Διανυσματικοί Υπολογισμοί (Vectorized computations):
Benchmark Julia vs Matlab

Νικόλαος Κουκουδάκης

Εαρινό Εξάμηνο 2021

1 Πράξεις με μονοδιάστατους πίνακες - διανύσματα

Έστω $n \in \mathbb{N}$ και διάνυσμα $t \in \mathbb{R}^{100n+1}$, που αποτελείται από ισαπέχοντες αριθμούς από το 0 ως το n σε αύξουσα σειρά. Θέλουμε να υπολογίσουμε το τετράγωνο των στοιχείων του t. Υπάρχουν δύο τρόποι:
Ο 1ος γίνεται κατά συντεταγμένες σε Matlab:

```
function notvectorized(n)
  i = 0;
  for t = 0 : .01 : n
    i = i + 1;
    y(i) = t(i)^2;
  end
```

Και σε Julia:

```
function notvectorized(n)
    i = 0
    y = zeros(Float64, 1, 100*n+1)
    @time begin
    for t in 0 : 0.01 : n
        i = i + 1
        y[i] = t[i]^2
    end
    end
end
```

Ο 2ος εκτελείται διανυσματικά σε Matlab:

```
function vectorized(n)
```

```
2    t = 0 : .01 : n;
3    y = t.^2;
```

Και σε Julia:

```
1  function vectorized(n)
2      t = range(0, n, length = 100*n)
3      y = t.^2
4  end
```

Μάλιστα, αν θεωρήσουμε t_1 και t_2 τους χρόνους της κατά συντεταγμένες και διανυσματικής εκτέλεσης αντίστοιχα, ορίζουμε:

$$\lambda = \frac{t_1}{t_2},$$

δηλαδή την αναλογία των χρόνων εκτέλεσης των δύο μεθόδων.

Σε Julia λαμβάνονται τα εξής αποτελέσματα:

n	Κατά συντεταγμένες	Διανυσματικά	Αναλογία χρόνων (λ)
100	$9.6 \cdot 10^{-5} sec$	$2.2 \cdot 10^{-5} sec$	4.3
1,000	$2.6 \cdot 10^{-4} sec$	$2.5 \cdot 10^{-5} sec$	10.1
10,000	$2.7 \cdot 10^{-3} sec$	$6.8 \cdot 10^{-5} sec$	39
100,000	$2.5 \cdot 10^{-2} sec$	$6.3 \cdot 10^{-4} sec$	40
1,000,000	$0.2 sec$	$3.7 \cdot 10^{-3} sec$	66
10,000,000	$2.4 sec$	$0.04 sec$	63

Σε Matlab λαμβάνονται τα εξής αποτελέσματα:

n	Κατά συντεταγμένες	Διανυσματικά	Αναλογία χρόνων (λ)
100	$3 \cdot 10^{-4} sec$	$8.8 \cdot 10^{-5} sec$	3.4
1,000	$5.9 \cdot 10^{-4} sec$	$2.2 \cdot 10^{-4} sec$	2.6
10,000	$6.3 \cdot 10^{-3} sec$	$2.5 \cdot 10^{-3} sec$	2.5
100,000	$5.5 \cdot 10^{-3} sec$	$3.9 \cdot 10^{-3} sec$	1.4
1,000,000	$0.56 sec$	$0.38 sec$	1.5
10,000,000	$5 min\ 12 sec$	$3 min\ 50 sec$	1.4

Στο Octave λαμβάνονται τα εξής αποτελέσματα:

n	Κατά συντεταγμένες	Διανυσματικά	Αναλογία χρόνων (λ)
100	$0.15 sec$	$0.001 sec$	15
1,000	$1.1 sec$	$0.006 sec$	183
10,000	$13.3 sec$	$0.05 sec$	266
100,000	$5 min$	$0.5 sec$	600
1,000,000	$5 h\ 30 min$	$6.1 sec$	3245
10,000,000	$\approx 28 days$	$1 min\ 50 sec$	$1.4 \cdot 10^4$

2 Γινόμενο Πίνακα με Διάνυσμα (Matrix-Vector Product)

Έστω $A \in \mathbb{R}^{m \times n}$ και θέλουμε να υπολογίσουμε το αποτέλεσμα πίνακα - διάνυσμα $y = Ax$, όπου $x \in \mathbb{R}^n$.

2.1 Κατά συντεταγμένες

Ο συνηθισμένος τρόπος για τον υπολογισμό του γινομένου αυτού είναι να υπολογίσουμε τα:

$$y_i = \sum_{j=1}^{n} a_{ij} x_j, \ i = 1, ..., m$$

Αυτό οδηγεί στον ακόλουθο αλγόριθμο:

```
function notvecmvp(n)
    y = zeros(Float64, 1, n)
    x = rand(Float64, n, 1)
    A = rand(Float64, n, n)
    for i in 1:n
        for j in 1:n
            k = y[i] + A[i,j]*x[j]
            y[i] = k
        end
    end
end
```

2.2 Κατά γραμμές

Το παραπάνω διπλό loop δεν είναι απαραίτητο. Έτσι, αντικαθιστούμε το j-loop με το εσωτρικό γινόμενο της i-οστής γραμμής του A και του διανύσματος x και προκύπτει ο παρακάτω κώδικας:

```
function vecmvpRows(n)
    y = zeros(Float64, 1, n)
    x = rand(Float64, n, 1)
    A = rand(Float64, n, n)
    for i in 1:n
        k = A[i,:]'x;
        y[i] = k[1]
    end
end
```

2.3 Κατά στήλες

Αντίστοιχα, με εσωτερικό γινόμενο της j-οστής στήλης του A με το x προκύπτει το ίδιο y με κώδικα ως εξής:

```
1  function vecmvpCols(n)
2      y = zeros(Float64, 1, n)
3      x = rand(Float64, n, 1)
4      A = rand(Float64, n, n)
5      for j in 1:n
6          k = A[:,j]'x;
7          y[j] = k[1]
8      end
9  end
```

2.4 Ολικό γινόμενο

Τέλος μπορούμε να χρησιμοποιήσουμε τον έτοιμο τύπο:

$$y = Ax$$

```
1  function vecmvp(n)
2      x = rand(Float64, n, 1)
3      A = rand(Float64, n, n)
4      y = A*x
5  end
```

Οι επιδόσεις των προαναφερθέντων αλγορίθμων βρίσκονται στον ακόλουθο πίνακα.

n	Κατά συντεταγμένες	Κατά γραμμές	Κατά στήλες	Ολικό γινόμενο
100	$0.001sec$	$0.001sec$	$0.001sec$	$0.001sec$
1,000	$0.04sec$	$0.008sec$	$0.003sec$	$0.001sec$
5,000	$0.3sec$	$0.4sec$	$0.1sec$	$0.02sec$
10,000	$1.8sec$	$2.1sec$	$0.5sec$	$0.09sec$
20,000	$9.4sec$	$11.4sec$	$2.6sec$	$0.38sec$
30,000	$25sec$	$32sec$	$3.8sec$	$1.25sec$

2.5 Συμπεράσματα

Παρατηρούμε ότι οι διανυσματικά εκτελούμενοι αλγόριθμοι τρέχουν πολύ πιο γρήγορα απ΄ ό,τι κατά συντεταγμένες, με τη διαφορά να ανεβαίνει ραγδαία όσο μεγαλώνει το μήκος του διανύσματος (ή οι διαστάσεις του πίνακα). Αυτή βελτίωση είναι πιο εύκολα ορατή όταν σταδιακά εισάγουμε τους διανυσματικούς υπολογισμούς (§2.2). Τελευταίο αλλά εξίσου σημαντικό είναι το γεγονός πως ο αλγόριθμος κατά στήλες εκτελείται εξαρετικά πιο γρήγορα απ΄ ό,τι κατά γραμμές.