Scalability of Parallel Algorithms for Matrix Multiplication

Presented by: Abhishek Kumar

Motivation

• Analyze the performance and scalability of a number of parallel formulations of the matrix multiplication algorithm
• Predict the conditions under which each formulation is better than the other
Introduction

• Matrix multiplication is used in a variety of application
• Matrix multiplication formulations:
  • Cannon’s algorithm
  • Berntsen’s algorithm
  • DNS algorithm
• Near linear speedups for sufficiently large matrices
• Isoefficiency metric to analyze the scalability
Cannon’s Algorithm

• Two $n \times n$ matrices $A$ and $B$ are divided into square submatrices of size $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ among $p$ processors.

• Data from block $A_{ij}$ is sent to processor $(i, (j+i)\mod\sqrt{p})$. Similarly, block $B_{ij}$ sends data to processor $((i+j)\mod\sqrt{p}, j)$.

• Sub-blocks of $A$ are rolled one step left and $B$ sub-blocks are rolled one step up and multiplied.

• Multiplication of $A$ and $B$ is complete after $\sqrt{p}$ steps.
Cannon’s Algorithm

Total parallel execution time: \( \frac{n^3}{p} + 2t_s \sqrt{p} + 2t_w \frac{n^2}{\sqrt{p}} \)
Berntsen’s Algorithm

• Berntsen proposed a algorithm which exploits greater connectivity of the hypercube.
• $p = 2^{3q}$ processors with $p \leq n^{3/2}$ restriction
• Matrix A is split by columns and matrix B by rows into $2^q$ parts
• Hypercube is split into $2q$ sub-cubes, each performing a submatrix multiplication between submatrices $A \left( \frac{n}{2^{2q}} \times \frac{n}{2^{2q}} \right)$ and $B \left( \frac{n}{2^{2q}} \times \frac{n}{2^{2q}} \right)$ using Cannon’s algorithm
Berntsen’s Algorithm

Total parallel execution time: \[ \frac{n^3}{p} + 2t_s p^{1/3} + \frac{1}{3} t_s \log p + 3t_w \frac{n^2}{p^{2/3}} \]

- Terms associated with both \( t_s \) and \( t_w \) are smaller than Cannon’s algorithm
DNS Algorithm

• Initially, \( p = n^3 = 2^{3q} \) processors

• Completed the task of \( O(n^3) \) matrix multiplication in \( O(\log n) \) time using \( n^3 \) processors

• A proposed variant could work with \( n^2r \) processors \((1 < r < n)\)

• Logical processor array of \( r^3 \) superprocessors is used, each comprising of \((n/r)^2\) hypercube processors

• Multiplication of \((n/r) \times (n/r)\) blocks is performed on \( \frac{n}{r} \times \frac{n}{r} \) subarrays of processors using Cannon’s algorithm
GK Algorithm

• Another scheme to adapt the DNS algorithm to use fewer than $n^3$ processors

• $p = 2^{3q}$ processors with $q < \frac{1}{3} \log n$

• Matrices are divided into sub-blocks of $\frac{n}{2^{2q}} \times \frac{n}{2^{2q}}$ elements

• In this variant of the DNS algorithm, all the single element operations are replaced by sub-block operations

• Total parallel execution time: $\frac{5}{3} t_s p \log p + \frac{5}{3} t_w n^2 p^{1/3} \log p$
Comparison of various algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Total Overhead function</th>
<th>Asymptotic Isoefficiency</th>
<th>Applicability range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cannon’s</td>
<td>$2t_s\sqrt{p} + 2t_w \frac{n^2}{\sqrt{p}}$</td>
<td>$O(p^{1.5})$</td>
<td>$1 \leq p \leq n^2$</td>
</tr>
<tr>
<td>Berntsen’s</td>
<td>$2t_s p^{1/3} + \frac{1}{3} t_s \log p + 3t_w \frac{n^2}{p^{2/3}}$</td>
<td>$O(p^2)$</td>
<td>$1 \leq p \leq n^{3/2}$</td>
</tr>
<tr>
<td>DNS</td>
<td>$(t_s + t_w) \left(\frac{5}{3} p \log p + 2n^3\right)$</td>
<td>$O(p \log p)$</td>
<td>$n^2 \leq p \leq n^3$</td>
</tr>
<tr>
<td>GK</td>
<td>$\frac{5}{3} t_s p \log p + \frac{5}{3} t_w n^2 p^{1/3} \log p$</td>
<td>$O(p (\log p)^3)$</td>
<td>$1 \leq p \leq n^3$</td>
</tr>
</tbody>
</table>

- Only asymptotic scalabilities
- None of the algorithms is strictly better than others for all possible problem sizes and number of processors
- Compare pairwise the total overhead functions – GK vs Cannon’s:
  - $t_s$ term for GK is always smaller than Cannon’s
  - Even if $t_s=0$, the $t_w$ for GK becomes smaller than Cannon’s for $p > 130$ million (irrespective of $n$)
  - For reasonable values of $t_s$, GK performs better than Cannon’s for very practical values of $p$ and $n$
  - $n = \sqrt[3]{\frac{(5/3p \log p - 2 p^{3/2})t_s}{t_w - 5/3 p^{1/3} \log p}}$
Comparison of various algorithms

• Regions marked ‘x’ is where non of the algorithms apply and $p>n^3$

• Region ‘a’ (GK), ‘b’ (Brentsen’s), ‘c’ (Cannon’s), ‘d’ (DNS algorithm)
Conclusion

• Scalability analysis provides insights into relative superiority under different conditions
• Predict the condition under which each formulation outperforms the other
• Can be used by smart preprocess/compiler based on different parameters
• Small expression of communication overload ≠ best algorithm
  • Berntsen’s algorithm with the least communication overhead is the least scalable with $O(p^2)$ isoefficiency