Parallella algoritmer för periodiska ekvationer av Sylvestertyp

Per Andersson

4 oktober 2007
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Robert Granat
Examiner: Per Lindström

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN

Sammanfattning

Detta examensarbete presenterar dels en prestandautvärdering av biblioteket SCASY, då det körs på en parallelldator med distribuerat minne och där varje nod består av en symmetrisk multiprocessor, och dels en parallell algoritm för att lösa den triangulära periodiska tidskontinuerliga Sylvesterekvationen (TRPSYCT). TRPSYCT har formen $A_kX_k - X_{k\oplus 1}B_k = C_k$ där $k = 0, 1, \ldots, p-1$ och p är perioden.

En modifierad version, för att köras på en parallelldator med distribuerat minne där varje nod är en symmetrisk multiprocessor, av SCASY jämförs med en omodifierad version av SCASY. Resultaten visar att det är möjligt att modifiera SCASY till den aktuella konfigurationen och att prestandan för den modifierade och den omodifierade version är nästan identisk för vissa utseenden av processorgrid.

Algoritmen som används för att lösa TRPSYCT är en vågfrontsalgoritm som bygger på att de involverade matriserna har en 2-dimensionell blockcyklisk distribution. Resultaten av testkörningar där implementationen av algoritmen har testas visar att denna algoritm ger prestandaförbättringar jämfört med seriella algoritmer.

Abstract

In this Master's Thesis a performance analysis of the library SCASY, when it runs on a parallel computer with distributed memory where the nodes consists of symmetric multiprocessors, is performed and a parallel algorithm for the triangular periodic continuous-time Sylvester equation (TRPSYCT), $A_kX_k - X_{k\oplus 1}B_k = C_k$ where $k = 0, 1, \ldots, p-1$ and p is the period, is presented.

A modified version, to run on a parallel computer with distributed memory where the nodes consists of symmetric multiprocessors, of SCASY is compared with a unmodified version of SCASY. The results shows that is possible to run SCASY under this configuration and that the performance are almost equal for some processorgrids as for the unmodified version.

The algorithm for solving TRPSYCT is a wavefront algorithm which is based on the 2-dimension block-cyclic distribution of the involved matrices. The result shows that this parallel algorithm gives better performance then serial algorithms.

Innehåll

Figurer

Tabeller

Algoritmer

Kapitel 1

Introduktion

1.1 Bakgrund

Linjär algebra förekommer inom många olika områden. T.ex. inom matematik, naturvetenskap, datavetenskap och även inom samhällsvetenskap och ekonomi. Ofta uppstår problem som genererar stora linjära ekvationssystem och för att lösa dessa stora problem används datorer. För att kunna lösa en del av de riktigt stora problemen på rimlig tid används hjälp av parallelldatorer.

För att underlätta och effektivisera användandet finns det ett antal bibliotek innehållande optimerade och effektiva seriella och parallella rutiner för linjär algebra tillgängliga. Dessa bibliotek utvecklas fortlöpande för att erhålla bättre prestanda och funktionalitet.

1.2 Problembeskrivning

Detta examensarbete består av två delar. Den första delen består av att utvärdera prestandan i biblioteket SCASY (se stycke 2.6.7), vilket är ett bibliotek innehållande parallella rutiner för att lösa 44 varianter av ekvationer av Sylvestertyp. T.ex. den tidskontinuerliga Sylvesterekvationen $AX - XB = C$ (se stycke 2.1). Biblioteket ska utvärderas på en parallelldator med distribuerat minne där varje nod består av två processorer vilka kan ses som en symmetrisk multiprocessor (se stycke 2.3.1). Prestandan ska jämföras mellan testkörningar där hela parallelldatorn ses som parallelldator med distribuerat minne, och testkörningar där varje nod i parallelldatorn med distribuerat minne består av en symmetrisk multiprocessor.

I den andra delen av examensarbetet ska en parallell algoritm för den triangulära periodiska tidskontinuerliga Sylvesterekvationen (TRPSYCT) implementeras och utvärderas. TRPSYCT har formen

$$A_k X_k - X_{k\oplus 1} B_k = C_k,$$

där $A_k \in \mathbb{R}^{m\times m}$, $B_k \in \mathbb{R}^{n\times n}$, $C_k, X_k \in \mathbb{R}^{m\times n}$, $k = 0,1,\ldots,p-1$, p är perioden, $a \oplus b = a + b$ mod p, A_k och B_k är övertriangulära, A_s och $B_{s'}$ är övertriangulära bestående av 1×1- och/eller 2×2-block på diagonalen och $s, s' \in \{0,1,\ldots,p-1\}$.

Algoritmen som ska användas för att lösa TRPSYCT är en generaliserad version av algoritmen som i SCASY används för att lösa den triangulära tidskontinuerliga Sylvesterekvationen (TRSYCT).

1.3 Mål

Målet för första delen av arbetet som består av att utvärdera prestandan i SCASY är först och främst att se om det är möjligt att köra SCASY på en parallelldator med distribuerat minne där varje beräkningselement består av en symmetrisk multiprocessor. Om det är möjligt ska prestandan testas och utvärderas för att se om det är möjligt att få en prestandaförbättring med denna metod jämfört med den mer traditionella där hela parallelldatorn är en parallelldator med distribuerat minne.

Målet för andra delen, vilken består av att utveckla en parallell rutin för att lösa triangulära periodiska tidskontinuerliga Sylvesterekvationer är dels att se om den tänkta metoden fungerar och dels om den ger en tillfredsställande prestanda.

1.4 Notation

I denna rapport kommer följande notationer att användas om inte annat anges. Skalärer betecknas med små bokstäver, t.ex. a, s och x. Vektorer betecknas med små bokstäver i fet stil, t.ex. **a**, **s** och **x**. Matriser betcknas med stora bokstäver, t.ex. A och X.

I_m Betecknar indentitetsmatrisen av storleken $m \times m$.

$vec(X)$ Betecknar en vektor där kolumnerna från vänster till höger i X är staplade på varandra. Alltså, om $X = [\mathbf{x_1}\mathbf{x_2}\ldots\mathbf{x_n}]$, där $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ är kolumnvektorer, så är

$$vec(X) = \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \vdots \\ \mathbf{x_n} \end{bmatrix}.$$

$A \otimes B$ Är Kroneckerprodukten av matriserna A och B där blockelement (i, j) består av $a_{ij}B$. T.ex om matriserna A och B har har formen

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

respektive

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

så ger det Kroneckerprodukten

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & b_{22}B \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & b_{11}b_{22} & a_{12}b_{21} & b_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & b_{21}b_{22} & a_{22}b_{21} & b_{22}b_{22} \end{bmatrix}.$$

$op(A)$	$op(A)$ står för antingen A eller A^T.
X_k	Anger en matris k i en matrissekvens $X_1, X_2, X_3, \ldots$. En matrissekvens är periodisk med perioden p om $X_k = X_{k+\alpha p}$ där $\alpha \in \mathbb{Z}$.
X_{ij}^k	Betecknar blocket (i, j) av en matris k i en matrissekvens $X_1, X_2, X_3, \ldots$. Matrissekvensen X_{ij} är det samma som $X_{ij}^1, X_{ij}^2, X_{ij}^3, \ldots$.
$a \oplus b$	Betecknar $a + b$ mod p när periodiska ekvationer med perioden p diskuteras.

1.5 Rapportens disposition

Denna rapport är indelad i ett antal kapitel som behandlar följande:

Kapitel 1 innehåller en introduktion till ämnet och en genomgång av problemet och målsättningen med arbetet.

Kapitel 2 innehåller nödvändig bakgrundsteori om ekvationer av Sylvestertyp, Bartels-Stewarts metod, parallella beräkningar, rekursiva algoritmer och mjukvara för linjär algebra.

Kapitel 3 redogör för prestandautvärderingen av SCASY. Kapitlet innehåller en beskrivning av utförandet samt en redovisning av resultaten.

Kapitel 4 behandlar vår algoritm för att lösa triangulära periodiska tidskontinuerliga Sylvesterekvationer. Kapitlet innehåller algoritmbeskrivning, implementationsdetaljer och resultat.

Kapitel 5 innehåller slutsatser och diskussion om arbetet.

Kapitel 2

Teori

I detta kapitel kommer nödvändig teori att gås igenom. Läsaren antas ha kunskaper inom linjär algebra och allmänna kunskaper om datorer och programmering.

2.1 Sylvesterekvationer

Den tidskontinuerliga Sylvesterekvationen (SYCT) har formen

$$AX - XB = C,$$

där $A \in \mathbb{R}^{m\times m}$, $B \in \mathbb{R}^{n\times n}$ och $C, X \in \mathbb{R}^{m\times n}$. SYCT hör till en grupp av linjära ekvationer som är av Sylvestertyp. Tabell 2.1 visar åtta vanligt förekommande ekvationer av Sylvestertyp.

Den triangulära tidskontinuerliga Sylvesterekvationen (TRSYCT) har samma form som SYCT men med tillägget att A och B är på Schurform. Schurformen innebär att A och B är övertriangulära innehållandes 1×1- och/eller 2×2-block på diagonalen där 2×2-blocken består av komplexa konjugatpar. Dekomposition av en matris till Schurform kan utföras med QR-algoritmen. Hur den fungerar diskuteras i bland annat [6].

Ekvationer av Sylvestertyp spelar en viktig roll inom linjär algebra och förekommer ofta i egenvärdesproblem och inom kontrollteori.

SYCT har en unik lösning om och endast om A och B inte har några gemensamma egenvärden [16]. De andra ekvationerna av Sylvestertyp som listas i Tabell 2.1 har liknande villkor för att generera en unik lösning, se exempelvis diskussion i [12, 13].

2.1.1 Periodiska Sylvesterekvationer

Den tidskontinuerliga periodiska Sylvesterekvationen (PSYCT) har formen

$$A_k X_k - X_{k\oplus 1} B_k = C_k,$$

där $A_k \in \mathbb{R}^{m\times m}$, $B_k \in \mathbb{R}^{n\times n}$, $\{C_k, X_k\} \in \mathbb{R}^{m\times n}$, $k = 0, 1, ..., p-1$, p är perioden och $a \oplus b = a + b$ mod p.

Om man exempelvis har en PSYCT med perioden 4 så ger den upphov till ekvationssystemet

$$\begin{aligned} A_0X_0 - X_1B_0 &= C_0 \\ A_1X_1 - X_2B_1 &= C_1 \\ A_2X_2 - X_3B_2 &= C_2 \\ A_3X_3 - X_0B_3 &= C_3 \end{aligned}$$

Tabell 2.1: De vanligaste Sylvester- och Lyapunovekvationerna.

Ekvation	Namn	Akronym
$AX - XB = C$	Tidskontinuerlig Sylvester	SYCT
$AXB - X = C$	Tidsdiskret Sylvester	SYDT
$AX - XA^T = C$	Tidskontinuerlig Lyapunov	LYCT
$AXA^T - X = C$	Tidsdiskret Lyapunov	LYDT
$AXB - CXD = E$	Generaliserad Sylvester	GSYL
$AX - YB = C$ $DX - YE = F$	Generaliserad kopplad Sylvester	GCSY
$AXE^T + EXA^T = C$	Tidskontinuerlig Generaliserad Lyapunov	GLYCT
$AXA^T - EXE^T = C$	Tidsdiskret Generaliserad Lyapunov	GLYDT

som innehåller fyra ekvationer och fyra obekanta matriser.

Den triangulära tidskontinuerliga periodiska Sylvesterekvationen (TRPSYCT) har samma form som PSYCT med tillägget att matrissekvenserna A och B är på periodisk Schurform. Detta innebär att A_s och $B_{s'}$ där $\{s,s'\} \in \{1,2,\ldots,p\}$ är på Schurform medan A_t där $t = 1,2,\ldots,s-1,s+1,\ldots,p$ och $B_{t'}$ där $t' = 1,2,\ldots,s'-1,s'+1,\ldots,p$ är övertriangulära. Hur en dekomposition av en matrissekvens till periodisk Schurform går till diskuteras i [1].

2.2 Bartels-Stewarts metod

En metod för att lösa den tidskontinuerliga Sylvesterekvationen, $AX - XB = C$, föreslogs 1972 av Bartels och Stewart [22]. Metoden består av följande fyra steg:

1. Transformera A och B till Schurform.
2. Uppdatera C med hänsyn till transformeringen av A och B.
3. Lös den triangulära matrisekvationen som uppstår.
4. Transformera tillbaka X till det ursprungliga koordinatsystemet.

I första steget transformerar man A och B till Schurform enligt

$$A' = U^T AU = \begin{bmatrix} A'_{11} & A'_{12} & \cdots & A'_{1m} \\ & A'_{22} & \cdots & A'_{2m} \\ & & \ddots & \vdots \\ & & & A'_{mm} \end{bmatrix}$$

och

$$B' = V^T BV = \begin{bmatrix} B'_{11} & B'_{12} & \cdots & B'_{1n} \\ & B'_{22} & \cdots & B'_{2n} \\ & & \ddots & \vdots \\ & & & B'_{nn} \end{bmatrix}$$

där U och V är ortogonala och A' och B' är övertriangulära bestående av 1×1 och/eller 2×2 block på diagonalen där 2×2 blocken består av komplexa konjugatpar. I andra steget används U och V, vilka erhålls i första steget, för att uppdatera C enligt

$$C' = U^T CV.$$

Antag sedan att X' har formen

$$X' = U^T XV.$$

Av A', B', C' och X' bildas sedan det triangulära ekvationssystemet

$$A'X' - X'B' = C'.$$

Ur detta triangulära ekvationssystem kan sedan X' beräknas. I det fjärde steget transformeras X' tillbaka till X enligt

$$X = UX'V^T$$

vilket ger lösningen på ekvationen $AX - XB = C$.

Denna metod går att generalisera för de andra ekvationerna av Sylvestertyp som nämns i Tabell 2.1, och även för de periodiska varianterna av dessa.

2.3 Parallella beräkningar

Parallella beräkningar är när exekveringen av ett program är uppdelat i flera uppgifter som körs samtidigt på mer än en processor. Behovet för parallella beräkningar är framförallt för att kunna utföra beräkningar snabbare. Anledningen kan i vissa fall även vara att kunna lösa stora problem som på grund av sin storlek inte är möjliga att lösa med seriella datorer då dessa saknar tillräckligt med minne.

Kostnadsbesparing kan också vara en anledning till att använda parallella beräkningar. Istället för att använda en dyr superdator kan istället billiga enkla delar användas för att bygga en parallelldator. Den kan även finnas fysiska begränsningar för beräkningskraften hos en seriell dator vilka inte finns hos parallelldatorer.

Parallella beräkningar används inom många områden som t.ex. naturvetenskap, ingenjörskonst, datavetenskap, ekonomi och samhällsvetenskap och används bland annat för att beräkna fysikaliska fenomen, i web- och databasservrar och för att göra väderförutsägelser.

2.3.1 Parallelldatorsystem

Parallelldatorsystem är datorsystem som innehåller mer än en processor eller mer än en processorkärna. Parallelldatorsystem kan delas in i två huvudgrupper som baseras på deras minnesarkitektur. De två grupperna är parallelldatorer med distribuerat minne och parallelldatorer med gemensamt minne. I parallelldatorer med gemensamt minne har alla processorer access till hela minnet och alla processorer har samma adressrymd Parallelldatorer med gemensamt minne kan dessutom delas in i de två undergrupperna UMA (Uniform Memory Access), där accesstiden till hela minnet är densamma, och NUMA (Non-Uniform Memory Access), där accesstiden till olika delar av minnet varierar. Om alla processorer i en parallelldator med gemensamt minne och UMA är identiska och kan beräkna samma saker är det en symmetrisk multiprocessor (SMP).

I parallelldatorer med distribuerat minne har däremot varje processor sitt eget lokala minne som de andra processorerna inte har access till utan de utbyter istället data med hjälp av något slags nätverk. I en parallelldator med distribuerat minne är en nod detsamma som ett beräkningselement.

Förutom de två huvudgrupperna har det på senare tid blivit vanligt med en typ av system som är ett mellanting mellan de två tidigare nämnda och som går under benämningen multicoreprocessor. En multicoreprocessor innehåller två eller flera individuella processorkärnor på ett och samma chip. Minnesarkitekturen i multicoreprocessorer kan variera från att kärnorna delar alla nivåer av cache till att varje prcessorkärnorna har sin helt egna cacheherarki.

2.3.2 Prestandametriker

För att utvärdera prestanda på parallella program används till viss del andra metriker än de som används vid utvärdering av prestanda för seriella program.

Den vanligaste metriken som används är uppsnabbning, eller *speedup*, vilken definieras av

$$S_p = \frac{T_s}{T_p}$$

där T_s är exekveringstiden på en processor och T_p är exekveringstiden på p processorer. Speedup anger alltså hur mycket snabbare en parallell körning på p processorer är jämfört med en seriell körning. Linjär speedup åstadkoms när $S_p = p$ och superlinjär speedup åstadkoms när $S_p > p$. Utifrån uppsnabbning, S_p, kan man sedan definiera effektivitet enligt

$$E = \frac{S_p}{p}.$$

Linjär speedup eller effektivitet $E = 1$ är i praktiken nästan omöjligt att uppnå förutom för en del väldigt speciella problem. Detta förklaras genom Almdahls lag som menar att ett problem för det mesta består av en seriell del som inte går att parallellisera. Almdahls lag definierar utifrån detta den parallella tiden T_p till

$$T_p = \beta T_s + \frac{(1-\beta)T_s}{p}$$

där βT_s anger tiden för den del av problemet som inte går att parallellisera och $\frac{(1-\beta)T_s}{p}$ anger tiden för den del av problemet som går att parallellisera. Genom att använda detta T_p i ekvationen för speedup, S_p, fås följande ekvation

$$S_p = \frac{T_s}{\beta T_s + \frac{(1-\beta)T_s}{p}} = \frac{1}{\beta + \frac{1-\beta}{p}}.$$

Då $p \to \infty$ ger det att $\frac{1-\beta}{p} \to 0$ vilket innebär att $S_p = \frac{1}{\beta}$ blir är en övre gräns för speedup för det aktuella problemet. Detta innebär t.ex. att ett problem där 20% av arbetet inte går att parallellisera maximalt kan uppnå en speedup på 5.

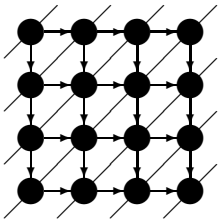
Problemet med Almdahls lag är att den inte tar någon hänsyn till problemstorleken. 1988 visade Gustafson [10] att om man samtidigt som antalet processorer ökar också ökar problemstorleken kan man nå bra parallell prestanda för att stort antal processorer. Detta ledde fram till Gustafson-Barsis lag för speedup, S_p, som lyder

$$S_p = p + \beta(1-p)$$

där p är antal processorer och β anger den andel av problemet som inte går att parallellisera.

2.4 Rekursivt blockade algoritmer

I dagens datorer där processorerna blir snabbare och snabbare men där prestandan på minnet inte utvecklas i samma takt som processorerna är det viktigt att utnyttja minnet effektivt för att uppnå bra prestanda. Dagens datorer har oftast en djup minneshierarki som består av register, olika nivåer av cacheminne och primärminne, och där åtkomsttiden skiljer sig väsentligt mellan de olika nivåerna.



Figur 2.1: Exempel på databeroende för problem som kan lösas med en vågfrontsalgoritm.

En metod för att utnyttja djupa minneshierarkier effektivt är att använda sig av rekursiva algoritmer. Rekursiva algoritmer bygger på att man delar upp beräkningarna rekursivt för att på det sättet få en blockning av problemet som fyller minneshierarkin effektivt.

Ett exempel är en vanlig matrismultiplikation $A \cdot B$. Om man delar A och B vertikalt och horisontellt enligt

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \end{bmatrix}$$

, så ser man att det uppstår 8 nya oberoende matrismultiplikationer. Dessa kan sedan också delas upp på samma sätt rekursivt. Fördelen med denna metod är att storleken på delmatriserna i något steg i rekursionen kommer att passa i respektive nivå utav cache.

Grundligare information om rekursivt blockade algoritmer för linjär algebra presenteras av Elmroth et al [7].

2.5 Vågfrontsalgoritmer

Vågfrontsalgoritmer är vanliga inom parallella beräkningar då det gäller att lösa problem som har databeroenden. Ett exempel på databeroende visas i Figur 2.1. Då beroendena i detta fallet går från övre vänstra hörnet till det nedre högra hörnet går det lösa alla delproblem som ligger på samma diagonal parallellt. Problem inom linjär algebra och särskilt de som innehåller triangulära matriser kan ofta formuleras på detta sätt.

När man pratar om vågfrontsalgoritmer brukar man ofta prata om deras riktning. När beräkningarna startar i det övre vänstra hörnet och slutar i det nedre högra hörnet har algoritmen en sydostlig riktning, när de startar i det övre högra hörnet och slutar i det nedre vänstra hörnet har de en sydvästlig riktning, när de startar i det nedre vänstar hörnet och slutar i det övre högra hörnet har det en nordostlig riktning och när de startar i det nedre högra hörnet och slutar i det övre vänstar hörnet har de en nordvästlig riktning.

2.6 Mjukvara

För att underlätta utvecklandet av och för att effektivisera program som löser problem inom linjär algebra finns det ett antal gränssnitt och bibliotek tillgängliga. En genomgång av de relevanta för detta examensarbete ges här.

Tabell 2.2: Skillnaden i antal operationer och anatal minnesreferenser i olika nivåer av BLAS.

	operation	flop	minnesref.	flop/minnesref.
Nivå 1 BLAS	$y \leftarrow \alpha\mathbf{x} + \mathbf{y}$	$2n$	$3n$	$2/3$
Nivå 2 BLAS	$y \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$	$2n^2$	n^2	2
Nivå 3 BLAS	$C \leftarrow \alpha AB + \beta C$	$2n^3$	$4n^2$	$n/2$

2.6.1 BLAS

BLAS [5] (Basic Linear Algebra Subprograms) är ett gränssnitt för rutiner som utför (enkla) vektor- och matrisoperationer. Från början var tanken med BLAS att alla datortillverkare skulle tillhandahålla sina egna versioner av BLAS så att dessa skulle vara optimerade för den specifika hårdvaran. Detta skulle innebära att program som använder sig av BLAS skulle vara effektiva på alla datorsystem. Detta är dock inte verkligheten idag då ett fåtal optimerade plattformsoberoende implementationer av BLAS är dominerande. GotoBLAS [8] och ATLAS [2] är två sådana implementationer.

BLAS är uppdelat i tre nivåer, eller *levels*. Nivå 1 innehåller vektor-vektor-operationer, nivå 2 innehåller vektor-matris-operationer och nivå 3 innehåller matris-matris-operationer. Prestandan för de olika nivåerna skiljer sig mycket. Nivå 3 har bäst prestanda och det förklaras med att nivå 3 utför betydligt fler operationer per minnesreferens än de två övriga nivåerna [3]. Tabell 2.2 innehåller en jämförelse av antal operationer och minnesreferenser för en funktion från varje nivå av BLAS [3].

2.6.2 LAPACK

LAPACK [14] (Linear Algebra PACKage) är ett bibliotek som främst innehåller rutiner för att lösa linjära ekvationssystem, minstakvadratproblem och egenvärdesproblem. Matrisfaktoriseringar (LU, Cholesky, QR, SVD, Schur och generaliserad Schur) och andra beräkningar (t.ex. omordning av Schurfaktorisering och uppskattning av villkorstal) som är nära förknippade med de nämnda problemen ingår också i LAPACK.

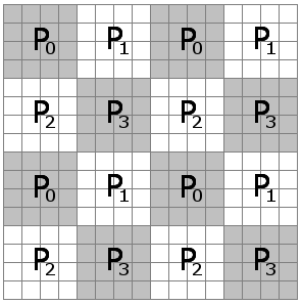
I LAPACK ingår rutiner för både täta matriser och för bandmatriser och av dessa finns det också versioner där både reella och komplexa matriser hanteras. Det finns versioner för både enkel precision och dubbel precision av flyttal.

LAPACK använder så mycket av nivå 3 BLAS som möjligt för att utföra sina beräkningar och säkerställer genom detta också att rutinerna ger bra prestanda. Detta förutsätter dock, med tanke på den aktuella maskinvaran, att en effektiv och optimerad version av BLAS används.

2.6.3 BLACS

BLACS [4] (Basic Linear Algebra Communication Subprograms) är ett gränssnitt för rutiner som utför distribution av data, samt kommunikation mellan noderna i ett parallelldatorsystem. BLACS är framtaget för att användas till linjär algebra och är därför speciellt anpassat till detta till skillnad från de flesta andra gränssnitt för distribution av data och kommunikation i ett parallelldatorsystem.

Vid distribution av data används en 2-dimensionell blockcyklisk distribution. Figur 2.2 visar ett exempel på hur en sådan distribution ser ut då en 16×16-matris med blockstorleken 4×4 distribueras på en 2×2-processorgrid.



Figur 2.2: Exempel på en 2-dimensionell blockcyklisk datadistribution av en 16×16-matris på en 2×2-processorgrid där blockstorleken är 4×4.

BLACS innehåller 3 olika typer av kommunikationsrutiner. De första två typerna av rutiner är rutiner för en-till-en-kommunikation och en-till-alla-kommunikation. Den tredje typen av rutiner är kombinations rutiner för elementvis summering och elementvis maximum eller minimum. En-till-alla-kommunikation och kombinationsrutinerna kan utföras i tre olika omgivningar. Dessa är radvis, kolumnvis eller då alla noder ingår.

2.6.4 PBLAS

PBLAS [17] (Parallel Basic Linear Algebra Subprograms) är ett bibliotek som innehåller parallella versioner av de flesta rutinerna som finns i BLAS.

PBLAS använder sig av BLACS för att sköta kommunikation och uppdelning mellan noderna och BLAS för att uppnå bra prestanda för beräkningarna på varje nod.

2.6.5 ScaLAPACK

Biblioteket ScaLAPACK [24] (Scalable Linear Algebra PACKage) innehåller parallella versioner av en delmängd av rutinerna i LAPACK.

ScaLAPACK använder sig av PBLAS och LAPACK för att utföra effektiva beräkningar och BLACS för datadistribution och kommunikation.

2.6.6 RECSY

RECSY [23] är ett bibliotek som använder rekursiva algoritmer för att lösa 42 (se Tabell 2.3) olika varianter av triangulära ekvationer av Sylvestertyp. RECSY bygger på algoritmer presenterade av Jonsson och Kågström [12, 13].

RECSY använder sig av 3 olika rekursionsmönster beroende på värdena av m och n då $X \in \mathbb{R}^{m \times n}$. För TRSYCT utnyttjas det första rekursionsmönstret då $n \leq 2m$ och ger att A delas båda vertikalt och horisontellt medan X och C delas horisontellt. Det ger upphov till ekvationen

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} - \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} B = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}.$$

Att lösa denna ekvation är ekvivalent med att lösa de två TRSYCT

$$A_{11}X_1 - X_1B = C_1 - A_{12}X_2$$

Tabell 2.3: De 42 varianter av Sylvester- och Lyapunovekvationer som kan beräknas med hjälp av RECSY.

$op(A)X \pm Xop(B) = C$	Tidskontinuerlig Sylvester
$op(A)Xop(B) \pm X = C$	Tidsdiskret Sylvester
$op(A)X + Xop(A^T) = C$	Tidskontinuerlig Lyapunov
$op(A)Xop(A^T) - X = C$	Tidsdiskret Lyapunov
$op(A)Xop(B) \pm op(C)Xop(D) = E$	Generaliserad Sylvester
$op(A)X \pm Yop(B) = C$ $op(D)X \pm Yop(E) = F$	Generaliserad kopplad Sylvester
$op(A)Xop(E^T) + op(E)Xop(A^T) = C$	Tidskontinuerlig Generaliserad Lyapunov
$op(A)Xop(A^T) - op(E)Xo(E)^T = C$	Tidsdiskret Generaliserad Lyapunov

och

$$A_{22}X_2 - X_2B = C_2.$$

Detta görs genom att först lösa $A_{22}X_2 - X_2B = C_2$ och sedan använda sig av X_2 för att lösa $A_{11}X_1 - X_1B = C_1 - A_{12}X_2$.

Det andra rekursionsmönstret används då $m \leq 2n$ och innebär att B delas både vertikalt och horisontellt medan X och C delas vertikalt enligt

$$A \begin{bmatrix} X_1 & X_2 \end{bmatrix} - \begin{bmatrix} X_1 & X_2 \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ & B_{22} \end{bmatrix} = \begin{bmatrix} C_1 & C_2 \end{bmatrix}$$

Detta ger upphov till följande TRSYCT

$$AX_1 - X_1B_{11} = C_1$$

och

$$AX_2 - X_2B_{22} = C_2 + X_1B_{12}$$

som löses analogt med ekvationerna som uppstår vid det första rekursionsmönstret.

Det tredje rekursionsmönstret innebär att A, B, C och X alla delas både vertikalt och horisontellt enligt

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} - \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Detta ger upphov till följande TRSYCT

$$A_{11}X_{11} - X_{11}B_{11} = C_{11} - A_{12}X_{21},$$

$$A_{11}X_{12} - X_{12}B_{22} = C_{12} - A_{12}X_{22} + X_{11}B_{12},$$

$$A_{22}X_{21} - X_{21}B_{11} = C_{21}$$

och

$$A_{22}X_{22} - X_{22}B_{22} = C_{22} + X_{21}B_{12}.$$

Här löses $A_{22}X_{21} - X_{21}B_{11} = C_{21}$ först för att sedan använda sig av X_{21} för att lösa $A_{11}X_{11} - X_{11}B_{11} = C_{11} - A_{12}X_{21}$ och $A_{22}X_{22} - X_{22}B_{22} = C_{22} + X_{21}B_{12}$ och slutligen används X_{11} och X_{22} för att lösa $A_{11}X_{12} - X_{12}B_{22} = C_{12} - A_{12}X_{22} + X_{11}B_{12}$.

Tabell 2.4: De 44 varianter av Sylvester- och Lyapunovekvationer som kan beräknas med hjälp av SCASY.

$op(A)X \pm Xop(B) = C$	Tidskontinuerlig Sylvester
$op(A)Xop(B) \pm X = C$	Tidsdiskret Sylvester
$op(A)X + Xop(A^T) = C$	Tidskontinuerlig Lyapunov
$op(A)Xop(A^T) - X = C$	Tidsdiskret Lyapunov
$op(A)Xop(B) \pm op(C)Xop(D) = E$	Generaliserad Sylvester
$op(A)X \pm Yop(B) = C$ $op(D)X \pm Yop(E) = F$	Generaliserad kopplad Sylvester
$op(A)Xop(E^T) + op(E)Xop(A^T) = C$	Tidskontinuerlig Generaliserad Lyapunov
$op(A)Xop(A^T) - op(E)Xo(E)^T = C$	Tidsdiskret Generaliserad Lyapunov

Rekursionen i RECSY avbryts då $m,n \leq 4$ och ekvationerna löses sedan med hjälp av snabba kärnor som utnyttjar att ekvationerna av Sylvestertyp kan skrivas om som den linjära ekvationen $Zx = c$ där Z är en Kroneckerprodukt som representerar den aktuella ekvationen, $x = vec(X)$ och $c = vec(C)$. Kroneckerprodukten för den tidskontinuerliga Sylvesterekvationen är $Z = I_n \otimes A - B^T \otimes I_m$ [12, 13].

RECSY innehåller även parallella rutiner som bygger på OpenMP [15] för att användas på symmetriska multiprocessorer.

2.6.7 SCASY

SCASY [25] är ett bibliotek som innehåller parallella rutiner för att lösa 44 (se Tabell 2.4) ekvationer av Sylvestertyp. För att lösa ekvationerna används Bartels-Stewarts metod.

För att lösa de triangulära ekvationerna som är ett av stegen i Bartels-Stewarts metod används vågfrontsalgoritmer presenterade av Granat och Kågström [20, 21]. Algoritmen bygger på blockning där t.ex. TRSYCT kan skrivas om som

$$A_{ii}X_{ij} - X_{ij}B_{jj} = C_{ij} - \left(\sum_{k=i+1}^{D_A} A_{ik}X_{kj} - \sum_{k=1}^{j-1} X_{ik}B_{kj} \right)$$

där $i = 1,2,\ldots,D_A$, $j = 1,2,\ldots,D_B$, D_A anger antalet diagonalblock i A och D_B anger antalet diagonalblock i B. Eftersom A och B är övertriangulära innehåller summorna då X_{ij} beräknas enbart X_{kl} där $k \geq j$ och $l \leq i$. Detta innebär att alla block X_{ij} endast är beroende av block X_{kl} som ligger under och till vänster om sig självt och därmet kan beräknas om X_{kl} redan har beräknats. Detta ger upphov till Algoritm 1 som är den parallella algoritm för att lösa TRSYCT som används i SCASY.

På liknande sätt går det att skriva om de andra ekvationerna av Sylvestertyp presenterade i Tabell 2.4 på blockad form och konstruera algoritmer för att lösa dessa. Mer information om dessa ekvationer och deras respektive algoritm finns presenterat av Granat och Kågström [20, 21].

För att lösa de små ekvationsystemen som uppstår använder sig SCASY av rutiner från RECSY för att uppnå bra prestanda lokalt på varje nod.

Algoritm 1 Parallell algoritm för TRSYCT

Indata: Matriserna A, B och C. A och B är i Schurform
Utdata: Lösningsmatrisen X.

```
for l = 1, antal blockdiagonaler i C do
  % Lös delsystem på aktuell blockdiagonal parallellt
  if min nod innehar C_ij then
    if min nod inte innehar A_ii och/eller B_jj then
      Kommunicera för att få A_ii och/eller B_jj
    end if
    Lös A_ii X_ij − X_ij B_jj = C_ij seriellt
    Sänd X_ij till processorer som håller block i blockrad i eller blockkolumn j
  else if min nod behöver X_ij then
    Ta emot X_ij
  end if
  if min nod inte innehar nödvändiga block i A för att uppdatera blockkolumn j then
    Kommunicera för att få nödvändiga block i A
  end if
  Uppdatera blockkolumn j i C parallellt
  if min nod inte innehar nödvändiga block i B för att uppdatera blockrad i then
    Kommunicera för att få nödvändiga block i B
  end if
  Uppdatera blockrad i i C parallellt
end for
```

Kapitel 3

Prestandautvärdering av SCASY

Detta kapitel behandlar prestandautvärderingen av SCASY då det körs på en parallelldator med distribuerat minne och där varje nod består av en symmetrisk multiprocessor innehållande två processorer.

3.1 Utförande

Prestandan för en omodifierad version av SCASY jämförs i denna sektion med en version av SCASY som är modifierad för att köra på en parallelldator med distribuerat minne där varje nod består av en symmetrisk multiprocessor.

Då största delen av arbetet som SCASY utför vid lösning av ekvationer av Sylvestertyp består av matrismultiplikation, då rader och kolumner ska uppdateras, samt lösning av små ekvationssystem, med hjälp av RECSY, så använder sig den modifierade version av en trådad version av BLAS samt OpenMP-versionerna av rutinerna i RECSY för att på detta sätt utnyttja kapaciteten hus en symmetrisk multiprocessor.

I den modifierade versionen kan antal trådar som den trådade versionen av BLAS använder sig av samt antal trådar som OpenMP använder sig av ställas in.

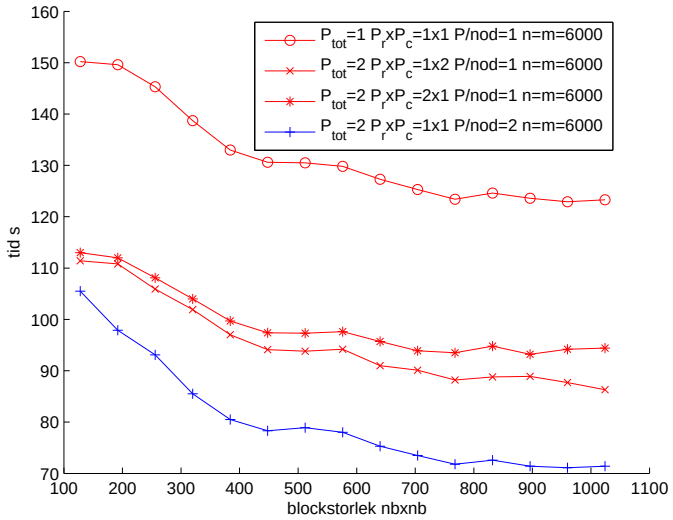
3.2 Resultat

Nedan presenteras prestandaresultat för den modifierade versionen av SCASY där den jämförs med en omodifierad version av SCASY.

3.2.1 Hård- och mjukvara

Alla testkörningar utfördes på Linuxklustret Sarek som består av 384 64-bitars AMD Opteron processorer på 2,2 GHz som är fördelade på 192 duala noder. Varje nod har 1,5 TB RAM och de är sammankopplade med ett switchat Myrinet-2000 nätverk. Klustret har en maxprestanda på 1,69 Tflops och har uppmätt en prestanda på 1,33 Tflops med HP-Linpack benchmark [11].

Vid kompilering användes mpif77 och vid länkning mpif90. De bibliotek som användes var BLACS 1.1, GotoBLAS r0.94 [8], LAPACK 3.0, ScaLAPACK 1.7 samt RECSY 0.01 alpha. Till den modifierade versionen av SCASY användes en trådad version av GotoBLAS samt OpenMP-versionerna av rutinerna i RECSY.



Figur 3.1: Prestandaresultat för den omodifierade och modifierade version av SCASY vid körning på 1 och 2 processorer.

3.2.2 Prestanda

Prestandaresultat för körningar på 1 och 2 processorer för den omodifierade samt den modifierade versionen av SCASY presenteras i Figur 3.1. Resultaten visar att det ger bättre prestanda att använda sig av en nod som en parallelldator med gemensamt minne än som en parallelldator med distribuerat minne. Det syns även att det är när blockstorleken är 400×400 och däröver som skillnaden är som störst.

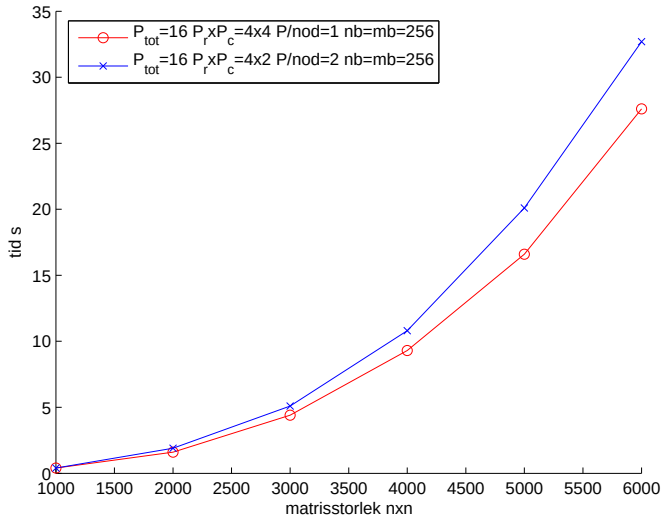
Tabell 3.1 innehåller prestandatester för 16 och 32 processorer både som 4×2- och 4×4-processorgrid där varje nod består av en symmetrisk multiprocessor med 2 processorer och som 4×4- och 4×8-processorgrid där varje nod endast består av en processor. Man kan se av tiderna i Tabell 3.1 och av Figur 3.2 att då 16 processorer används har den omodifierade versionen bättre prestanda än den modifierade. Men som man kan se av tiderna i Tabell 3.1 och av Figur 3.3 så är prestandan nästan identisk mellan de olika versionenerna när 32 processorer används. Förklaringen till detta är troligtvis att SCASY fungerar bättre på en kvadratisk processorgrid då SCASY vid körning på en processorgrid av storlek $P_r \times P_c$ kan lösa $min(P_r, P_c)$ delsystem på en diagonal parallellt [20].

3.3 Begränsningar

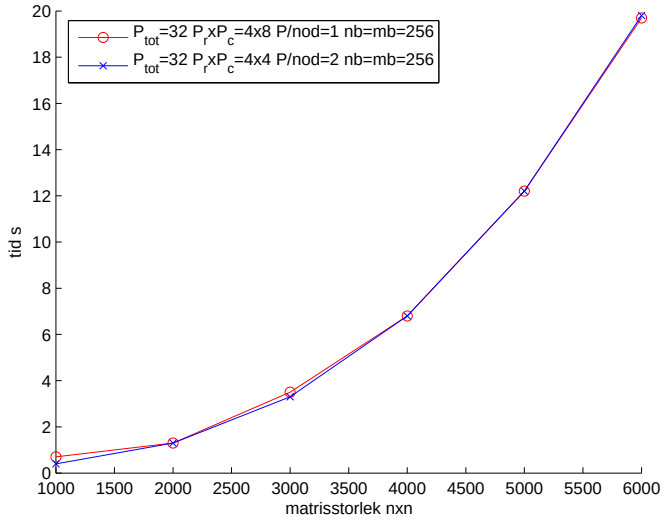
Då BLAS anropas både direkt från SCASY och indirekt via RECSY uppstår problem då ett trådat BLAS anropas från en OpenMP-tråd. Av denna anledning har endast en OpenMP-tråd använts vilket är ekvivalent med att använda sig av de RECSY-rutiner som inte använder sig av OpenMP.

Tabell 3.1: Prestandaresultat för SCASY på 32 processorer. Resultat både för en 4×8-processorgrid utan smp och en 4×4-processorgrid med smp.

$m \times n$	$P_r \times P_c$	P/nod	BLAS T	OMP T	$mb \times nb$	Tid
1000×1000	4×2	2	2	1	256×256	$0,4$
2000×2000	4×2	2	2	1	256×256	$1,9$
3000×3000	4×2	2	2	1	256×256	$5,1$
4000×4000	4×2	2	2	1	256×256	$10,8$
5000×5000	4×2	2	2	1	256×256	$20,1$
6000×6000	4×2	2	2	1	256×256	$32,7$
1000×1000	4×4	2	2	1	256×256	$0,4$
2000×2000	4×4	2	2	1	256×256	$1,3$
3000×3000	4×4	2	2	1	256×256	$3,3$
4000×4000	4×4	2	2	1	256×256	$6,8$
5000×5000	4×4	2	2	1	256×256	$12,2$
6000×6000	4×4	2	2	1	256×256	$19,8$
1000×1000	4×4	1	1	1	256×256	$0,4$
2000×2000	4×4	1	1	1	256×256	$1,6$
3000×3000	4×4	1	1	1	256×256	$4,4$
4000×4000	4×4	1	1	1	256×256	$9,3$
5000×5000	4×4	1	1	1	256×256	$16,6$
6000×6000	4×4	1	1	1	256×256	$27,6$
1000×1000	4×8	1	1	1	256×256	$0,7$
2000×2000	4×8	1	1	1	256×256	$1,3$
3000×3000	4×8	1	1	1	256×256	$3,5$
4000×4000	4×8	1	1	1	256×256	$6,8$
5000×5000	4×8	1	1	1	256×256	$12,2$
6000×6000	4×8	1	1	1	256×256	$19,7$



Figur 3.2: Prestandajämförelse mellan att köra SCASY på en 4×4-processorgrid utan SMP på noderna och att köra på en 4×2-processorgrid med SMP med 2 processorer på noderna.



Figur 3.3: Prestandajämförelse mellan att köra SCASY på en 4×8-processorgrid utan SMP på noderna och att köra på en 4×4-processorgrid med SMP med 2 processorer på noderna.

Kapitel 4

Algoritmer för triangulära periodiska tidskontinuerliga Sylvesterekvationer

Det här kapitlet handlar om den parallella algoritmen PTRPSYCTD som löser den triangulära periodiska tidskontinuerliga Sylvesterekvationen (TRPSYCT). Kapitlet innehåller en beskrivning av algoritmen som används, implementationsdetaljer samt prestandatester.

4.1 Algoritm

Algoritmen som används är en generaliserad version av den parallella algoritmen för att lösa TRSYCT presenterad av Granat och Kågström [20, 21] och som används i SCASY. Algoritmen är generaliserad så att den förutom att kunna lösa TRSYCT även löser TRPSYCT. Den generaliserade algoritmen presenteras i Algoritm 2.

Algoritmen är en vågfrontsalgoritm som består av två faser som bygger på att TRPSYCT kan skrivas på blockad form enligt

$$A_{ii}^{(k)} X_{ij}^{(k)} - X_{ij}^{(k\oplus 1)} B_{jj}^{(k)} = C_{ij}^{(k)} - \left(\sum_{k=i+1}^{D_A} A_{ik}^{(k)} X_{kj}^{(k)} - \sum_{k=1}^{j-1} X_{ik}^{(k\oplus 1)} B_{kj}^{(k)} \right)$$

vilket innebär att matrissekvensen X_{ij} endast beror av matrissekvenser X_{kl} där $i < k$ och $j > k$. I den första fasen löses delsystemen, $A_{ii}^{(k)} X_{ij}^{(k)} - X_{ij}^{(k\oplus 1)} B_{jj}^{(k)} = C_{ij}^{(k)}$, på en blockdiagonal parallellt. I den andra fasen så uppdateras berörda kolumner och rader med avseende på resultatet för matrissekvensen X_{ij}, som beräknas i första fasen, enligt $C_{lj}^{(k)} = C_{lj}^{(k)} - A_{li}^{(k)} X_{ij}^{(k)}$ där $l > i$ respektive $C_{il}^{(k)} = C_{il}^{(k)} + X_{ij}^{(k\oplus 1)} B_{jl}^{(k)}$ där $l < j$.

4.1.1 Transponeringsvarianter

PTRPSYCT ska klara av att lösa fyra olika transponeringsvarianter av TRPSYCT. De fyra varianterna är

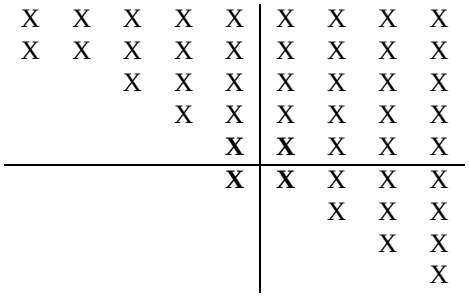
$$A_k X_k - X_{k\oplus 1} B_k = C_k,$$
$$A_k^T X_k - X_{k\oplus 1} B_k = C_k,$$

Algoritm 2 Parallell algoritm för TRPSYCT

Indata: Matriserna A_k, B_k och C_k där $k = 0,1,\ldots,p-1$ och p är perioden. A_s och $B_{s'}$ där $\{s,s'\} \in \{0,1,\ldots,p-1\}$ är i Schurform medan A_t där $t = 1,2,\ldots,s-1,s+1,\ldots,p$ och $B_{t'}$ där $t' = 1,2,\ldots,s'-1,s'+1,\ldots,p$ är triangulära.

Utdata: Lösningsmatriserna X_k där $k = 1,2,\ldots,p$

```
% q ∈ {1,2,...,p}
% r = 1,2,...,p
for l = 1, antal blockdiagonaler i C_q do
  % Lös delsystem på aktuell blockdiagonal parallellt
  if min nod innehar C_ij^(r) then
    if min nod inte innehar A_ii^(r) och/eller B_jj^(r) then
      Kommunicera för att få A_ii^(r) och/eller B_jj^(r)
    end if
    Lös A_ii^(r) X_ij^(r) − X_ij^(r⊕1) B_jj^(r) = C_ij^(r) seriellt
    Sänd X_ij^(r) till processorer som håller block i blockrad i eller blockkolumn j
  else if min nod behöver X_ij^(r) then
    Ta emot X_ij^(r)
  end if
  if min nod inte innehar nödvändiga block i A_r för att uppdatera blockkolumn j then
    Kommunicera för att få nödvändiga block i A_r
  end if
  Uppdatera blockkolumn j i C_r parallellt
  if min nod inte innehar nödvändiga block i B_r för att uppdatera blockrad i then
    Kommunicera för att få nödvändiga block i B_r
  end if
  Uppdatera blockrad i i C_r parallellt
end for
```



Figur 4.1: Exempel på ett 2×2-block på diagonalen som är delat mellan två olika block i den blockcykliska distributionen.

$$A_k X_k - X_{k\oplus 1} B_k^T = C_k$$

och

$$A_k^T X_k - X_{k\oplus 1} B_k^T = C_k.$$

Skillnaden mellan de olika fallen blir att vågfrontsalgoritmen har olika riktning och att uppdateringen av rader och kolumner har en annan riktning [18]. Då varken matriserna i matrissekvens A eller matrissekvens B är transponerade kommer algoritmen att ha nordostlig riktning, när endast matriserna i matrissekvens A är transponerad kommer algoritmen att ha en sydostlig riktning, när endast matriserna i matrissekvens B är transponerad kommer algoritmen att ha en nordvästlig riktning och när både matriserna i matrissekvens A och matrissekvens B är transponerade kommer algoritmen att ha en sydvästlig riktning.

4.1.2 Delade 2×2-block på diagonalen

Då matriserna $A^{(s)}$ och $B^{(s')}$ där $\{s,s'\} \in \{1,2,\ldots,p\}$ är på Schurform, och därmed är övertriangulär med 1×1- och/eller 2×2-block på diagonalen, och distribuerad blockcyklisk med block av storleken $mb \times nb$ kan det hända att 2×2-block på diagonalen delas mellan olika processorer enligt Figur 4.1.

För att lösa detta problem används en explicit omdistribution av elementen som beskrivs i [9]. Omdistributionen bygger på att om ett 2×2-block är delat mellan olika processorer så utvidgas ett block medan det andra förminskas. Om $A_{ij}^{(s)}$ och $A_{(i+1)(j+1)}^{(s')}$ delar ett 2×2 block kommer matrissekvensen A_{ij} att utvidgas med en rad och en kolumn och matrissekvensen $C_{(ij)}$ utvidgas med en rad. Om $B_{ij}^{(s)}$ och $B_{(i+1)(j+1)}^{(s')}$ delar ett 2×2 block kommer matrissekvensen B_{ij} att utvidgas med en rad och en kolumn och matrissekvensen C_{ij} utvidgas med en kolumn.

För omdistribution används tre rutiner som ingår i SCASY. PDEXTCHK beräknar hur omdistributionen ser ut, PDIMPRED utför omdistributionen och PDBCKRD distribuerar tillbaka till ursprunglig distribution. Användadet av dessa rutiner beskrivs i Algoritm 3.

4.2 Implementation

PTRPSYCT kan lösa TRPSYCT

$$op(A_k) X_k + s \cdot X_{k\oplus 1} op(B_k) = C_k$$

Algoritm 3 Omdistribution av delade 2×2 block

Indata: Matriserna A_k, B_k och C_k där $k = 0,1,\ldots,p-1$ och p är perioden. A_s och $B_{s'}$ där $\{s,s'\} \in \{1,2,\ldots,p\}$ är i Schurform medan A_t där $t = 1,2,\ldots,s-1,s+1,\ldots,p$ och $B_{t'}$ där $t' = 1,2,\ldots,s'-1,s'+1,\ldots,p$ är triangulära.

Utdata: Lösningsmatriserna $X^{(k)}$ där $k = 1,2,\ldots,p$

```
% q ∈ {1,2,...,p}
% r = 1,2,...,p
% Beräkna hur omdistrubitionen ska ske m.a.p. A_s och B_s'.
Anropa PDEXTCHK(A_s,B_s')
% Omdistribuera matriserna A_r, B_r och C_r
Anropa PDIMPRED(A_r,B_r,C_r)
% Lös ekvationen enligt Algoritm 2
% Återställ den ursprungliga distrubitionen av C^(r)
Anropa PDBCKRD(C_r)
```

där $op(A_k)$ anger A_k eller A_k^T, $op(B_k)$ anger B_k eller B_k^T och s är antingen 1 eller -1. Detta innebär att PTRPSYCTD kan lösa 8 olika tecken och transponeringvarianter av TRPSYCT.

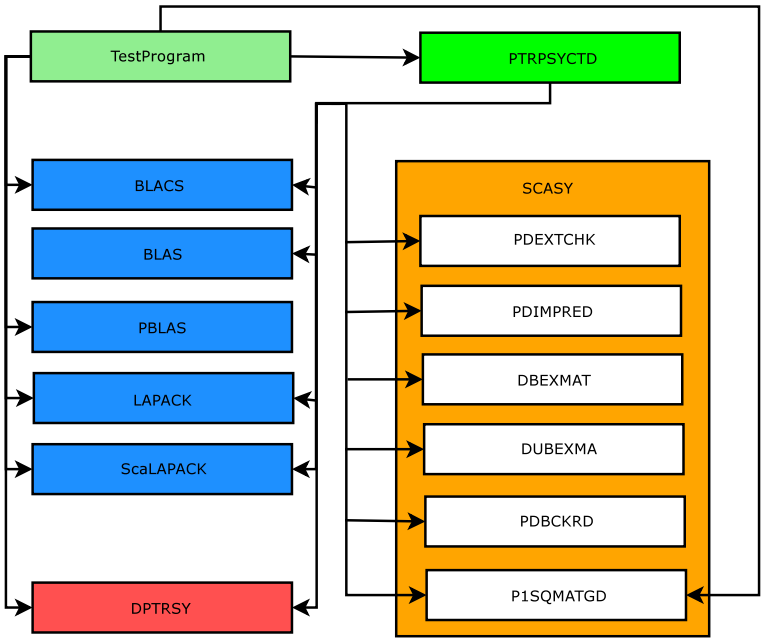
PTRPSYCTD är implementerat i ANSI Fortran 77 och använder sig av biblioteken BLACS, BLAS, LAPACK och ScaLAPACK. PTRPSYCTD använder sig dessutom av rutiner från SCASY samt DPTRSY som är en seriell lösare för TRPSYCT. Förutom PTRPSYCTD är ett testprogram, TestProgram, implementerat för att testa prestandan på PTRPSYCTD. En komplett systemöversikt där det framgår vilka bibliotek och rutiner TestProgram och PTRPSYCT använder sig av presenteras i Figur 4.2.

4.2.1 Parametrar

PTRPSYCTD har följande funktionshuvud

```
SUBROUTINE PTRPSYCTD( P, TRANSA, TRANSB, ISGN, M, N, ASCHUR, BSCHUR, A,
IA, JA, DESCA, B, IB, JB, DESCB, C, IC, JC, DESCC, DWORK, LDWORK, IWORK,
LIWORK, INFO ).
```

Parametern P anger ekvationens period. TRANSA och TRANSB kan anta värdena 'N' eller 'T' och anger om matriserna i matrissekvens A respektive B ska vara transponerade eller inte, och alltså anger vilken av transponeringsvarianterna som ska användas. ISGN kan anta värdena 1 eller -1 och anger att $op(A_k)X_k + X_{k\oplus 1}op(B_k) = C_k$ respektive $op(A_k)X_k - X_{k\oplus 1}op(B_k) = C_k$ ska lösas, och alltså anger vilken teckenvariant som ska lösas. M anger antal rader i matriserna i matrissekvenserna C och X samt antal rader och kolumner i matriserna i matrissekvens A. N anger antalet kolumner i matriserna i matrissekvenserna C och X samt antal rader och kolumner i matriserna i matrissekvens B. ASCHUR och BSCHUR antar värdena s respektive s' så att $\{s,s'\} \in \{1,2,\ldots,p\}$ och att A_s och $B_{s'}$ är i Schurform. A, B och C är de distribuerade och periodiska matrissekvenserna A, B respektive C. IA, JA, IB, JB, IC och JC anger start för delmatriser inuti matriserna i matrissekvenserna A, B respektive C. DESCA, DESCB och DESCC är BLACS deskriptorer för att beskriva de distribuerade och periodiska matrissekvenserna A, B respektive C. DWORK är arbetsminne bestående av flyttal och dess storlek är LDWORK. IWORK är arbetsminne bestående av heltal och dess storlek är LIWORK. INFO returnerar felmeddelanden.



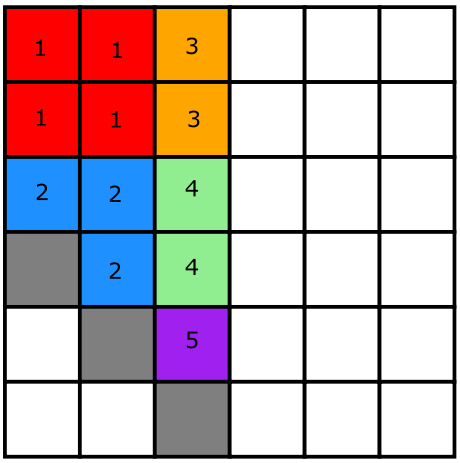
Figur 4.2: Systemöversikt där det framgår vilka bibliotek och funktioner som TestProgram och PTRPSYCT använder sig av.

4.2.2 Omordning av delade 2×2-block

För omordningen av 2×2-block på diagonalen som är delade mellan olika block i den block-cykliska datadistributionen används ett antal rutiner från SCASY. PDEXTCHK går igenom matriserna A_s och $B_{s'}$ som är på Schurform där $s \in \{0, 1, \ldots, p-1\}$ och p är perioden och söker efter 2×2-block på diagonalen som är delade mellan två processorer. PDEXCHK returnerar information om vilka block som ska utökas och vilka som ska förminskas. Denna information används i PDIMPRED för att utföra omdistributionen på alla matriser i matrissekvenserna A, B och C. Detta innebär p anrop till PDIMPRED. När beräkningen är klar används samma information av PDBCKRD för att återställa den ursprungliga distributionen av matriserna i matrissekvensen C och även detta kräver p anrop. Då PDIMPRED inte påverkar de ursprungliga matriserna utan istället spar utvidgningarna i egna arrayer behövs rutinerna DBEXMAT och DUBEXMA för att bygga ihop de tillfälliga blocken respektive plocka isär de tillfälliga blocken.

4.2.3 Lösning av delsystem

Då alla processorer har en global bild av hur matriserna i matrissekvenserna A, B och C är distribuerade bland processorerna så vet alla till vem eller från vem som block ska skickas för att delsystemen ska kunna lösas. Delsystemet $A_{ii}^{(k)} X_{ij}^{(k)} - X_{ij}^{(k \oplus 1)} B_{jj}^{(k)} = C_{ij}^{(k)}$ löses av processorn P som håller matrissekvensen C_{ij} vilket innebär att matrissekvenserna A_{ii} och B_{jj} kommer att skickas till P om inte P redan innehar dessa.



Figur 4.3: Uppdateringsordning vid uppdatering av A.

4.2.4 Uppdateringsfasen

I uppdateringfasen då en processorgrid med P processorer används kan P block uppdateras parallellt. Eftersom matriserna i matrissekvenserna A och B har en blockcyklisk distribution hålls $P_r \times P_c$ block som ligger bredvid varandra av olika processorer. Detta innebär att block om $P_r \times P_c$ kan uppdateras parallellt. Ett exempel på i vilken ordning blocken i A uppdateras kan ses i Figur 4.3.

4.2.5 Nodlösare

För att lösa den seriella TRPSYCT som uppstår i Algoritm 2 används två olika rutiner. Om ekvationen har perioden 1, d.v.s. ekvationen kan behandlas som en TRSYCT, används rutinen DTRSYL som ingår i LAPACK. Om ekvationen har en period större än 1 används rutinen DPTRSY, som diskuteras i [19].

4.2.6 Testprogram

TestProgram används för att testa korrekthet och prestanda på PTRPSYCTD. I TestProgram kan perioden och storleken på matrissekvenserna A, B och C, antal processorer, storlek på de distribuerade blocken och vilken tecken och transponeringsvariant som ska användas ställas in.

TestProgram redovisar tiden för de olika testkörningarna samt som ett mått på felet i beräkningen redovisas frobeniusnormen

$$E = max_k(||X_k - \hat{X}_k||_F)$$

där $k = 0, 1, \ldots, p-1$, p är perioden, X_k den exakta lösningen och $\hat{X}_k$ lösningen beräknad med PTRPSYCTD.

4.3 Resultat

Nedan presenteras resultat från testkörningar av PTRPSYCTD. Optimala värden på *mb* och *nb* har beräknas samt att prestandatester har utförts.

4.3.1 Hård- och mjukvara

Alla testkörningar utfördes på Linuxklustret Sarek. För kompilering och länkning användes mpif77. De bibliotek som användes var BLACS 1.1, GotoBLAS r0.94, LAPACK 3.0 och ScaLAPACK 1.7.

4.3.2 Generering av testdata

Testdatan är genererad så att A_1 och B_1 är på Schurform och har värdena $1,2,3\ldots$ respektive $-1,-2,-3,\ldots$ på diagonalen och där $\frac{1}{4}$ av talen består av komplexa konjugatpar. Talen nedanför diagonalen är 0 medan talen ovanför diagonalen är slumpade mellan 0 och 1. Testdatan för A_k och B_k, där $k = 2,3,4,\ldots p$, har ettor på diagonalen, nollor nedanför diagonalen och slumpade tal mellan 0 och 1 ovanför diagonalen. Matrissekvensen X fylls med slumptal mellan 0 och 1 och används sedan tillsammans med matrissekvenserna A och B för att beräkna matrissekvensen C.

4.3.3 Korrekthet

I Tabell 4.1 kan man se resultatet från testkörningar av PTRPSYCTD. Då perioden $p = 1$ och DTRSYL används som nodlösare är felet i beräkningen väldigt litet. Om perioden $p > 1$ och DPTRSY används som nodlösare ökar felet i beräkningen ju större matriser som beräknas. Då det inte är någon skillnad i algoritmen för de två fallen förutom att olika nodlösare används och att DPTRSY visar likadana fel i beräkningen då den körs seriellt (se Tabell 4.2) beror beräkningsfelet troligvis antigen på att testdatan är dåligt konditionerat eller på dålig stabilitet hos DPSYL2, som DPTRSY använder sig av.

4.3.4 Optimal blockstorlek

Tabell 4.3 innehåller resultat från testkörningar av PTRPSYCTD på en 2×2-processorgrid med problem av storlek 2000×2000 och perioden 4. Blockstorleken varierar från 64×64 till 1024×1024 och av resultaten i Tabell 4.3 och Figur 4.4 kan man se att PTRPSYCTD har bäst prestanda för problem med blockstorlek från ungefär 200×200 till problem med blockstorlek runt 450×450.

Med dessa resultat som grund har alla andra testkörningar där prestandan utvärderas utförts med blockstorleken 200×200.

4.3.5 Prestanda

Prestandaresultat för PTRPSYCTD när TRPSYCT med perioden 2 och 4 har lösts presenteras i Tabell 4.4 respektive Tabell 4.5. Av resultaten ser man att en ganska bra effektivitet åstadkoms för två processorer och att den sedan blir lägre och lägre ju fler processorer som används. Dock kan det vara så att problemstorleken är för liten för att nå upp i full prestanda vilket syns i Figur 4.5 och Figur 4.6 där graferna inte planar ut.

Värt att notera är att en 1×2 processorgrid ger bättre prestanda än en 2×1 processorgrid. Anledningen till detta är oklar men troligtvis beror det på hårdvaran eller den underliggande mjukvaran som troligvis fungerar bättre på en 1×2 processorgrid än på en 2×1 processorgrid.

Tabell 4.1: Resultat från testkörningar av PTRPSYCTD där felet, E, redovisas.

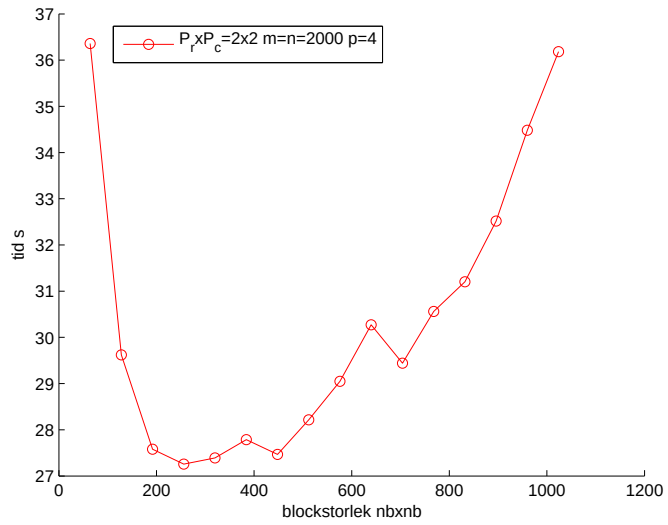
$m \times n$	$P_r \times P_c$	$mb \times nb$	p	E
10×10	2×2	10×10	1	$0,632 \times 10^{-15}$
60×60	2×2	10×10	1	$0,0125 \times 10^{-12}$
110×110	2×2	10×10	1	$0,0308 \times 10^{-12}$
160×160	2×2	10×10	1	$0,0554 \times 10^{-12}$
210×210	2×2	10×10	1	$0,0828 \times 10^{-12}$
260×260	2×2	10×10	1	$0,114 \times 10^{-12}$
310×310	2×2	10×10	1	$0,149 \times 10^{-12}$
10×10	2×2	10×10	2	$2,25 \times 10^{-15}$
60×60	2×2	10×10	2	$0,241 \times 10^{-9}$
110×110	2×2	10×10	2	$0,0565 \times 10^{-3}$
160×160	2×2	10×10	2	$0,271 \times 10^{3}$
210×210	2×2	10×10	2	$8,06 \times 10^{6}$
260×260	2×2	10×10	2	$0,0270 \times 10^{15}$
310×310	2×2	10×10	2	$3,49 \times 10^{18}$
10×10	2×2	10×10	4	$4,54 \times 10^{-15}$
60×60	2×2	10×10	4	$0,370 \times 10^{-6}$
110×110	2×2	10×10	4	$0,0258 \times 10^{3}$
160×160	2×2	10×10	4	$0,249 \times 10^{9}$
210×210	2×2	10×10	4	$0,274 \times 10^{18}$
260×260	2×2	10×10	4	$0,278 \times 10^{24}$
310×310	2×2	10×10	4	$8,26 \times 10^{30}$

Tabell 4.2: Resultat från seriella testkörningar av DPTRSY där felet, E, redovisas.

$m \times n$	$P_r \times P_c$	$mb \times nb$	p	E
10×10	1×1	10×10	2	$2,25 \times 10^{-15}$
60×60	1×1	10×10	2	$1,19 \times 10^{-9}$
110×110	1×1	10×10	2	$0,0480 \times 10^{-3}$
160×160	1×1	10×10	2	$0,144 \times 10^{3}$
210×210	1×1	10×10	2	$7,18 \times 10^{6}$
260×260	1×1	10×10	2	$0,0159 \times 10^{15}$
310×310	1×1	10×10	2	$5,97 \times 10^{18}$
10×10	1×1	10×10	4	$4,54 \times 10^{-15}$
60×60	1×1	10×10	4	$0,522 \times 10^{-6}$
110×110	1×1	10×10	4	$0,0359 \times 10^{3}$
160×160	1×1	10×10	4	$0,209 \times 10^{9}$
210×210	1×1	10×10	4	$0,309 \times 10^{18}$
260×260	1×1	10×10	4	$0,226 \times 10^{24}$
310×310	1×1	10×10	4	$3,40 \times 10^{30}$

Tabell 4.3: Prestandaresultat för PTRPSYCTD där $m = n = 2000$, $P_r = P_c = 2$ och $p = 4$ men där *mb* och *nb* varierar.

$m \times n$	$P_r \times P_c$	$mb \times nb$	p	Tid
2000×2000	2×2	64×64	4	$36,4$
2000×2000	2×2	128×128	4	$29,6$
2000×2000	2×2	192×192	4	$27,6$
2000×2000	2×2	256×256	4	$27,3$
2000×2000	2×2	320×320	4	$27,4$
2000×2000	2×2	384×384	4	$27,8$
2000×2000	2×2	448×448	4	$27,5$
2000×2000	2×2	512×512	4	$28,2$
2000×2000	2×2	576×576	4	$29,0$
2000×2000	2×2	640×640	4	$30,3$
2000×2000	2×2	704×704	4	$29,4$
2000×2000	2×2	768×768	4	$30,6$
2000×2000	2×2	832×832	4	$31,2$
2000×2000	2×2	896×896	4	$32,5$
2000×2000	2×2	960×960	4	$34,5$
2000×2000	2×2	1024×1024	4	$36,2$



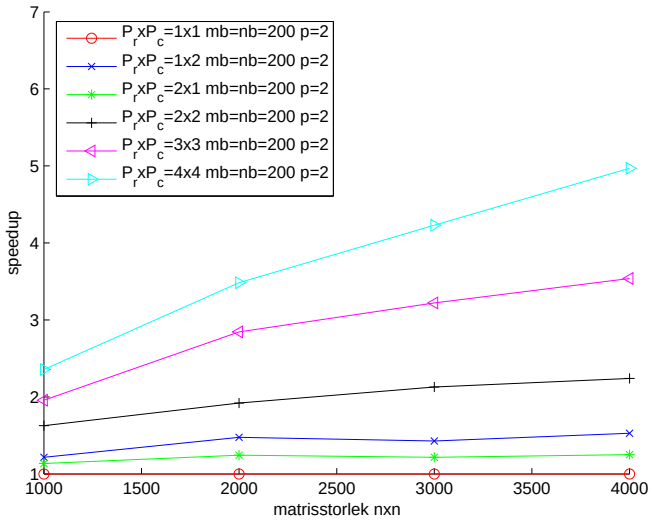
Figur 4.4: Prestandaresultat för PTRPSYCTD där $m = n = 2000$, $P_r = P_c = 2$ och $p = 4$ men där olika *mb* och *nb* varierar.

Tabell 4.4: Prestandaresultat för PTRPSYCTD där $p = 2$ och $mb = nb = 200$. Då $P_r = P_c = 1$ används DPTRSY istället för PTRSYCTD.

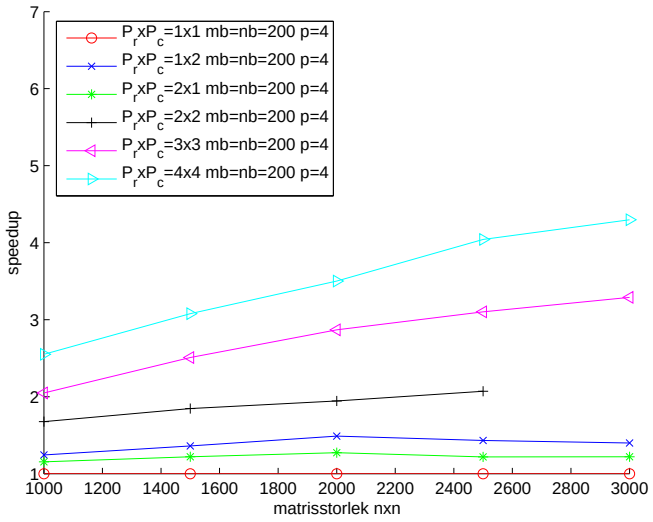
$m \times n$	$P_r \times P_c$	$mb \times nb$	p	Tid	Speedup	Effektivitet
1000×1000	1×1	200×200	2	3,86	1	1
2000×2000	1×1	200×200	2	20,8	1	1
3000×3000	1×1	200×200	2	58,4	1	1
4000×4000	1×1	200×200	2	124	1	1
1000×1000	1×2	200×200	2	3,17	1,22	0,610
2000×2000	1×2	200×200	2	14,1	1,48	0,740
3000×3000	1×2	200×200	2	40,9	1,43	0,715
4000×4000	1×2	200×200	2	81,3	1,53	0,765
1000×1000	2×1	200×200	2	3,40	1,14	0,570
2000×2000	2×1	200×200	2	16,7	1,25	0,625
3000×3000	2×1	200×200	2	50,0	1,46	0,730
4000×4000	2×1	200×200	2	99,3	1,25	0,625
1000×1000	2×2	200×200	2	2,37	1,63	0,407
2000×2000	2×2	200×200	2	10,8	1,93	0,482
3000×3000	2×2	200×200	2	27,4	2,13	0,532
4000×4000	2×2	200×200	2	55,4	2,24	0,560
1000×1000	3×3	200×200	2	1,97	1,96	0,218
2000×2000	3×3	200×200	2	7,31	2,85	0,317
3000×3000	3×3	200×200	2	18,1	3,23	0,359
4000×4000	3×3	200×200	2	35,1	3,53	0,392
1000×1000	4×4	200×200	2	1,63	2,37	0,148
2000×2000	4×4	200×200	2	5,97	3,48	0,217
3000×3000	4×4	200×200	2	13,8	4,23	0,264
4000×4000	4×4	200×200	2	25,0	4,96	0,310

Tabell 4.5: Prestandaresultat för PTRPSYCTD där $p = 4$ och $mb = nb = 200$. Då $P_r = P_c = 1$ används DPTRSY istället för PTRSYCTD.

$m \times n$	$P_r \times P_c$	$mb \times nb$	p	Tid	Speedup	Effektivitet
1000×1000	1×1	200×200	4	8,46	1	1
1500×1500	1×1	200×200	4	21,9	1	1
2000×2000	1×1	200×200	4	44,5	1	1
2500×2500	1×1	200×200	4	76,4	1	1
3000×3000	1×1	200×200	4	119	1	1
1000×1000	1×2	200×200	4	6,80	1.24	0,620
1500×1500	1×2	200×200	4	16,1	1,36	0,68
2000×2000	1×2	200×200	4	29,9	1,49	0,745
2500×2500	1×2	200×200	4	53,4	1,43	0,715
3000×3000	1×2	200×200	4	85,3	1,40	0,700
1000×1000	2×1	200×200	4	7,33	1,15	0,575
1500×1500	2×1	200×200	4	18,0	1,22	0,610
2000×2000	2×1	200×200	4	35,0	1,27	0,635
2500×2500	2×1	200×200	4	62,6	1,22	0,610
3000×3000	2×1	200×200	4	97,8	1.22	0,610
1000×1000	2×2	200×200	4	5,05	1,68	0,420
1500×1500	2×2	200×200	4	11,9	1,84	0,460
2000×2000	2×2	200×200	4	22,9	1,94	0,485
2500×2500	2×2	200×200	4	36,9	2,07	0,517
3000×3000	2×2	200×200	4	–	–	–
1000×1000	3×3	200×200	4	4,13	1,96	0,218
1500×1500	3×3	200×200	4	8,75	2,50	0,278
2000×2000	3×3	200×200	4	15,5	2,87	0,319
2500×2500	3×3	200×200	4	24,6	3,11	0,346
3000×3000	3×3	200×200	4	36,3	3.28	0,364
1000×1000	4×4	200×200	4	3,31	2,56	0,160
1500×1500	4×4	200×200	4	7,13	3,07	0,192
2000×2000	4×4	200×200	4	12,7	3,50	0,219
2500×2500	4×4	200×200	4	18,9	4.04	0,250
3000×3000	4×4	200×200	4	27,8	4,28	0,267



Figur 4.5: Prestandaresultat för PTRPSYCTD där perioden, p, är 2, och där processorgridens storlek, $P_r \times P_c$, varierar.



Figur 4.6: Prestandaresultat för PTRPSYCTD där perioden, p, är 4, och där processorgridens storlek, $P_r \times P_c$, varierar.

4.4 Begränsningar

PTRPSYCTD ställer vissa krav på utseendet på matrissekvenserna *A*, *B* och *C*. För det första måste matriserna i varje matrissekvens ligga efter varandra i minnet och för det andra måste alla delmatriser i en matrissekvens vara identiskt distribuerade bland processorerna. Förutom detta är det några samband mellan matrissekvenserna *A*, *B* och *C* som måste stämma. Om *A* har blockstorlek $mb \times mb$ och *B* har blockstorlek $nb \times nb$ måste *C* ha blockstorlek $mb \times nb$ och dessutom lika många blockrader som *A* och lika många blockkolumner som *B*.

Prestandan för PTRPSYCTD är begränsad när TRPSYCT med perioden 1 löses då en nodlösare från LAPACK används. Istället för en nodlösare från LAPACK skulle en mycket effektivare nodlösare från RECSY kunna användas.

Kapitel 5

Slutsatser och diskussion

Detta kapitel innehåller slutsatser och diskussion om de två delar som detta arbete omfattar.

5.1 Prestandautvärdering av SCASY

Detta arbete visar att det går att köra SCASY på en parallelldator med distribuerat minne och där varje nod består av en symmetrisk parallelldator. Resultaten visar ett det inte blir någon prestandaförsämring med denna metod jämför med den ursprungliga metoden som används i SCASY då denna metoden körs på en kvadratisk processorgrid och den ursprungliga på en icke kvadratisk processorgrid . Om däremot den ursprungliga metoden använder sig av en kvadratisk processorgrid och denna metod av en icke kvadratisk processorgrid är den ursprungliga metoden bättre.

Det är möjligt att om man istället för att ha en parallelldator med distribuerat minne där varje nod består av en symmetrisk parallelldator med två processorer istället har en där varje nod består av en symmetrisk parallelldator som har av fler an två processorer uppnår bättre prestanda än vad som uppnåtts i dessa tester. Det är också möjligt att den modifierade versionen ger bättre prestanda än den ursprungliga versionen av SCASY om en sådan parallelldator används.

5.2 Algoritmer för triangulära periodiska tidskontinuerliga Sylvesterekvationer

Detta arbete har visat att det går att lösa TRPSYCT med metoden som används och att denna metod är prestandaeffektiv. En speedup på nästan 5 för 16 processorer är troligtvis inte maximalt vad denna algoritm kan prestera då det verkar som att storleken på problemet har varit för litet för att uppnå full prestanda.

Då denna metod fungerar och ger tillfredsställande prestanda för TRPSYCT är det också troligt att de andra periodiska ekvationerna av Sylvestertyp går att lösa och ge tillfredsställande prestanda med denna metod.

Kapitel 6

Tack

Först vill jag tacka min handledare Robert Granat som föreslog detta examensarbeta och som har hjälp mig att genomföra detsamma. Sedan vill jag tacka HPC2N för att jag har fått använda deras datorsystem och HPC2Ns supportpersonal för deras snabba hjälp när problem har uppstått.

Referenser

[1] A. Bojanczyk, G. Golub och P. van Dooren. The Periodic Schur Decomposition. Algorithms and Applications. *Proceedings of the SPIE - The International Society for Optical Engineering*, 1770:31–42, 1992.

[2] ATLAS. ATLAS – Automatically Tuned Linear Algebra Software. Se http://math-atlas.sourceforge.net/.

[3] S. Blackford. ScaLAPACK user's Guide - ScaLAPACK Tutorial. http://www.netlib.org/scalapack/tutorial (besökt 2007-07-27).

[4] BLACS. BLACS – Basic Linear Algebra Communication Subprograms. Se http://www.netlib.org/blacs/index.html.

[5] BLAS. BLAS – Basic Linear Algebra Subprograms. Se http://www.netlib.org/blas/index.html.

[6] E. Andersson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney och D. Sorensen. LAPACK User's Guide. Thirt Edition. *SIAM Publications*, 1999.

[7] E. Elmroth, F. Gustavson, I. Jonsson och Bo Kågström. Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software. *SIAM Review*, 46(1):3–45, 2004.

[8] GotoBLAS. GotoBLAS – BLAS implementation av Kazushige Goto. Se http://www.tacc.utexas.edu/resources/software/.

[9] R. Granat. A Parallel ScaLAPACK-style Sylvester Solver. *Master's Thesis*, umnad 435/03, Department of Computer Science, 2003.

[10] J. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533, 1988.

[11] HPC2N. HPC2N – High Performance Computing Center North. Se http://www.hpc2n.umu.se.

[12] I. Jonsson och B. Kågström. Recursive Blocked Algorithms for Solving Triangular Systems-Part 1: One-Sided and Coupled Sylvester-Type Matrix Equations. *ACM Transactions on Mathematical Software*, 28(4):392–415, 2002.

[13] I. Jonsson och B. Kågström. Recursive Blocked Algorithms for Solving Triangular Systems-Part 2: Two-Sided and Generalized Sylvester and Lyapunov Matrix Equations. *ACM Transactions on Mathematical Software*, 28(4):416–435, 2002.

[14] LAPACK. LAPACK – Linear Algebra PACKage. Se http://www.netlib.org/lapack/index.html.

[15] OpenMP. OpenMP – Open Multi Processing. Se http://www.openmp.org.

[16] P. Lancaster och L. Rodman. *Algebraic Riccati Equations*. Oxford University Press, 1995.

[17] PBLAS. PBLAS – Parallel Basic Linear Algebra Subprograms. Se http://www.netlib.org/scalapack/pblas_qref.html.

[18] R. Granat, B. Kågström och Peter Poromaa. Parallel ScaLAPACK-style Algorithms for Solving Continous-Time Sylvester Equations. *I H. Kosch et al (Eds), Euro-Par 2003 Parallel Processing. LCNS, Springer Verlag*, 2790:800–809, 2003.

[19] R. Granat, I. Jonsson och B. Kågström. Recursive Blocked Algorithms for Solving Periodic Triangular Sylvester-type Matrix Equations. *kommer i PARA'06 State-of-the-Art in Scientific Computing Conference Proceedings, LCNS, Springer Verlag*, 2007.

[20] R. Granat och B. Kågström. Parallel Solvers for Sylvester-Type Matrix Equations with Applications in Condition Estimation, Part I: Theory and Algorithms. *Inlämnad till ACM Transactions on Mathematical Software*, juni 2007.

[21] R. Granat och B. Kågström. Parallel Solvers for Sylvester-Type Matrix Equations with Applications in Condition Estimation, Part II: The SCASY Software Library. *Inlämnad till ACM Transactions on Mathematical Software*, juni 2007.

[22] R. H. Bartels och G. W. Stewart. Solution of the Matrix Equation AX + XB = C. *Communications of the ACM*, 15:820–826, 1972.

[23] RECSY. RECSY – High Performance Library for Sylvester-Type Matrix Equations. Se http://www.cs.umu.se/∼isak/recsy/index.html.

[24] ScaLAPACK. ScaLAPACK – Scalable Linear Algebra PACKage. Se http://www.netlib.org/scalapack/index.html.

[25] SCASY. SCASY – Parallel ScaLAPACK-style High Performance Library for Sylvester-Type Matrix Equations. Se http://www.cs.umu.se/∼granat/scasy.html.