1991

Unstructured Scheduling in Parallel PDE Sparse Solvers on Distributed Memory Machines

Mo Mu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
91-077

UNSTRUCTURED SCHEDULING IN PARALLEL PDE SPARSE SOLVERS ON DISTRIBUTED MEMORY MACHINES

Mo Mu
John R. Rice

CSD-TR-91-077
November 1991

TR - 91-077

UNSTRUCTURED SCHEDULING IN PARALLEL PDE SPARSE SOLVERS ON DISTRIBUTED MEMORY MACHINES

Mo Mu*
and
John R. Rice**

Computer Science Department
Purdue University
West Lafayette, IN 47907

October 25, 1991
Oak Ridge, TN

---

* Supported by NSF grant CCR-86-19817.
** Supported in part by AFOSR grant 88-0243 and the Strategic Defense Initiative through ARO contract DAAG03-90-0107.

OUTLINE

- Background
- Underlying Algorithm
- Load Imbalance
- Unstructured Scheduling
- Other Optimization Strategies
- Conclusions

BACKGROUND

MOTIVATION

- Parallel ELLPACK

Solution Library
- Iterative Methods
- **Direct Methods**

- Distributed memory machines

PDE PROBLEM

- General coefficients
- General boundary condition types
- General geometric domains

DISCRETIZATION

- Various Discretizations and Grids

    Finite differences

        Standard
        High order

    Finite elements

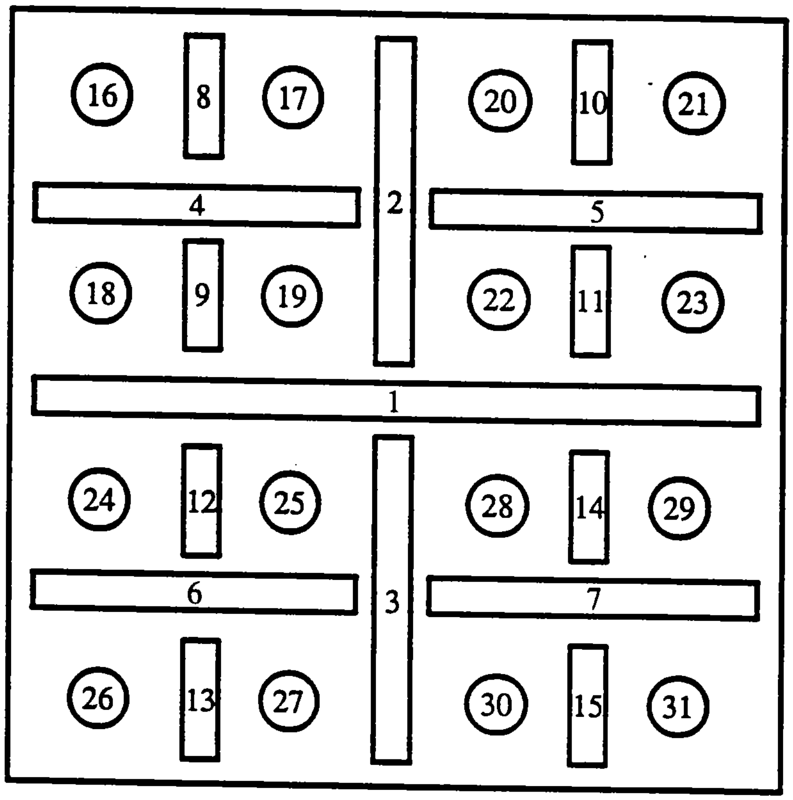
        Collocation
        Galerkin
        on triangles or rectangles

    Hybrid schemes

- Distributed Over Processors

INDEXING

Incomplete Nested Dissection
(domain decomposition based)



- *within each subdomain ("circle")*

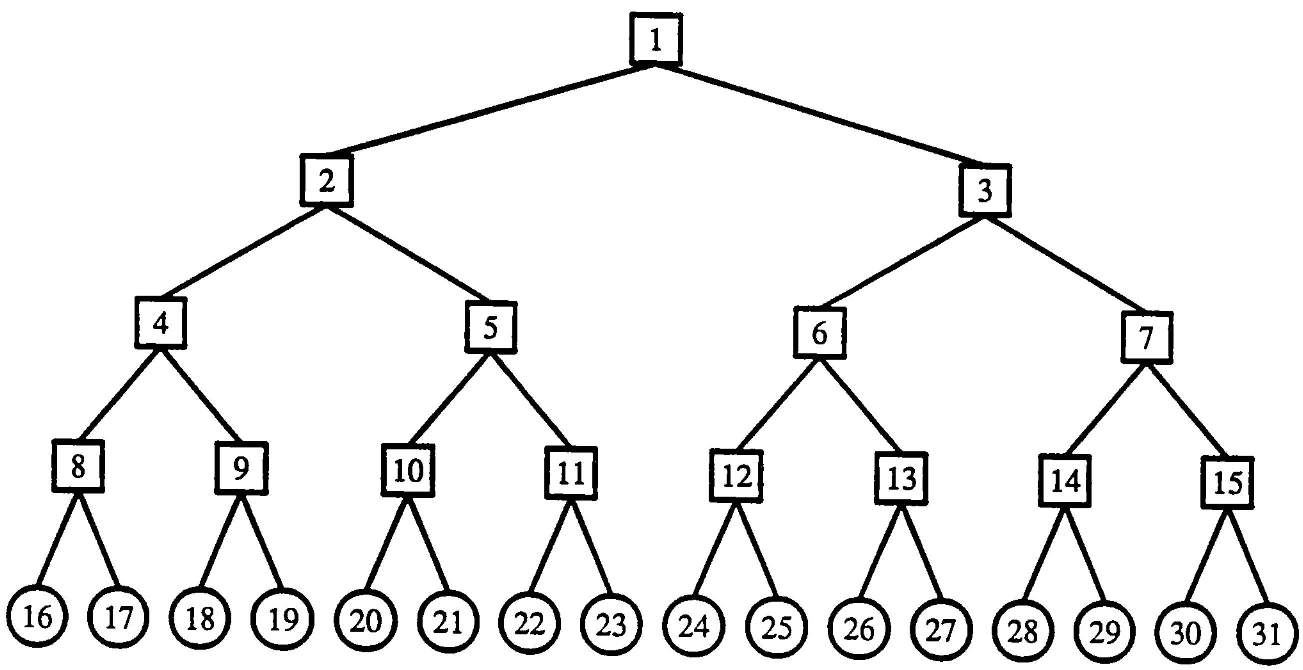
nested dissection
(potentially any efficient indexing scheme)

- *interface (the set of "boxes")*

nested dissection

INDEXING (CONTINUED)

Elimination Tree

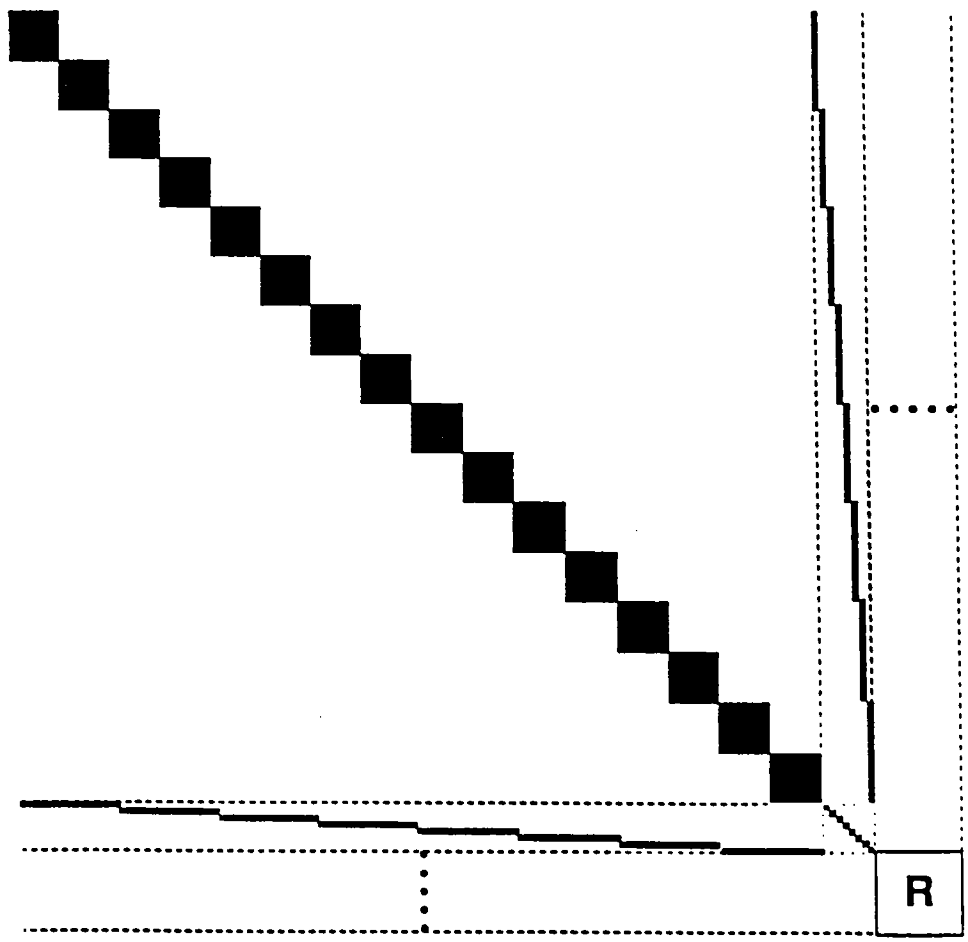


MATRIX PROBLEM

- Very large, sparse
- Nonsymmetric
- Block structured
- Distributed by row
- Numerically stable
- No symbolic factorization

MATRIX PROBLEM (CONTINUED)

Sparse Matrix Structure

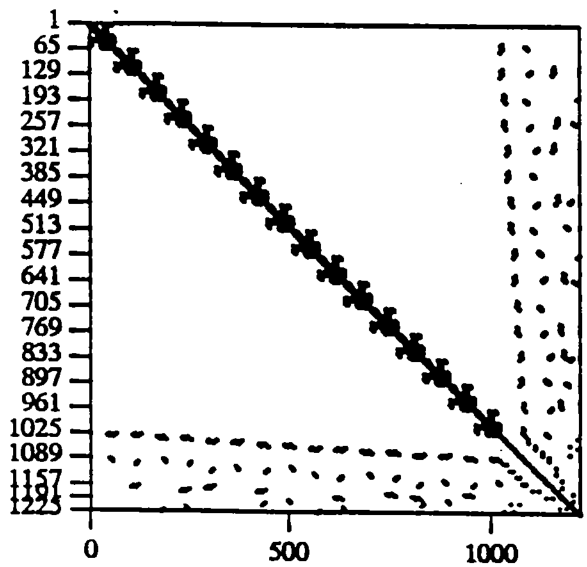


The sparse matrix structure for $p = 16$ processors. For the first two levels the solid boxes are where nonzero matrix elements might be (actually, these blocks are sparse also). The lower right box R contains diagonal blocks for the other 3 levels. Dots indicate sparse rows and columns. The relative sizes are correct for $n^2 = 100$, the number of grid points in one subdomain.

MATRIX PROBLEM (CONTINUED)

Sparse Matrix Structure

(a)



(b)

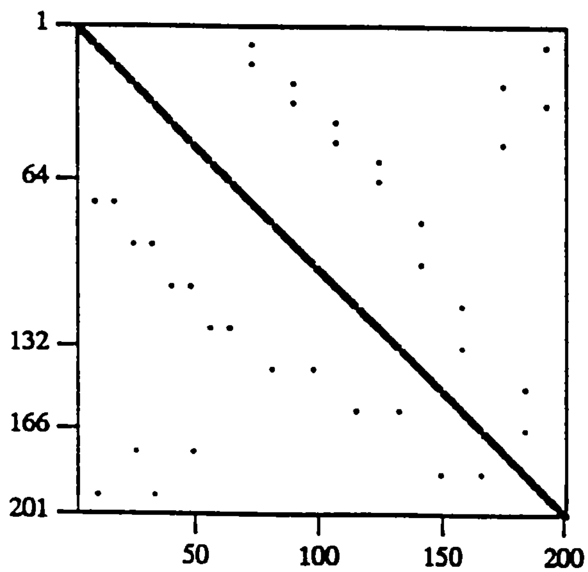
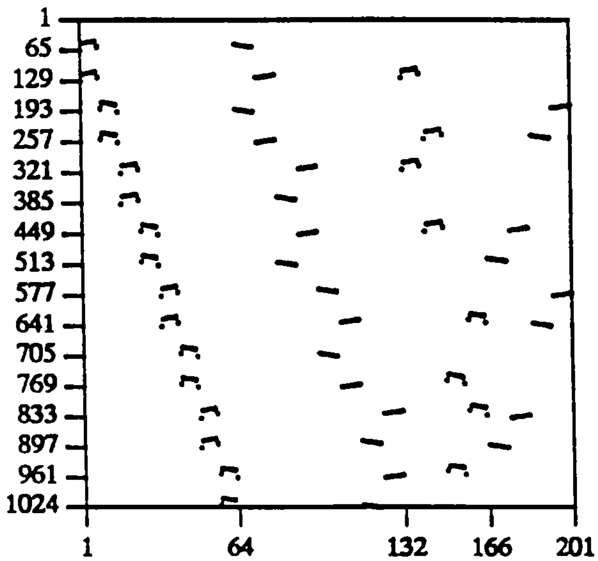


Figure 10: (a) Actual non-zero structure with $p = 16$, $n = 8$. The equation numbers are listed on the left. (b) The lower right block (everything except level 0) before the elimination starts.

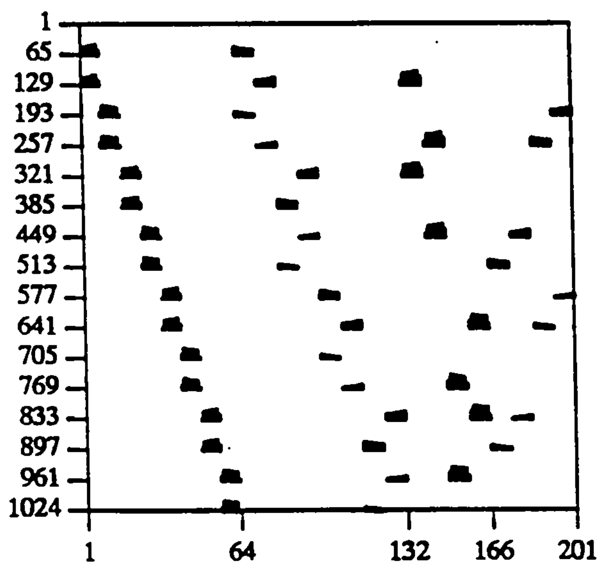
MATRIX PROBLEM (CONTINUED)

Sparse Matrix Structure

(a)



(b)

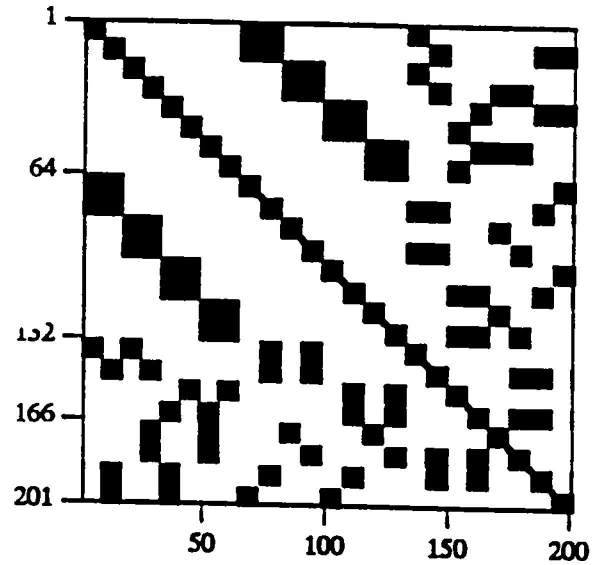


(a) The non-zero structure of the upper right matrix B before the elimination starts. Note that the display is distorted. B has 1024 rows and 201 columns. (b) The upper right matrix $\bar{B}$ after the level 0 elimination.

MATRIX PROBLEM (CONTINUED)

Sparse Matrix Structure

(a)



(b)

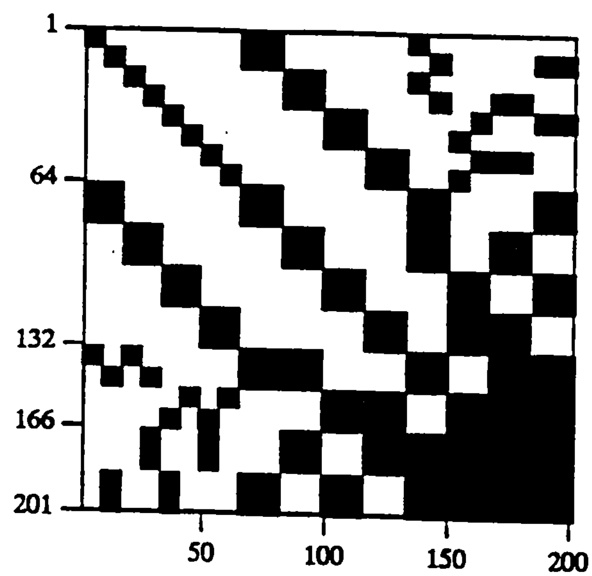


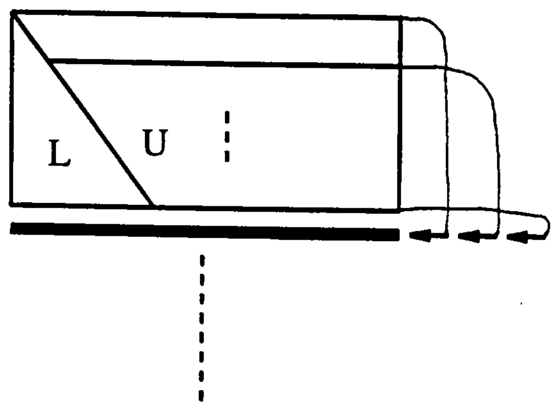
Figure 11: (a) The effect of the level 0 elimination on the lower right block. $\tilde{D}$ is given by (5). (b) The lower right block at the end of the elimination.

UNDERLYING

ALGORITHM

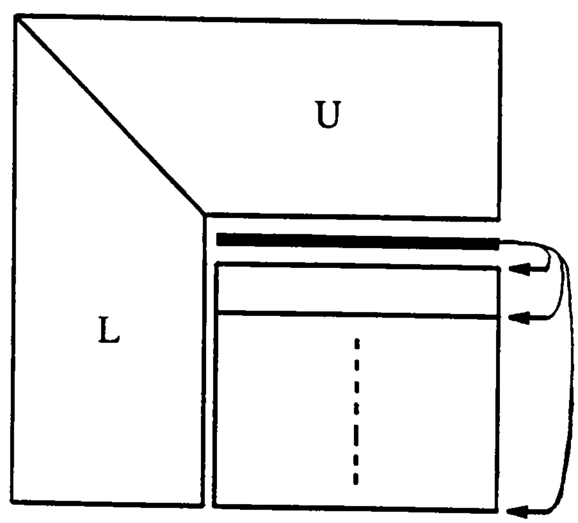
COMPUTATION ORGANIZATIONS

- up-looking



Do everything for an equation when you reach it.

- down-looking

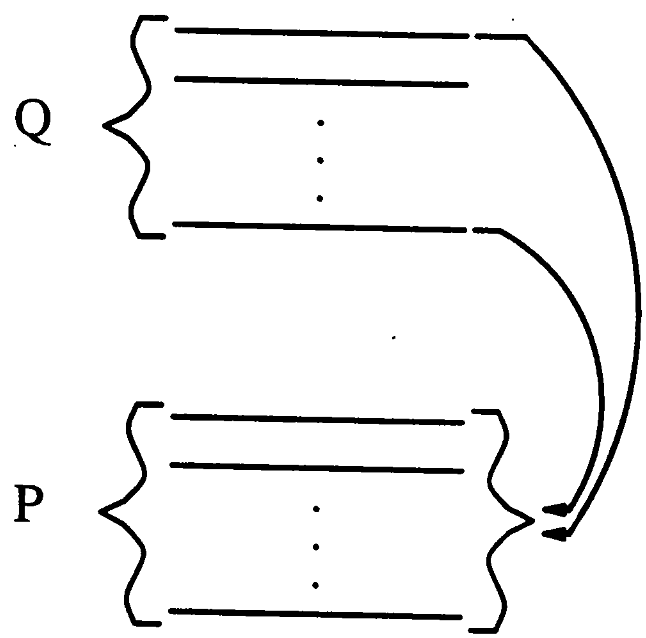


Have the effects of elimination in an equation propagated before going on to the next equation.

COMMUNICATION ORGANIZATIONS

Q = Source P = Destination

- fan-out

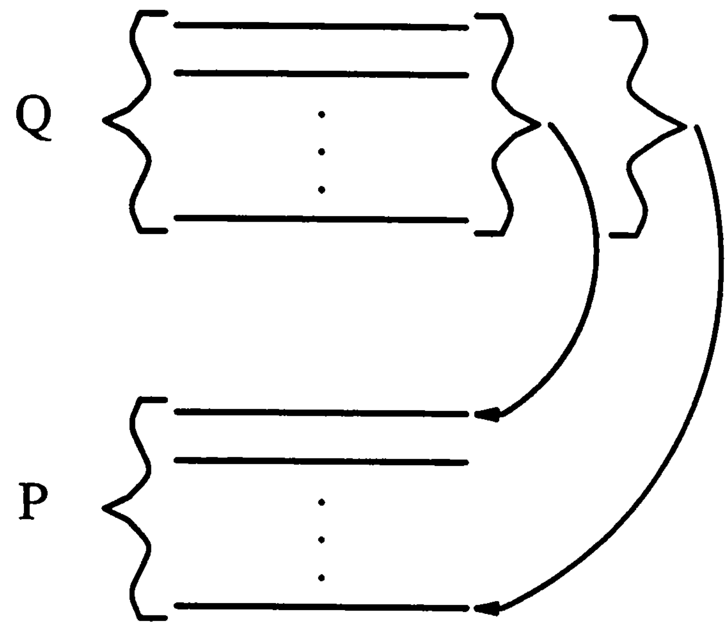


When processing an equation organize and pass on everything to later equations that they will need.

COMMUNICATION ORGANIZATIONS (CONTINUED)

Q = Source P = Destination

- fan-in



When processing an equation get everything from preceding equations that is needed.

$$Q : \mathbf{r}_i^q = \sum_{k \in K} (a_{ik}/a_{kk})^* \text{row}_k$$

$$= \sum_{k \in K} (a_{ki}/a_{kk})^* \text{row}_k \quad \text{(if A is symmetric)}$$

$$P : \text{row}_i = \text{row}_i - \mathbf{r}_i^q$$

OBSERVATIONS AND FACTS

- Up-looking is better than down-looking in sparse data structure manipulation

- Fan-in has less communication overhead than fan-out

- Fan-out is suitable for down-looking

- Fan-in is suitable for up-looking

- Fan-in is not applicable to nonsymmetric matrices
  - (a) rows in the partial sum are in the source processor while the corresponding multipliers are in the destination processor;
  - (b) all multipliers of an equation in the destination processor have to be computed in a strictly sequential order by using rows distributed among various source processors

Possible way:

redistribute data and compute row i and column i at the same time

OUR SITUATION

Problem and Choice:

- Nonsymmetric matrices
- Fan-out communication organization
- Down-looking computation organization

Difficulties:

- Heavier communication overhead
- Communication buffer limit
- Destination list
- Up-looking used with fan-out requires a big storage buffer or repeated sending of same message.

OUR APPROACH

Adapt ideas from other PDE solving methods, such as

- Domain Decomposition
- Substructuring

to direct sparse solvers

MATRIX FORMULATION

$$\begin{bmatrix} A_{11} & & & & & B_1 \\ & A_{22} & & & & B_2 \\ & & \cdot & & & \cdot \\ & & & \cdot & & \cdot \\ & & & & \cdot & \cdot \\ & & & & A_{pp} & B_p \\ C_1 & C_2 & C_1 & \cdots & C_p & D \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_p \\ x_d \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_p \\ f_d \end{bmatrix}$$

Schur Complement or Capacitance Matrix

$$S = D - \sum_{i=1}^{p} C_i A_{ii}^{-1} B_i$$

$$S x_d = f_d - \sum_{i=1}^{p} C_i A_{ii}^{-1} f_i$$

$$A_{ii} x_i = f_i - B_i x_d \qquad i = 1, \ldots, p$$

MAJOR STEPS

- factoring A_{ii}

$$A_{ii} = L_i U_i$$

- forming Schur Complement S
- factoring S

COMPUTING SCHUR COMPLEMENT

$$S = D - \sum_{i=1}^{p} C_i U_i^{-1} L_i^{-1} B_i$$

- Ordinary Gauss elimination algorithm

$$S = D - \sum_{i=1}^{p} (C_i U_i^{-1})(L_i^{-1} B_i)$$

- Implicit block factorization does not modify C_i matrices

$$S = D - \sum_{i=1}^{p} C_i(U_i^{-1}(L_i^{-1} B_i))$$

Advantages:

- sparsity of C_i matrices never lost
- reduced communication requirements similar to fan-in (next slide)
- static destination information is available from C_i matrices

COMPUTING SCHUR COMPLEMENT (CONTINUED)

Explicitly computing $A^{-1}B$ is too expensive!!!

$$CA^{-1}B = \sum_k \text{col}_k\,(C) * \text{row}_k\,(A^{-1}B)$$

for ($\text{col}_k\,(C) \neq$ null) **do**:

- solve $U^T y_k = e_k$ (triangular system of order $n-k+1$)
- $\text{row}_k\,(A^{-1}B) = y_k^T (L^{-1}B)$

end k loop

- only subdomain boundary layer unknowns have $\text{col}_k(C) \neq$ null, each of which corresponds to one communication with its partial sum (in the fan-in terminology, the modification vector, but it is much shorter here)
- very moderate increase in the computation overhead, which is compensated by the saving in the data structure manipulation for C
- flexible choices of ordering within the k-loop
- independent of local indexing

DATA STRUCTURES USED

- Subdomain equations — sparse
- Schur Complement — dense

ALGORITHMS

- **subdomains**

  up-looking with "fan-in" type communication

- **interface**

  down-looking with fan-out communication

Algorithm Outline

1. Apply up-looking Gauss elimination to subdomain equations

   —— fully parallel

2. Participate in computing Schur Complement with "fan-in" type communication

   —— parallel and synchronized

3. Participate in factoring Schur Complement according to the elimination tree using down-looking with fan-out

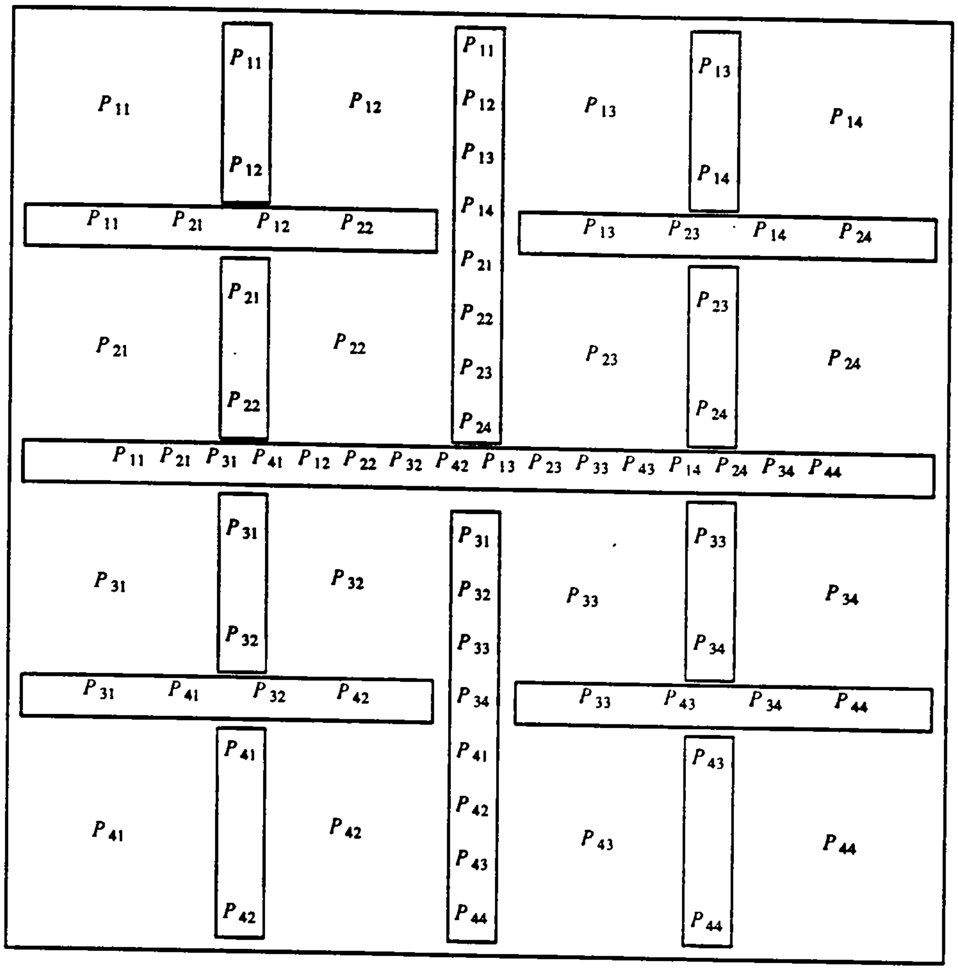
   —— parallel and synchronized

LOAD IMBALANCE

ASSIGNMENT

Equations to Processors

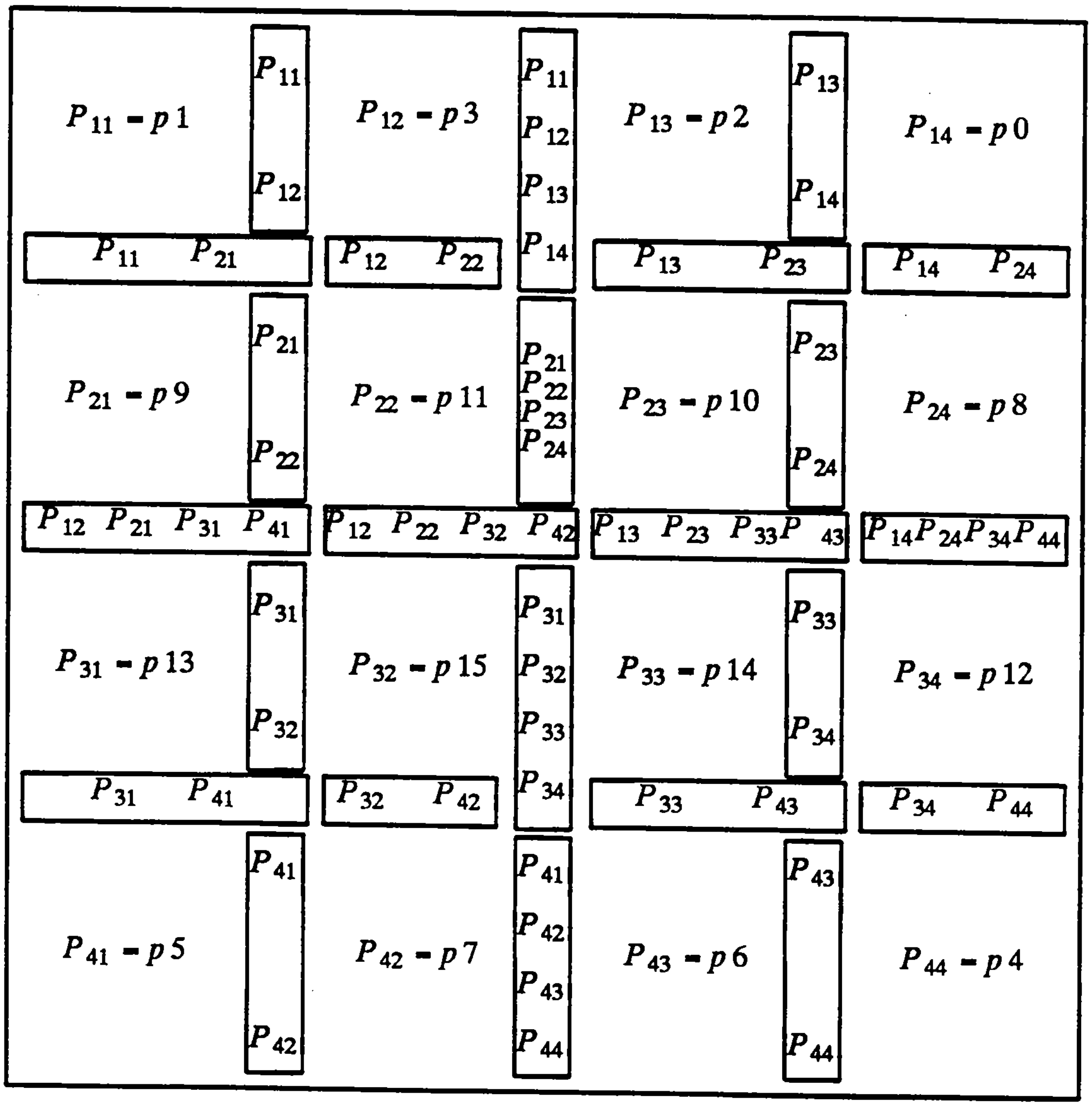
SUBCUBE-SUBTREE (Standard)



Standard subtree-subcube assignment for 16 processors. Within each box unknowns are assigned in wrapping manner to processors shown in the box.

ASSIGNMENT (CONTINUED)

GRID-SUBCUBE-SUBTREE (Grid)



Grid based subtree-subcube assignment for 16 processors. Within the subdomain interfaces we show how the processors are assigned to unknowns in parts of the separators.

PERFORMANCE, 16 PROCESSORS

- on the NCUBE/2

Grid	Sequential time	Parallel time	Speedup
21 × 21	0.578	0.118	4.90
25 × 25	1.05	0.173	6.07
29 × 29	1.77	0.244	7.25
33 × 33	2.73	0.340	8.03
37 × 37	4.03	0.489	8.24
41 × 41	5.69	0.659	8.63
45 × 45	7.73	0.843	9.17
49 × 49	10.23	1.07	9.56
53 × 53	13.21	1.397	9.46
57 × 57	16.78	1.75	9.59
61 × 61	20.87	2.09	9.98
65 × 65	25.67	2.46	10.43

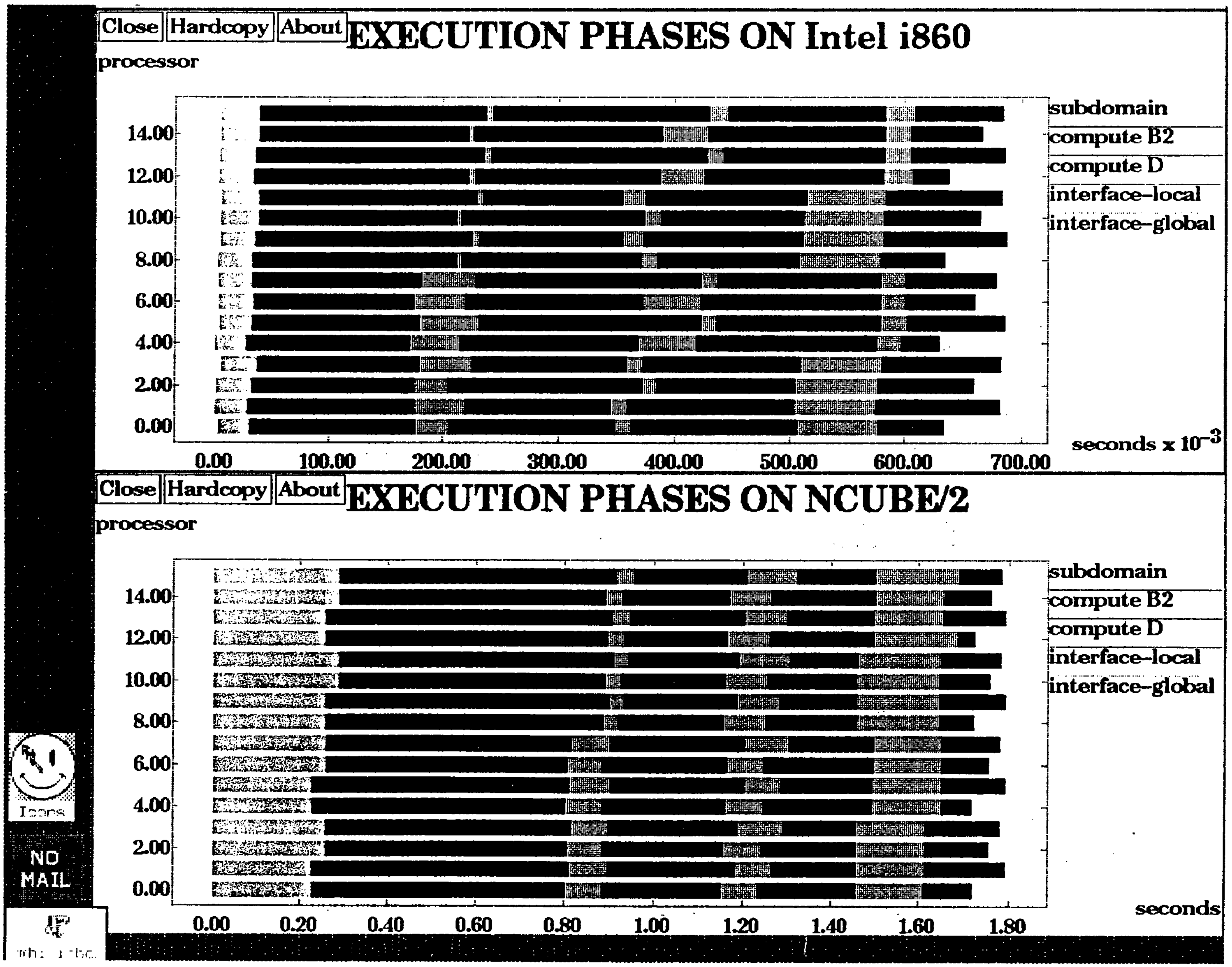
- on the Intel i860

Grid	Sequential time	Parallel time	Speedup
21 × 21	0.071	0.094	xxx
57 × 57	1.87	0.673 (4)	~~2.78~~ 2.91

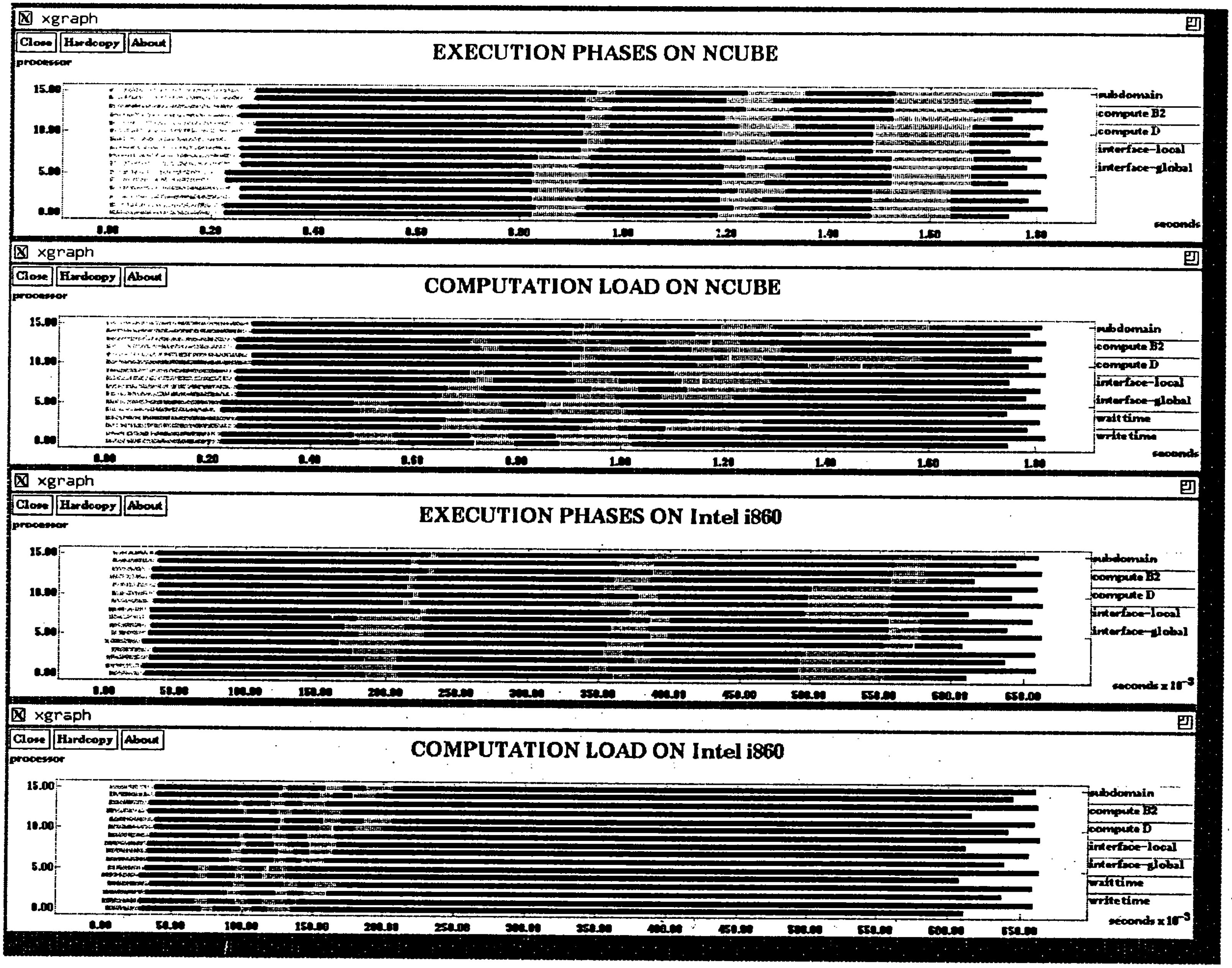
VISUALIZING PERFORMANCE

- subdomain — almost load balanced
- $A^{-1}B$ — very unbalanced
- $CA^{-1}B$ — a lot of idle time
- interface — a lot of synchronization
- sending message — substantial overhead on the Intel i860
- varying grid — similar performance behavior

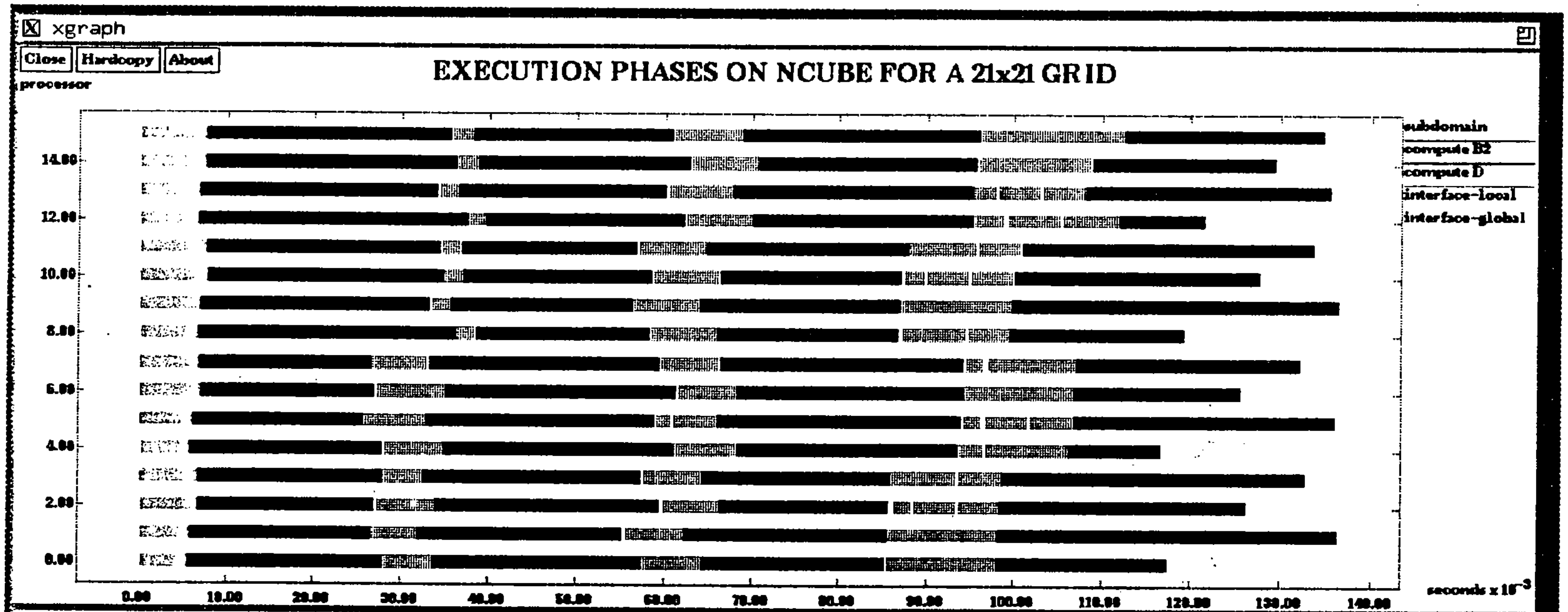
5/x5/ grid
Slide 27a



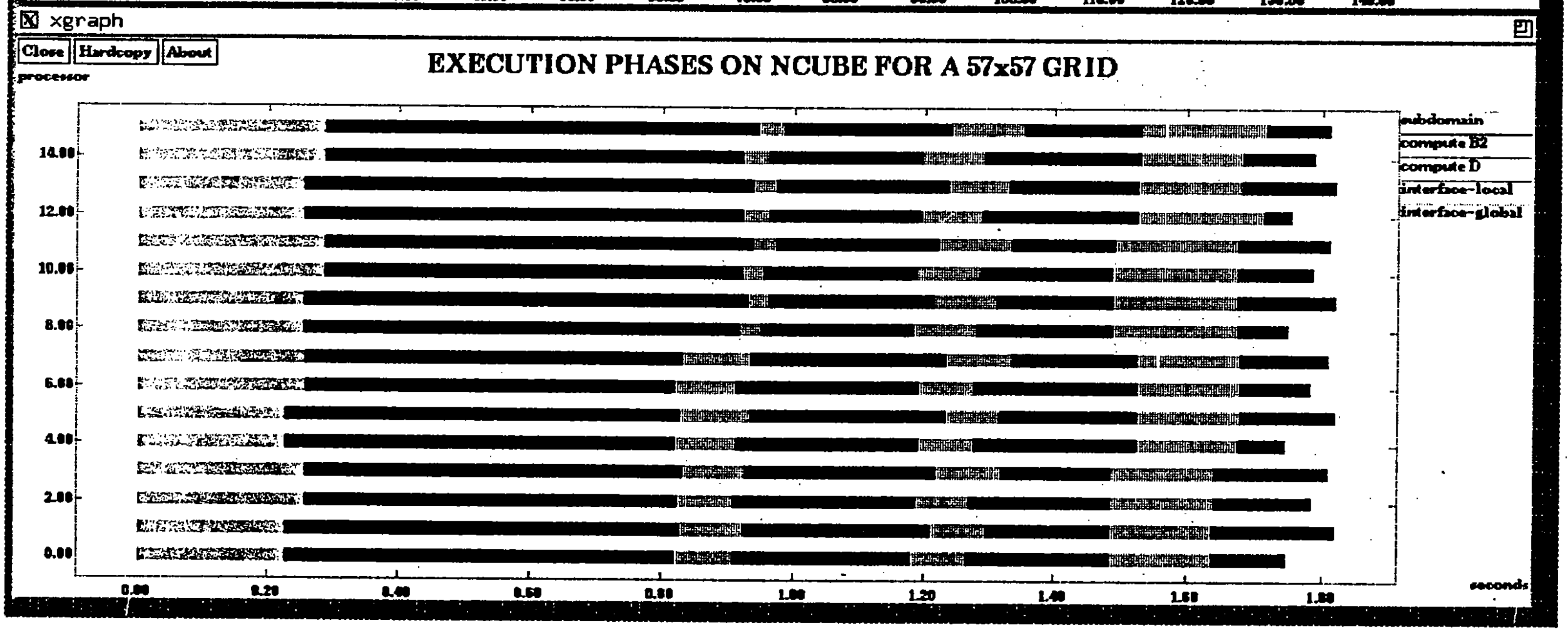
5/x5 / green
Steele 276



Slide 27c



xgraph
Close
Hardcopy
About
EXECUTION PHASES ON NCUBE FOR A 21x21 GRID
processor
subdomain
compute B2
compute D
interface-local
interface-global
seconds x 10^-3



xgraph
Close
Hardcopy
About
EXECUTION PHASES ON NCUBE FOR A 57x57 GRID
processor
subdomain
compute B2
compute D
interface-local
interface-global
seconds

UNSTRUCTURED SCHEDULING

REORGANIZE COMPUTATION AND COMMUNICATION IN FORMING SCHUR COMPLEMENT

To reduce synchronization time, compute rows of $A^{-1}B$ in an order that sends work first to idle processors using the following priorities.

- priority 1 — corner processors:

  P0, P1, P4 and P5

- priority 2 — other border processors:

  P2, P3, P6, P7, P8, P9, P12, P13

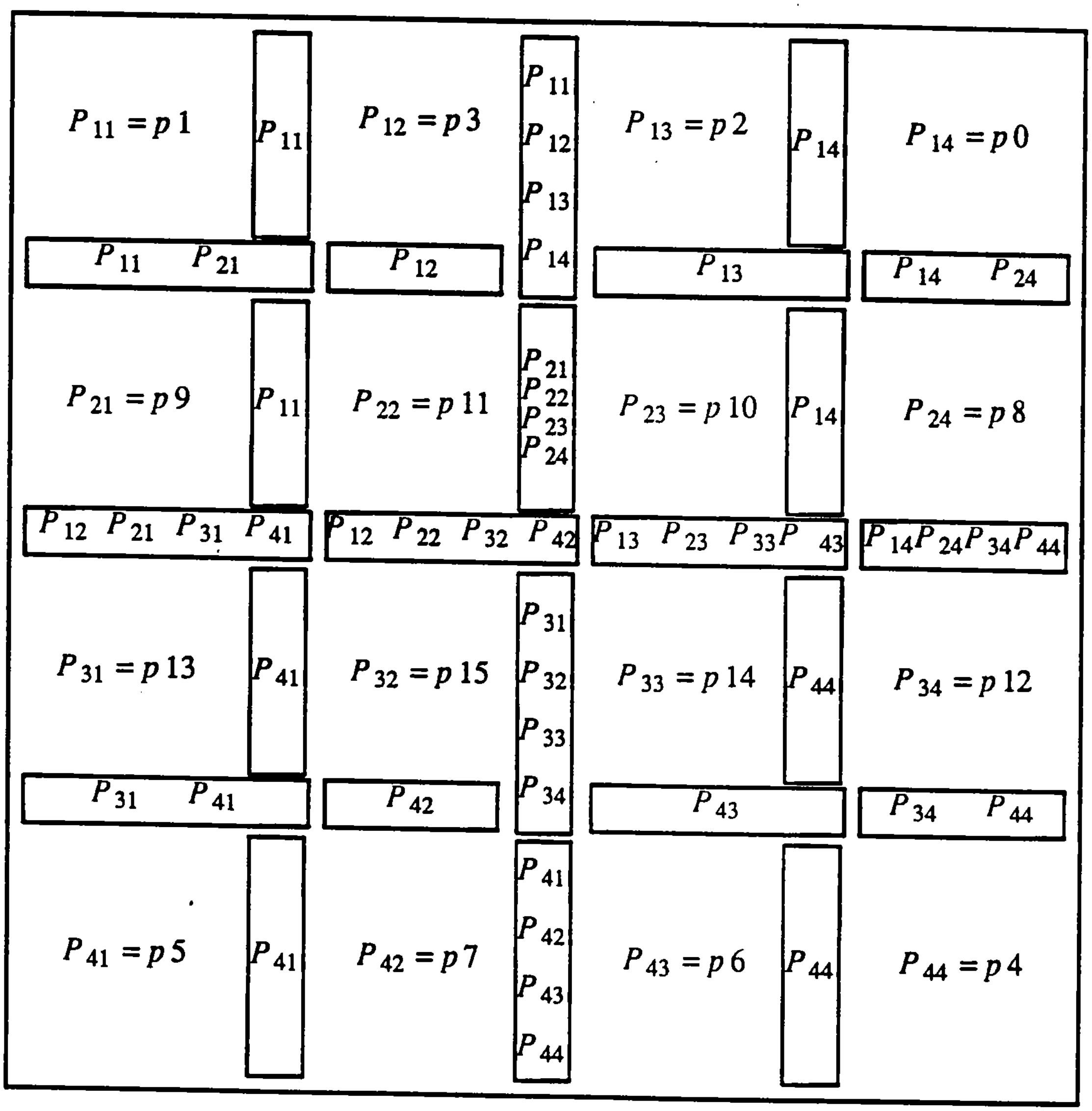
- priority 3 — center processors:

  P10, P11, P14, P15

REASSIGN THE DATA AND TASKS

- move tasks from busy processors to idle processors
- overlap computation and communication

REASSIGNMENT



EFFECTS OF RESCHEDULING

- On the NCUBE/2

  57×57 grid:

parallel time	$1.75 \rightarrow 1.54$
speedup	$9.59 \rightarrow 10.89$

  61×61 grid:

parallel time	$2.09 \rightarrow 1.87$
speedup	$9.98 \rightarrow 11.15$

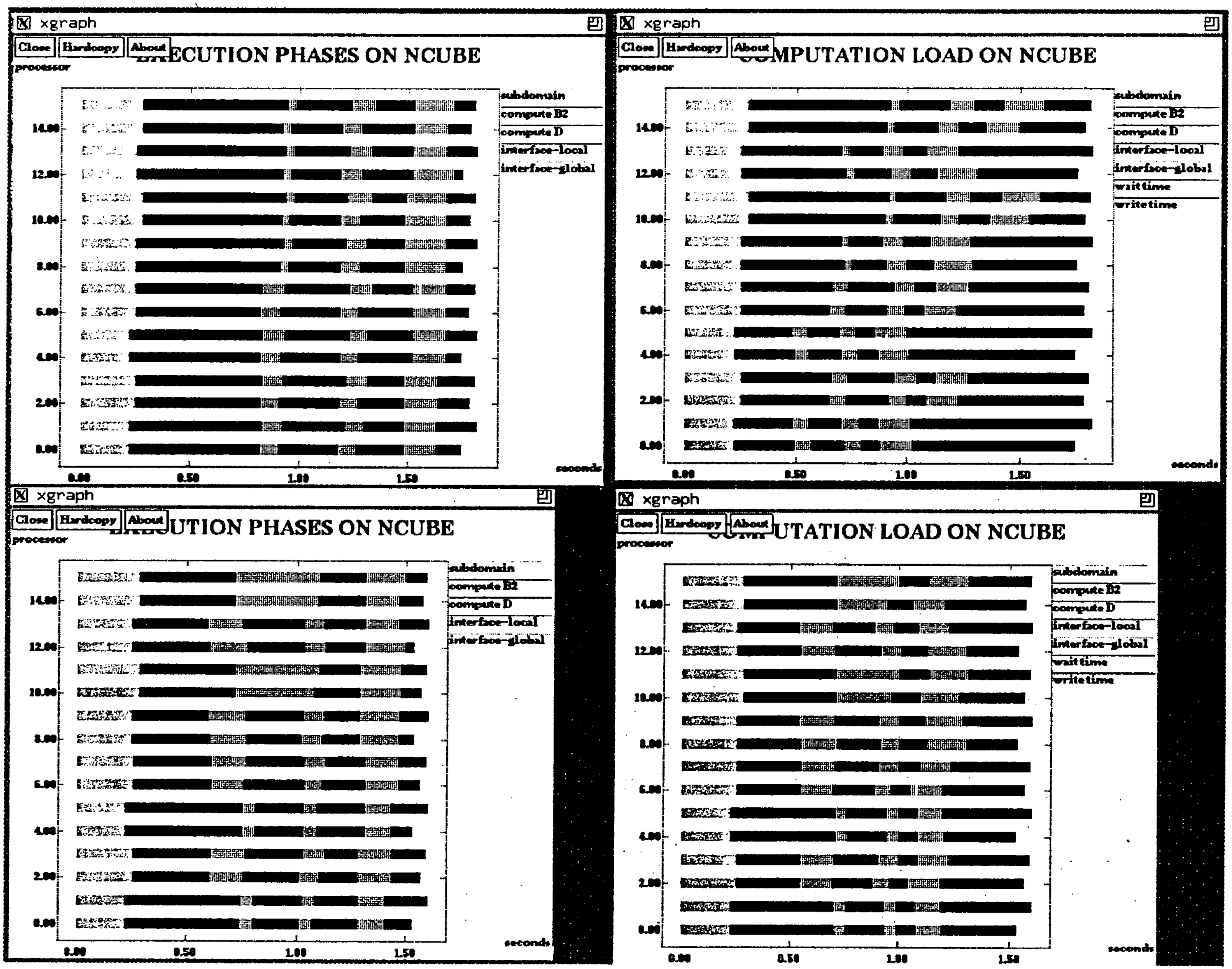
- On the i860

  no improvement

  (a) the effect of communication dominates that of the load imbalance too much

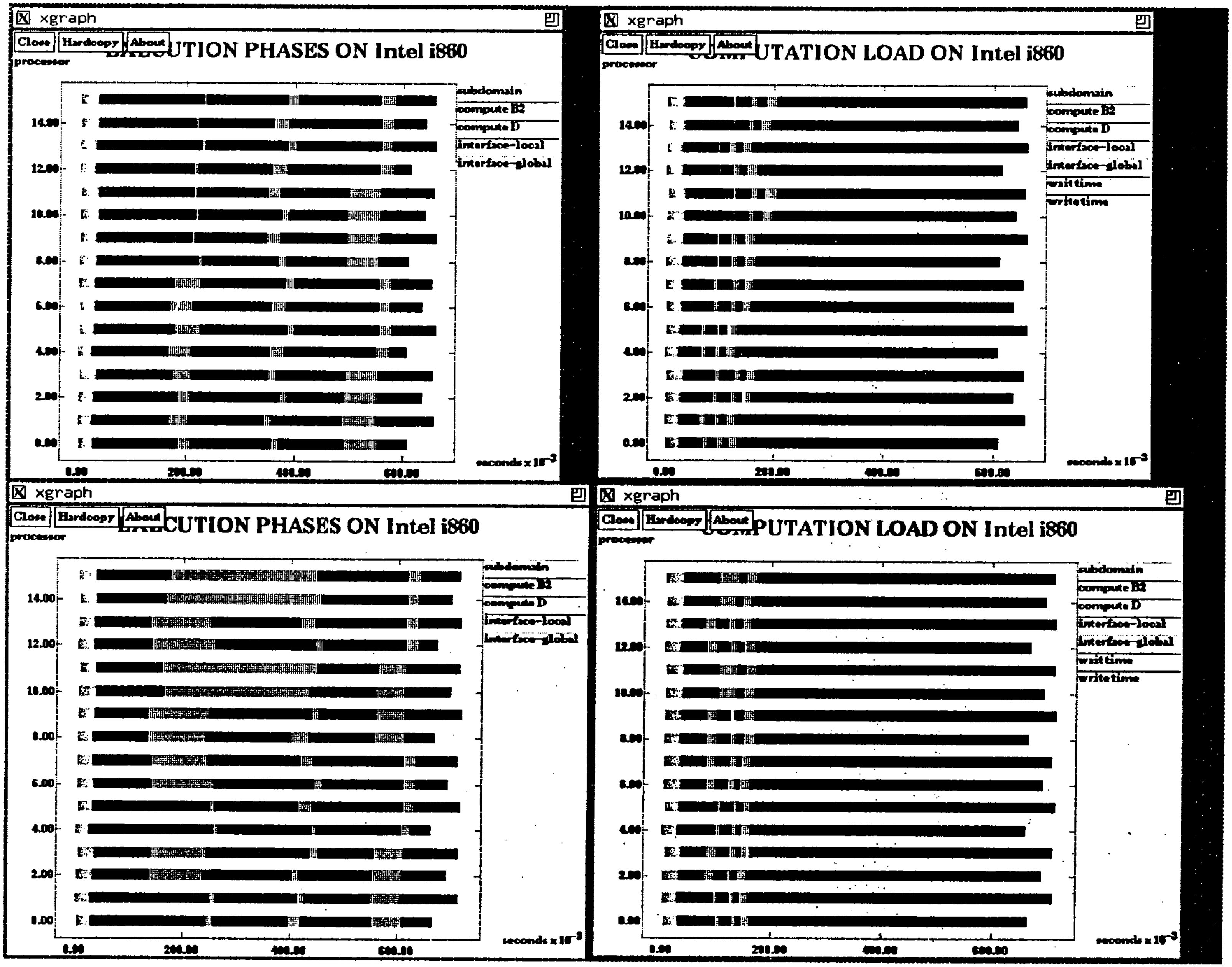
  (b) heavy overhead of sending message

Slide 32a



xgraph
Close
Hardcopy
About
ECUTION PHASES ON NCUBE
processor
subdomain
compute B2
compute D
interface-local
interface-global
seconds
0.00
0.50
1.00
1.50
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
xgraph
Close
Hardcopy
About
MPUTATION LOAD ON NCUBE
processor
subdomain
compute B2
compute D
interface-local
interface-global
wait time
write time
seconds
xgraph
Close
Hardcopy
About
UTION PHASES ON NCUBE
processor
subdomain
compute B2
compute D
interface-local
interface-global
seconds
xgraph
Close
Hardcopy
About
UTATION LOAD ON NCUBE
processor
subdomain
compute B2
compute D
interface-local
interface-global
wait time
write time
seconds

Slide 32b


xgraph
Close
Hardcopy
About
UTION PHASES ON Intel i860
processor
subdomain
compute B2
compute D
interface-local
interface-global
seconds x 10^-3
xgraph
Close
Hardcopy
About
UTATION LOAD ON Intel i860
processor
subdomain
compute B2
compute D
interface-local
interface-global
wait time
write time
seconds x 10^-3
xgraph
Close
Hardcopy
About
CUTION PHASES ON Intel i860
processor
subdomain
compute B2
compute D
interface-local
interface-global
seconds x 10^-3
xgraph
Close
Hardcopy
About
PUTATION LOAD ON Intel i860
processor
subdomain
compute B2
compute D
interface-local
interface-global
wait time
write time
seconds x 10^-3

OPTIMAL SCHEDULINGS

- Very unstructured
- Mutual interactions of load balancing in rescheduling and synchronization in computing S
- Coarse grid analysis

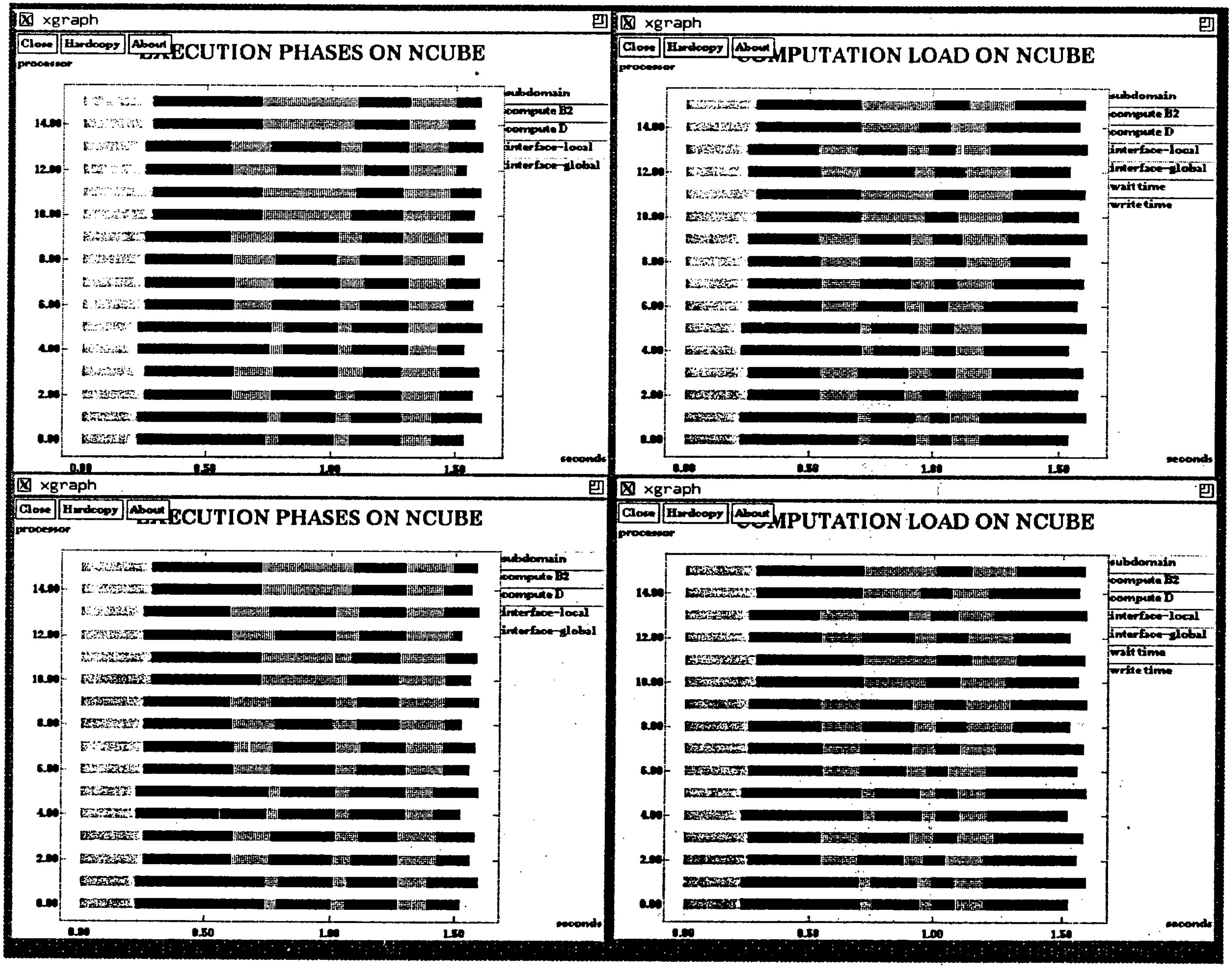
OTHER OPTIMIZATION STRATEGIES

PACKING VS. PIPELINING

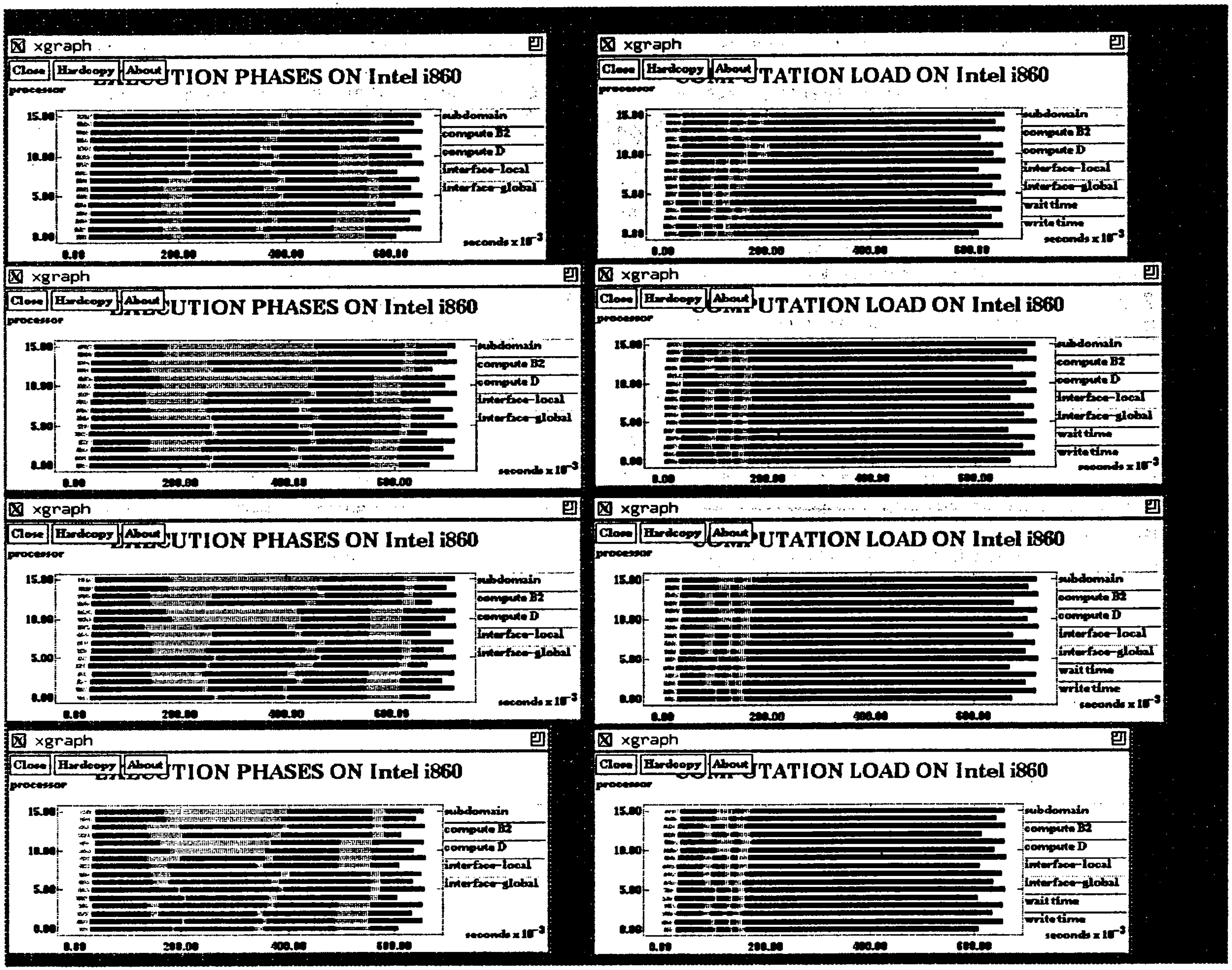
- Pack messages when pipelining is not important

- Trade-off between packing and pipelining by adjusting a grain_control parameter in rescheduling

Slide 35a


xgraph
Close
Hardcopy
About
EXECUTION PHASES ON NCUBE
processor
seconds
subdomain
compute B2
compute D
interface-local
interface-global
xgraph
COMPUTATION LOAD ON NCUBE
wait time
write time

Slide 35b


xgraph
Close
Hardcopy
About
TION PHASES ON Intel i860
processor
15.00
10.00
5.00
0.00
subdomain
compute B2
compute D
interface-local
interface-global
seconds x 10^-3
0.00
200.00
400.00
600.00
xgraph
Close
Hardcopy
About
TATION LOAD ON Intel i860
processor
15.00
10.00
5.00
0.00
subdomain
compute B2
compute D
interface-local
interface-global
wait time
write time
seconds x 10^-3
0.00
200.00
400.00
600.00

OTHER STRATEGIES

- Replace multicast by broadcast when the remaining matrix becomes much denser

- Use irregular grids

CONCLUSIONS

- The parallel PDE sparse solver is load unbalanced with the standard scheduling
- The parallel PDE sparse solver can gain high speedup by reorganizing and overlapping computation and communication using proper schedulings
- The i860 machine is an unbalanced design for many more scientific applications than the NCUBE 2 or Intel iPSC/2