Paweł Kliber

Optymalizacja kodu w R na przykładzie zastosowań finansowych

Dlaczego tak powoli

R jest językiem interpretowanym – kod jest tłumaczony „w locie”, a nie kompilowany

W przypadku dłuższych bloków składniowych (np. pętle) tłumaczenie zachodzi wielokrotnie

Dodatkowo – argumenty są przekazywane przez wartość (muszą być kopiowane)

Ogólne zasady

- Unikaj pętli
- Gdy się da – korzystaj z zapisu macierzowego, wektorowego
- Przetwarzaj całe struktury, a nie pojedyncze elementy
- Stosuj funkcje wewnętrzne dla operacji (spychaj co się da na niski poziom)
- Szukaj pakietów, które mają zoptymalizowane funkcje
- Stosuj `apply` i rodzinę

Wolno działający kod

```
for(i in 1:n) {
  for(j in 1:m) {
     .....
       A[i, j] <- fun(A, i, j)
  }
}
```

Za każdym razem tworzona jest kopia całej macierzy

Przykłady

- Wektoryzacja
- Funkcje wbudowane
- Unikanie ,,przyrostów” struktury
- Funkcja `apply`

Rodzina funkcji apply

- `apply(X, margin, f)` – operacja na macierzach
- `lapply(list, f)` – operacja na elementach listy
- `sapply(list, f)` – na listach, wektoryzowane
- `mapply(f, list1, list2, ...)` – `sapply` na wielu listach (dla funkcji o wielu argumentach)

Funkcja Reduce

- x=(x1, x2, x3, x4, ...)
- `Reduce(f, x, accumulate=FALSE) :`
- (f(x1, x2), x3, x4, ...)
- (f(f(x1, x2), x3), x4, ...)
- (f(f(f(x1, x2), x3), x4), ...)
- ....
- Przykład – wygładzanie wykładnicze

Wykorzystanie pakietów przyśpieszających obliczenia

- `bigmemory` – pozwala m.in. na przekazywanie argumentów do przez referencję
- `data.table` – ulepszona wersja `data.frame` – też możliwe wykorzystanie referencji

Kompilacja funkcji w R

- Pakiet `compiler` - od wersji 2.13 dołączany standardowo do R, od wersji 2.14 standardowe funkcji i pakiet są skompilowane.
- Kompilacja do kodu bajtowego (pośredniego), który szybciej się interpretuje.
- JIT (Just-In-Time compilation) – nadal interpretacja, ale wykorzystuje bufor.
- Funkcje: `enableJIT`, `cmpfun`, `cmpfile`, `loadcmp`

Przygotowanie funkcji w języku niższego poziomu (C, Fortran)

- Potrzebne pliki nagłówkowe R.h, Rmath.h, Rinternals.h,…
- Wymaga skompilowania dynamicznie dołączanej biblioteki (.so, .dll)
- Bibliotekę ładujemy dyn.load(file)
- Obudowanie funkcjami R
- Wywołanie funkcji: `.C(fun, args)`

Konwersja typów

R	C
integer	int*
numeric	double* /float*
complex	Rcomplex*
logical	int*
character	char**
raw	unsigned char*
list	SEXP*
other	SEXP

Problem

- Analiza danych finansowych wysokiej częstotliwości
- Dane transakcyjne z GPW. Dokładność czasowa 1s.
- Przetworzenie plików danych. Dane surowe dla dni sesyjnych. Wydobycie akcji i indektów (Python).
- Próbkowanie danych w różnych częstotliwościach. Obliczanie stóp zwrotu i statystyk.

Fragment pliku z danymi (notowania KGHM S.A.)

2009-01-07,09:00:16,1,32416,3,33.1
2009-01-07,09:00:17,1,32417,3,33.15
2009-01-07,09:00:17,1,32417,3,33.19
2009-01-07,09:00:17,1,32417,3,33.19
2009-01-07,09:00:18,1,32418,3,33.19
2009-01-07,09:00:21,1,32421,3,33.11
2009-01-07,09:00:21,1,32421,3,33.11
2009-01-07,09:00:23,1,32423,3,33.19
2009-01-07,09:00:29,1,32429,3,33.2
2009-01-07,09:00:29,1,32429,3,33.2
2009-01-07,09:00:29,1,32429,3,33.23
2009-01-07,09:00:29,1,32429,3,33.25
2009-01-07,09:00:36,1,32436,3,33.25
2009-01-07,09:00:36,1,32436,3,33.28
2009-01-07,09:00:36,1,32436,3,33.29
2009-01-07,09:00:36,1,32436,3,33.3
2009-01-07,09:00:36,1,32436,3,33.3
2009-01-07,09:00:36,1,32436,3,33.3
2009-01-07,09:00:36,1,32436,3,33.33
2009-01-07,09:00:36,1,32436,3,33.33
2009-01-07,09:00:36,1,32436,3,33.4
2009-01-07,09:00:36,1,32436,3,33.4

Dane

- 583 dni sesyjne
- Wielkość plików: od 5 MB (Agora) do 300 MB (FW20)
- Od 150tys. do 740tys. transakcji
- Brak możliwości wektoryzowania przy próbkowaniu.
- Czasami trzeba próbować dla jednego dnia, czasami wszystkie dane.

```
#include <R.h>
#include <math.h>

void resample(double *tt, double *pp, int *pdt, int *pstart, int *pend, int *ptrans_N, double *pout)  {
	int start, end, dt, trans_N;
	start = *pstart;
	end = *pend;
	dt = *pdt;
	trans_N = *ptrans_N;

	int t, trans, curr;
	double p;
	t = start;
	trans = 0;
	curr = 0;

	p=pout[0]=pp[0];
	curr++;
	t+=dt;

	while(t<=end) {
		while(trans<trans_N && tt[trans]<=t)
				p = pp[trans++];
		pout[curr++] = p;
		t += dt;
	}
}
```

Kompilacja

- W Linuksie – do biblioteki .so

```
gcc -shared resample.c -lm -fPIC -I/usr/include/R -o
   resample.so
```

- Kompilacja za pomocą R:

```
R CMD SHLIB resampling.c
```

- W Windows – do biblioteki .dll (np. w Visual Studio)

Obudowanie funkcji

```
dyn.load("resample.dll")

resample<-function(d, dt) {
  trans_N<-length(d$p)
  pp<-d$p
  tt<-d$t
  m<-1+floor((end-start)/dt)
  out<-numeric(m)
  p<-.C("resample", tt=as.double(tt), pp=as.double(pp), pdt=as.integer(dt),
     pstart=as.integer(start), pend=as.integer(end),
     ptrans_N=as.integer(trans_N), pout=as.double(out))$pout
 t<-seq(start, end, dt)
 data.frame(t=t, p=p)
}
```

Pakiet Rcpp

- Łatwiejsza integracja R i C++
- Kompilowanie wykonywane jest automatycznie
- Należy najpierw zainstalować Rtools http://cran.r-project.org/bin/windows/Rtools/
- Możliwość korzystania z biblioteki STL
- Dostęp do R z C++

Działanie

- Deklaracje

```
#include <Rcpp.h>
using namespace Rcpp;
```

- Zaznaczenie funkcji do eksportu

```
// [[Rcpp::export]]
```

- Typy argumentów: `NumericVector`, `NumericMatrix`, `CharacterVector`,… są obiektami (dostęp przez metody)
- Przekazywanie argumentu przez referencję

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
int dodaj(int x, int y) {
   int s;
   s=x+y;
   return(s);
}
```

```
// [[Rcpp::export]]
void add10(NumericVector v) {
  int i, n;
  n = v.size();
  for(i=0; i<n; i++)
      v[i]+=10;
}

// [[Rcpp::export]]
NumericVector add10a(NumericVector v) {
  int i, n;
  n = v.size();
  for(i=0; i<n; i++)
      v[i]+=10;
  return(v);
}

// [[Rcpp::export]]
NumericVector add10b(NumericVector v) {
  int i, n;
  n = v.size();
  NumericVector out(n);
  for(i=0; i<n; i++)
      v[i]+=10;
  return(v);
}
```