Monte-Carlo examples in R

Serial vs parallel code

Meelis Utt

Setup

Let's source the setup (function, analytical solution, number of iterations).

```
source("Setup.R",echo = T)
```

```
##
## > options(scipen = 1000)
##
## > n <- 10000000
##
## > header <- c("n", "computational", "analytical", "error",
## +     "walltime", "type")
##
## > f <- function(x) {
## +     return(x^2 + x^4 + sin(x) + cos(x) + x^25)
## + }
##
## > analytical <- integrate(f, 0, 1)$value
```

Serial implementations

Let's start with a simple implementation of Monte-Carlo method.

```
MCser1 <- function(n){
  start <- Sys.time()
  i <- runif(n,0,1)
  EX <- mean(f(i))
  end <- Sys.time()
  time <- difftime(end,start)
  error <- (EX - analytical) %>% abs
  return(c(n,EX,analytical,error,time,"MCser1"))
}
data.table(t(MCser1(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87353741604257 1.87296355073463 0.000573865307943233
##            walltime   type
## 1: 2.04202342033386 MCser1
```

Let's try a bit more vectorized solution, using the *apply function.

```
MCser2 <- function(n,ncols=1000){
  start <- Sys.time()
```

```
  dt <- matrix(runif(n,0,1),ncol = ncols)
  EX <- sapply(1:ncols,function(i,dt){
      EX <- dt[,i] %>% f %>% mean
    },dt) %>% mean
  end <- Sys.time()
  time <- difftime(end,start)
  error <- (EX - analytical) %>% abs
  return(c(n,EX,analytical,error,time,"MCser2"))
}
data.table(t(MCser2(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87266314578146 1.87296355073463 0.000300404953171185
##            walltime   type
## 1: 1.60613822937012 MCser2
```

Let's try an approach using data.table.

```
MCser3 <- function(n){
  start <- Sys.time()
  dt <- data.table(unif = runif(n,0,1))
  EX <- dt[,.(EX = mean(f(unif)))] %>% unlist %>% unname
  end <- Sys.time()
  time <- difftime(end,start)
  error <- (EX - analytical) %>% abs
  return(c(n,EX,analytical,error,time,"MCser3"))
}
data.table(t(MCser3(n))) %>% setNames(header)
```

```
##           n    computational       analytical                  error
## 1: 10000000 1.87295819430205 1.87296355073463 0.00005356432578373937
##            walltime   type
## 1: 1.68406891822815 MCser3
```

Let's try divide-and-conquer approach with data.table.

```
MCser4 <- function(n,ncols=1000){
  start <- Sys.time()
  dt <- matrix(runif(n,0,1),ncol=ncols) %>% data.table
  EX <- dt[,lapply(.SD,f)][,lapply(.SD,mean)][,.(EX = sum(.SD)/ncols)] %>% unlist %>% unname
  end <- Sys.time()
  time <- difftime(end,start)
  error <- (EX - analytical) %>% abs
  return(c(n,EX,analytical,error,time,"MCser4"))
}
data.table(t(MCser4(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87330731982586 1.87296355073463 0.00034376909123246
##            walltime   type
## 1: 1.88429856300354 MCser4
```

Parallel implementations

Let's try different parallel implementations. First let's start with package *parallel*.

```
MCpar1 <- function(n){
  start <- Sys.time()
  # Calculate the number of cores
  no_cores <- detectCores()
  # Initiate cluster
  cl <- makeCluster(no_cores)
  intermean <- parSapply(cl, rep(n/no_cores,no_cores),function(ni,f){
      EX <- mean(f(runif(ni,0,1)))
    },f
  )
  on.exit(stopCluster(cl))
  EX <- mean(intermean)
  end <- Sys.time()
  time <- difftime(end,start)
  error <- (EX - analytical) %>% abs
  return(c(n,EX,analytical,error,time,"MCpar1"))
}
data.table(t(MCpar1(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87339102185438 1.87296355073463 0.000427471119751388
##            walltime   type
## 1: 1.32567048072815 MCpar1
```

Now let's try approach analogical to MCser2.

```
MCpar2 <- function(n){
  start <- Sys.time()
  # Calculate the number of cores
  no_cores <- detectCores()
  cl <- makeCluster(no_cores)
  dt <- matrix(runif(n,0,1),ncol = no_cores)
  intermean <- parSapply(cl, 1:no_cores,function(i,f,dt){
      EX <- mean(f(dt[,i]))
    },f,dt
  )
  on.exit(stopCluster(cl))
  EX <- mean(intermean)
  end <- Sys.time()
  error <- (EX - analytical) %>% abs
  time <- difftime(end,start)
  return(c(n,EX,analytical,error,time,"MCpar2"))
}
data.table(t(MCpar2(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87304020083651 1.87296355073463 0.000076650101882958
##            walltime   type
## 1: 4.93123769760132 MCpar2
```

This approach was not very good. But let's have one more try at analogical solution to MCser2.

```
MCpar2_2 <- function(n,ncols=1000){
  start <- Sys.time()
  # Calculate the number of cores
  no_cores <- detectCores()
```

```
  cl <- makeCluster(no_cores)
  dt <- matrix(runif(n,0,1),ncol = ncols)
  intermean <- parSapply(cl, 1:ncols,function(i,f,dt){
      EX <- mean(f(dt[,i]))
    },f,dt
  )
  on.exit(stopCluster(cl))
  EX <- mean(intermean)
  end <- Sys.time()
  error <- (EX - analytical) %>% abs
  time <- difftime(end,start)
  return(c(n,EX,analytical,error,time,"MCpar2_2"))
}
data.table(t(MCpar2_2(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87279591153091 1.87296355073463 0.000167639203717762
##            walltime     type
## 1: 4.82105684280396 MCpar2_2
```

This solution was bit better, but still worse than the previous examples.
Let's try the package *foreach* now.

```
MCpar3 <- function(n){
  start <- Sys.time()
  # Calculate the number of cores
  no_cores <- detectCores()
  # Initiate cluster
  cl<-makeCluster(no_cores)
  # registerDoParallel(cl)
  EX <- foreach(ni = rep(n/no_cores,no_cores),.combine=mean,.export="f") %dopar%
    mean(f(runif(ni,0,1)))
  on.exit(stopCluster(cl))
  # stopImplicitCluster()
  end <- Sys.time()
  time <- difftime(end,start)
  error <- (EX - analytical) %>% abs
  return(c(n,EX,analytical,error,time,"MCpar3"))
}
data.table(t(MCpar3(n))) %>% setNames(header)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87332913732254 1.87296355073463 0.000365586587907529
##            walltime   type
## 1: 2.03123497962952 MCpar3
```

Benchmarking

Now, let's visualize the walltimes of implemented solutions.

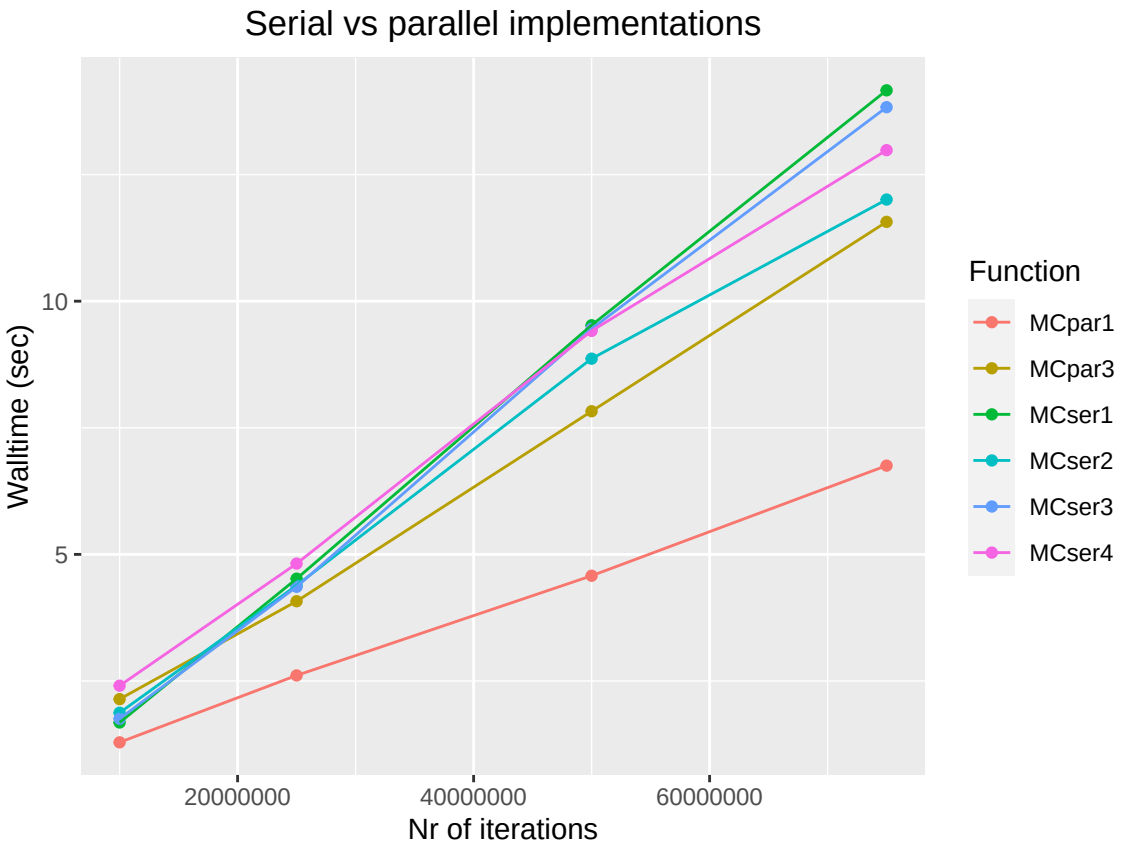
```
iterations <- c(1,2.5,5,7.5)*10**(7)#(5:7)
funs <- c(
  MCser1,MCser2,MCser3,MCser4,
```

```
  MCpar1,MCpar3 #MCpar2,MCpar2_2,
)
data <- sapply(funs,function(f,iterations){
  sapply(iterations,function(n){
    f(n)
  })
},iterations) %>%
  matrix(ncol=6,byrow=T) %>%
  data.table %>%
  setNames(header) %>%
  mutate_at(header[-grep(x=header,pattern="type")],as.numeric)

data
```

```
##            n computational analytical         error  walltime   type
##  1: 10000000      1.873144   1.872964 0.00018006442  1.679950 MCser1
##  2: 25000000      1.872934   1.872964 0.00002993648  4.520063 MCser1
##  3: 50000000      1.872776   1.872964 0.00018749121  9.524911 MCser1
##  4: 75000000      1.873037   1.872964 0.00007335057 14.163665 MCser1
##  5: 10000000      1.873000   1.872964 0.00003679440  1.872934 MCser2
##  6: 25000000      1.873036   1.872964 0.00007210419  4.387838 MCser2
##  7: 50000000      1.872938   1.872964 0.00002604306  8.864731 MCser2
##  8: 75000000      1.872995   1.872964 0.00003098338 12.007296 MCser2
##  9: 10000000      1.872698   1.872964 0.00026519163  1.755220 MCser3
## 10: 25000000      1.872981   1.872964 0.00001748658  4.360301 MCser3
## 11: 50000000      1.872957   1.872964 0.00000608254  9.454168 MCser3
## 12: 75000000      1.872898   1.872964 0.00006552740 13.830856 MCser3
## 13: 10000000      1.873285   1.872964 0.00032096333  2.406975 MCser4
## 14: 25000000      1.872999   1.872964 0.00003529201  4.818219 MCser4
## 15: 50000000      1.872987   1.872964 0.00002313410  9.413631 MCser4
## 16: 75000000      1.872992   1.872964 0.00002803494 12.982518 MCser4
## 17: 10000000      1.872892   1.872964 0.00007109093  1.288364 MCpar1
## 18: 25000000      1.873019   1.872964 0.00005506899  2.609506 MCpar1
## 19: 50000000      1.872944   1.872964 0.00001965172  4.578090 MCpar1
## 20: 75000000      1.872952   1.872964 0.00001173007  6.752777 MCpar1
## 21: 10000000      1.872822   1.872964 0.00014118409  2.141526 MCpar3
## 22: 25000000      1.873535   1.872964 0.00057168676  4.074641 MCpar3
## 23: 50000000      1.872938   1.872964 0.00002530814  7.826763 MCpar3
## 24: 75000000      1.872941   1.872964 0.00002210452 11.566591 MCpar3
##            n computational analytical         error  walltime   type
```

```
ggplot(data=data,aes(x=n,y=walltime,group=type,color=type)) +
  geom_point() +
  geom_line() +
  labs(
       title="Serial vs parallel implementations",
       x="Nr of iterations",
       y="Walltime (sec)"
       ) +
  theme(
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5)
       ) +
  guides(color=guide_legend(title="Function"))
```


Serial vs parallel implementations
Walltime (sec)
10
5
20000000
40000000
60000000
Nr of iterations
Function
MCpar1
MCpar3
MCser1
MCser2
MCser3
MCser4