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
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

Slide 8
Computing About Physical Objects
MATRIX PROBLEM

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

(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 $\tilde{B}$ after the level 0 elimination.
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

\[ \begin{array}{c}
\text{L} \\
\text{U}
\end{array} \]

Do everything for an equation when you reach it.

- down-looking

\[ \begin{array}{c}
\text{U} \\
\text{L}
\end{array} \]

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

\[ Q : r_i^q = \sum_{k \in K} \left( \frac{a_{ik}}{a_{kk}} \right) \cdot \text{row}_k \]

\[ = \sum_{k \in K} \left( \frac{a_{ki}}{a_{kk}} \right) \cdot \text{row}_k \quad \text{(if A is symmetric)} \]

\[ P : \text{row}_i = \text{row}_i - r_i^q \]
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
MATRICES FORMULATION

\[
\begin{bmatrix}
A_{11} & B_1 \\
A_{22} & B_2 \\
\vdots & \vdots \\
C_1 & C_2 & C_1 & \ldots & C_p & D
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_p \\
x_d
\end{bmatrix}
= 
\begin{bmatrix}
f_1 \\
f_2 \\
\vdots \\
f_p \\
f_d
\end{bmatrix}
\]

Schur Complement or Capacitance Matrix

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

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

\[
A_{ii} x_i = f_i - B_{i} x_d \quad i = 1, \ldots, p
\]
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
Explicitly computing $A^{-1}B$ is too expensive!!!

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

for ($\text{col}_k(C) \neq \text{null}$) 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 $\text{col}_k(C) \neq \text{null}$, 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
Standard subtree-subcube assignment for 16 processors. Within each box unknowns are assigned in wrapping manner to processors shown in the box.
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

<table>
<thead>
<tr>
<th>Grid</th>
<th>Sequential time</th>
<th>Parallel time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 x 21</td>
<td>0.578</td>
<td>0.118</td>
<td>4.90</td>
</tr>
<tr>
<td>25 x 25</td>
<td>1.05</td>
<td>0.173</td>
<td>6.07</td>
</tr>
<tr>
<td>29 x 29</td>
<td>1.77</td>
<td>0.244</td>
<td>7.25</td>
</tr>
<tr>
<td>33 x 33</td>
<td>2.73</td>
<td>0.340</td>
<td>8.03</td>
</tr>
<tr>
<td>37 x 37</td>
<td>4.03</td>
<td>0.489</td>
<td>8.24</td>
</tr>
<tr>
<td>41 x 41</td>
<td>5.69</td>
<td>0.569</td>
<td>8.63</td>
</tr>
<tr>
<td>45 x 45</td>
<td>7.73</td>
<td>0.843</td>
<td>9.17</td>
</tr>
<tr>
<td>49 x 49</td>
<td>10.23</td>
<td>1.07</td>
<td>9.56</td>
</tr>
<tr>
<td>53 x 53</td>
<td>13.21</td>
<td>1.397</td>
<td>9.46</td>
</tr>
<tr>
<td>57 x 57</td>
<td>16.78</td>
<td>1.75</td>
<td>9.59</td>
</tr>
<tr>
<td>61 x 61</td>
<td>20.87</td>
<td>2.09</td>
<td>9.98</td>
</tr>
<tr>
<td>65 x 65</td>
<td>25.67</td>
<td>2.46</td>
<td>10.43</td>
</tr>
</tbody>
</table>

- on the Intel i860

<table>
<thead>
<tr>
<th>Grid</th>
<th>Sequential time</th>
<th>Parallel time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 x 21</td>
<td>0.071</td>
<td>0.094</td>
<td>xxx</td>
</tr>
<tr>
<td>57 x 57</td>
<td>1.87</td>
<td>0.673</td>
<td>2.78</td>
</tr>
</tbody>
</table>
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
EXECUTION PHASES ON Intel i860

EXECUTION PHASES ON NCUBE/2
UNSTRUCTURED SCHEDULING
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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{11} = p_1$</td>
<td>$P_{12} = p_3$</td>
<td>$P_{13} = p_2$</td>
<td>$P_{14} = p_0$</td>
<td></td>
</tr>
<tr>
<td>$P_{11}$</td>
<td>$P_{12}$</td>
<td>$P_{13}$</td>
<td>$P_{14}$</td>
<td></td>
</tr>
<tr>
<td>$P_{21} = p_9$</td>
<td>$P_{22} = p_{11}$</td>
<td>$P_{23} = p_{10}$</td>
<td>$P_{24} = p_8$</td>
<td></td>
</tr>
<tr>
<td>$P_{12}$</td>
<td>$P_{13}$</td>
<td>$P_{14}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_{31} = p_{13}$</td>
<td>$P_{32} = p_{15}$</td>
<td>$P_{33} = p_{14}$</td>
<td>$P_{34} = p_{12}$</td>
<td></td>
</tr>
<tr>
<td>$P_{12}$</td>
<td>$P_{22}$</td>
<td>$P_{32}$</td>
<td>$P_{42}$</td>
<td></td>
</tr>
<tr>
<td>$P_{31}$</td>
<td>$P_{32}$</td>
<td>$P_{33}$</td>
<td>$P_{34}$</td>
<td></td>
</tr>
<tr>
<td>$P_{41}$</td>
<td>$P_{42}$</td>
<td>$P_{43}$</td>
<td>$P_{44}$</td>
<td></td>
</tr>
</tbody>
</table>

Computing About Physical Objects

Slide 31
EFFECTS OF RESCHEDULING

- On the NCUBE/2

  \[57 \times 57\] grid:

  \[
  \begin{array}{lcl}
  \text{parallel time} & 1.75 \rightarrow 1.54 \\
  \text{speedup} & 9.59 \rightarrow 10.89 \\
  \end{array}
  \]

  \[61 \times 61\] grid:

  \[
  \begin{array}{lcl}
  \text{parallel time} & 2.09 \rightarrow 1.87 \\
  \text{speedup} & 9.98 \rightarrow 11.15 \\
  \end{array}
  \]

- On the i860

  no improvement

  (a) the effect of communication dominates that of the load imbalance too much
  (b) heavy overhead of sending message
EXECUTION PHASES ON NCUBE

COMPUTATION LOAD ON NCUBE
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
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.