعضها أي أنه من الممكن ان ننجز كل واحدة منها في الوقت ذاته؛ لأنها لا تعتمد على بعضها البعض، وهذه الخطوات المستقلة هي ايجاد مجموع $\sum_{j=1}^{i-1} a_{ij} x_j^{(k)}$ و $\sum_{j=i+1}^{n} a_{ij} x_j^{(k+1)}$ ، ولكل $i=1..n$ فمن الممكن ايجاد مجموع كلاهما كلاً على حدا وتنفيذها على معالج مستقل، وبالتالي يكون الوقت المحسوب لتنفيذ هذه الخطوات هو اعلى الوقتين لكل منهما.

فسيكون لدينا ٣ معالجات، اذ ان المعالج الاول لإيجاد $\sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)}$ وإرسالها إلى المعالج الثالث، اما المعالج الثاني لايجاد $\sum_{j=i+1}^{n} a_{ij} x_j^{(k)}$ وارسالها الى المعالج الثالث، ولكل $i=1..n$، وكما موضح في الشكل ادناه:

CPU2

١– يقوم بإيجاد

$$U_i = \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}$$

٢– يرسل القيم الى *CPU3*

CPU1

١– يستلم قيم $x_i^{(k+1)}$ من *CPU3*

٢– يقوم بإيجاد $L_i = \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)}$

٣– يرسل القيم الى *CPU3*

CPU3

١. يعطي قيمة ابتدائية $x_i^{(0)} = b_i \ , \ i = 1..n$ ويسلمها إلى جميع المعالجات.
٢. يستلم القيم من *CPU1*، *CPU2*.
٣. لكل $i=1..n$
٤. احسب قيم $x_i^{(k+1)} = \frac{1}{a_{ii}}[b_i - L_i - U_i]$
٥. يرسل القيم الجديدة $x_i^{(k+1)}$ إلى المعالج *CPU1*
٦. يختبر تقارب الحل $||x^{(k+1)} - x^{(k)}|| < tolerance$.
٧. فإذا نعم يتوقف وهذا الحل هو الحل الامثل.
٨. والا يقوم بزيادة $k=k+1$ ويعيد تنفيذ العمليات.

شكل (٦): توزيع الأوامر على المعالجات لطريقة كاوس–سيدل المتوازية.

٤ – مناقشة النتائج:

٤–١ برمجة الطرق المتوازية

كما ذكرنا سابقاً ان حاسبات من نوع MIMD غير متوفرة في بلادنا، فلذلك تم تنفيذ كل إجراء على حدا وحساب الوقت له، ويكون الوقت الأكبر هو الوقت المعتمد في حساب تنفيذ الإجراءات المتوازية.

وقد تمت برمجة الطرائق المتوازية الثلاث وتنفيذها باستخدام لغة C++ على حاسبة PIV بمعالج 2GHz وباستخدام دالة Delay(120000) لعدة أنظمة وكانت النتائج المستحصلة كالاتي:

جدول (١): يمثل الوقت المحسوب في تنفيذ طريقة جاكوبي وكاوس–سيدل التكراريتين مع الطرائق المتوازية، ولعدد من التكرارات (٢٠) تكرار.

سعة المصفوفة	طريقة جاكوبي التكرارية	طريقة جاكوبي المتوازية الاولى	طريقة جاكوبي المتوازية الثانية	طريقة كاوس–سيدل التكرارية	طريقة كاوس–سيدل المتوازية
٥٠×٥٠	٢,٣٩ (1 CPU)	١,١٧ (3 CPU)	٠,٧٦٤ (51 CPU)	1.68 (1 CPU)	0.8757 (3 CPU)
١٠٠×١٠٠	٩,٤٨ (1 CPU)	٤,٧٩ (3 CPU)	٣,١٤٢ (101 CPU)	7.024 (1 CPU)	3.522 (3 CPU)
٢٠٠×٢٠٠	٣٧,٦ (1 CPU)	١٩ (3 CPU)	١٠,٢٨ (201 CPU)	28.138 (1 CPU)	14.241 (3 CPU)

فعند حساب عامل التسريع Speed up بين الطريقة العادية والطرق المتوازية الثلاث نحصل على:

التسريع Sp =

وقت التنفيذ لـM1 باستخدام معالج واحد (Ts)

وقت التنفيذ لـM باستخدام n من ال

اذ ان (M) هي تطوير عن خوارزمية متسلسلة (M1) باستخدام (P) من المعالجات.

جدول (٢): يمثل عامل التسريع للطرائق المتوازية.

سعة المصفوفة	طريقة جاكوبي المتوازية الأولى	طريقة جاكوبي المتوازية الثانية	*طريقة كاوس–سيدل المتوازية*
٥٠×٥٠	٢,٠٣٧٥١٠٦٦	٣,١٢٨٢٧٢٢٥	*١,٩١٨٤٦٥٢٣*
١٠٠×١٠٠	١,٩٧٩٩٤٩٨٧	٣,٠١٧١٨٦٥١	*١,٩٩٤٣٢١٤١*
٢٠٠×٢٠٠	*١,٩٨٤٦٩٦٥٧*	*٣,٦٥٧٥٨٧٥*	*١,٩٧٥٨٤٤٣٩*

إذن كما نلاحظ ان عامل تسريع الطريقة الثانية اقل من الطريقتين الاولى والثالثة وان عامل التسريع في الطريقة الثالثة هو افضل من الطريقة الاولى، اضافة الى ان عامل التسريع في الطريقتين الاولى والثالثة يزداد بنسبة ملحوظة عن الطريقة الثانية اذا زادت سعة المصفوفة.

٥. مقارنة بين الطرائق المتوازية المقترحة:

من خلال النتائج التي ظهرت لنا يمكن أن نلاحظ فروقاً بين الطرائق يمكن تلخيصها بالجدول الآتي:

جدول (٣): الفروق بين الطرائق المتوازية المقترحة لطريقتي جاكوبي وكاوس–سيدل التكراريتين

	طريقة جاكوبي المتوازية الاولى	طريقة جاكوبي المتوازية الثانية	طريقة كاوس–سيدل المتوازية
١.	نحتاج إلى ٣ من المعالجات مهما كان عدد المعادلات	نحتاج إلى n+1 من المعالجات اذ ان n هو عدد المعادلات	نحتاج إلى ٣ من المعالجات مهما كان عدد المعادلات
٢.	عندما تكون المسألة كبيرة تأخذ وقتاً كبيراً.	عندما تكون المسألة كبيرة يكون وقت التنفيذ أقل من الطريقة الأولى.	عندما تكون المسألة كبيرة تأخذ وقتاً كبيراً.
٣.	غير مكلفة بالنسبة للمسائل الكبيرة لأن كافة المسائل تحتاج إلى ٣ معالجات.	تعد مكلفة للمسائل الكبيرة ولهذا يفضل استخدامها في المسائل الصغيرة.	غير مكلفة بالنسبة للمسائل الكبيرة لأن كافة المسائل تحتاج إلى ٣ معالجات.
٤.	الحمل يكون على المعالجات بشكل متساوٍ تقريباً.	الحمل على CPU1 و CPUn متساوٍ والحمل يكون اكثر على بقية المعالجات.	الحمل يكون على CPU1 و CPU2 بشكل متساوٍ تقريباً، والحمل الاكبر يكون على CPU3.

المصادر:

١- الالوسي، د.أحمد صالح وعادل زينل البياتي (١٩٨٩): "*مقدمة في التحليل العددي*"، مطبعة التعليم العالي في الموصل.

٢- اميل شكر الله، (٢٠٠٣)، "*التحليل العددي التطبيقي، النظرية التطبيقات والطرق التقريبية*"، جامعة المنوفية، مصر ، الطبعة الثانية.

٣- سيفي، د.علي محمد صادق و د. ابتسام كمال الدين (١٩٨٦): "*مبادىء التحليل العددي*"، مديرية دار الكتب للطباعة والنشر،جامعة الموصل.

٤- عبد الحبيب عبدالله أحمد مرشد (٢٠٠٠): "تقصي خوارزميات عددية لحل المعادلات التفاضلية الاعتيادية الصلبة الملائمة للحاسبات المتوازية"، أطروحة دكتوراه، كلية العلوم، جامعة الموصل ، وزارة التعليم العالي والبحث العلمي.

٥- عماد فتاش، (٢٠٠٤) "*الحاسبات المتوازية والخوارزميات المتوازية (دراسة)*"، كلية العلوم، جامعة الملك سعود، المملكة العربية السعودية.

٦- عمر محمد التومي، احمد الشوشة، (٢٠٠٥) "*التحليل العددي*"، كتاب الكتروني، منتديات التقنية، www.tkne.net .

٧- يحيى قاسم إبراهيم القاضلي (٢٠٠٢م): "حل مسألة التخصيص باستخدام خوارزمية متوازية"، مشروع ماجستير ، كلية علوم الحاسبات والرياضيات، جامعة الموصل، وزارة التعليم العالي والبحث العلمي.

8- Autar Kaw, (2007) "*Introduction of Matrix Algebra*", http://numericalmethods.eng.usf.edu.

9- Bashir M. S., Khalaf and Khlil K.Abbo (2001): "Parallel Revised Simplex Method", Raf. Jour. Sci., Vol. 13, No. 1, pp51-60.

10- Flynn M. J. (1972): "Some Computer Organization and their Effectiveness". IEEE Transaction on /computers, Vol. C-21, pp. 948-960.

11- Ian Foster, (2002) "Designing and building Parallel Program", www.unix.ms.anl.gov/dbpp/text/node8.html.

12- J. M. McDonough, (2004) "*Lectures in Basic Computational Numerical Analysis*", University of Kentucky.

13- Kincho H. Law, "*Parallel Methods in Numerical Analysis*", http://adl.stanford.edu/cme342-wiki

14- Lori Anne McNally, (2005), "*Parallel Processing Numerical Algorithms on the IBM SP*", Project submitted to College of Computer Science, BScEE—University of New Brunswick.

15- Michael Wasilewski (2004), "*Parallel Gaussian Elimination*", University of Waterloo.

16- Y. F. Fung, et al; "*Teaching Parallel Computing Concepts with A Desktop Computer"* International Journal of Electrical Engineering Education, ISSN: 0020-7209, Vol 41 Issue 2, April (2004), pp 113-125.

17- Yousef Saad, (2000), "*Iterative Methods for sparse Linear Systems*", E-Book, http://www-users.cs.umn.edu/~saad/ books.html.

Parallel Iteration Methods

Mohammed Wajid Al-Nema
Education College for Girls , Mosul University, Mosul , Iraq
(Received 17 / 8 / 2008 , Accepted 1 / 3 / 2009)

Abstract

The aim of the project is in develop parallel approaches for Iteration Methods (Jacobi & Gauss-Seidel Methods) that are used in linear programming to solve linear module systems.

Most of these models are time consuming when executed and processing in the sequential microprocessor computers. During the project we try to decrease this time and increase the efficiency of the algorithm for this two Methods, through developing parallel methods appropriate to be executed on MIMD type computers.

In this paper, three algorithms were suggested for paralleling, two for a developed algorithm of Jacboi Iteration Method and one for a developed algorithm of Gauss-Seidel Iteration Method and a comparison was made between the three algorithms and the original.

In general, the practical results and the suggested programs for theses new algorithms proved to be better in performance than their analogues that are executed in computers of sequential processor in view of the two elements of execution time and algorithm time