To Sue DeAngelis

From Eleanor Chu

Date Oct 6 '89

memo

University of Waterloo

Sue,

Would you please send me 5 copies of report CS-88-45 and CS-88-46 as soon as you receive the reprints.

Thanks a lot.

Eleanor

gave to Eleanor Oct. 17

81502

Printing Requisition / Graphic Services

1. Please complete unshaded areas on form as applicable.
2. Distribute copies as follows: White and Yellow to Graphic Services. Retain Pink Copies for your records.
3. On completion of order the Yellow copy will be returned with the printed material.
4. Please direct enquiries, quoting requisition number and account number, to extension 3451.

TITLE OR DESCRIPTION

Updating and Downdating the Inverse Cholesky Factor on a Hypercube Multiprocessor CS-~~89~~-88-46

DATE REQUISITIONED: ~~Sept.~~ OCT. 6/89

DATE REQUIRED: ASAP

ACCOUNT NO.: 4 1 2 4 1 0 0 6 0

REQUISITIONER - PRINT: J.A. George

PHONE: 2192

SIGNING AUTHORITY: [signature]

MAILING INFO - NAME: Sue DeAngelis

DEPT.: C.S.

BLDG. & ROOM NO.: DC 2314

[X] DELIVER [ ] PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 29

NUMBER OF COPIES: 50

TYPE OF PAPER STOCK: [X] BOND [ ] NCR ___ PT. [X] COVER [ ] BRISTOL [X] SUPPLIED [ ] ___

PAPER SIZE: [X] 8½ x 11 [ ] 8½ x 14 [ ] 11 x 17 [ ] ___

PAPER COLOUR: [X] WHITE [ ] ___

INK: [X] BLACK [ ] ___

PRINTING: [ ] 1 SIDE ___ PGS. [X] 2 SIDES ___ PGS.

NUMBERING: FROM TO

BINDING/FINISHING: 3 down left side

[X] COLLATING [X] STAPLING [ ] ___ HOLE PUNCHED [ ] PLASTIC RING

FOLDING/PADDING

CUTTING SIZE

Special Instructions

Math fronts and backs enclosed.

COPY CENTRE	OPER. NO.	BLDG.	MACH. NO.

DESIGN & PASTE-UP	OPER. NO.	TIME	LABOUR CODE
			D01
			D01
			D01

TYPESETTING	QUANTITY			LABOUR CODE
PAP00000				T01
PAP00000				T01
PAP00000				T01

PROOF				
PRF				
PRF				
PRF				

NEGATIVES	QUANTITY	OPER. NO.	TIME	LABOUR CODE
FLM				C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01

PMT				
PMT				C01
PMT				C01
PMT				C01

PLATES				
PLT				P01
PLT				P01
PLT				P01

STOCK				
				001
				001
				001
				001

BINDERY				
RNG				B01
RNG				B01
RNG				B01
MIS00000				B01

OUTSIDE SERVICES

$ ___ COST

TAXES - PROVINCIAL [ ] FEDERAL [ ]

GRAPHIC SERV. OCT. 85 482-2

Printing Requisition/Graphic Services

81501

1. Please complete unshaded areas on form as applicable.
2. Distribute copies as follows: White and Yellow to Graphic Services. Retain Pink Copies for your records.
3. On completion of order the Yellow copy will be returned with the printed material.
4. Please direct enquiries, quoting requisition number and account number, to extension 3451.

TITLE OR DESCRIPTION

A Balanced Submatrix Merging Algorithm for Multiprocessor Architectures CS-88-45

DATE REQUISITIONED: OCT. 6/89

DATE REQUIRED: ASAP

ACCOUNT NO.: 4 1 2 4 1 0 0 6 0

REQUISITIONER – PRINT: J.A. George

PHONE: 2192

SIGNING AUTHORITY: [signature]

MAILING INFO – NAME: Sue DeAngelis DEPT.: C.S. BLDG. & ROOM NO.: DC 2314 [X] DELIVER [ ] PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 30 NUMBER OF COPIES: 50

TYPE OF PAPER STOCK: [X] BOND [ ] NCR ___ PT. [X] COVER [ ] BRISTOL [X] SUPPLIED [ ] ___

PAPER SIZE: [X] 8½ x 11 [ ] 8½ x 14 [ ] 11 x 17 [ ] ___

PAPER COLOUR: [X] WHITE [ ] ___ INK: [X] BLACK [ ] ___

PRINTING: [ ] 1 SIDE ___ PGS. [X] 2 SIDES ___ PGS. NUMBERING: FROM TO

BINDING/FINISHING: 3 down left side [X] COLLATING [X] STAPLING [ ] ___ HOLE PUNCHED [ ] PLASTIC RING

FOLDING/PADDING CUTTING SIZE

Special Instructions

Math fronts and backs enclosed.

COPY CENTRE — OPER. NO. / BLDG. / MACH. NO.

DESIGN & PASTE-UP — OPER. NO. / TIME / LABOUR CODE

		D01
		D01
		D01

TYPESETTING — QUANTITY

PAP00000				T01
PAP00000				T01
PAP00000				T01

PROOF

PRF				
PRF				
PRF				

NEGATIVES — QUANTITY / OPER. NO. / TIME / LABOUR CODE

FLM				C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01

PMT

PMT				C01
PMT				C01
PMT				C01

PLATES

PLT				P01
PLT				P01
PLT				P01

STOCK

				001
				001
				001
				001

BINDERY

RNG				B01
RNG				B01
RNG				B01
MIS00000				B01

OUTSIDE SERVICES

$ ___ COST

TAXES – PROVINCIAL [ ] FEDERAL [ ]

GRAPHIC SERV. OCT. 85 482-2

2850

Printing Requisition/Graphic Services

81502

1. Please complete unshaded areas on form as applicable.
2. Distribute copies as follows: White and Yellow to Graphic Services. Retain Pink Copies for your records.
3. On completion of order the Yellow copy will be returned with the printed material.
4. Please direct enquiries, quoting requisition number and account number to extension 3451.

TITLE OR DESCRIPTION

Updating and Downdating the Inverse Cholesky Facéor on a Hypercube Multiprocessor CS-89-46

DATE REQUISITIONED: OCT 6/89

DATE REQUIRED: ASAP

ACCOUNT NO.: 4 1 2 4 1 0 0 6 0

REQUISITIONER – PRINT: J.A. George

PHONE: 2192

SIGNING AUTHORITY: Colin Bragg

MAILING INFO – NAME: Sue DeAngelis

DEPT.: C.S.

BLDG. & ROOM NO.: DC 2314

[X] DELIVER [ ] PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 29

NUMBER OF COPIES: 50

TYPE OF PAPER STOCK: [X] BOND [ ] INCR. ___ PT. [X] COVER [ ] BRISTOL [X] SUPPLIED [ ]

PAPER SIZE: [X] 8½ x 11 [ ] 8½ x 14 [ ] 11 x 17 [ ] 10m

PAPER COLOUR: [X] WHITE [ ]

INK: [X] BLACK [ ]

PRINTING: [ ] 1 SIDE ___ PGS. [X] 2 SIDES ___ PGS.

NUMBERING: FROM TO

BINDING/FINISHING: 3 down left side

[X] COLLATING [X] STAPLING [ ] PUNCHED [ ] PLASTIC RING

FOLDING/PADDING: STAPLES

CUTTING SIZE

Special Instructions

Math fronts and backs enclosed.

NEGATIVES	QUANTITY	OPER. NO.	TIME	LABOUR CODE
FLM				C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01

PMT				
PMT				C01
PMT				C01
PMT				C01

PLATES				
PLT		15		P01
PLT				P01
PLT				P01

STOCK				
		15		001
				001
				001
				001

BINDERY				
RNG		26		B01
RNG		26	20	B01
RNG				B01
MISC000				B01

COPY CENTRE	OPER. NO.	BLDG.	MACH. NO.

DESIGN & PASTE-UP	OPER. NO.	TIME	LABOUR CODE
			D01
			D01
			D01

TYPESETTING	QUANTITY			
PAP00000				T01
PAP00000				T01
PAP00000				T01

PROOF				
PRF				
PRF				
PRF				

OUTSIDE SERVICES

$ COST

TAXES – PROVINCIAL [ ] FEDERAL [ ]

GRAPHIC SERV. OCT 85 469-2

Printing Requisition/Graphic Services

2850

81501

1. Please complete unshaded areas on form as applicable.
2. Distribute copies as follows: White and Yellow to Graphic Services. Retain Pink Copies for your records.
3. On completion of order the Yellow copy will be returned with the printed material.
4. Please direct enquiries, printing requisition number and account number, to extension 3451.

TITLE OR DESCRIPTION: A Balanced Submatrix Merging Algorithm for Multiprocessor Architectures CS-88-45

DATE REQUISITIONED: Oct. 6/89

DATE REQUIRED: ASAP

ACCOUNT NO.: 4 1 2 4 1 0 0 6 0

REQUISITIONER - PRINT: J.A. George

PHONE: 2192

SIGNING AUTHORITY: [signature]

MAILING INFO — NAME: Sue DeAngelis DEPT.: C.S. BLDG. & ROOM NO.: DC 2314 [X] DELIVER [ ] PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 30 NUMBER OF COPIES: 50

TYPE OF PAPER STOCK: [X] BOND [ ] WEB. ___ PT. [X] COVER [ ] BRISTOL [X] SUPPLIED [ ]

PAPER SIZE: [X] 8½ x 11 [ ] 8½ x 14 [ ] 11 x 17 [ ] 10m

PAPER COLOUR: [X] WHITE [ ] ___ INK: [X] BLACK [ ] ___

PRINTING: [ ] 1 SIDE ___ PGS. [X] 2 SIDES ___ PGS. NUMBERING: FROM TO

BINDING/FINISHING: 3 down left side [X] COLLATING [X] STAPLING [ ] ___ HOLE PUNCHED [ ] PLASTIC RING

FOLDING/PADDING: 3 STAPLES CUTTING SIZE

Special Instructions

Math fronts and backs enclosed.

NEGATIVES	QUANTITY	OPER. NO.	TIME	LABOUR CODE
FLM				C01
FLM				C01
FLM				C01
FLM				C01
FLM				C01
PMT				
PMT				C01
PMT				C01
PMT				C01
PLATES				
PLT		15		P01
PLT				P01
PLT				P01
STOCK				
		15		001
				001
				001
				001
BINDERY				
RNG		26		B01
RNG		26	20	B01
RNG				B01
MIS 00000				B01

COPY CENTRE	OPER. NO.	BLDG.	MACH. NO.	
DESIGN & PASTE-UP	OPER. NO.	TIME	LABOUR CODE	
			D01	
			D01	
			D01	
TYPESETTING	QUANTITY			
PAP 00000			T01	
PAP 00000			T01	
PAP 00000			T01	
PROOF				
PRF				
PRF				
PRF				

OUTSIDE SERVICES

$ ___ COST

TAXES — PROVINCIAL [ ] FEDERAL [ ]

GRAPHIC SERV. OCT. 89 482-2

A Balanced Submatrix Merging Algorithm for Multiprocessor Architectures

Eleanor Chu
Alan George

Department of Computer Science

Research Report CS-88-45
November 1988

A Balanced Submatrix Merging Algorithm for Multiprocessor Architectures *

Eleanor Chu
Alan George

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Research Report CS-88-45

November 1988

Abstract

In this article we describe a parallel algorithm which applies Givens rotations to selectively annihilate $k(k+1)/2$ nonzero elements from two $k \times n$ ($k \leq n$) upper trapezoidal submatrices. The new algorithm we propose is suitable for implementation on *either* a pair of directly connected local-memory processors *or* two clusters of tightly-coupled processors. We show in both cases that the proposed algorithms may achieve optimal speed-up by balancing the work load distribution and masking inter-processor or inter-cluster communication by computation. In the context of solving large scale least squares problems, this submatrix merging step is repetitively needed during the entire computation, and, furthermore, there are usually many pairs of such submatrices to be merged with each submatrix stored in the memory of a processor or a cluster of processors. The proposed algorithm can be applied to each pair of submatrices concurrently and thus parallelizes an important step in solving the least squares problems.

*Research supported in part by NASA Grant No. NAG-1-803, and by the University of Waterloo

Contents

List of Figures

1 Introduction

In this article we study some effective ways to merge submatrices on multiprocessor architectures, and propose a cure to the unbalanced load distribution problem which the algorithms currently known to us have experienced. The particular submatrix merging operation we consider can be understood as eliminating $k(k+1)/2$ nonzeros by Givens rotations from a pair of $k \times n$ $(k \leq n)$ upper trapezoidal submatrices as depicted in Fig. 1, where $k = 6$, $n = 12$.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

Figure 1: Merging two upper trapezoidal submatrices.

The need to reduce multiple pairs of such submatrices arises in both dense and sparse matrix computations. An example of the former case can be found in the recursive fine partitioning (*rfp*) scheme proposed by Pothen et. al. in [2] for implementing dense QR factorization on a hypercube. An example of the latter case occurs in the parallel block schemes proposed by Golub et. al. in [1] for large scale least squares computations.

In our study of this submatrix merging process, we use the following definitions and observations.

1. We refer to the computations incurred in eliminating one nonzero element as a task.

2. Each task involves applying a Givens rotation to two rows with their leading nonzeros in the same position. Given in Fig. 2 is the sequential algorithm which implements the task for the following transformation:

$$\begin{pmatrix} a_{i,\ell} & a_{i,\ell+1} & a_{i,\ell+2} & \cdots & a_{i,n} \\ a_{j,\ell} & a_{j,\ell+1} & a_{j,\ell+2} & \cdots & a_{j,n} \end{pmatrix} \longrightarrow \begin{pmatrix} \tilde{a}_{i,\ell} & \tilde{a}_{i,\ell+1} & \tilde{a}_{i,\ell+2} & \cdots & \tilde{a}_{i,n} \\ 0 & \tilde{a}_{j,\ell+1} & \tilde{a}_{j,\ell+2} & \cdots & \tilde{a}_{j,n} \end{pmatrix}.$$

   Note that in Fig. 2, $a_{i,q}$ and $a_{j,q}$ $(\ell \leq q \leq n)$ are overwritten by $\tilde{a}_{i,q}$ and $\tilde{a}_{j,q}$.

3. If $(n - \ell + 1)$ is the number of nonzeros in each of the two rows, then the size of the task is measured by $4(n - \ell + 1)$ multiplicative operations.

```
if |a_{j,ℓ}| ≥ |a_{i,ℓ}| then
    t ← |a_{i,ℓ}|/|a_{j,ℓ}|
    s ← 1/√(1 + t^2)
    c ← st
else
    t ← |a_{j,ℓ}|/|a_{i,ℓ}|
    c ← 1/√(1 + t^2)
    s ← ct
for q = ℓ, ℓ+1, ···, n do
    v ← a_{i,q}
    w ← a_{j,q}
    a_{i,q} ← cv + sw
    a_{j,q} ← −sv + cw
```

Figure 2: Implementing the task of annihilating $a_{j,\ell}$ by Givens rotation.

4. A single task can be equally divided between two cooperative processors if each processor can access both rows but updates only one of them; i.e., both processors concurrently execute all of the steps given in Fig. 2 except for executing only

$$a_{i,q} \leftarrow cv + sw$$

or

$$a_{j,q} \leftarrow -sv + cw$$

in the **for** loop.

Throughout this manuscript whenever we divide a single task among two processors, we assume an even distribution of work load as described above.

5. All tasks involving disjoint pairs of rows can potentially be performed in parallel by different processors.

6. There are $k(k+1)/2$ nonzero elements to be eliminated in merging two $k \times n$ ($k \leq n$) upper trapezoidal submatrices. We note that these $k(k+1)/2$ elements do not necessarily come from the same submatrix as identified by $\otimes$ in Fig. 3. Instead, a Givens rotation can be applied to selectively zero out the $k(k+1)/2$ "$\otimes$" elements in Fig. 4. The same reduced matrix can be obtained by permuting the appropriate rows (as well as the corresponding right-hand-side elements) as shown in Fig. 5.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & & \otimes & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 3: $\otimes$: the elements to be annihilated.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \otimes & \times & \times & \times & \times & \times & \times \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 4: $\otimes$: the elements to be annihilated.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 5: Same result may be obtained by permuting the rows after the annihilation process.

2 The Multiprocessor Environments

In [1] the target machine is the University of Illinois Cedar system consisting of clusters of processors, where each cluster has a shared-memory system and the clusters are in turn interconnected via a single, system-wide shared memory. The data mapping strategy employed in [1] dictates that each cluster of processors have one $k \times n$ upper trapezoidal submatrix in their memory. To merge two such submatrices two clusters will cooperate with the aim to exploiting parallelism and minimizing inter-cluster communication. In [2] a parallel algorithm was proposed for merging two upper trapezoidal submatrices stored in the local memory of two directly connected processors. This algorithm was then embedded in the recursive fine partitioning scheme proposed in the same paper for implementing dense QR factorization on a hypercube multiprocessor.

In this study we shall propose a new submatrix merging algorithm which can be applied beneficially in either one of the multiprocessor environments considered above. However, in order to be clear and precise in our presentation, we shall postpone all discussion relating only to the Cedar system until the last section.

3 Submatrix Merging Algorithms

When merging two $k \times n$ upper trapezoidal submatrices on a single processor, the $k(k+1)/2$ nonzeros from one of the submatrices may be eliminated in many different orderings. For example, they may be eliminated column by column as shown in Fig. 6, or row by row as shown in Fig. 7 or diagonal by diagonal as shown in Fig. 8.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
1 & 3 & 6 & 10 & 15 & 21 & \times & \times & \times & \times & \times & \times \\
 & 2 & 5 & 9 & 14 & 20 & \times & \times & \times & \times & \times & \times \\
 & & 4 & 8 & 13 & 19 & \times & \times & \times & \times & \times & \times \\
 & & & 7 & 12 & 18 & \times & \times & \times & \times & \times & \times \\
 & & & & 11 & 17 & \times & \times & \times & \times & \times & \times \\
 & & & & & 16 & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

Figure 6: Column-by-column elimination sequence.

In order to devise an elimination sequence which is most suitable for parallel implementation, it is helpful to study the data access pattern of these three elimination sequences under the constraint that *a task may not involve data from both submatrices unless it cannot*

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \times & \times & \times & \times & \times & \times \\
 & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} & \mathbf{11} & \times & \times & \times & \times & \times & \times \\
 & & \mathbf{12} & \mathbf{13} & \mathbf{14} & \mathbf{15} & \times & \times & \times & \times & \times & \times \\
 & & & \mathbf{16} & \mathbf{17} & \mathbf{18} & \times & \times & \times & \times & \times & \times \\
 & & & & \mathbf{19} & \mathbf{20} & \times & \times & \times & \times & \times & \times \\
 & & & & & \mathbf{21} & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 7: Row-by-row elimination sequence.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\mathbf{1} & \mathbf{7} & \mathbf{12} & \mathbf{16} & \mathbf{19} & \mathbf{21} & \times & \times & \times & \times & \times & \times \\
 & \mathbf{2} & \mathbf{8} & \mathbf{13} & \mathbf{17} & \mathbf{20} & \times & \times & \times & \times & \times & \times \\
 & & \mathbf{3} & \mathbf{9} & \mathbf{14} & \mathbf{18} & \times & \times & \times & \times & \times & \times \\
 & & & \mathbf{4} & \mathbf{10} & \mathbf{15} & \times & \times & \times & \times & \times & \times \\
 & & & & \mathbf{5} & \mathbf{11} & \times & \times & \times & \times & \times & \times \\
 & & & & & \mathbf{6} & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 8: Diagonal-by-diagonal elimination sequence.

proceed without doing so. For each elimination sequence we identify the tasks which must access data from both submatrices and display such tasks and the required data in Fig. 9, 10 and 11.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\mathbf{1} & \mathbf{3} & \mathbf{6} & \mathbf{10} & \mathbf{15} & \mathbf{21} & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

Figure 9: Column-by-column elimination sequence: data from both submatrices are needed for tasks 1, 3, 6, 10, 15 and 21.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\mathbf{1} & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \mathbf{7} & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \mathbf{12} & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \mathbf{16} & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \mathbf{19} & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \mathbf{21} & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

Figure 10: Row-by-row elimination sequence: data from both submatrices are needed for tasks 1, 7, 12, 16, 19 and 21.

We observe from Fig. 9, 10 and 11 that each row of data from the top submatrix is used in exactly one task. Since the two submatrices are each located in a different processor or a different cluster of processors in the multiprocessor environment considered in [2] and [1], the tasks identified above are also those which require inter-processor or inter-cluster communication. Since each row of data in the top submatrix must participate in at least one task during the entire merging operation and all such tasks annihilate nonzeros in the bottom submatrix which is stored in a different processors (or a different cluster), the goal of *minimizing* inter-processor (or inter-cluster) communication can be achieved via any one elimination sequence described here. The particular sequence chosen for parallel implementation in [2] and [1] is the diagonal-by-diagonal elimination sequence.

However, using the approach above the tasks requiring inter-processor communication

$$
\begin{array}{c}
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\end{pmatrix} \\
\begin{array}{cccccccccccc}
\mathbf{1} & \mathbf{7} & \mathbf{12} & \mathbf{16} & \mathbf{19} & \mathbf{21} & \times & \times & \times & \times & \times & \times
\end{array}
\end{array}
$$

Figure 11: Diagonal-by-diagonal elimination sequence: data from both submatrices are needed for tasks 1, 7, 12, 16, 19 and 21.

are the only tasks which can be performed by the two processors (or clusters) concurrently. Minimizing the number of such tasks can thus cause unbalanced work load distribution. This problem is more serious when two local-memory processors instead of two clusters are in question. In fact, it can be easily verified that when maintaining minimum inter-processor communication as suggested earlier, the parallel algorithm running on two processors has the same arithmetic complexity as the sequential algorithm.

4 A Balanced Submatrix Merging Algorithm

In order to balance the work load and minimize inter-processor communication, we propose to implement the alternative transformation in Fig. 12, which can be viewed as consisting of the annihilation process in Fig. 4 and the permutation process in Fig. 5.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \otimes & \times & \times & \times & \times & \times & \times \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 12: An alternative elimination sequence ($\otimes$: the elements to be annihilated).

Assuming as before that each submatrix resides in a different processor, we employ the following two strategies to balance the work load and minimize inter-processor communication.

1. To balance the load, the tasks corresponding to the $k(k+1)/2$ nonzeros to be annihilated are evenly divided among the two processors.

2. To help reduce inter-processor communication, a "$\otimes$" may be zeroed by a processor other than the one it is originally stored in.

The parallel algorithm we propose can be best explained when applying to the transformation in Fig. 12. Let us denote the processor storing the top submatrix as P_A and the processor storing the bottom submatrix as P_B. We first specify the particular ordering these tasks are to be performed in the left diagram in Fig. 13, where the tasks to be performed by processors P_A and P_B are each labelled by its scheduled time step. We have distinguished P_A's tasks from P_B's by labelling P_A's i^{th} task by i and P_B's by $[i]$. The two tasks scheduled for the same step can potentially be performed concurrently by two processors provided the communication can be masked by computation. This point will be clear after we explain the inter-processor communication scheme. We next consult the diagram to the right in Fig. 13, which identifies the tasks requiring data from both submatrices.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & 2 & 3 & 4 & 5 & 6 & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & 8 & 9 & 10 & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & 12 & \times & \times & \times & \times & \times & \times \\
1 & [3] & [4] & [5] & [6] & [7] & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & 7 & [9] & [10] & [11] & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & 11 & [13] & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & 2 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & 8 & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & 12 & \times & \times & \times & \times & \times & \times \\
1 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & 7 & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & 11 & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & & & & & & \\
 & & & & & & & & & & &
\end{pmatrix}
$$

Figure 13: Concurrent scheduling of P_A's and P_B's tasks.

We now explain the inter-processor communication scheme using the example in Fig. 13. Our algorithm requires processor P_B to send the top row of the bottom submatrix to P_A so that P_A can complete tasks 1 and 2 as shown in Fig. 14, where the elements are labelled by "A" or "B" depending on in which processor they are originally stored.

$$
\begin{pmatrix}
A & A & A & A & A & A & A & A & A & A & A & A \\
 & A & A & A & A & A & A & A & A & A & A & A \\
B & B & B & B & B & B & B & B & B & B & B & B
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & A & A & A & A & A & A & A & A & A & A & A \\
0 & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B}
\end{pmatrix}
$$

$$
\longrightarrow
\begin{pmatrix}
\tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & 0 & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
0 & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}}
\end{pmatrix}
$$

Figure 14: Processor P_A performs tasks 1 and 2.

The row from P_B is thus appropriately updated by P_A and is sent back to P_B immediately after task 2 is completed. P_A can then proceed to complete tasks 3, 4, 5 and 6 without inter-processor communication as shown in Fig. 15. P_B will do the same with respect to its

$$
\begin{pmatrix}
\tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & 0 & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & & A & A & A & A & A & A & A & A & A & A \\
 & & & A & A & A & A & A & A & A & A & A \\
 & & & & A & A & A & A & A & A & A & A \\
 & & & & & A & A & A & A & A & A & A
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & 0 & 0 & 0 & 0 & 0 & \hat{A} & \hat{A} & \hat{A} & \hat{A} & \hat{A} & \hat{A} \\
 & & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & & & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & & & & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & & & & & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A}
\end{pmatrix}
$$

Figure 15: Processor P_A performs tasks 3, 4, 5 and 6.

tasks [3], [4], $\cdots$, [7] after receiving back the modified top row as shown in Fig. 16.

$$
\begin{pmatrix}
0 & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} \\
 & B & B & B & B & B & B & B & B & B & B & B \\
 & & B & B & B & B & B & B & B & B & B & B \\
 & & & B & B & B & B & B & B & B & B & B \\
 & & & & B & B & B & B & B & B & B & B \\
 & & & & & B & B & B & B & B & B & B
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & \hat{B} & \hat{B} & \hat{B} & \hat{B} & \hat{B} & \hat{B} \\
 & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} \\
 & & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} \\
 & & & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} \\
 & & & & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} \\
 & & & & & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B}
\end{pmatrix}
$$

Figure 16: Processor P_B performs tasks [3], [4], $\cdots$, [7].

We next explain how to mask communication by computation. The strategy is for each processor to send out the row needed by the other processor as early as possible. For example, processor P_B should send its row 3 immediately after it completes task [4] so that it would have arrived in P_A when P_A completes task 6, and P_A should send the updated row back to P_B as soon as it completes tasks 7 and 8, and so on. In the next section we shall introduce a task precedence graph which is instrumental in our analysis of the performance

of the proposed algorithm and allows us to conveniently formalize the notion of masking communication by computation.

5 Performance Analysis of the Proposed Algorithm

In order to analyze the performance of this algorithm, we make use of a task precedence graph, where each vertex identified by a step number represents the task of annihilating the nonzero in that position of one submatrix. As an example, we display in Fig. 17 the task precedence graph set up according to the scheduling of P_A's and P_B's tasks in Fig. 13. The precedence relationship identified by $\rightarrow$ is established considering both data and processor availability subject to the condition that communication can be completely masked by computation. We show next how the latter condition is indeed satisfied by the particular strategy we employ for masking communication by computation.

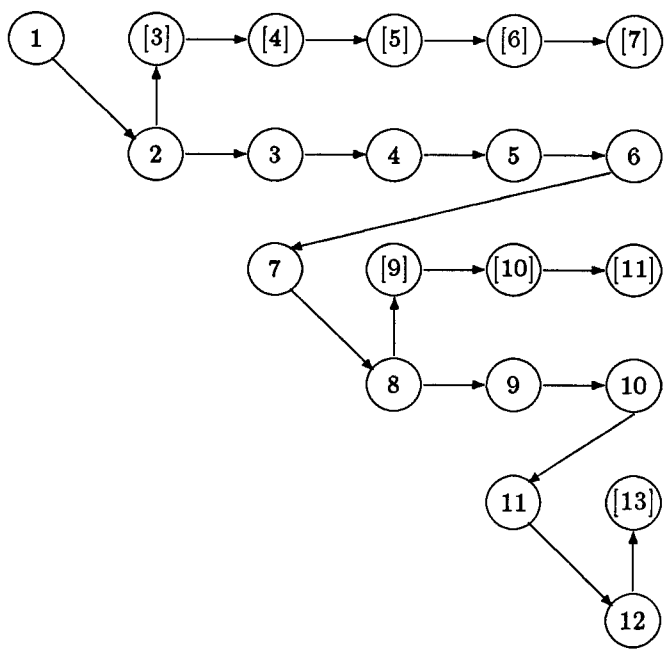


Figure 17: Task precedence graph of the example in Fig. 13.

In Fig. 18 we identify the data communication path by double arrows. The following observations are helpful in studying this graph.

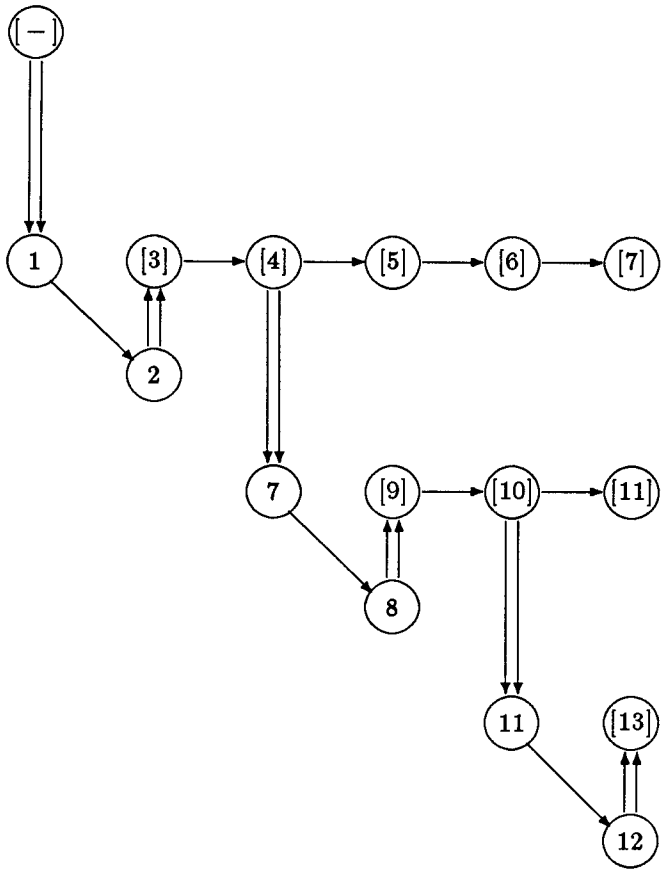


Figure 18: Task communication path graph of the example in Fig. 13.

1. We introduce a dummy task node ([−]) to account for the initial data communication from processor P_B to P_A.

2. The two tasks connected by double arrows are each executed by a different processor.

3. Data communication follows the arrow direction, namely that the processor executing the task at the tail sends the data to the processor executing the task at the head.

4. The placement of double arrows traces the actual data flow of our communication algorithm. For example, processor P_A sends the modified top row of the bottom submatrix back to P_B immediately after task 2 is completed and P_B needs this row to perform task [3]. The data transfer from P_A to P_B is faithfully reflected by the double arrow pointing from vertex (2) to vertex ([3]) in the communication path graph in Fig. 18.

In order to show that communication can be masked by computation in our algorithm, we need to adapt our analytical model to account for the time actually taken for communication. To motivate our proof, let us allocate one time step for communication and obtain the modified precedence graph as well as communication path graph for the example above in Fig. 19 and 20. We now contrast the critical paths embedded in the two precedence graphs in Fig. 21, which are established by assuming that the tasks scheduled for step i and $[i]$ finish at the same time. Consequently, step number i occurs only once in each critical path identified in Fig. 21 and the arrows connecting vertex i to vertex j, $i < j$, have been omitted. Note that the critical path in the left is identified from Fig. 17 assuming that communication takes no time at all, whereas the critical path in the right is identified from Fig. 19 assuming that communication takes one time step. To be technically precise, the latter assumption implies that sending one row of size $(n - q + 1)$ to another directly connected processor takes no more time than $4(n - q + 1)$ multiplicative operations. Another technical point is that the number of operations involved in step $[i]$ are not exactly equal to that of step i. The difference is $4(n-q+1)$ $(1 \leq q \leq k)$ for step $[i]$ versus either $4(n-q)$ for step i in Fig. 17 and 18 or $4(n-q-1)$ for step i in Fig. 19 and 20. We see that in either case the difference amounts to less than *eight* multiplicative operations, which is negligible when $n \gg k$. We shall thus assume that time step $[i]$ is of the same length as time step i throughout our analysis.

We now make the following important observation from Fig. 21, namely that the *delay* caused by communication does not affect the critical path until the very last three steps and the *total delay* amounts to three time steps exactly. An immediate question, of course, is whether this result holds for any given pair of $k \times n$ upper trapezoidal submatrices. It turns out that when k is even the delay amounts to three time steps and when k is odd the delay becomes four time steps. Our proof makes use of a generalized precedence graph given in Fig. 22 for k being an even number and the one given in Fig. 23 for k being an

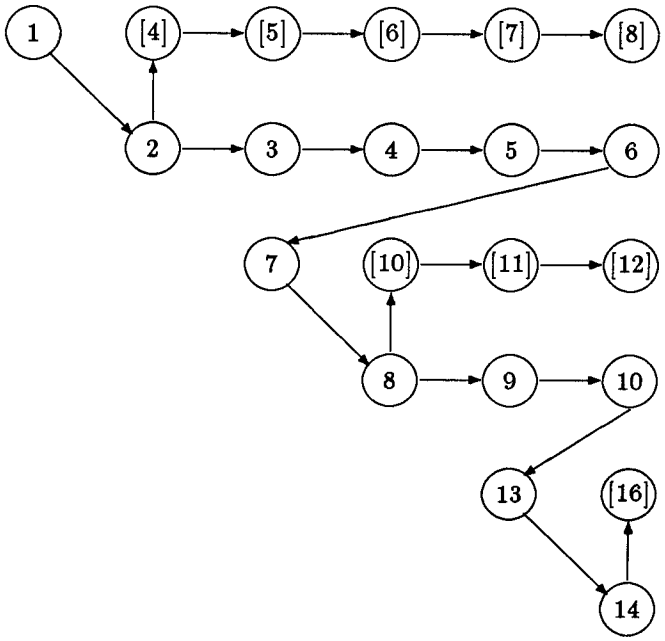


Figure 19: Task precedence graph (modified) of the example in Fig. 13.

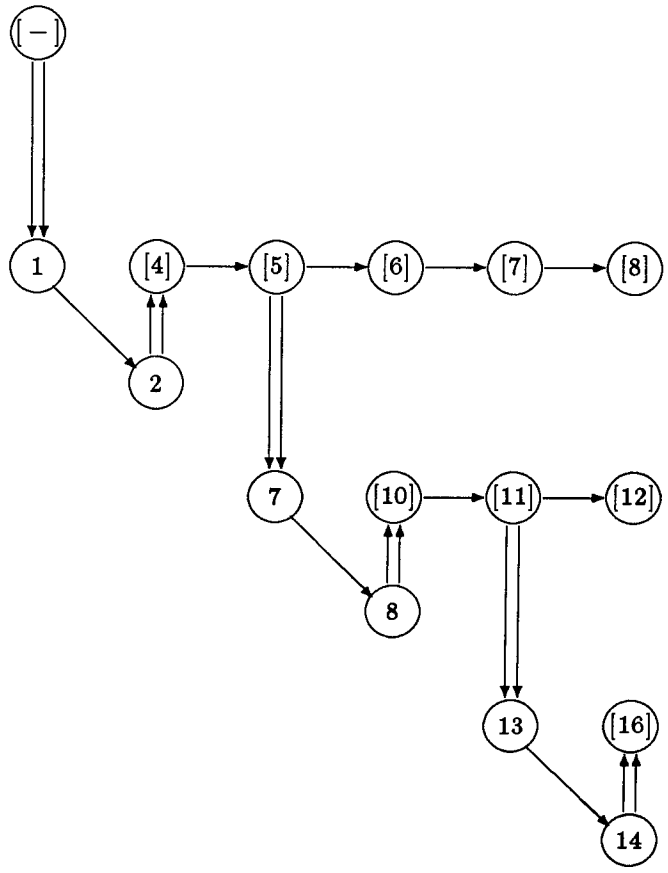


Figure 20: Task communication path graph (modified) of the example in Fig. 13.

1					
	2	3	4	5	6
		7			
			8	9	10
				11	[13]
					12

1					
	2	3	4	5	6
		7			
			8	9	10
				13	[16]
					14

Figure 21: The critical paths identified from the precedence graphs in Fig. 17 and 19.

odd number. We offer some explanation of the setup of the generalized precedence graph before we proceed.

Recall that our algorithm would zero out odd-numbered rows from the bottom triangular matrix and even-numbered rows from the top one. It is convenient to arrange the vertices of our task precedence graph as an upper triangle to reflect the locations of the elements to be eliminated. Since exactly one nonzero is annihilated by performing one task, the mapping from the task nodes to the nonzeros is *one-to-one* and *onto*. We label each task node by an integer i or $[i]$ depending on whether the task is performed by processor P_A or P_B. The vertices in Fig. 22 and 23 should be viewed as connected by arrows in the same manner as those of the task precedence graph in Fig. 19, although the arrows are not explicitly shown here due to lack of space. From our description of the algorithm, P_A would zero out the even-numbered rows from the top submatrix as well as the leading nonzeros of the odd-numbered rows from the bottom submatrix, whereas P_B would zero out the odd-numbered rows from the bottom submatrix except for their leading elements. We summarize the implication of such basic understanding below.

$$
\begin{array}{cccccccccccc}
1 & [4] & [5] & [6] & [7] & [8] & \cdots & \cdots & \cdots & [k_1-2] & [k_1-1] & [k_1] \\
 & 2 & 3 & 4 & 5 & 6 & \cdots & \cdots & \cdots & k_1-4 & k_1-3 & k_1-2 \\
 & & k_1-1 & [k_1+2] & [k_1+3] & [k_1+4] & \cdots & \cdots & \cdots & [k_3-2] & [k_3-1] & [k_3] \\
 & & & k_1 & k_1+1 & k_1+2 & \cdots & \cdots & \cdots & k_3-4 & k_3-3 & k_3-2 \\
 & & & & k_3-1 & [k_3+2] & \cdots & \cdots & \cdots & [k_5-2] & [k_5-1] & [k_5] \\
 & & & & & k_3 & \cdots & \cdots & \cdots & k_5-4 & k_5-3 & k_5-2 \\
 & & & & & & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
 & & & & & & & \cdots & \cdots & \cdots & \cdots & \cdots \\
 & & & & & & & & k_m-1 & [k_m+2] & [k_{m+2}-1] & [k_{m+2}] \\
 & & & & & & & & & k_m & k_{m+2}-3 & k_{m+2}-2 \\
 & & & & & & & & & & k_{m+2}+1 & [k_{m+2}+4] \\
 & & & & & & & & & & & k_{m+2}+2
\end{array}
$$

Figure 22: A generalized precedence graph (k is an even number).

1	[4]	[5]	[6]	[7]	[8]	$\cdots$	$\cdots$	$\cdots$	$[k_1 - 2]$	$[k_1 - 1]$	$[k_1]$
	2	3	4	5	6	$\cdots$	$\cdots$	$\cdots$	$k_1 - 4$	$k_1 - 3$	$k_1 - 2$
		$k_1 - 1$	$[k_1 + 2]$	$[k_1 + 3]$	$[k_1 + 4]$	$\cdots$	$\cdots$	$\cdots$	$[k_3 - 2]$	$[k_3 - 1]$	$[k_3]$
			k_1	$k_1 + 1$	$k_1 + 2$	$\cdots$	$\cdots$	$\cdots$	$k_3 - 4$	$k_3 - 3$	$k_3 - 2$
				$k_3 - 1$	$[k_3 + 2]$	$\cdots$	$\cdots$	$\cdots$	$[k_5 - 2]$	$[k_5 - 1]$	$[k_5]$
					k_3	$\cdots$	$\cdots$	$\cdots$	$k_5 - 4$	$k_5 - 3$	$k_5 - 2$
						$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
							$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
							$k_m - 1$	$[k_m + 2]$	$[k_{m+2} - 2]$	$[k_{m+2} - 1]$	$[k_{m+2}]$
								k_m	$k_m + 1$	$k_{m+2} - 3$	$k_{m+2} - 2$
									k_{m+2}	$[k_{m+2} + 3]$	$[k_{m+2} + 4]$
										$k_{m+2} + 1$	$k_{m+2} + 2$
											$k_{m+2} + 6$

Figure 23: A generalized precedence graph (k is an odd number).

1. The tasks along each even-numbered row are scheduled consecutively, as are the tasks (except for the leading one) along each odd-numbered row. This is evident from our examples in Fig. 17 and 19 as well as the two generalized precedence graphs in Fig. 22 and 23.

2. We note also that since the leading task of each odd-numbered row is performed by the same processor performing the tasks along the following even-numbered row, the sequencing of these tasks is also consecutive.

3. From the task precedence graph given in Fig. 17, it is clear that the data dependency prevents P_B from processing the second leading task of the odd-numbered row earlier than the completion of the leading task in the following even-numbered row. We show in Fig. 19 that the actual time taken for data transmission causes P_B to be two time steps behind P_A on completing the last task of each row. This fact is faithfully reflected on our generalized precedence graphs in Fig. 22 and 23, where the last task of each odd-numbered $(2i + 1)^{th}$ row is labelled "$[k_{2i+1}]$", and the last task of the following even-numbered row is labelled "$k_{2i+1} - 2$". It follows that subject to the condition that "$k - (2i + 1) - 1 \geq 4$" the leading tasks of the following two rows would

be numbered $k_{2i+1} - 1$ and k_{2i+1}, from the latter we further infer that the second leading task of the next odd-numbered row would be labelled $[k_{2i+1} + 2]$.

4. Note that the condition "$k - (2i + 1) - 1 \geq 4$" is violated when the row number $m = 2i + 1 = k - 3$ if k is even or $m = 2i + 1 = k - 4$ if k is odd.

5. It should now be clear that the critical path of the algorithm can be established by tracing Processor P_A's execution sequence and taking into account the delay caused by the violation of the condition "$k - (2i + 1) - 1 \geq 4$". As examples, we identify the two critical paths corresponding to the two cases for $k = 12$ and $k = 13$ in Fig. 24 and 25.

1											
	2	3	4	5	6	7	8	9	10	11	12
		13									
			14	15	16	17	18	19	20	21	22
				23							
					24	25	26	27	28	29	30
						31					
							32	33	34	35	36
								37			
									38	39	40
										43	[46]
											44

Figure 24: The critical path for $k = 12$.

1												
	2	3	4	5	6	7	8	9	10	11	12	13
		14										
			15	16	17	18	19	20	21	22	23	24
				25								
					26	27	28	29	30	31	32	33
						34						
							35	36	37	38	39	40
								41				
									42	43	44	45
										47		
											48	49
												53

Figure 25: The critical path for $k = 13$.

We complete our analysis by proposing the following theorems.

Theorem 1 *Suppose we employ two directly connected local-memory processors to merge two $k \times n$ upper trapezoidal submatrices. Using the balanced submatrix merging algorithm we proposed in section 4 of this article, the length (in terms of time steps) of the critical path is*

$$\frac{1}{2}\left(\frac{k(k+1)}{2} + \frac{k}{2} + 8\right)$$

for k even, and

$$\frac{1}{2}\left(\frac{k(k+1)}{2} + \frac{k-1}{2} + 9\right)$$

for k odd.

Proof: We first consider the case when k is an even number. Referring to the generalized precedence graph given in Fig. 22, we recall our observation that the condition "$k - (2i+1) - 1 \geq 4$" is violated when the odd row number $m + 2 = 2i + 1 = k - 3$, which is the fourth row from the bottom. Using our notation the last task of the $(m+2)^{th}$ row is labelled by time step $[k_{m+2}]$, it follows that the third task on this row must be completed in time step $[k_{m+2} - 1]$. It is now straightforward to verify that the last three time steps are $k_{m+2} + 1$, $k_{m+2} + 2$, followed by $[k_{m+2} + 4]$. Noting that these three steps would be scheduled as $k_{m+2} - 1$, k_{m+2} and $[k_{m+2} + 1]$ in the ideal situation when the communication time is ignored entirely, henceforth the total delay is exactly three time steps.

To compute the length of the critical path we count processor P_A's tasks and take into account the delay of three time steps. By simple algebra we immediately obtain the following formula.

$$1+\frac{1}{2}\left(\frac{k(k+1)}{2}+\frac{k}{2}\right)+3=\frac{1}{2}\left(\frac{k(k+1)}{2}+\frac{k}{2}+8\right) . \tag{1}$$

The proof for the second case is similar except for noting that the condition "$k-(2i+1)-1 \geq 4$" is violated when $m+2=2i+1=k-4$, which is the fifth row from the bottom. The effect is that the last four steps are labelled k_{m+2}, $k_{m+2}+1$, $k_{m+2}+2$ and lastly $k_{m+2}+6$ instead of $k_{m+2}-1$, k_{m+2}, $k_{m+2}+1$ and $k_{m+2}+2$. The total delay is exactly four time steps in this case. The total number of time steps is thus given by

$$\frac{1}{2}\left(\frac{k(k+1)}{2}+\frac{k-1}{2}+1\right)+4=\frac{1}{2}\left(\frac{k(k+1)}{2}+\frac{k-1}{2}+9\right) . \tag{2}$$

□

Theorem 2 *Using the balanced submatrix merging algorithm we proposed in section 4 of this article for merging two $k \times n$ upper trapezoidal submatrices, the total data exchanged between the two directly connected processors are $\lceil k/2 \rceil$ rows in the annihilation process and $\lfloor k/2 \rfloor$ rows in the permutation process.*

Proof: The proof can be immediately obtained by noting that (i) only the odd-numbered rows from the bottom submatrix must be sent back and forth (one exchange) between the two processors during the annihilation process, and (ii) only the even-numbered rows from the bottom submatrix must be exchanged with the even-numbered rows from the top submatrix during the permutation process. □

6 Further Parallelization on Two Clusters of Processors

We described in the last section a new parallel algorithm for merging two upper trapezoidal submatrices on two directly connected processors. We also show that the work load is evenly divided between the two processors and that communication is well masked by computation assuming that the time for transmitting n floating-point numbers is no longer than the time taken for $4n$ multiplicative operations. In this section we shall analyze the performance of the proposed algorithm when two clusters of processors are employed and compare our speed-up result with that of the algorithm proposed in [1].

Our analysis models the same multiprocessor environment considered in [1], where the target machine consists of clusters of processors. Each cluster is a shared-memory system and the clusters are in turn interconnected via a single, system-wide shared memory. Since the submatrices to be merged are each located in the local shared-memory of a cluster, we

find it convenient to model the inter-cluster communication using the notion of message-passing, which can be easily implemented by writing to and reading from the global shared-memory.

Let us assume that each cluster has p processors. We thus have $2p$ processors at our disposal by employing two clusters. Since the algorithm proposed in [1] exploits parallelism within only one of the two clusters by overlapping the annihilation of the top row of the bottom submatrix (which requires intercluster communication) with the annihilation of the rest of the elements of the same submatrix (which requires only local communication), clearly the maximum possible speed-up is p, resulting in an efficiently of $p/2p \leq 50\%$.

While the algorithm we proposed in section 4 can be easily adapted for implementation on two p-processor clusters, its performance needs to be carefully re-examined. We first establish the shortest critical path (in terms of time steps) of the proposed algorithm in Lemma 3. For simplicity in presentation we shall only consider the case when k is an even number throughout the rest of the manuscript. Besides, as far as performance is concerned, it is adequate to analyze one case of k and similar performance is expected for the case of $k+1$ for any algorithm with medium data granularity.

Lemma 3 *We adapt the algorithm proposed in section 4 to merge two $k \times n$ upper trapezoidal submatrices when P_A and P_B are each a cluster of processors. We assume that the $k(k+1)/2$ tasks are assigned to P_A and P_B exactly as before. We further assume that within each cluster the tasks along the same row are performed sequentially by the same processor, and that there are enough free processors to start processing each row of tasks as early as permitted by the availability of data. Subject to the assumptions above, the shortest critical path of the parallel algorithm consists of $3k-2$ time steps.*

Proof: In Fig. 26 we present the task precedence graph for the case $k = 12$ taking into account the actual communication time but assuming a free processor exists when data is available. Recall that the leading tasks of the odd-numbered rows are performed by P_A and observe that data dependency and the time taken for communication dictates that there is a gap of six time steps between the leading tasks of two consecutive odd-numbered rows. As examples, the $k/2$ such tasks are numbered 1, 7, 13, 19, 25 and 31 in Fig. 26. It is a straightforward exercise to generalize the numbering sequence for any given value of k and obtain the critical path of

$$1 + 6\left(\frac{k}{2} - 1\right) + 3 = 3k - 2\,. \tag{3}$$

□

Substituting k in equation (3) by the value of 12, we obtain a shortest critical path of 34 time steps for our example as verified in Fig. 26.

Since the serial algorithm takes $k(k+1)/2$ time steps and the shortest critical path takes $3k-2$ time steps, the speed-up is $k(k+1)/(6k-4) > k/6$. Note that when $k \ll n$,

1	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
	2	3	4	5	6	7	8	9	10	11	12
		7	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]
			8	9	10	11	12	13	14	15	16
				13	[16]	[17]	[18]	[19]	[20]	[21]	[22]
					14	15	16	17	18	19	20
						19	[22]	[23]	[24]	[25]	[26]
							20	21	22	23	24
								25	[28]	[29]	[30]
									26	27	28
										31	[34]
											32

Figure 26: Concurrent scheduling of cluster P_A's and cluster P_B's tasks assuming that free processors exist when data is available.

the difference between the time taken for $4n$ multiplicative operations and the time taken for $4n - i$ multiplicative operations, $1 \leq i \leq k - 1$, is negligible. Therefore, it is sensible to measure the performance of the algorithm by the number of time steps. To obtain the efficiency, we need to know how many processors are needed to achieve the shortest critical path. We obtain the result in Lemma 4 by analyzing the task precedence graph in Fig. 26.

Lemma 4 *In order to complete the merge of two $k \times n$ upper trapezoidal matrices in $3k-2$ time steps when employing two clusters of processors, each cluster must have $\lceil k/6 \rceil$ processors.*

Proof: Suppose each cluster has p processors. The p processors of cluster P_A will each start a consecutive sequence of tasks at time step 1, 7, $\cdots$, and $1 + 6(p - 1)$. Clearly the processor which processes tasks 1, 2, 3, $\cdots$, and k will finish first and is able to start the next sequence of tasks, of which the leading task may start earliest at time step $1 + 6p$. Since no processor is free until time step $k+1$, we must have $1+6p \geq k+1$, i.e., $p = \lceil k/6 \rceil$, to achieve the shortest critical path.

We next show that when this condition is satisfied, the remaining $p - 1$ processors in cluster P_A can all begin the following sequences of tasks at the earliest possible scheduled time by simply observing the following.

1. The p processors in cluster P_A become free one by one after k, $k+4$, $\cdots$, and $k+4(p-1)$ time steps respectively.

2. The next p sequences of tasks are scheduled for time steps $1+6p$, $1+6(p+1)$, $1+6(p+2)$, $\cdots$, and $1+6(2p-1)$.

3. The inequality $1+6p \geq k+1$ implies $1+6p+6i \geq k+4i+1$ for $1 \leq i \leq p-1$.

Finally, we need to establish that under the same condition all processors in cluster P_B will also be available to process their assigned tasks at the earliest possible time step. To show that this is indeed the case we simply note that the last task of each odd-numbered row is finished two time steps behind the last task of the following even-numbered row, whereas the first task scheduled for processors in P_B in the following odd-numbered row begins three time steps later than the leading task. Since the same condition $1+6p \geq k+1$ implies $(1+6p)+3 > (k+2)+1$, we have proved that all tasks assigned to cluster P_B can all proceed as scheduled to achieve the shortest critical path. □

From Lemma 3 and 4, a speed-up of $k/6$ is obtained while employing a total of $k/3$ processors, resulting in an efficiency of 50%. However, in contrast to the upper bound of 50% efficiency for the algorithm in [1], we shall show that the 50% efficiency is a "lower bound" of our algorithm when $p \leq \lceil k/6 \rceil$ processors are employed in each cluster. Before we proceed, we first use an example to explain how the proposed algorithm works when $p < \lceil k/6 \rceil$ are available in each cluster. Let us refer to Fig. 27, where we present the task precedence graph for $k = 16$ when each cluster has two processors. We first note that since $p = 2$ and $\lceil k/6 \rceil = 3$, the condition $p = \lceil k/6 \rceil$ in Lemma 4 is violated and the shortest critical path cannot be achieved. We therefore must establish the critical path for an arbitrary choice of $p < \lceil k/6 \rceil$ taking into account not only the availability of data but also the existence of free processors. The first four rows of task in the precedence graph are scheduled taking into account only the data availability because all four processors (two in each cluster) are free initially. However, when we reach the fifth row of the graph, the leading task can no longer be scheduled for time step 13 because no processor in cluster P_A is free to take on the task until time step 17. When we reach the thirteenth row, we encounter a case when there exists a free processor to start the leading task at time step 37 but the data are not available for the processor to take on this task until time step 41. For this example, the critical path consists of 50 time steps.

Theorem 5 *We consider merging two $k \times n$ upper trapezoidal submatrices on two clusters of processors by adapting the algorithm proposed in section 4 as described above. If each cluster has p processors, then the merging process can be completed in*

$$\frac{k^2}{4p} + \frac{k}{2} + 6p - 2 \tag{4}$$

time steps, resulting in an asymptotic speed-up of $2p$ and 100% efficiency.

1	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		7	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]
			8	9	10	11	12	13	14	15	16	17	18	19	20
				17	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]
					18	19	20	21	22	23	24	25	26	27	28
						23	[26]	[27]	[28]	[29]	[30]	[31]	[32]	[33]	[34]
							24	25	26	27	28	29	30	31	32
								29	[32]	[33]	[34]	[35]	[36]	[37]	[38]
									30	31	32	33	34	35	36
										35	[38]	[39]	[40]	[41]	[42]
											36	37	38	39	40
												41	[44]	[45]	[46]
													42	43	44
														47	[50]
															48

Figure 27: Concurrent scheduling of tasks for two clusters with each having 2 processors.

Proof: To establish the critical path length as given in equation (4), recall again that we have used i to label the time steps of the tasks assigned to processors in P_A and $[i]$ to label the time steps of those assigned to processors in P_B. We assume that all of the $[i]$ tasks along the odd-numbered row of the precedence graph are processed sequentially by one processor in P_B, whereas the leading i task of an odd-numbered row and all of the i tasks of the following even-numbered row are processed sequentially by one processor in P_A.

Our algorithm requires that each sequence of tasks are performed by the earliest available processor. Since the data dependency dictates that the processor assigned tasks along row $2i+1$ or $2i+2$ finishes before the processor assigned tasks along row $2i+3$ or $2i+4$ respectively, the mapping strategy is, in fact, equivalent to wrap-mapping the rows of tasks around the p processors within each cluster. To be technically precise, we group the leading task of each odd-numbered row with the tasks along the following even-numbered row. In other words, for any given values of k and p, assuming that k is an integral multiple of $2p$, our mapping strategy assigns $k/2p$ rows of tasks to each processor, while each block of consecutive $2p$ rows of tasks are performed by $2p$ different processors.

We now trace the execution path of one particular processor in cluster P_A. We naturally choose the processor which begins the merging process by performing task 1. Let us denote

this processor by $P_A^{(1)}$. Given below are the key observations leading to the proof of this theorem.

The first task $P_A^{(1)}$ performs in each of the assigned $k/2p$ rows is the leading task of the top odd-numbered row of each block of consecutive $2p$ rows. Referring to our example in Fig. 27, $P_A^{(1)}$ begins each sequence of tasks by taking on tasks 1, 17, 29 and 41, which are the leading tasks of rows 1, 5, 9 and 13, or 1, $1+2p$, $1+4p$ and $1+6p$ for $p=2$. Consequently, if we can schedule these tasks taking into account both data dependency and processor availability, then we can derive the length of the critical path of the algorithm. In order to do so, observe that if $P_A^{(1)}$ has completed its sequence of tasks in the jth block in step k_m-2, where $m=2p(j-1)+1$ using our notation in the generalized precedence graph, then the time step $P_A^{(1)}$ can begin with its tasks in the $(j+1)^{th}$ block is the maximum of "k_m-1" and "$k_m-1-2p(k/2p-j+1)+6p$". There is delay only when the choice must be the latter. We next find out in which block this would occur. That is, we would like to find out the value of "$j+1$" such that

$$k_m-1-2p(k/2p-j+1)+6p > k_m-1\ . \tag{5}$$

Simplifying inequality (5), we obtain

$$j > \frac{k}{2p}-2\ . \tag{6}$$

The minimum value of j satisfying the inequality (6) is $k/2p-1$. We therefore have proved that the delay will not occur until the $(j+1=k/2p)^{th}$ block, which is the very last block! The step number $k_m-1-2p(k/2p-j+1)+6p$ of the leading task in the $(j+1=k/2p)^{th}$ block can be simplified to be k_m+2p-1, with k_m computed from equation (7).

$$\begin{aligned} k_m &= 2+\sum_{j=0}^{\frac{k}{2p}-2}(k-2pj) \\ &= \frac{k^2}{4p}+\frac{k}{2}-2p+2\ . \end{aligned} \tag{7}$$

It is now straightforward to compute the critical path length by

$$\begin{aligned} k_m+2p-1+6(p-1)+3 &= k_m+8p-4 \\ &= \frac{k^2}{4p}+\frac{k}{2}+6p-2\ . \end{aligned} \tag{8}$$

As an example, we substitute $k=16$ and $p=2$ in the formula above and obtain a critical path of 50 time steps as verified in Fig. 27.

Since the serial algorithm requires $k(k+1)/2$ time steps, with the parallel algorithm completed in $k^2/4p+O(k)$ time steps we obtain an asymptotic speed-up of $2p$ and efficiency of 100%. □

7 Conclusions

The problem of merging two $k \times n$ ($k \leq n$) upper trapezoidal submatrices on multiprocessor machines is considered in this paper. The parallel algorithms we present are designed for implementation on either a pair of directly connected local-memory processors or two clusters of tightly-coupled processors. Our analysis of the proposed algorithms shows that in both environments the work load is evenly distributed, communication can be well masked by computation, and the optimal speed-up may be achieved. In the orthogonal factorization phase described in [1] and [2] for solving large scale dense or sparse least squares problems, multiple pairs of processors or clusters may repetitively apply the proposed algorithm to merge multiple pairs of submatrices concurrently throughout the computation. While the proposed algorithm has improved an important step in solving the least squares problem on clusters of processors, the distribution of data remain suitable for employing the parallel schemes available in [1] for back substitution and the calculation of certain elements of the covariance matrix. Furthermore, the unit operation of the proposed algorithm involves applying a Givens rotation to two rows of data, which is not different from the unit operation defined in [1] and is thus also suitable for exploiting the vector capacity of each processor within the cluster.

Acknowledgement

The authors thank Professor Joseph Liu for bringing reference [1] to their attention.

References

[1] G. H. Golub, R. J. Plemmons, and A. Sameh. *Parallel block schemes for large scale least squares computations.* Technical Report CSRD Rpt. No. 574, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL 61801-2932, April 1986.

[2] A. Pothen, J. Somesh, and U. Vemulapati. Orthogonal factorization on a distributed memory multiprocessor. In M. T. Heath, editor, *Proc. Hypercube Multiprocessors 1987*, pages 587–596, SIAM, Philadelphia, PA, 1987.

To Eleanor

From Sue DeAngelis

Date Sept. 27

memo

University of Waterloo

I am out of stock of CS-88-45 "A Balanced Submatrix..." and CS-88-46 "Updating and Downdating...". Please advise ASAP if you would like more copies made.

Thanks.

Sue

Sue, please have another 50 made of each report. Thanks. Eleanor

Printing Requisition / Graphic Services

26245

1. **Please complete unshaded areas** on form as applicable.
2. **Distribute copies** as follows: White and Yellow to Graphic Services. **Retain Pink Copies for your records.**
3. On completion of order the **Yellow copy will be returned** with the printed material.
4. **Please direct enquiries, quoting requisition number and account number,** to **extension 3451.**

TITLE OR DESCRIPTION

A Balanced Submatrix Merging Algorithm for Multiprocessor Architectures CS-88-45

DATE REQUISITIONED: **Dec. 5/88**

DATE REQUIRED: **ASAP**

ACCOUNT NO.: 4 1 2 4 1 0 0 6 0

REQUISITIONER – PRINT: **J.A. George**

PHONE:

SIGNING AUTHORITY: J. Allen Brown

MAILING INFO – NAME: **Sue DeAngelis** DEPT.: **C.S.** BLDG. & ROOM NO.: **DC 2314** [X] DELIVER [ ] PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES: 30 NUMBER OF COPIES: 50

TYPE OF PAPER STOCK: [X] BOND [ ] NCR ____ PT. [X] COVER [ ] BRISTOL [X] SUPPLIED [ ] ____

PAPER SIZE: [X] 8½ x 11 [ ] 8½ x 14 [ ] 11 x 17 [ ] ____

PAPER COLOUR: [X] WHITE [ ] ____ INK: [X] BLACK [ ] ____

PRINTING: [ ] 1 SIDE ____ PGS. [X] 2 SIDES ____ PGS. NUMBERING: FROM TO

BINDING/FINISHING: [X] COLLATING [X] STAPLING [ ] ____ HOLE PUNCHED [ ] PLASTIC RING

3 down left side

FOLDING/PADDING: CUTTING SIZE:

Special Instructions

Math fronts and backs enclosed.

COPY CENTRE OPER. NO. BLDG. MACH. NO.

DESIGN & PASTE-UP OPER. NO. TIME LABOUR CODE

D01
D01
D01

TYPESETTING QUANTITY

PAP000000 T01
PAP000000 T01
PAP000000 T01

PROOF

PRF
PRF
PRF

NEGATIVES QUANTITY OPER. NO. TIME LABOUR CODE

FLM C01
FLM C01
FLM C01
FLM C01
FLM C01

PMT

PMT C01
PMT C01
PMT C01

PLATES

PLT P01
PLT P01
PLT P01

STOCK

001
001
001
001

BINDERY

RNG B01
RNG B01
RNG B01
MIS000000 B01

OUTSIDE SERVICES

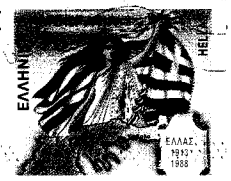
$ ____ COST

TAXES – PROVINCIAL [ ] FEDERAL [ ]

GRAPHIC SERV. OCT. 85 482-2

N. MISSIRLIS
University of Athens
Unit of Applied Mathematics
Panepistemiopolis Athens (157 10) - Greece

Prof. I



Research Report Secretary,
Department of Computer Science
University of Waterloo,
Waterloo, Ontario
N2L 3G1
USA.

Dear Sir/Madam,

I should be very obliged to you for sending me a reprint of your paper:

1) CS-88-45 – A balanced automatrix....
2) CS-88-46 – Updating and downdating...

Thanking you in advance

Sent March 20/89

Yours Sincerely

A Balanced Submatrix Merging Algorithm for Multiprocessor Architectures

Eleanor Chu
Alan George

Department of Computer Science

Research Report CS-88-45
November 1988

Faculty of Mathematics

University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

A Balanced Submatrix Merging Algorithm for Multiprocessor Architectures

Eleanor Chu
Alan George

Department of Computer Science

Research Report CS-88-45
November 1988

A Balanced Submatrix Merging Algorithm for Multiprocessor Architectures *

Eleanor Chu
Alan George

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Research Report CS-88-45

November 1988

Abstract

In this article we describe a parallel algorithm which applies Givens rotations to selectively annihilate $k(k+1)/2$ nonzero elements from two $k \times n$ ($k \leq n$) upper trapezoidal submatrices. The new algorithm we propose is suitable for implementation on *either* a pair of directly connected local-memory processors *or* two clusters of tightly-coupled processors. We show in both cases that the proposed algorithms may achieve optimal speed-up by balancing the work load distribution and masking inter-processor or inter-cluster communication by computation. In the context of solving large scale least squares problems, this submatrix merging step is repetitively needed during the entire computation, and, furthermore, there are usually many pairs of such submatrices to be merged with each submatrix stored in the memory of a processor or a cluster of processors. The proposed algorithm can be applied to each pair of submatrices concurrently and thus parallelizes an important step in solving the least squares problems.

*Research supported in part by NASA Grant No. NAG-1-803, and by the University of Waterloo

Contents

List of Figures

1 Introduction

In this article we study some effective ways to merge submatrices on multiprocessor architectures, and propose a cure to the unbalanced load distribution problem which the algorithms currently known to us have experienced. The particular submatrix merging operation we consider can be understood as eliminating $k(k+1)/2$ nonzeros by Givens rotations from a pair of $k \times n$ $(k \leq n)$ upper trapezoidal submatrices as depicted in Fig. 1, where $k = 6$, $n = 12$.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

Figure 1: Merging two upper trapezoidal submatrices.

The need to reduce multiple pairs of such submatrices arises in both dense and sparse matrix computations. An example of the former case can be found in the recursive fine partitioning (*rfp*) scheme proposed by Pothen et. al. in [2] for implementing dense QR factorization on a hypercube. An example of the latter case occurs in the parallel block schemes proposed by Golub et. al. in [1] for large scale least squares computations.

In our study of this submatrix merging process, we use the following definitions and observations.

1. We refer to the computations incurred in eliminating one nonzero element as a task.

2. Each task involves applying a Givens rotation to two rows with their leading nonzeros in the same position. Given in Fig. 2 is the sequential algorithm which implements the task for the following transformation:

$$\begin{pmatrix} a_{i,\ell} & a_{i,\ell+1} & a_{i,\ell+2} & \cdots & a_{i,n} \\ a_{j,\ell} & a_{j,\ell+1} & a_{j,\ell+2} & \cdots & a_{j,n} \end{pmatrix} \longrightarrow \begin{pmatrix} \tilde{a}_{i,\ell} & \tilde{a}_{i,\ell+1} & \tilde{a}_{i,\ell+2} & \cdots & \tilde{a}_{i,n} \\ 0 & \tilde{a}_{j,\ell+1} & \tilde{a}_{j,\ell+2} & \cdots & \tilde{a}_{j,n} \end{pmatrix}.$$

   Note that in Fig. 2, $a_{i,q}$ and $a_{j,q}$ $(\ell \leq q \leq n)$ are overwritten by $\tilde{a}_{i,q}$ and $\tilde{a}_{j,q}$.

3. If $(n - \ell + 1)$ is the number of nonzeros in each of the two rows, then the size of the task is measured by $4(n - \ell + 1)$ multiplicative operations.

```
if |a_{j,ℓ}| ≥ |a_{i,ℓ}| then
    t ← |a_{i,ℓ}|/|a_{j,ℓ}|
    s ← 1/√(1 + t^2)
    c ← st
else
    t ← |a_{j,ℓ}|/|a_{i,ℓ}|
    c ← 1/√(1 + t^2)
    s ← ct
for q = ℓ, ℓ+1, ···, n do
    v ← a_{i,q}
    w ← a_{j,q}
    a_{i,q} ← cv + sw
    a_{j,q} ← −sv + cw
```

Figure 2: Implementing the task of annihilating $a_{j,\ell}$ by Givens rotation.

4. A single task can be equally divided between two cooperative processors if each processor can access both rows but updates only one of them; i.e., both processors concurrently execute all of the steps given in Fig. 2 except for executing only

$$a_{i,q} \leftarrow cv + sw$$

or

$$a_{j,q} \leftarrow -sv + cw$$

in the **for** loop.

Throughout this manuscript whenever we divide a single task among two processors, we assume an even distribution of work load as described above.

5. All tasks involving disjoint pairs of rows can potentially be performed in parallel by different processors.

6. There are $k(k+1)/2$ nonzero elements to be eliminated in merging two $k \times n$ ($k \leq n$) upper trapezoidal submatrices. We note that these $k(k+1)/2$ elements do not necessarily come from the same submatrix as identified by $\otimes$ in Fig. 3. Instead, a Givens rotation can be applied to selectively zero out the $k(k+1)/2$ "$\otimes$" elements in Fig. 4. The same reduced matrix can be obtained by permuting the appropriate rows (as well as the corresponding right-hand-side elements) as shown in Fig. 5.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & & \otimes & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 3: $\otimes$: the elements to be annihilated.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \otimes & \times & \times & \times & \times & \times & \times \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 4: $\otimes$: the elements to be annihilated.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 5: Same result may be obtained by permuting the rows after the annihilation process.

2 The Multiprocessor Environments

In [1] the target machine is the University of Illinois Cedar system consisting of clusters of processors, where each cluster has a shared-memory system and the clusters are in turn interconnected via a single, system-wide shared memory. The data mapping strategy employed in [1] dictates that each cluster of processors have one $k \times n$ upper trapezoidal submatrix in their memory. To merge two such submatrices two clusters will cooperate with the aim to exploiting parallelism and minimizing inter-cluster communication. In [2] a parallel algorithm was proposed for merging two upper trapezoidal submatrices stored in the local memory of two directly connected processors. This algorithm was then embedded in the recursive fine partitioning scheme proposed in the same paper for implementing dense QR factorization on a hypercube multiprocessor.

In this study we shall propose a new submatrix merging algorithm which can be applied beneficially in either one of the multiprocessor environments considered above. However, in order to be clear and precise in our presentation, we shall postpone all discussion relating only to the Cedar system until the last section.

3 Submatrix Merging Algorithms

When merging two $k \times n$ upper trapezoidal submatrices on a single processor, the $k(k+1)/2$ nonzeros from one of the submatrices may be eliminated in many different orderings. For example, they may be eliminated column by column as shown in Fig. 6, or row by row as shown in Fig. 7 or diagonal by diagonal as shown in Fig. 8.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
1 & 3 & 6 & 10 & 15 & 21 & \times & \times & \times & \times & \times & \times \\
 & 2 & 5 & 9 & 14 & 20 & \times & \times & \times & \times & \times & \times \\
 & & 4 & 8 & 13 & 19 & \times & \times & \times & \times & \times & \times \\
 & & & 7 & 12 & 18 & \times & \times & \times & \times & \times & \times \\
 & & & & 11 & 17 & \times & \times & \times & \times & \times & \times \\
 & & & & & 16 & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

Figure 6: Column-by-column elimination sequence.

In order to devise an elimination sequence which is most suitable for parallel implementation, it is helpful to study the data access pattern of these three elimination sequences under the constraint that *a task may not involve data from both submatrices unless it cannot*

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \times & \times & \times & \times & \times & \times \\
 & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} & \mathbf{11} & \times & \times & \times & \times & \times & \times \\
 & & \mathbf{12} & \mathbf{13} & \mathbf{14} & \mathbf{15} & \times & \times & \times & \times & \times & \times \\
 & & & \mathbf{16} & \mathbf{17} & \mathbf{18} & \times & \times & \times & \times & \times & \times \\
 & & & & \mathbf{19} & \mathbf{20} & \times & \times & \times & \times & \times & \times \\
 & & & & & \mathbf{21} & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 7: Row-by-row elimination sequence.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\mathbf{1} & \mathbf{7} & \mathbf{12} & \mathbf{16} & \mathbf{19} & \mathbf{21} & \times & \times & \times & \times & \times & \times \\
 & \mathbf{2} & \mathbf{8} & \mathbf{13} & \mathbf{17} & \mathbf{20} & \times & \times & \times & \times & \times & \times \\
 & & \mathbf{3} & \mathbf{9} & \mathbf{14} & \mathbf{18} & \times & \times & \times & \times & \times & \times \\
 & & & \mathbf{4} & \mathbf{10} & \mathbf{15} & \times & \times & \times & \times & \times & \times \\
 & & & & \mathbf{5} & \mathbf{11} & \times & \times & \times & \times & \times & \times \\
 & & & & & \mathbf{6} & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 8: Diagonal-by-diagonal elimination sequence.

proceed without doing so. For each elimination sequence we identify the tasks which must access data from both submatrices and display such tasks and the required data in Fig. 9, 10 and 11.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\mathbf{1} & \mathbf{3} & \mathbf{6} & \mathbf{10} & \mathbf{15} & \mathbf{21} & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

Figure 9: Column-by-column elimination sequence: data from both submatrices are needed for tasks 1, 3, 6, 10, 15 and 21.

$$\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\mathbf{1} & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \mathbf{7} & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \mathbf{12} & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \mathbf{16} & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \mathbf{19} & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \mathbf{21} & \times & \times & \times & \times & \times & \times
\end{pmatrix}$$

Figure 10: Row-by-row elimination sequence: data from both submatrices are needed for tasks 1, 7, 12, 16, 19 and 21.

We observe from Fig. 9, 10 and 11 that each row of data from the top submatrix is used in exactly one task. Since the two submatrices are each located in a different processor or a different cluster of processors in the multiprocessor environment considered in [2] and [1], the tasks identified above are also those which require inter-processor or inter-cluster communication. Since each row of data in the top submatrix must participate in at least one task during the entire merging operation and all such tasks annihilate nonzeros in the bottom submatrix which is stored in a different processors (or a different cluster), the goal of *minimizing* inter-processor (or inter-cluster) communication can be achieved via any one elimination sequence described here. The particular sequence chosen for parallel implementation in [2] and [1] is the diagonal-by-diagonal elimination sequence.

However, using the approach above the tasks requiring inter-processor communication

$$
\begin{array}{c}
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
\end{pmatrix} \\
\begin{array}{cccccccccccc}
\mathbf{1} & \mathbf{7} & \mathbf{12} & \mathbf{16} & \mathbf{19} & \mathbf{21} & \times & \times & \times & \times & \times & \times
\end{array}
\end{array}
$$

Figure 11: Diagonal-by-diagonal elimination sequence: data from both submatrices are needed for tasks 1, 7, 12, 16, 19 and 21.

are the only tasks which can be performed by the two processors (or clusters) concurrently. Minimizing the number of such tasks can thus cause unbalanced work load distribution. This problem is more serious when two local-memory processors instead of two clusters are in question. In fact, it can be easily verified that when maintaining minimum inter-processor communication as suggested earlier, the parallel algorithm running on two processors has the same arithmetic complexity as the sequential algorithm.

4 A Balanced Submatrix Merging Algorithm

In order to balance the work load and minimize inter-processor communication, we propose to implement the alternative transformation in Fig. 12, which can be viewed as consisting of the annihilation process in Fig. 4 and the permutation process in Fig. 5.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \otimes & \times & \times & \times & \times & \times & \times \\
\otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \otimes & \otimes & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \otimes & \otimes & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times & \times & \times & \times
\end{pmatrix}
$$

Figure 12: An alternative elimination sequence ($\otimes$: the elements to be annihilated).

Assuming as before that each submatrix resides in a different processor, we employ the following two strategies to balance the work load and minimize inter-processor communication.

1. To balance the load, the tasks corresponding to the $k(k+1)/2$ nonzeros to be annihilated are evenly divided among the two processors.

2. To help reduce inter-processor communication, a "$\otimes$" may be zeroed by a processor other than the one it is originally stored in.

The parallel algorithm we propose can be best explained when applying to the transformation in Fig. 12. Let us denote the processor storing the top submatrix as P_A and the processor storing the bottom submatrix as P_B. We first specify the particular ordering these tasks are to be performed in the left diagram in Fig. 13, where the tasks to be performed by processors P_A and P_B are each labelled by its scheduled time step. We have distinguished P_A's tasks from P_B's by labelling P_A's i^{th} task by i and P_B's by $[i]$. The two tasks scheduled for the same step can potentially be performed concurrently by two processors provided the communication can be masked by computation. This point will be clear after we explain the inter-processor communication scheme. We next consult the diagram to the right in Fig. 13, which identifies the tasks requiring data from both submatrices.

$$
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & 2 & 3 & 4 & 5 & 6 & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & 8 & 9 & 10 & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & 12 & \times & \times & \times & \times & \times & \times \\
1 & [3] & [4] & [5] & [6] & [7] & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & 7 & [9] & [10] & [11] & \times & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & 11 & [13] & \times & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times & \times & \times & \times
\end{pmatrix}
\begin{pmatrix}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & 2 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & 8 & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & 12 & \times & \times & \times & \times & \times & \times \\
1 & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & 7 & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
 & & & & 11 & \times & \times & \times & \times & \times & \times & \times \\
 & & & & & & & & & & & \\
 & & & & & & & & & & &
\end{pmatrix}
$$

Figure 13: Concurrent scheduling of P_A's and P_B's tasks.

We now explain the inter-processor communication scheme using the example in Fig. 13. Our algorithm requires processor P_B to send the top row of the bottom submatrix to P_A so that P_A can complete tasks 1 and 2 as shown in Fig. 14, where the elements are labelled by "A" or "B" depending on in which processor they are originally stored.

$$
\begin{pmatrix}
A & A & A & A & A & A & A & A & A & A & A & A \\
 & A & A & A & A & A & A & A & A & A & A & A \\
B & B & B & B & B & B & B & B & B & B & B & B
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & A & A & A & A & A & A & A & A & A & A & A \\
0 & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B}
\end{pmatrix}
$$

$$
\longrightarrow
\begin{pmatrix}
\tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & 0 & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
0 & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}}
\end{pmatrix}
$$

Figure 14: Processor P_A performs tasks 1 and 2.

The row from P_B is thus appropriately updated by P_A and is sent back to P_B immediately after task 2 is completed. P_A can then proceed to complete tasks 3, 4, 5 and 6 without inter-processor communication as shown in Fig. 15. P_B will do the same with respect to its

$$
\begin{pmatrix}
\tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & 0 & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & & A & A & A & A & A & A & A & A & A & A \\
 & & & A & A & A & A & A & A & A & A & A \\
 & & & & A & A & A & A & A & A & A & A \\
 & & & & & A & A & A & A & A & A & A
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
\tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & 0 & 0 & 0 & 0 & 0 & \hat{A} & \hat{A} & \hat{A} & \hat{A} & \hat{A} & \hat{A} \\
 & & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & & & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & & & & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} \\
 & & & & & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A} & \tilde{A}
\end{pmatrix}
$$

Figure 15: Processor P_A performs tasks 3, 4, 5 and 6.

tasks [3], [4], $\cdots$, [7] after receiving back the modified top row as shown in Fig. 16.

$$
\begin{pmatrix}
0 & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} & \tilde{\tilde{B}} \\
 & B & B & B & B & B & B & B & B & B & B & B \\
 & & B & B & B & B & B & B & B & B & B & B \\
 & & & B & B & B & B & B & B & B & B & B \\
 & & & & B & B & B & B & B & B & B & B \\
 & & & & & B & B & B & B & B & B & B
\end{pmatrix}
\longrightarrow
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & \hat{B} & \hat{B} & \hat{B} & \hat{B} & \hat{B} & \hat{B} \\
 & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} \\
 & & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} \\
 & & & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} \\
 & & & & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} \\
 & & & & & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B} & \tilde{B}
\end{pmatrix}
$$

Figure 16: Processor P_B performs tasks [3], [4], $\cdots$, [7].

We next explain how to mask communication by computation. The strategy is for each processor to send out the row needed by the other processor as early as possible. For example, processor P_B should send its row 3 immediately after it completes task [4] so that it would have arrived in P_A when P_A completes task 6, and P_A should send the updated row back to P_B as soon as it completes tasks 7 and 8, and so on. In the next section we shall introduce a task precedence graph which is instrumental in our analysis of the performance

of the proposed algorithm and allows us to conveniently formalize the notion of masking communication by computation.

5 Performance Analysis of the Proposed Algorithm

In order to analyze the performance of this algorithm, we make use of a task precedence graph, where each vertex identified by a step number represents the task of annihilating the nonzero in that position of one submatrix. As an example, we display in Fig. 17 the task precedence graph set up according to the scheduling of P_A's and P_B's tasks in Fig. 13. The precedence relationship identified by $\rightarrow$ is established considering both data and processor availability subject to the condition that communication can be completely masked by computation. We show next how the latter condition is indeed satisfied by the particular strategy we employ for masking communication by computation.

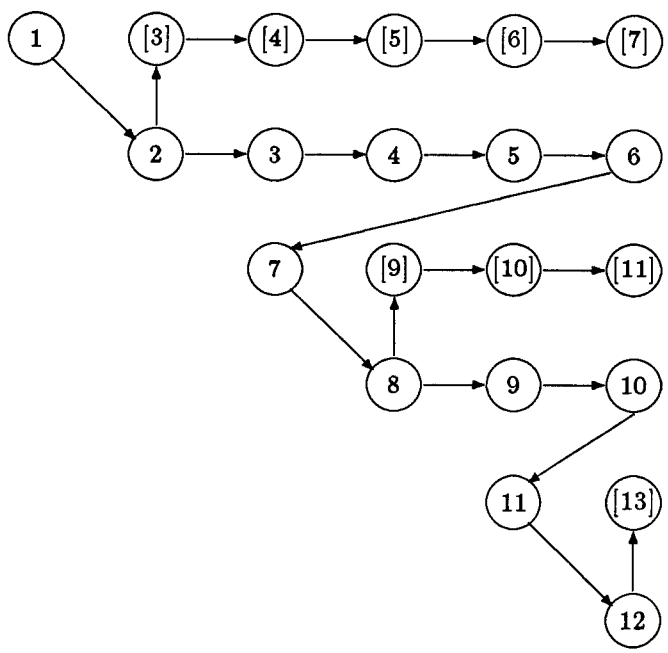


Figure 17: Task precedence graph of the example in Fig. 13.

In Fig. 18 we identify the data communication path by double arrows. The following observations are helpful in studying this graph.

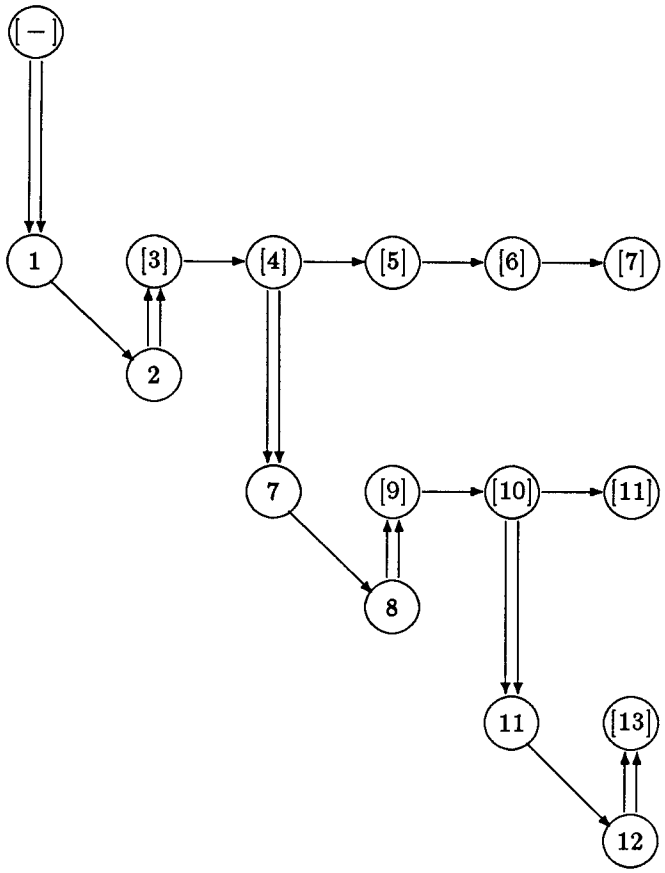


Figure 18: Task communication path graph of the example in Fig. 13.

1. We introduce a dummy task node ([−]) to account for the initial data communication from processor P_B to P_A.

2. The two tasks connected by double arrows are each executed by a different processor.

3. Data communication follows the arrow direction, namely that the processor executing the task at the tail sends the data to the processor executing the task at the head.

4. The placement of double arrows traces the actual data flow of our communication algorithm. For example, processor P_A sends the modified top row of the bottom submatrix back to P_B immediately after task 2 is completed and P_B needs this row to perform task [3]. The data transfer from P_A to P_B is faithfully reflected by the double arrow pointing from vertex (2) to vertex ([3]) in the communication path graph in Fig. 18.

In order to show that communication can be masked by computation in our algorithm, we need to adapt our analytical model to account for the time actually taken for communication. To motivate our proof, let us allocate one time step for communication and obtain the modified precedence graph as well as communication path graph for the example above in Fig. 19 and 20. We now contrast the critical paths embedded in the two precedence graphs in Fig. 21, which are established by assuming that the tasks scheduled for step i and $[i]$ finish at the same time. Consequently, step number i occurs only once in each critical path identified in Fig. 21 and the arrows connecting vertex i to vertex j, $i < j$, have been omitted. Note that the critical path in the left is identified from Fig. 17 assuming that communication takes no time at all, whereas the critical path in the right is identified from Fig. 19 assuming that communication takes one time step. To be technically precise, the latter assumption implies that sending one row of size $(n - q + 1)$ to another directly connected processor takes no more time than $4(n - q + 1)$ multiplicative operations. Another technical point is that the number of operations involved in step $[i]$ are not exactly equal to that of step i. The difference is $4(n-q+1)$ $(1 \leq q \leq k)$ for step $[i]$ versus either $4(n-q)$ for step i in Fig. 17 and 18 or $4(n-q-1)$ for step i in Fig. 19 and 20. We see that in either case the difference amounts to less than *eight* multiplicative operations, which is negligible when $n \gg k$. We shall thus assume that time step $[i]$ is of the same length as time step i throughout our analysis.

We now make the following important observation from Fig. 21, namely that the *delay* caused by communication does not affect the critical path until the very last three steps and the *total delay* amounts to three time steps exactly. An immediate question, of course, is whether this result holds for any given pair of $k \times n$ upper trapezoidal submatrices. It turns out that when k is even the delay amounts to three time steps and when k is odd the delay becomes four time steps. Our proof makes use of a generalized precedence graph given in Fig. 22 for k being an even number and the one given in Fig. 23 for k being an

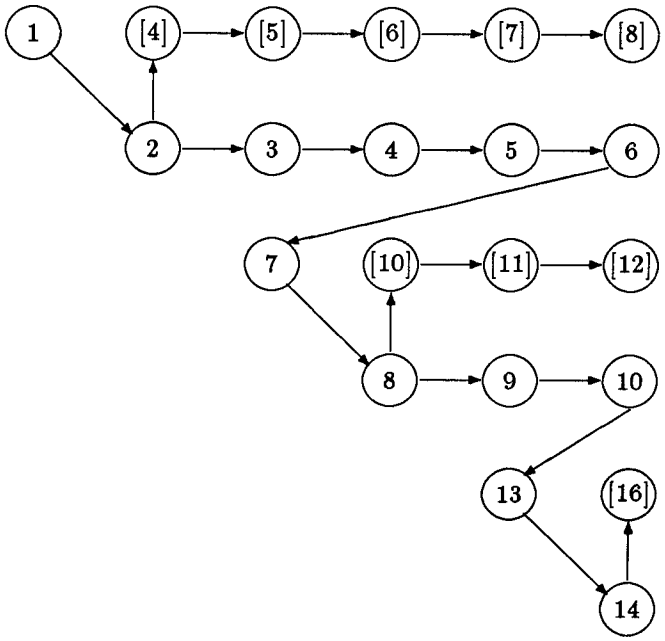


Figure 19: Task precedence graph (modified) of the example in Fig. 13.

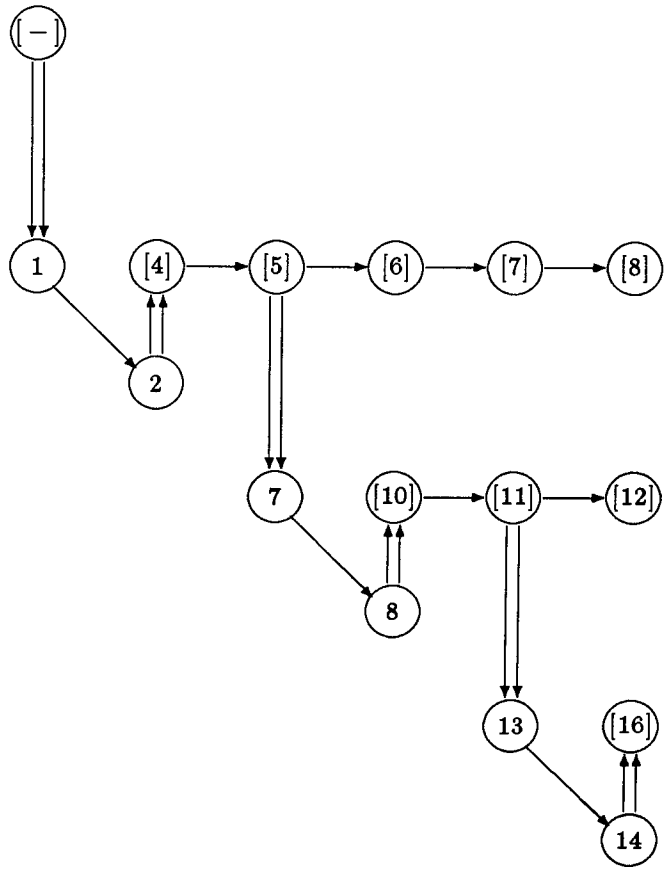


Figure 20: Task communication path graph (modified) of the example in Fig. 13.

1					
	2	3	4	5	6
		7			
			8	9	10
				11	[13]
					12

1					
	2	3	4	5	6
		7			
			8	9	10
				13	[16]
					14

Figure 21: The critical paths identified from the precedence graphs in Fig. 17 and 19.

odd number. We offer some explanation of the setup of the generalized precedence graph before we proceed.

Recall that our algorithm would zero out odd-numbered rows from the bottom triangular matrix and even-numbered rows from the top one. It is convenient to arrange the vertices of our task precedence graph as an upper triangle to reflect the locations of the elements to be eliminated. Since exactly one nonzero is annihilated by performing one task, the mapping from the task nodes to the nonzeros is *one-to-one* and *onto*. We label each task node by an integer i or $[i]$ depending on whether the task is performed by processor P_A or P_B. The vertices in Fig. 22 and 23 should be viewed as connected by arrows in the same manner as those of the task precedence graph in Fig. 19, although the arrows are not explicitly shown here due to lack of space. From our description of the algorithm, P_A would zero out the even-numbered rows from the top submatrix as well as the leading nonzeros of the odd-numbered rows from the bottom submatrix, whereas P_B would zero out the odd-numbered rows from the bottom submatrix except for their leading elements. We summarize the implication of such basic understanding below.

$$
\begin{array}{cccccccccccc}
1 & [4] & [5] & [6] & [7] & [8] & \cdots & \cdots & \cdots & [k_1-2] & [k_1-1] & [k_1] \\
 & 2 & 3 & 4 & 5 & 6 & \cdots & \cdots & \cdots & k_1-4 & k_1-3 & k_1-2 \\
 & & k_1-1 & [k_1+2] & [k_1+3] & [k_1+4] & \cdots & \cdots & \cdots & [k_3-2] & [k_3-1] & [k_3] \\
 & & & k_1 & k_1+1 & k_1+2 & \cdots & \cdots & \cdots & k_3-4 & k_3-3 & k_3-2 \\
 & & & & k_3-1 & [k_3+2] & \cdots & \cdots & \cdots & [k_5-2] & [k_5-1] & [k_5] \\
 & & & & & k_3 & \cdots & \cdots & \cdots & k_5-4 & k_5-3 & k_5-2 \\
 & & & & & & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
 & & & & & & & \cdots & \cdots & \cdots & \cdots & \cdots \\
 & & & & & & & & k_m-1 & [k_m+2] & [k_{m+2}-1] & [k_{m+2}] \\
 & & & & & & & & & k_m & k_{m+2}-3 & k_{m+2}-2 \\
 & & & & & & & & & & k_{m+2}+1 & [k_{m+2}+4] \\
 & & & & & & & & & & & k_{m+2}+2
\end{array}
$$

Figure 22: A generalized precedence graph (k is an even number).

1	[4]	[5]	[6]	[7]	[8]	$\cdots$	$\cdots$	$\cdots$	$[k_1 - 2]$	$[k_1 - 1]$	$[k_1]$
	2	3	4	5	6	$\cdots$	$\cdots$	$\cdots$	$k_1 - 4$	$k_1 - 3$	$k_1 - 2$
		$k_1 - 1$	$[k_1 + 2]$	$[k_1 + 3]$	$[k_1 + 4]$	$\cdots$	$\cdots$	$\cdots$	$[k_3 - 2]$	$[k_3 - 1]$	$[k_3]$
			k_1	$k_1 + 1$	$k_1 + 2$	$\cdots$	$\cdots$	$\cdots$	$k_3 - 4$	$k_3 - 3$	$k_3 - 2$
				$k_3 - 1$	$[k_3 + 2]$	$\cdots$	$\cdots$	$\cdots$	$[k_5 - 2]$	$[k_5 - 1]$	$[k_5]$
					k_3	$\cdots$	$\cdots$	$\cdots$	$k_5 - 4$	$k_5 - 3$	$k_5 - 2$
						$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
							$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
							$k_m - 1$	$[k_m + 2]$	$[k_{m+2} - 2]$	$[k_{m+2} - 1]$	$[k_{m+2}]$
								k_m	$k_m + 1$	$k_{m+2} - 3$	$k_{m+2} - 2$
									k_{m+2}	$[k_{m+2} + 3]$	$[k_{m+2} + 4]$
										$k_{m+2} + 1$	$k_{m+2} + 2$
											$k_{m+2} + 6$

Figure 23: A generalized precedence graph (k is an odd number).

1. The tasks along each even-numbered row are scheduled consecutively, as are the tasks (except for the leading one) along each odd-numbered row. This is evident from our examples in Fig. 17 and 19 as well as the two generalized precedence graphs in Fig. 22 and 23.

2. We note also that since the leading task of each odd-numbered row is performed by the same processor performing the tasks along the following even-numbered row, the sequencing of these tasks is also consecutive.

3. From the task precedence graph given in Fig. 17, it is clear that the data dependency prevents P_B from processing the second leading task of the odd-numbered row earlier than the completion of the leading task in the following even-numbered row. We show in Fig. 19 that the actual time taken for data transmission causes P_B to be two time steps behind P_A on completing the last task of each row. This fact is faithfully reflected on our generalized precedence graphs in Fig. 22 and 23, where the last task of each odd-numbered $(2i + 1)^{th}$ row is labelled "$[k_{2i+1}]$", and the last task of the following even-numbered row is labelled "$k_{2i+1} - 2$". It follows that subject to the condition that "$k - (2i + 1) - 1 \geq 4$" the leading tasks of the following two rows would

be numbered $k_{2i+1} - 1$ and k_{2i+1}, from the latter we further infer that the second leading task of the next odd-numbered row would be labelled $[k_{2i+1} + 2]$.

4. Note that the condition "$k - (2i + 1) - 1 \geq 4$" is violated when the row number $m = 2i + 1 = k - 3$ if k is even or $m = 2i + 1 = k - 4$ if k is odd.

5. It should now be clear that the critical path of the algorithm can be established by tracing Processor P_A's execution sequence and taking into account the delay caused by the violation of the condition "$k - (2i + 1) - 1 \geq 4$". As examples, we identify the two critical paths corresponding to the two cases for $k = 12$ and $k = 13$ in Fig. 24 and 25.

1											
	2	3	4	5	6	7	8	9	10	11	12
		13									
			14	15	16	17	18	19	20	21	22
				23							
					24	25	26	27	28	29	30
						31					
							32	33	34	35	36
								37			
									38	39	40
										43	[46]
											44

Figure 24: The critical path for $k = 12$.

1												
	2	3	4	5	6	7	8	9	10	11	12	13
		14										
			15	16	17	18	19	20	21	22	23	24
				25								
					26	27	28	29	30	31	32	33
						34						
							35	36	37	38	39	40
								41				
									42	43	44	45
										47		
											48	49
												53

Figure 25: The critical path for $k = 13$.

We complete our analysis by proposing the following theorems.

Theorem 1 *Suppose we employ two directly connected local-memory processors to merge two $k \times n$ upper trapezoidal submatrices. Using the balanced submatrix merging algorithm we proposed in section 4 of this article, the length (in terms of time steps) of the critical path is*

$$\frac{1}{2}\left(\frac{k(k+1)}{2} + \frac{k}{2} + 8\right)$$

for k even, and

$$\frac{1}{2}\left(\frac{k(k+1)}{2} + \frac{k-1}{2} + 9\right)$$

for k odd.

Proof: We first consider the case when k is an even number. Referring to the generalized precedence graph given in Fig. 22, we recall our observation that the condition "$k - (2i+1) - 1 \geq 4$" is violated when the odd row number $m + 2 = 2i + 1 = k - 3$, which is the fourth row from the bottom. Using our notation the last task of the $(m+2)^{th}$ row is labelled by time step $[k_{m+2}]$, it follows that the third task on this row must be completed in time step $[k_{m+2} - 1]$. It is now straightforward to verify that the last three time steps are $k_{m+2} + 1$, $k_{m+2} + 2$, followed by $[k_{m+2} + 4]$. Noting that these three steps would be scheduled as $k_{m+2} - 1$, k_{m+2} and $[k_{m+2} + 1]$ in the ideal situation when the communication time is ignored entirely, henceforth the total delay is exactly three time steps.

To compute the length of the critical path we count processor P_A's tasks and take into account the delay of three time steps. By simple algebra we immediately obtain the following formula.

$$1+\frac{1}{2}\left(\frac{k(k+1)}{2}+\frac{k}{2}\right)+3=\frac{1}{2}\left(\frac{k(k+1)}{2}+\frac{k}{2}+8\right) . \tag{1}$$

The proof for the second case is similar except for noting that the condition "$k-(2i+1)-1 \geq 4$" is violated when $m+2=2i+1=k-4$, which is the fifth row from the bottom. The effect is that the last four steps are labelled k_{m+2}, $k_{m+2}+1$, $k_{m+2}+2$ and lastly $k_{m+2}+6$ instead of $k_{m+2}-1$, k_{m+2}, $k_{m+2}+1$ and $k_{m+2}+2$. The total delay is exactly four time steps in this case. The total number of time steps is thus given by

$$\frac{1}{2}\left(\frac{k(k+1)}{2}+\frac{k-1}{2}+1\right)+4=\frac{1}{2}\left(\frac{k(k+1)}{2}+\frac{k-1}{2}+9\right) . \tag{2}$$

□

Theorem 2 *Using the balanced submatrix merging algorithm we proposed in section 4 of this article for merging two $k \times n$ upper trapezoidal submatrices, the total data exchanged between the two directly connected processors are $\lceil k/2 \rceil$ rows in the annihilation process and $\lfloor k/2 \rfloor$ rows in the permutation process.*

Proof: The proof can be immediately obtained by noting that (i) only the odd-numbered rows from the bottom submatrix must be sent back and forth (one exchange) between the two processors during the annihilation process, and (ii) only the even-numbered rows from the bottom submatrix must be exchanged with the even-numbered rows from the top submatrix during the permutation process. □

6 Further Parallelization on Two Clusters of Processors

We described in the last section a new parallel algorithm for merging two upper trapezoidal submatrices on two directly connected processors. We also show that the work load is evenly divided between the two processors and that communication is well masked by computation assuming that the time for transmitting n floating-point numbers is no longer than the time taken for $4n$ multiplicative operations. In this section we shall analyze the performance of the proposed algorithm when two clusters of processors are employed and compare our speed-up result with that of the algorithm proposed in [1].

Our analysis models the same multiprocessor environment considered in [1], where the target machine consists of clusters of processors. Each cluster is a shared-memory system and the clusters are in turn interconnected via a single, system-wide shared memory. Since the submatrices to be merged are each located in the local shared-memory of a cluster, we

find it convenient to model the inter-cluster communication using the notion of message-passing, which can be easily implemented by writing to and reading from the global shared-memory.

Let us assume that each cluster has p processors. We thus have $2p$ processors at our disposal by employing two clusters. Since the algorithm proposed in [1] exploits parallelism within only one of the two clusters by overlapping the annihilation of the top row of the bottom submatrix (which requires intercluster communication) with the annihilation of the rest of the elements of the same submatrix (which requires only local communication), clearly the maximum possible speed-up is p, resulting in an efficiently of $p/2p \leq 50\%$.

While the algorithm we proposed in section 4 can be easily adapted for implementation on two p-processor clusters, its performance needs to be carefully re-examined. We first establish the shortest critical path (in terms of time steps) of the proposed algorithm in Lemma 3. For simplicity in presentation we shall only consider the case when k is an even number throughout the rest of the manuscript. Besides, as far as performance is concerned, it is adequate to analyze one case of k and similar performance is expected for the case of $k+1$ for any algorithm with medium data granularity.

Lemma 3 *We adapt the algorithm proposed in section 4 to merge two $k \times n$ upper trapezoidal submatrices when P_A and P_B are each a cluster of processors. We assume that the $k(k+1)/2$ tasks are assigned to P_A and P_B exactly as before. We further assume that within each cluster the tasks along the same row are performed sequentially by the same processor, and that there are enough free processors to start processing each row of tasks as early as permitted by the availability of data. Subject to the assumptions above, the shortest critical path of the parallel algorithm consists of $3k-2$ time steps.*

Proof: In Fig. 26 we present the task precedence graph for the case $k = 12$ taking into account the actual communication time but assuming a free processor exists when data is available. Recall that the leading tasks of the odd-numbered rows are performed by P_A and observe that data dependency and the time taken for communication dictates that there is a gap of six time steps between the leading tasks of two consecutive odd-numbered rows. As examples, the $k/2$ such tasks are numbered 1, 7, 13, 19, 25 and 31 in Fig. 26. It is a straightforward exercise to generalize the numbering sequence for any given value of k and obtain the critical path of

$$1 + 6\left(\frac{k}{2} - 1\right) + 3 = 3k - 2\,. \tag{3}$$

□

Substituting k in equation (3) by the value of 12, we obtain a shortest critical path of 34 time steps for our example as verified in Fig. 26.

Since the serial algorithm takes $k(k+1)/2$ time steps and the shortest critical path takes $3k-2$ time steps, the speed-up is $k(k+1)/(6k-4) > k/6$. Note that when $k \ll n$,

1	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
	2	3	4	5	6	7	8	9	10	11	12
		7	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]
			8	9	10	11	12	13	14	15	16
				13	[16]	[17]	[18]	[19]	[20]	[21]	[22]
					14	15	16	17	18	19	20
						19	[22]	[23]	[24]	[25]	[26]
							20	21	22	23	24
								25	[28]	[29]	[30]
									26	27	28
										31	[34]
											32

Figure 26: Concurrent scheduling of cluster P_A's and cluster P_B's tasks assuming that free processors exist when data is available.

the difference between the time taken for $4n$ multiplicative operations and the time taken for $4n - i$ multiplicative operations, $1 \leq i \leq k - 1$, is negligible. Therefore, it is sensible to measure the performance of the algorithm by the number of time steps. To obtain the efficiency, we need to know how many processors are needed to achieve the shortest critical path. We obtain the result in Lemma 4 by analyzing the task precedence graph in Fig. 26.

Lemma 4 *In order to complete the merge of two $k \times n$ upper trapezoidal matrices in $3k-2$ time steps when employing two clusters of processors, each cluster must have $\lceil k/6 \rceil$ processors.*

Proof: Suppose each cluster has p processors. The p processors of cluster P_A will each start a consecutive sequence of tasks at time step 1, 7, $\cdots$, and $1 + 6(p - 1)$. Clearly the processor which processes tasks 1, 2, 3, $\cdots$, and k will finish first and is able to start the next sequence of tasks, of which the leading task may start earliest at time step $1 + 6p$. Since no processor is free until time step $k+1$, we must have $1+6p \geq k+1$, i.e., $p = \lceil k/6 \rceil$, to achieve the shortest critical path.

We next show that when this condition is satisfied, the remaining $p - 1$ processors in cluster P_A can all begin the following sequences of tasks at the earliest possible scheduled time by simply observing the following.

1. The p processors in cluster P_A become free one by one after k, $k+4$, $\cdots$, and $k+4(p-1)$ time steps respectively.

2. The next p sequences of tasks are scheduled for time steps $1+6p$, $1+6(p+1)$, $1+6(p+2)$, $\cdots$, and $1+6(2p-1)$.

3. The inequality $1+6p \geq k+1$ implies $1+6p+6i \geq k+4i+1$ for $1 \leq i \leq p-1$.

Finally, we need to establish that under the same condition all processors in cluster P_B will also be available to process their assigned tasks at the earliest possible time step. To show that this is indeed the case we simply note that the last task of each odd-numbered row is finished two time steps behind the last task of the following even-numbered row, whereas the first task scheduled for processors in P_B in the following odd-numbered row begins three time steps later than the leading task. Since the same condition $1+6p \geq k+1$ implies $(1+6p)+3 > (k+2)+1$, we have proved that all tasks assigned to cluster P_B can all proceed as scheduled to achieve the shortest critical path. □

From Lemma 3 and 4, a speed-up of $k/6$ is obtained while employing a total of $k/3$ processors, resulting in an efficiency of 50%. However, in contrast to the upper bound of 50% efficiency for the algorithm in [1], we shall show that the 50% efficiency is a "lower bound" of our algorithm when $p \leq \lceil k/6 \rceil$ processors are employed in each cluster. Before we proceed, we first use an example to explain how the proposed algorithm works when $p < \lceil k/6 \rceil$ are available in each cluster. Let us refer to Fig. 27, where we present the task precedence graph for $k = 16$ when each cluster has two processors. We first note that since $p = 2$ and $\lceil k/6 \rceil = 3$, the condition $p = \lceil k/6 \rceil$ in Lemma 4 is violated and the shortest critical path cannot be achieved. We therefore must establish the critical path for an arbitrary choice of $p < \lceil k/6 \rceil$ taking into account not only the availability of data but also the existence of free processors. The first four rows of task in the precedence graph are scheduled taking into account only the data availability because all four processors (two in each cluster) are free initially. However, when we reach the fifth row of the graph, the leading task can no longer be scheduled for time step 13 because no processor in cluster P_A is free to take on the task until time step 17. When we reach the thirteenth row, we encounter a case when there exists a free processor to start the leading task at time step 37 but the data are not available for the processor to take on this task until time step 41. For this example, the critical path consists of 50 time steps.

Theorem 5 *We consider merging two $k \times n$ upper trapezoidal submatrices on two clusters of processors by adapting the algorithm proposed in section 4 as described above. If each cluster has p processors, then the merging process can be completed in*

$$\frac{k^2}{4p} + \frac{k}{2} + 6p - 2 \tag{4}$$

time steps, resulting in an asymptotic speed-up of $2p$ and 100% efficiency.

1	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		7	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]
			8	9	10	11	12	13	14	15	16	17	18	19	20
				17	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]
					18	19	20	21	22	23	24	25	26	27	28
						23	[26]	[27]	[28]	[29]	[30]	[31]	[32]	[33]	[34]
							24	25	26	27	28	29	30	31	32
								29	[32]	[33]	[34]	[35]	[36]	[37]	[38]
									30	31	32	33	34	35	36
										35	[38]	[39]	[40]	[41]	[42]
											36	37	38	39	40
												41	[44]	[45]	[46]
													42	43	44
														47	[50]
															48

Figure 27: Concurrent scheduling of tasks for two clusters with each having 2 processors.

Proof: To establish the critical path length as given in equation (4), recall again that we have used i to label the time steps of the tasks assigned to processors in P_A and $[i]$ to label the time steps of those assigned to processors in P_B. We assume that all of the $[i]$ tasks along the odd-numbered row of the precedence graph are processed sequentially by one processor in P_B, whereas the leading i task of an odd-numbered row and all of the i tasks of the following even-numbered row are processed sequentially by one processor in P_A.

Our algorithm requires that each sequence of tasks are performed by the earliest available processor. Since the data dependency dictates that the processor assigned tasks along row $2i+1$ or $2i+2$ finishes before the processor assigned tasks along row $2i+3$ or $2i+4$ respectively, the mapping strategy is, in fact, equivalent to wrap-mapping the rows of tasks around the p processors within each cluster. To be technically precise, we group the leading task of each odd-numbered row with the tasks along the following even-numbered row. In other words, for any given values of k and p, assuming that k is an integral multiple of $2p$, our mapping strategy assigns $k/2p$ rows of tasks to each processor, while each block of consecutive $2p$ rows of tasks are performed by $2p$ different processors.

We now trace the execution path of one particular processor in cluster P_A. We naturally choose the processor which begins the merging process by performing task 1. Let us denote

this processor by $P_A^{(1)}$. Given below are the key observations leading to the proof of this theorem.

The first task $P_A^{(1)}$ performs in each of the assigned $k/2p$ rows is the leading task of the top odd-numbered row of each block of consecutive $2p$ rows. Referring to our example in Fig. 27, $P_A^{(1)}$ begins each sequence of tasks by taking on tasks 1, 17, 29 and 41, which are the leading tasks of rows 1, 5, 9 and 13, or 1, $1+2p$, $1+4p$ and $1+6p$ for $p=2$. Consequently, if we can schedule these tasks taking into account both data dependency and processor availability, then we can derive the length of the critical path of the algorithm. In order to do so, observe that if $P_A^{(1)}$ has completed its sequence of tasks in the jth block in step k_m-2, where $m=2p(j-1)+1$ using our notation in the generalized precedence graph, then the time step $P_A^{(1)}$ can begin with its tasks in the $(j+1)^{th}$ block is the maximum of "k_m-1" and "$k_m-1-2p(k/2p-j+1)+6p$". There is delay only when the choice must be the latter. We next find out in which block this would occur. That is, we would like to find out the value of "$j+1$" such that

$$k_m-1-2p(k/2p-j+1)+6p > k_m-1\ . \tag{5}$$

Simplifying inequality (5), we obtain

$$j > \frac{k}{2p}-2\ . \tag{6}$$

The minimum value of j satisfying the inequality (6) is $k/2p-1$. We therefore have proved that the delay will not occur until the $(j+1=k/2p)^{th}$ block, which is the very last block! The step number $k_m-1-2p(k/2p-j+1)+6p$ of the leading task in the $(j+1=k/2p)^{th}$ block can be simplified to be k_m+2p-1, with k_m computed from equation (7).

$$\begin{aligned} k_m &= 2+\sum_{j=0}^{\frac{k}{2p}-2}(k-2pj) \\ &= \frac{k^2}{4p}+\frac{k}{2}-2p+2\ . \end{aligned} \tag{7}$$

It is now straightforward to compute the critical path length by

$$\begin{aligned} k_m+2p-1+6(p-1)+3 &= k_m+8p-4 \\ &= \frac{k^2}{4p}+\frac{k}{2}+6p-2\ . \end{aligned} \tag{8}$$

As an example, we substitute $k=16$ and $p=2$ in the formula above and obtain a critical path of 50 time steps as verified in Fig. 27.

Since the serial algorithm requires $k(k+1)/2$ time steps, with the parallel algorithm completed in $k^2/4p+O(k)$ time steps we obtain an asymptotic speed-up of $2p$ and efficiency of 100%. □

7 Conclusions

The problem of merging two $k \times n$ ($k \leq n$) upper trapezoidal submatrices on multiprocessor machines is considered in this paper. The parallel algorithms we present are designed for implementation on either a pair of directly connected local-memory processors or two clusters of tightly-coupled processors. Our analysis of the proposed algorithms shows that in both environments the work load is evenly distributed, communication can be well masked by computation, and the optimal speed-up may be achieved. In the orthogonal factorization phase described in [1] and [2] for solving large scale dense or sparse least squares problems, multiple pairs of processors or clusters may repetitively apply the proposed algorithm to merge multiple pairs of submatrices concurrently throughout the computation. While the proposed algorithm has improved an important step in solving the least squares problem on clusters of processors, the distribution of data remain suitable for employing the parallel schemes available in [1] for back substitution and the calculation of certain elements of the covariance matrix. Furthermore, the unit operation of the proposed algorithm involves applying a Givens rotation to two rows of data, which is not different from the unit operation defined in [1] and is thus also suitable for exploiting the vector capacity of each processor within the cluster.

Acknowledgement

The authors thank Professor Joseph Liu for bringing reference [1] to their attention.

To Eleanor

From Sue DeAngelis

Date Sept. 27

memo

University of Waterloo

I am out of stock of CS-88-45 "A Balanced Submatrix..." and CS-88-46 "Updating and Downdating...". Please advise ASAP if you would like more copies made.

Thanks.

Sue

Sue, please have another 50 made of each report. Thanks. Eleanor