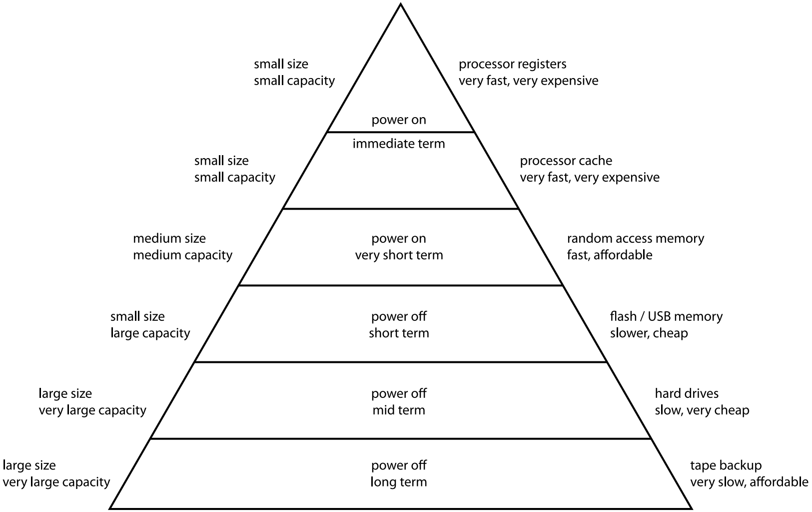
Programming (Econometrics)

Lecture 3: Memory organization

Tommi Tervonen

Econometric Institute, Erasmus School of Economics

Computer Memory Hierarchy



Memory organization

- Local variables such as loop counters can possibly be stored in registers
- All larger data structures have to be allocated to the main memory
- The random access memory is linear and addressed using integers pointing out the location (e.g. 0x400345CF)
- 32 bit adrressing = max 4Gb of memory

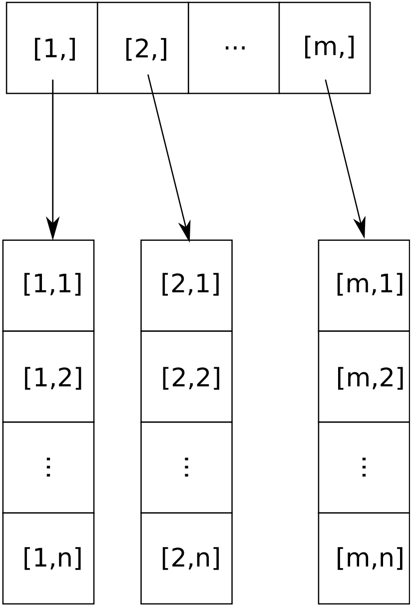
Matrices in Matlab

- Matrices are included in Matlab as a built-in data type

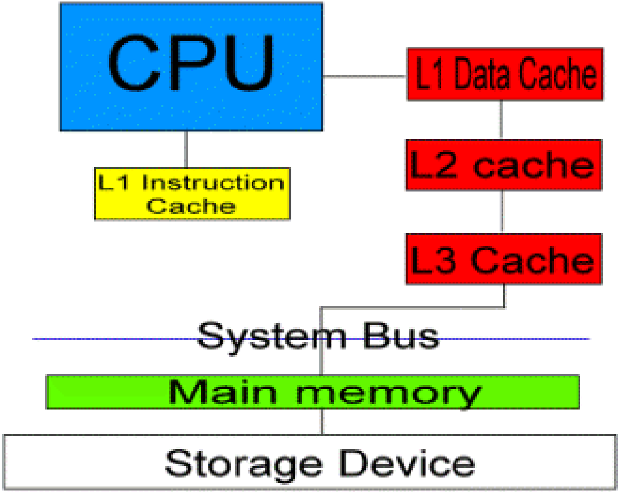
```
a = [3, 4];
b = '1';
c = a*b; % what's c now?
```

- How to represent $m \times n$ matrices?

Matrix representations: naive



CPU caches


CPU
L1 Data Cache
L1 Instruction Cache
L2 cache
L3 Cache
System Bus
Main memory
Storage Device

Matrix representations: efficient

- Memory is linear, so store the element $[a, b]$ in index $[(a-1)*n+b]$

1	2	$\cdots$	n	(n+1)	(n+2)	$\cdots$	(m*n)

- Row-major representation; in column-major one $[a, b]$ is in $[(b-1)*m+a]$
- In most programming languages the array indices start from 0 and the formulas are simpler

Row- and column-major representations: an example

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- As row-major: [1 2 3 4 5 6]
- As column-major: [1 4 2 5 3 6]

Special matrices: sparse

- If the matrix if *sparse*, i.e. it contains only a few elements, it is more efficient to store only the non-zero elements

- E.g.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Can be represented with ([3, 4, 2], [5, 1, 1])

Special matrices: diagonal and identity

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

- Can be represented with [1, 3, 2, 7, 4]

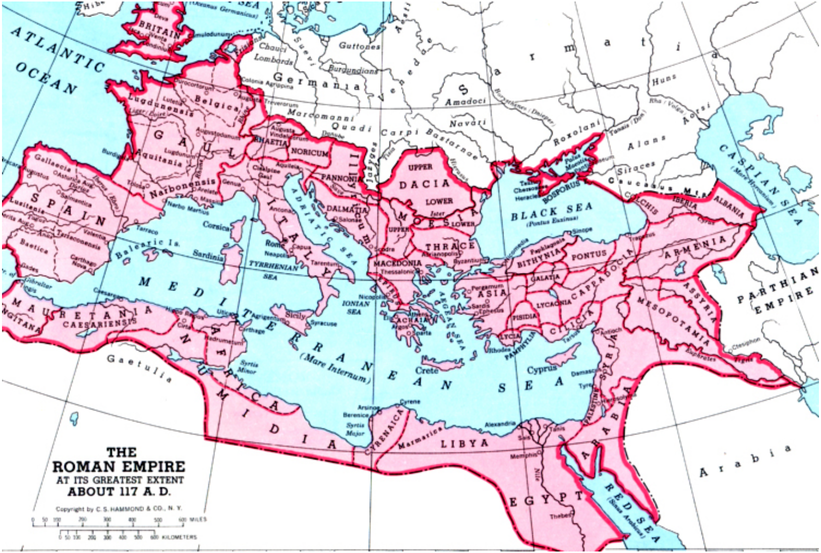
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- $= I_5$ and can be represented with a single integer 5

Matrix multiplication: naive

```
function C = multiply(A, B)
  C = zeros(rows(A), columns(B));
  for (i=1:rows(A))
    for (j=1:columns(B))
      s = 0;
      for (k=1:columns(A))
        s = s + A(i, k) * B(k, j);
      end
      C(i, j) = s;
    end
  end
end
```

- Complexity?


THE ROMAN EMPIRE
AT ITS GREATEST EXTENT
ABOUT 117 A. D.
Copyright by C. S. HAMMOND & CO., N. Y.
0 50 100 200 300 400 500 600 MILES
0 50 100 200 300 400 500 600 KILOMETERS
ATLANTIC OCEAN
BRITAIN
Chauci
Lombards
Suevi
Guttones
Burgundians
Germania
Venedae
Sarmatia
Amadoci
Navari
Huns
Alans
Siraces
Roxolani
Bastarnae
Carpi
Quadi
Marcomanni
Colonia Agrippina
Lugdunensis
Belgica
GAUL
Aquitania
Narbonensis
Gallaecia
SPAIN
Lusitania
Tarraconensis
Baetica
Balearic Is.
Corsica
Sardinia
RHAETIA
NORICUM
PANNONIA
ILLYRICUM
DALMATIA
ITALY
ADRIATIC SEA
TYRRHENIAN SEA
IONIAN SEA
Rome
Neapolis
Sicily
Syracuse
UPPER
DACIA
LOWER
MOESIA
UPPER
LOWER
THRACE
MACEDONIA
ACHAIA
AEGEAN SEA
Crete
BLACK SEA
(Pontus Euxinus)
BOSPORUS
BITHYNIA
PONTUS
GALATIA
CAPPADOCIA
ASIA
LYCAONIA
PISIDIA
LYCIA
PAMPHYLIA
CILICIA
Cyprus
ARMENIA
IBERIA
ALBANIA
COLCHIS
CASPIAN SEA
ASSYRIA
MESOPOTAMIA
PARTHIAN EMPIRE
SYRIA
PALESTINE
ARABIA
Arabia
MEDITERRANEAN SEA
(Mare Internum)
MAURETANIA CAESARIENSIS
NUMIDIA
AFRICA
Gaetulia
Carthage
CYRENAICA
Cyrene
LIBYA
Alexandria
Memphis
EGYPT
Nile
Thebes
RED SEA
(Sinus Arabicus)

Matrix multiplication: divide-and-conquer

- Assume that we are multiplying $n \times n$ matrices, where n is a power of 2

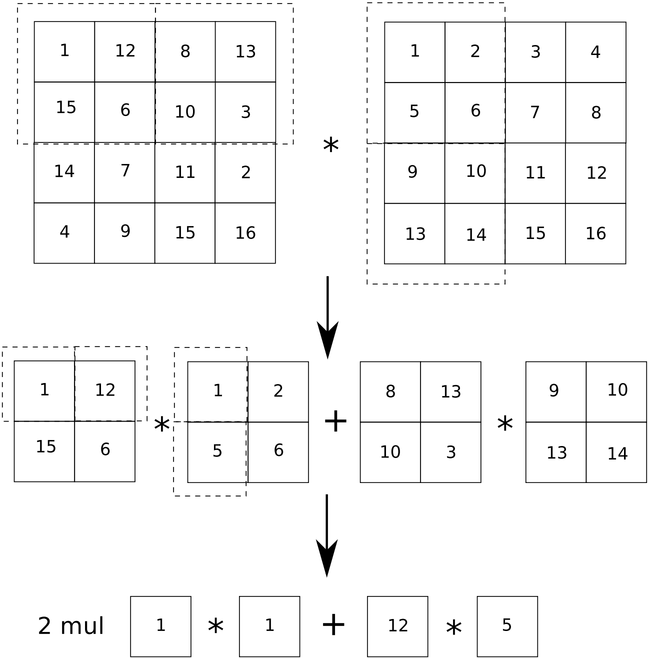
- Express $C = AB$ as

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

  that comes down to computing

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{1,2} \\ C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

- Proceed recursively until you multiply matrices of max size 1×1


1
12
8
13
15
6
10
3
14
7
11
2
4
9
15
16
*
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
1
12
15
6
*
1
2
5
6
+
8
13
10
3
*
9
10
13
14
2 mul
1
*
1
+
12
*
5

Complexity of divide-and-conquer multiplication

$$
\begin{aligned}
T(n) &= 8T(n/2) + n^2 \\
&= n^2 + 8((n/2)^2 + 8T(n/4)) \\
&= n^2 + 8((n/2)^2 + 8((n/4)^2 + 8T(n/16))) \\
&= n^2 + 2n^2 + 4n^2 + 8T(n/16)))
\end{aligned}
$$

Complexity of divide-and-conquer multiplication

$$
\begin{aligned}
T(n) &= 8T(n/2) + n^2 \\
&= n^2 + 8((n/2)^2 + 8T(n/4)) \\
&= n^2 + 8((n/2)^2 + 8((n/4)^2 + 8T(n/16))) \\
&= n^2 + 2n^2 + 4n^2 + 8T(n/16)))
\end{aligned}
$$

i^{th} term in the series is $2^{i-1}n^2$

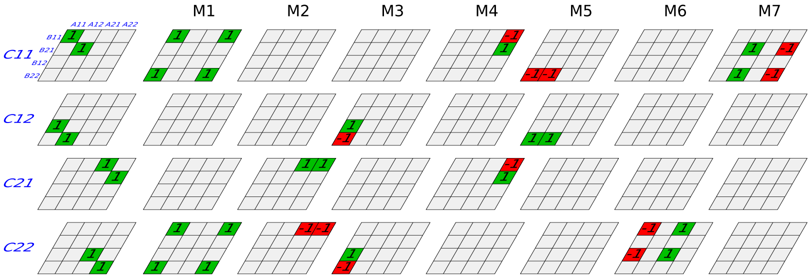
Complexity of divide-and-conquer multiplication

$$
\begin{aligned}
T(n) &= 8T(n/2) + n^2 \\
&= n^2 + 8((n/2)^2 + 8T(n/4)) \\
&= n^2 + 8((n/2)^2 + 8((n/4)^2 + 8T(n/16))) \\
&= n^2 + 2n^2 + 4n^2 + 8T(n/16)))
\end{aligned}
$$

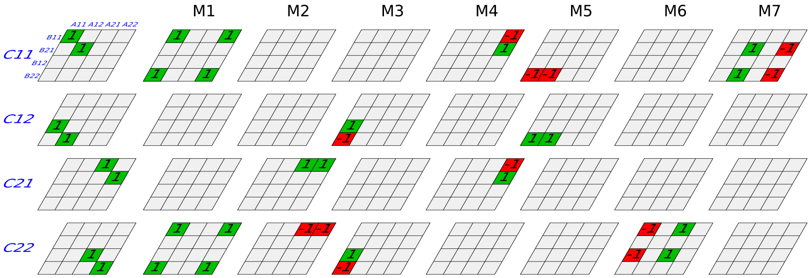
i^{th} term in the series is $2^{i-1}n^2$

$$
\begin{aligned}
T(n) &= n^2 + 2n^2 + 4n^2 + \cdots + 2^{\log_2 n} O(1) \\
&= n^2 \sum_{i=0}^{\log_2 n} 2^i + O(n^{\log_2 2}) \\
&= n^2 \frac{2^{\log_2(n+1)} - 1}{2 - 1} + O(n) \\
&\leq n^2 O(2^{\log_2 n}) + O(n) = n^2 O(n) + O(n) \\
&= O(n^3)
\end{aligned}
$$

Strassen's idea



Strassen's idea



Now we only need to do 7 multiplications, so the complexity becomes

$$
\begin{aligned}
T(n) &= 7T(n/2) + O(n^2) \\
&= O(n^{\log_2 7}) \approx O(n^{2.81})
\end{aligned}
$$

Static data structures

- Matrices and arrays are static data structures in the sense that although accessing an arbitrary element is efficient, adding an element is not
- Example: add an element into an array

Complexity of operations with matrices and arrays

For n elements

- Add element: O(n)
- Random access: O(1)
- Delete element: O(n)

For $n \times n$ matrices:

- Multiplication: O(n^3) (?)
- Inversion: as multiplication
- Determinant: O(n^3) with LU decomposition