Μέτρηση χρόνου εκτέλεσης στη Julia

Περιεχόμενα

1 Εισαγωγή

Σε αυτό το έγγραφο παρουσιάζονται ορισμένες από τις βασικότερες και απλούστερες εντολές της Julia που μπορούν να χρησιμοποιηθούν για τη μέτρηση του χρόνου εκτέλεσης ενός block κώδικα. Επιπλέον, επεξηγείται με παραδείγματα ο τρόπος χρήσης κάθε μίας εξ αυτών και αναλύονται τα πλεονεκτήματα και τα μειονεκτήματά της.

2 Διαθέσιμες Εντολές και Γενικός Τρόπος Χρήσης τους

Οι κύριες εντολές της Julia που μπορούν να αξιοποιηθούν για χρονομέτρηση είναι οι ακόλουθες:

1. @time
2. @timed
3. @elapsed

Πέρα από τις προηγούμενες, ιδιαίτερα χρήσιμες είναι και οι ακόλουθες εντολές της βιβλιοθήκης BenchmarkTools:[1]

1. @btime
2. @belapsed

Το σύμβολο '@' με το οποίο ξεκινούν όλες οι προηγούμενες εντολές δηλώνει ότι είναι μακροεντολές. Οι μακροεντολές έχουν τη χαρακτηριστική ιδιότητα ότι δρουν πάνω σε ένα τμήμα του προγράμματος στο οποίο ανήκουν.[2] Ας δούμε ένα απλό παράδειγμα χρήσης τους:

```
julia> @elapsed 1+1
5.0e-8
```

Εδώ το τμήμα του κώδικα πάνω στο οποίο δρα η μακροεντολή @elapsed είναι η πράξη $1 + 1$.

Στο παραπάνω παράδειγμα, ο κώδικας του οποίου μετράμε τον χρόνο εκτέλεσης είναι αρκετά απλός. Εάν θέλουμε να μετρήσουμε το χρόνο εκτέλεσης ενός block κώδικα περισσότερων γραμμών, μπορούμε να το κάνουμε με τους ακόλουθους τρόπους:

- Δηλώνοντας με τις εντολές **begin** και **end** ότι πρόκειται για ενιαίο block κώδικα:

```
@elapsed begin
εντολή 1
εντολή 2
end
```

[1] Για την εγκατάσταση του πακέτου αρκεί να εκτελεστεί σε διαδραστικό περιβάλλον Julia (Julia REPL) η ακόλουθη εντολή:

```
]add BenchmarkTools
```

Ο χαρακτήρας ']' ενεργοποιεί το package mode, οπότε το prompt μετατρέπεται από **julia**> σε **pkg**>. Για περισσότερες πληροφορίες σχετικά με τη βιβλιοθήκη παραθέτουμε την επίσημη ιστοσελίδα της.

[2] Η Julia μπορεί να διαχειριστεί τέτοιου είδους εντολές, επειδή ο ίδιος ο κώδικας αντιμετωπίζεται ως αντικείμενο, το οποίο μια κατάλληλη συνάρτηση, όπως οι παραπάνω, μπορεί να επεξεργαστεί. Για περισσότερες λεπτομέρειες: Julia Manual - Metaprogramming.

- Περικλείοντας τις εντολές σε μία συνάρτηση:

```
function function_name()
εντολή 1
εντολή 2
end

@elapsed function_name()
```

Γενικά, στη Julia είναι μια καλή πρακτική να περικλείουμε τον κώδικα σε συναρτήσεις, ανεξάρτητα από το αν μετράμε τον χρόνο εκτέλεσης.[3] Για τον λόγο αυτό, στη συνέχεια θα υιοθετήσουμε αποκλειστικά τον πρώτο τρόπο. Ο δεύτερος μπορεί να χρησιμοποιείται χωρίς επιπτώσεις στην ταχύτητα του προγράμματος μόνο όταν η εντολή @elapsed βρίσκεται ήδη μέσα σε κάποια συνάρτηση.

[3]Αυτό συμβαίνει επειδή ο compiler της γλώσσας διαχειρίζεται με μεγαλύτερη ευκολία αυτόν τον κώδικα, οπότε παράγει γρηγορότερο κώδικα μηχανής και ο χρόνος εκτέλεσης μειώνεται. Για περισσότερες πληροφορίες: Julia Documentation - Performance Tips.

3 Αναλυτική Εξέταση των Εντολών

3.1 Η εντολή @time

Ας χρονομετρήσουμε τον πολλαπλασιασμό ενός τυχαίου 2×3 πίνακα με τον ανάστροφό του, χρησιμοποιώντας την @time:

```
function matrix_mult()
A = rand(Float64, (2, 3))
A*A´
end

@time matrix_mult()
```

Τρέχοντας το πρόγραμμα, το αποτέλεσμα είναι το εξής:

```
  0.000657 seconds (21 allocations: 20.422 KiB)
```

Παρατηρούμε ότι, πέρα από το χρόνο εκτέλεσης, εκτυπώνεται και η μνήμη που χρησιμοποιήθηκε. Επιπλέον, όταν δεν είναι αμελητέο, εμφανίζεται το ποσοστό του χρόνου που αφιερώθηκε στο garbage collection, αφού είναι πιθανό να μην πρέπει να ληφθεί υπόψη. Αντίστοιχα, όταν δεν χρησιμοποιείται μνήμη για την αποθήκευση της τιμής κάποιας καθολικής μεταβλητής, το memory allocation παραλείπεται. Για παράδειγμα, στον ακόλουθο κώδικα, αφού δεν υπάρχουν καθολικές μεταβλητές (η a βρίσκεται εντός της συνάρτησης), το memory allocation είναι μηδενικό (και ο χρόνος εκτέλεσης μη μετρήσιμος):

```
function assign_value()
a = 3*10^3
end

@time assign_value()
```

```
  0.000000 seconds
```

Ωστόσο, όπως σημειώνεται στα Performance Tips στο Julia Documentation, χρειάζεται ιδιαίτερη προσοχή κατά τη χρονομέτρηση για να μη ληφθεί υπ΄ όψη ο χρόνος που απαιτείται για τη δημιουργία του κώδικα μηχανής. Αυτό μπορούμε να το δούμε με το ακόλουθο πρόγραμμα, το οποίο προσθέτει τα τετράγωνα όλων των φυσικών αριθμών που είναι μικρότεροι ή ίσοι από τον αριθμό που παρέχεται, εκτυπώνοντας σε κάθε βήμα το τρέχον άθροισμα:

```
function sum_of_squares(n)
s = 0
i = 1
while i<=n
s += i^2
println(s)
i += 1
end
end

@time sum_of_squares(10^3)
@time sum_of_squares(10^3)
@time sum_of_squares(10^3)
```

Αγνοώντας τα μερικά αθροίσματα, το πρόγραμμα εκτυπώνει τα ακόλουθα:

```
0.045306 seconds (22.20 k allocations: 948.455 KiB)
0.007956 seconds (9.07 k allocations: 277.062 KiB)
0.005942 seconds (9.07 k allocations: 277.062 KiB)
```

Παρατηρούμε ότι ο πρώτος χρόνος είναι πολύ μεγαλύτερος από τους προηγούμενους (και αντίστοιχα η χρήση μνήμης είναι επίσης κατά πολύ μεγαλύτερη). Για αυτό ευθύνεται το γεγονός ότι, την πρώτη φορά που εκτελείται η συνάρτηση sum_of_squares(10^3), πρέπει να μετατραπεί σε κώδικα μηχανής, ενώ τις επόμενες ο κώδικας μηχανής είναι ήδη έτοιμος. Επειδή σχεδόν πάντα θέλουμε να εστιάσουμε στο κομμάτι της εκτέλεσης του κώδικα μηχανής, *πρέπει πάντα να αγνοούμε το πρώτο αποτέλεσμα που λαμβάνουμε με την @time, καθώς και με όλες τις υπόλοιπες προεγκατεστημένες μακροεντολές που απαριθμήσαμε προηγουμένως.*

Μένει ένα ακόμα ερώτημα όσον αφορά τη μακροεντολή @time: εάν η συνάρτηση της οποίας μετράται ο χρόνος εκτέλεσης επιστρέφει κάποια τιμή, μετά τη χρονομέτρηση αυτή χάνεται ή είναι διαθέσιμη με κάποιον τρόπο· Η απάντηση είναι ότι αυτήν την τιμή επιστρέφει και η @time, δηλαδή μπορούμε να κρατήσουμε την τιμή αυτή με τον ακόλουθο τρόπο:

```
όνομα_μεταβλητής = @time όνομα_συνάρτησης()
```

3.2 Η εντολή @timed

Το κύριο μειονέκτημα της @time είναι ότι εκτυπώνει κατ΄ ευθείαν τα δεδομένα, οπότε δεν είναι δυνατή η περαιτέρω επεξεργασία τους. Ιδιαίτερα αν λάβουμε υπόψη το γεγονός ότι ορισμένα δεδομένα (π.χ. τα δεδομένα της πρώτης εκτέλεσης, ή σε ορισμένες περιπτώσεις το memory allocation) συχνά δεν έχουν κάποια χρησιμότητα, το γεγονός ότι αυτά αναγκαστικά εκτυπώνονται και δεν υπάρχει κάποιος προφανής τρόπος να απορριφθούν αποτελεί ένα σημαντικό πρόβλημα. Για τον λόγο αυτό, είναι πιθανό να προτιμηθεί η εντολή @timed, η οποία επιστρέφει σε μια πλειάδα την τιμή που επιστρέφει η συνάρτηση, καθώς και τις πληροφορίες που εκτυπώνει η @time:

```
function sum_of_squares(n)
s = 0
i = 1
while i<=n
s += i^2
println(s)
i += 1
end
return sn
end

@timed sum_of_squares(10^3)
t = @timed sum_of_squares(10^3)
println("Η συνάρτηση @timed επέστρεψε την τιμή ", t, " τύπου ", typeof(t))
```

```
Η συνάρτηση @timed επέστρεψε την τιμή (value = 333833.5,  time = 0.006693802,  bytes = 283712,  gctime = 0.0,  gcstats = Base.GC_Diff(283712, 0, 0, 9072, 0, 0, 0, 0, 0))
τύπου NamedTuple(:value, :time, :bytes, :gctime, :gcstats), TupleFloat64,Float64,Int64,Float64,Base.GC_Diff
```

Παρατηρούμε ότι πλέον έχουμε τον πλήρη έλεγχο του τι εκτυπώνεται και η πρώτη εκτέλεση αγνοείται εύκολα.

3.3 Η εντολή @elapsed

Συχνά το μόνο που χρειαζόμαστε από όσα επιστρέφει η @timed είναι ο χρόνος εκτέλεσης. Σε αυτές τις περιπτώσεις, καταλληλότερη είναι η εντολή @elapsed, η οποία επιστρέφει μόνο τη συγκεκριμένη τιμή (σε δευτερόλεπτα). Θα πρέπει να σημειώσουμε ότι με την @elapsed χάνεται και η τιμή που επιστρέφει η συνάρτηση.

3.4 Η εντολή @btime

Η συγκεκριμένη εντολή είναι η πρώτη από τα εργαλεία του BenchmarkTools που θα εξετάσουμε. Η λειτουργία της είναι παρόμοια με αυτή της @time (εκτυπώνει τα δεδομένα στην ίδια μορφή και επιστρέφει ό,τι επιστρέφει η χρονομετρούμενη συνάρτηση), αλλά ο χρόνος εκτέλεσής της είναι πολύ μεγαλύτερος. Αυτό συμβαίνει επειδή εκτελεί τη συνάρτηση πολλές φορές και εκτυπώνει τον ελάχιστο από τους χρόνους (η επιλογή αυτή γίνεται διότι οι όποιες παρεμβολές στη μέτρηση του χρόνου κατά κανόνα αυξάνουν την τελική τιμή, άρα ο ελάχιστος χρόνος είναι πιθανότατα ο πιο ακριβής). Επιπλέον, σε αυτή την εντολή, όπως και σε όλες τις υπόλοιπες της βιβλιοθήκης BenchmarkTools, στις παραμέτρους εισόδου πρέπει να εφαρμόζεται interpolation, το οποίο υλοποιείται προσθέτοντας το σύμβολο $ πριν από το όνομα της καθολικής μεταβλητής:

```
using BenchmarkTools

function my_addition(a, b)
return a+b
end

a = 1
b = rand()
@btime my_addition($a, $b)
```

```
  5.159 ns (0 allocations: 0 bytes)
```

Με την προσθήκη αυτών των συμβόλων, ο compiler αντιμετωπίζει κατά την εκτέλεση της συνάρτησης τις τιμές των μεταβλητών ως σταθερές και έτσι παράγει αποτελεσματικότερο κώδικα μηχανής.[4] Ας αφαιρέσουμε κατ' εξαίρεση το interpolation για να δούμε τις επιπτώσεις στο χρόνο:

```
using BenchmarkTools

function my_addition(a, b)
return a+b
end

a = 1
b = rand()
@btime my_addition(a, b)
```

```
  42.717 ns (1 allocation: 16 bytes)
```

3.5 Η εντολή @belapsed

Η εντολή αυτή λειτουργεί όπως η @elapsed (επιστρέφει μόνο τον χρόνο εκτέλεσης σε δευτερόλεπτα), αλλά έχει και τις ιδιαιτερότητες της @btime: η εκτέλεσή της παίρνει πολύ περισσότερη ώρα, καθώς επιστρέφει τον ελάχιστο χρόνο ύστερα από μεγάλο πλήθος εκτελέσεων του κώδικα που χρονομετράται και, στην περίπτωση που αυτός έχει τη μορφή συνάρτησης, πρέπει να εφαρμοσθεί interpolation στις παραμέτρους εισόδου.

[4] Βλ. Julia Packages - BenchmarkTools.

4 Συμπεράσματα και Προτάσεις Υλοποίησης

- Οι προεγκατεστημένες στη Julia συναρτήσεις προτείνονται για τη χρονομέτρηση σχετικά χρονοβόρου κώδικα, υπό την προϋπόθεση να μην απαιτείται ιδιαίτερα υψηλή ακρίβεια στην εκτίμηση του χρόνου εκτέλεσης (λόγω των πολλών εκτελέσεων του κώδικα, τα εργαλεία της βιβλιοθήκης BenchmarkTools μπορεί σε αυτές τις περιπτώσεις να χρειαστούν ιδιαίτερα μεγάλο χρόνο για να ολοκληρώσουν, γι΄ αυτό γενικά δεν ενδύκνειται η χρήση τους). Ειδικότερα, η συνάρτηση @elapsed προτείνεται εάν ενδιαφερόμαστε μόνο για τον χρόνο εκτέλεσης, ενώ αν θέλουμε προς επεξεργασία και άλλα δεδομένα (όπως η μνήμη που χρησιμοποιήθκε και η τιμή που επέστρεψε η συνάρτηση), ενδείκνυται η @timed. Τέλος, εάν είναι επιθυμητή η άμεση εκτύπωση του χρόνου εκτέλεσης (π.χ. προς πληροφόρηση του ίδιου του προγραμματιστή) και προς περαιτέρω επεξεργασία χρειαζόμαστε μόνο την τιμή που επιστρέφει η συνάρτηση, καταλληλότερη είναι η @time. Σε όλες αυτές τις συναρτήσεις, η πρώτη εκτέλεση δεν πρέπει να λαμβάνεται υπ΄ όψη.

- Οι συναρτήσεις της βιβλιοθήκης BenchmarkTools προτείνονται για σχετικά μικρούς χρόνους εκτέλεσης και ιδιαίτερα για εκτιμήσεις στις οποίες η ακρίβεια είναι ιδιαίτερα σημαντική. Ειδικότερα, η συνάρτηση @belapsed προτείνεται εάν ενδιαφερόμαστε μόνο για τον χρόνο εκτέλεσης, ενώ για άμεση εκτύπωση των δεδομένων, καθώς και παροχή περισσότερων πληροφοριών, ενδείκνυται η @btime. Κατά τη χρονομέτρηση συναρτήσεων που δέχονται ορίσματα θα πρέπει να πραγματοποιείται interpolation.