SCALABILITY ISSUES IN SPARSE SOLVERS

Manoj Gupta

CS497MH

APRIL 22 1999

Scalability

Definition:

A scalable algorithm is the one which maintains efficiency bounded away from zero as the number of processors (P) is increased and the size of the problem is increased by roughly O(P).

◇ Efficiency $= \frac{Speedup}{P}$.

◇ Quantifying scalability through Isoefficiency analysis:

- How should the problem size increase if we want the same efficiency ? $\rightarrow$ W=W(P)

◇ Who are the devils ??

- Communication costs
- Synchronization costs
- Serial components

◇ Scalability helps in answering:

<u>How Does</u>

- Number of processors
- Problem size
- Computation speed
- Communication speed
- Architecture
- Algorithmic limitations (amount of parallelism, load balance, cache behaviour etc.)

<u>affect the performance of an algorithm.</u>

◇ Defining Some Terms:

- Average flux $= \frac{\sum_{|m| \in M} md(m)}{|L|}\beta$
- Bisection width:

  $\text{flux}(V_0, V_1) = \sum_{src(m),dest(m)} |m|$

bound $= \frac{flux(V_0,V_1)}{sep(V_0,V_1)}\beta$

- Node congestion $= \max \sum_v |m|\beta$
- Edge contention $= \max \sum_\ell |m|\beta$

Mapping columns: Is it scalable ?

- Using Fan-Out Dense Cholesky.
- Mapping columns onto processors cyclically.

$map(j) \ = \ j \ mod \ \ P$

$$
\begin{array}{cccccc}
+ & & & & & \\
+ & + & & & & \\
+ & + & + & & & \\
. & . & . & . & & \\
. & . & . & . & . & \\
+ & + & + & + & + & + \\
\uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\
1 & 2 & . & P & 1 & 2\ldots
\end{array}
$$

- Operation count $= \frac{N^3}{3}\phi$

Minimum time due to parallelism$= \frac{N^3}{3P}\phi$

- Critical Path: cdiv(1)$\rightarrow$ cmod(2,1) $\rightarrow$ cdiv(2).....

cdiv(j) + cmod(j,k) = $(1 + 2)\phi$

Critical path minimum $= 3[N + (N-1) + ... + 1]\phi = \frac{3N^2}{2}\phi$

Minimum execution time $= N^2\phi \ \max(\frac{3}{2}, \frac{N}{3P})$

Machine topology: $\sqrt{P} * \sqrt{P}$ grid of processors

$$\sqrt{P}\ \begin{array}{cccccccc} + & + & + & + & + & + & + & + \\ + & + & + & + & + & + & + & + \\ + & + & + & + & + & + & + & + \\ + & + & + & + & + & + & + & + \\ + & + & + & + & + & + & + & + \\ + & + & + & + & + & + & + & + \\ + & + & + & + & + & + & + & + \\ + & + & + & + & + & + & + & + \\ & & & \sqrt{P} & & & & \end{array}$$

• After cdiv(k), processor map(k) sends column k to all processors $\{map(j)|\ j > k\}$.

◇ Two ways to send the columns

- Individual message casting.
- Multicast through a tree. Root → k.

Individual message

- First column sends N entries, second N-1, third N-3 ...

- Average distance = d(P) = $\frac{2\sqrt{P}}{3}$

- Each element (total elements = $\frac{N^2}{2}$) will travel on average d(P) and each element will travel to atmost P processors.

- Total Flux = $\frac{N^2}{2}(\frac{2\sqrt{P}}{3})P = \frac{N^2}{3}P^{3/2}$

- Total Machine Bandwidth = $\frac{2P}{\beta}$ = Total words machine can exchange in a second.

- Flux per communication link (avg)= Total Flux/ Total Machine Bandwidth = $\frac{1}{6}N^2\sqrt{P}\beta$ seconds.

Multicast

- Every element will travel over P links.

- Total Flux = $\frac{N^2}{2}P$

- Flux per communication link (avg)= Total Flux/ Total Machine Bandwidth = $\frac{1}{4}N^2\beta$ seconds.

Searching for most loaded links: Bisection width

• Bisect the machine into two halves and analyze the traffic through the bisector edges.

$$
\begin{array}{cccccccccc}
 & + & + & + & + & | & + & + & + & + \\
 & + & + & + & + & | & + & + & + & + \\
 & + & + & + & + & | & + & + & + & + \\
 & + & + & + & + & | & + & + & + & + \\
\sqrt{P} & + & + & + & + & | & + & + & + & + \\
 & + & + & + & + & | & + & + & + & + \\
 & + & + & + & + & | & + & + & + & + \\
 & + & + & + & + & | & + & + & + & + \\
 & & & & & \sqrt{P} & & & &
\end{array}
$$

Individual messages

- Each element crosses the bisector P/2 times
- The flux across the bisector = $(\frac{N^2}{4})(\frac{P}{2}) + (\frac{N^2}{4})(\frac{P}{2}) = (\frac{N^2}{4})P\beta$
- There are $\sqrt{P}$ links and therefore per link flux = $(\frac{N^2}{4})\sqrt{P}\beta$

Multicast

- Assuming on average, the tree uses $\sqrt{P}/2$ of the edges in the bisector.

- The flux across the bisector = $(\frac{N^2}{4})(\frac{\sqrt{P}}{2}) + (\frac{N^2}{4})(\frac{\sqrt{P}}{2}) = (\frac{N^2}{4})\sqrt{P}$
- per link flux = $\frac{N^2}{4}\beta$

Show me that column mapping is not scalable:

- Tree Multicast:

  Min. completion time $= max(\frac{N^3\phi}{3P}, \frac{3N^2\phi}{2}, \frac{N^2\beta}{4})$

  If $P > \frac{N\phi}{\beta}$ then efficiency degradation.

- Individual Messages:

  Min. completion time $= max(\frac{N^3\phi}{3P}, \frac{3N^2\phi}{2}, \frac{N^2\sqrt{P}\beta}{4})$

  $\frac{N^2\sqrt{P}\beta}{4}$ term from communication makes this non-scalable.

*Important point to remember: Tree multicasting is more scalable than individual messages.

Wrap mapping of elements

- View the machine as an $P_r * P_c$
- Map elements A_{ij} and L_{ij} to processor $(mapr(i)\ ,\ mapc(j))$
- Assume cyclic mapping: $mapr(i) = i \;\; mod \;\; P_r$ and similarly for mapc(j).

*Important point to remember: Wrap mapping of elements is better than column-wise mapping.

Have i forgotten Sparse matrices?

- Analyze Multifrontal method.
- Subtree-Subcube mapping.
- Block-cyclic mapping of matrices onto processors.
- Subtree-submesh mapping for 2-D architecture.
- Subforest-subcube mapping.

Subtree-subcube mapping

- Assuming an underlying hypercube architecture.
- The top-level supernode columns are mapped cyclically to the whole machine.
- The left subtree is mapped recursively to the left subcube.
- The right subtree is mappped recursively to the right subcube.
- This assignment is to map nodes to processors.
- The Frontal matrix at each node is then again distributed among the processors using either cyclic mapping or either by block-cyclic mapping.

Advantages:

- The communication between columns is now localized into subcubes.
- Less interference, less congestion.
- More load balanced.

Block-cyclic mapping

- The rows and columns are divided into groups.
- These groups assigned in cyclic fashion.

Advantages:

- Good pipelining.
- Less communication overhead due to less number of messages.

Disadvantage:

- Not good load balancing.

Subtree-submesh mapping

- Assuming that underlying architecture is a mesh of processors.
- The top supernode is mapped cyclically to the whole machine.
- The left subtree is mapped to the left half of the machine.
- The right subtree is mapped to the right half of the machine.
- Idea-wise same as subtree-subcube.

Conclusions

- Scalability important for large supercomputers.
- Tree multicasting is more scalable than individual message passing.
- Wrap mapped matrices are more scalable than column mapped matrices.
- For Sparse matrices, we employ subtree-subcube mapping for elimination tree and cyclic or block-cyclic subcube mapping for the dense frontal matrices corresponding to supernodes.