PRAM study of Linear Algebra Algorithms

Small (and fun) sequential observation

- Without parallelizing the Matrix – Matrix product, you can gain A LOT OF runtime by optimization of the 3 loops.

Solving a system of linear equations

- You want to solve $M \times x = y$, where $M$ is a $n \times n$ matrix.
  - And let us suppose that there is a unique solution.
- Use LU factorization
  - By Gaussian elimination, without pivoting.
    - for $k = 0; k < n-2; k++$
      - $M[k][k] = M[k][k] / M[k][k];$
      - for $i = k+1; i < n-1; i++$
        - $M[i][k] = M[i][k] + M[i][k] \times M[k][k];$
      - for $i = k+1; i < n-1; i++$
        - $M[i][i] = M[i][i] + M[i][k] \times M[k][i];$
  - In the end, $M$ is LU factorized.

Basic Linear Operations

- Scalar product: 2 input vectors $x,y$ of size $n$.
  - $res := 0$
    - for $i=0; i<n; i++$
      - $res = res + x[i] \times y[i];$
  - Matrix – Vector product
    - $res[i] := 0$
      - for $i=0; i<n; i++$
        - $res[i] = res[i] + M[i][i] \times y[i];$
  - Matrix – Matrix product
    - $res[i][j] := 0$
      - for $i=0; i<n; i++$
        - for $j=0; j<n; j++$
          - $res[i][j] = res[i][j] + M[i][i] \times M[k][j];$

LU factorization by D&C

- You want $M = L \times U$. Let us decompose this matrix product by blocks of size $(n/2) \times (n/2)$.
  - But then,
    - $M_1 = L_1 \times U_1$
    - $M_2 = L_2 \times U_1$
    - $M_3 = L_1 \times U_2$
    - $M_4 = L_2 \times U_2$
    - $M_5 = L_1 \times U_3$
    - $M_6 = L_2 \times U_3$
    - $M_7 = L_1 \times U_4$
    - $M_8 = L_2 \times U_4$
    - $M_{12} = L_1 \times U_{12}$
    - $M_{13} = L_2 \times U_{12}$
    - $M_{20} = L_2 \times U_{12}$
    - $M_{30} = L_1 \times U_{12}$

So how do you do this in parallel?

- Scalar product
  - Actually, it is a simple application of the sum of $n$ elements!
    - $T_{seq}(n) = \Theta(\log n)$, $P(n) = n^2 \Theta(\log n)$. Optimal.
- Matrix x Vector
  - The algorithm is trivially parallel: just compute the $n \times n$ components of $res$ in parallel.
    - Each one is a scalar product!
    - $T_{seq}(n) = \Theta(\log n)$, $P(n) = n^2 \Theta(\log n)$. Optimal.
- Matrix x Vector
  - The algorithm is trivially parallel: just compute the $n \times n$ components of $res$ in parallel.
    - Each one is a scalar product!
    - $T_{seq}(n) = \Theta(\log n)$, $P(n) = n^2 \Theta(\log n)$. Optimal.
The D&C algorithm

1. A LU factorization of size n/2 provides $L_{i/2}$ and $U_{i/2}$
2. Then, you have to invert 2 n/2 matrices ($U_{i/2}$ and $L_{i/2}$)
3. Then, with 2 matricial products, you get $L_i$ and $U_{i/2}$
4. Then, you can form the new matrix $M_{i/2}$: $L_{i/2} \times L_{i/2}$
   - One more matrix product, and a sum (subtraction).
5. Finally, one last LU factorization of this matrix yields $L_{i/2}$ and $U_{i/2}$.
   - And then you have all $L$ and all $U$.

\[
M_{i/2} = L_{i/2} \times U_{i/2} \\
L_{i/2} = \frac{1}{2} L_{i/2} \\
U_{i/2} = \frac{1}{2} U_{i/2} \\
\begin{pmatrix}
M_{i} = (L_{i/2} \times U_{i/2}) + (L_{i/2} \times U_{i/2})
\end{pmatrix}
\]

Parallele Programmierung
Nicolas Maillard, Marcus Ritt

PRAM Complexity

- Let $T_{LU}(n)$ be the parallel runtime ($T_{PRAM}(n)$) of the LU factorization of a matrix $n \times n$.
- $M_{i/2} = L_{i/2} \times U_{i/2}$
- $L_{i}$ and $U_{i/2}$
- $M_{i} = L_{i/2} \times U_{i/2}$
- $L_{i}$ and $U_{i/2}$

\[
T_{LU}(n) = T_{LU}(n/2) + \text{Inv}(n/2) + \text{Mul}(n/2) + \text{Mul}(n/2) + 1 + T_{LU}(n/2) = 2T_{LU}(n/2) + \text{Mul}(n/2) + \text{Inv}(n/2) + 1.
\]

- Where:
  - $\text{Mul}(n) = \log(n)$ (with $P(n) = n^2 \log n$ processors)
  - $\text{Inv}(n)$ is the parallel runtime to invert a triangular matrix of size $n$.

Parallele Programmierung
Nicolas Maillard, Marcus Ritt

Triangular Inversion

- So what is $\text{Inv}(n)$?
- You have $L$, triangular inferior, and want $T$ such that $LT = \text{Id}$:

\[
\begin{pmatrix}
L_{11} & (0) \\
0 & T_{22}
\end{pmatrix}
\begin{pmatrix}
L_{11} & (0) \\
0 & T_{22}
\end{pmatrix}
\begin{pmatrix}
T_{11} & T_{12} \\
T_{21} & T_{22}
\end{pmatrix}
\]

- But then,

\[
L_{11} = L_{11} \times T_{11}
\begin{pmatrix}
(0) \\
L_{12}
\end{pmatrix}
\begin{pmatrix}
(0) \\
L_{21}
\end{pmatrix}
L_{22} = L_{22} \times T_{22}
\]

- $T_{12} = L_{12} \times T_{21}$
- $T_{21} = L_{21} \times L_{11} \times T_{21}$

Parallele Programmierung
Nicolas Maillard, Marcus Ritt

PRAM complexity of the triangular inversion

- Then:

\[
\begin{align*}
\text{Inv}(n) = & \text{Inv}(n/2) + 2\text{ Mul}(n/2) + \text{Inv}(n/2) + \log(n) \\
= & \ldots + \text{Inv}(n/2) + \log(n) + \log(n) + \log(n) + \ldots + \log(n/2) \\
= & \log(n) + \log(n) + \log(n) \\
= & \log(n) / 2 \text{ for } k = \log(n). \\
\end{align*}
\]

- $P_{\text{pram}}(n) = \max \{ 2P_{\text{pram}}(n/2), P_{\text{pram}}(n/2) \}$

- The algorithm if efficient, but not optimal.

Parallele Programmierung
Nicolas Maillard, Marcus Ritt

Coming back to the LU factorization...

\[
\begin{align*}
\text{LU}(n) & = 2\text{ LU}(n/2) + 2\log(n) + \log^3 n + 1 \\
& \leq 2\text{ LU}(n/2) + 3\log^3 n \\
& \leq \ldots + 2\text{ LU}(n/2) + 3 \cdot (c_{\text{add}} + 2 \log^2 (n/2)) \\
& \text{for whatever } k \geq \log(n). \\
\end{align*}
\]

- Since $\log^2 (n/2) \leq \log^2 n$, the sum is less than $\log^2 n \times #\text{add} = (2^{k-1} - 1)\log^2 n = (2n-1)\log^2 n$, for $k = \log n$

- So, $\text{LU}(n) = O(n + 3n\log^2 n) = O(n\log^3 n)$

- Number of processors?

\[
\begin{align*}
P_{\text{pram}}(n) & = \max \{ P_{\text{pram}}(n/2), 2P_{\text{pram}}(n/2), 2P_{\text{pram}}(n/2), n^2 \} \\
& = \max \{ P_{\text{pram}}(n/2), n^3\log n, n^2 \} \\
& = O(n^3\log n)
\end{align*}
\]

- Conclusion: $C(n) = O(n\log n)$. The algorithm is not efficient.

Parallele Programmierung
Nicolas Maillard, Marcus Ritt

Conclusion about PRAM complexity

- Enables a quantification of how much parallel an algorithm is.
  - Scalar product, matrix product is very parallel and efficient.
  - LU factorization is accelerated by parallelism, but does not show as much parallelism as other algorithms.

- However, some parameters are not captured by the PRAM model:
  - Impact of the distribution of the data on the runtime?
  - What if the algorithm really accesses a lot the memory, including non-shared address spaces?

- The next lecture will give some examples to address these limitations.

Parallele Programmierung
Nicolas Maillard, Marcus Ritt