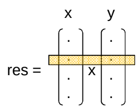
PRAM study of Linear Algebra Algorithms

Basic Linear Operations

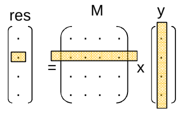
- **Scalar product: 2 input vectors x,y of size n.**

```
res := 0
for (i=1 ; i<=n ; i++)
   res = res + x[i] * y[i]
```



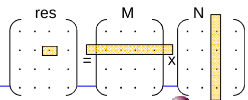
- **Matrix – Vector product**

```
res[:] = 0
for (i=1 ; i<=n ; i++)
  for (j=1 ; j<=n ; j++)
        res[i] = res[i] + M[i][j] * y[j]
```



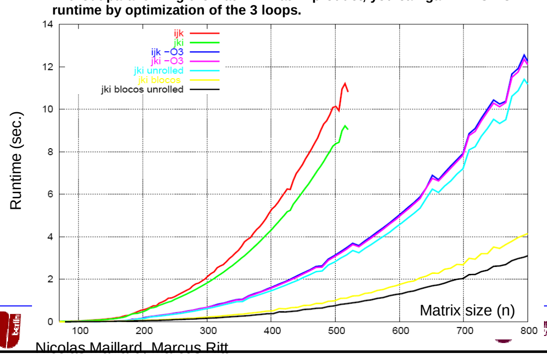
- **Matrix – Matrix product**

```
res[:][:] := 0
for (i=1 ; i<=n ; i++)
  for (j=1 ; j<=n ; j++)
    for (k=1 ; k<=n ; k++)
       res[i][j] = res[i][j] + M[i][k] * N[k][j]
```



Small (and fun) sequential observation

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

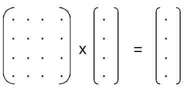


So how do you do this in parallel?

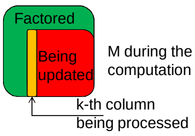
- **Scalar product**
  - **Actually, it is a simple application of the sum of n elements!**
  - **$T_{par}(n) = \theta(\log n)$, $P(n) = n/\log n$. Optimal.**
- **Matrix x Vector**
  - **The algorithm is trivially parallel: just compute the n components of 'res' in parallel.**
    - » **Each one is a scalar product!**
  - **$T_{par}(n) = \theta(\log n)$, $P(n) = n^2/\log n$. Optimal.**
- **Matrix x Vector**
  - **The algorithm is trivially parallel: just compute the n^2 components of 'res' in parallel.**
    - » **Each one is a scalar product!**
  - **$T_{par}(n) = \theta(\log n)$, $P(n) = n^3/\log n$. Optimal.**

Solving a system of linear equations

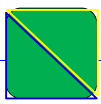
- **You want to solve Mx = y, where M is a n x n matrix.**
  - **And let us suppose that there is a unique solution.**
- **Use LU factorization**
  - **By Gaussian elimination, without pivoting.**




```
for (k = 0 ; k <= n-2; k++) {
   for (i = k+1 ; i <= n-1 ; k++)
      M[i][k] = M[i][k]  / M[k][k];
   for (j = k+1; j <= n-1; k++)
      for (i=k+1 ; i<=n-1; i++)
         M[i][j] = M[i][j] + M[i][k]*M[k][j];
```

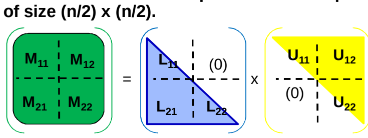


  - **In the end, M is LU factorized.**



LU factorization by D&C

- **You want M = LU. Let us decompose this matrix product by blocks of size (n/2) x (n/2).**



- **But then,**

$$\begin{cases} M_{11} = L_{11}U_{11} \\ M_{21} = L_{21}U_{11} \\ M_{12} = L_{11}U_{12} \\ M_{22} = L_{21}U_{12} + L_{22}U_{22} \end{cases} \quad \text{i.e.} \quad \begin{cases} M_{11} = L_{11} \times U_{11} \\ L_{21} = M_{21}\, U_{11}^{-1} \\ U_{12} = L_{11}^{-1}\, M_{12} \\ (M_{22} - L_{21}U_{12}) = L_{22}U_{22} \end{cases}$$

The D&C algorithm

1. A LU factorization of size n/2 provides L_{11} and U_{11}
2. Then, you have to invert 2 n/2 matrixes (U_{11} and L_{11})
3. Then, with 2 matricial products, you get L_{21} and U_{12}.
4. Then, you can form the new matrix $M_{22} - L_{21}U_{12}$.
   – One more matrix product, and a sum (substraction).
5. Finally, one last LU factorization of this matrix yields L_{22} and U_{22}.
   – And then you have all L and all U.

$$\begin{cases} M_{11} = L_{11} \times U_{11} \\ L_{21} = M_{21}\, U_{11}^{-1} \\ U_{12} = L_{11}^{-1}\, M_{12} \\ (M_{22} - L_{21}U_{12}) = L_{22}U_{22} \end{cases}$$

PRAM Complexity

- Let LU(n) be the parallel runtime ($T_{par}(n)$) of the LU factorization of a matrix n x n.

$$LU(n) = \overset{M_{11} = L_{11} \times U_{11}}{LU(n/2)} + \overset{L_{11}^{-1} \text{ and } U_{11}^{-1}}{Inv(n/2)} + \overset{L_{21} \text{ and } U_{12}}{Mul(n/2)} + \overbrace{Mul(n/2) + 1}^{M_{22} - L_{21}U_{12}} + \overset{L_{22} \text{ and } U_{22}}{LU(n/2)}$$

$$= 2\, LU(n/2) + 2Mul(n/2) + Inv(n/2) + 1.$$

- Where:
  – $Mul(n) = \log(n)$ (with $P(n) = n^3/\log n$ processors)
  – $Inv(n)$ is the parallel runtime to invert a triangular matrix of size n.

Triangular Inversion

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

$$\begin{pmatrix} I_{n/2} & (0) \\ (0) & I_{n/2} \end{pmatrix} = \begin{pmatrix} L_{11} & (0) \\ L_{21} & L_{22} \end{pmatrix} \times \begin{pmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{pmatrix}$$

- But then,

$I_{n/2} = L_{11} \times T_{11}$
$(0) = L_{21}T_{11} + L_{22}T_{21}$
$(0) = L_{11}T_{12}$
$I_{n/2} = L_{21}T_{12} + L_{22}T_{22}$

i.e.

$$\begin{cases} I_{n/2} = L_{11} \times T_{11} \\ T_{12} = (0) \\ I_{n/2} = L_{22} \times T_{22} \\ T_{21} = - L_{22}^{-1} L_{21}^{-1} L_{11}^{-1} \end{cases}$$

PRAM complexity of the triangular inversion

- Then:

$Inv(n) = Inv(n/2) + 2\, Mul(n/2) = Inv(n/2) + \log(n)$
$= \dots = Inv(n/2^k) + \log(n) + \log(n/2) + \dots + \log(n/2^k)$
$= k\log(n) - k(k+1)/2$
$= \theta(\log^2 n)$, for $k = \log(n)$.
$P_{inv}(n) = \max \{ 2P_{inv}(n/2), P_{mul}(n/2) \}$
$= \max \{ 2P_{inv}(n/2), n^3/\log n \} = O(n^3/\log n)$

- The algorithm if efficient, but not optimal.

$$\boxed{\begin{cases} I_{n/2} = L_{11} \times T_{11} \\ T_{12} = (0) \\ I_{n/2} = L_{22} \times T_{22} \\ T_{21} = - L_{22}^{-1} L_{21}^{-1} L_{11}^{-1} \end{cases}}$$

Coming back to the LU factorization...

$LU(n) = 2\, LU(n/2) + 2\log(n) + \log^2 n + 1$
$\leq 2\, LU(n/2) + 3 \log^2 n$
$\leq \dots.\ 2^k\, LU(n/2^k) + 3 \times (\sum_{i=0..k} 2^i \log^2 (n / 2^i))$ for whatever $k \leq \log(n)$.

- Since $\log^2 (n / 2^i) \leq \log^2 n$, the sum is less than
  $\log^2 n \times \sum_{i=0..k} 2^i = (2^{k+1} - 1)\log^2 n = (2n-1) \log^2 n$, for $k = \log n$
- So, $LU(n) = O(n + 3n \log^2 n) = O(n \log^2 n)$
- Number of processors?
  – $P_{LU}(n) = \text{Max}\{ P_{LU}(n/2) , 2\, P_{inv}(n/2), 2P_{mul}(n/2), n^2 \}$
  $= \text{Max}\{P_{LU}(n/2) , n^3/\log n , n^2 \}$
  $= O(n^3/\log n)$
- Conclusion: $C(n) = O(n^4 \log n)$. The algorithm is not efficient.

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 parallelim, 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.