Imperial College
London

Advanced Computational Methods in Statistics: Lecture 1 Monte Carlo Simulation & Parallel Computing

Axel Gandy

Department of Mathematics, Imperial College London
http://www2.imperial.ac.uk/~agandy

London Taught Course Centre
for PhD Students in the Mathematical Sciences
Autumn 2015

Lectures 2-5: Optimisation, MCMC, Bootstrap, Particle Filtering

Today's Lecture

Part I Monte Carlo Simulation

Part II Introduction to Parallel Computing

Part I

Monte Carlo Simulation

Outline

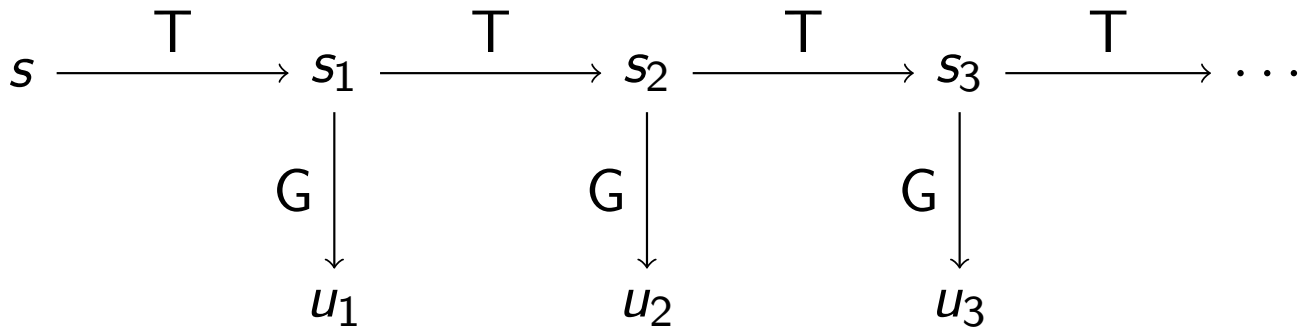
Uniform Random Number Generation

- Basic building block of simulation:
  stream of independent rv $U_1, U_2, \ldots \sim U(0,1)$
- "True" random number generators:
  - based on physical phenomena
  - Example `http://www.random.org/`; R-package *random*: "The randomness comes from atmospheric noise"
  - Disadvantages of physical systems:
    - cumbersome to install and run
    - costly
    - slow
    - cannot reproduce the exact same sequence twice [verification, debugging, comparing algorithms with the same stream]
- Pseudo Random Number Generators: Deterministic algorithms
- Example: linear congruential generators:

$$u_n = \frac{s_n}{M}, \quad s_{n+1} = (as_n + c) \mathrm{mod} M$$

General framework for Uniform RNG

(L'Ecuyer, 1994)



- s initial state ('seed')
- S finite set of states
- $T : S \to S$ is the transition function
- U finite set of output symbols
  (often $\{0, \ldots, m-1\}$ or a finite subset of $[0, 1]$)
- $G : S \to U$ output function
- $s_i := T(S_{i-1})$ and $u_i := G(s_i)$.
- output: $u_1, u_2, \ldots$

Some Notes for Uniform RNG

- S finite $\Longrightarrow$ u_i is periodic
- In practice: seed s often chosen by clock time as default.
- Good practice to be able to reproduce simulations:

  Save the seed!

- Default random number generator in R :
  Matsumoto, M. and Nishimura, T. (1998) Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Transactions on Modeling and Computer Simulation, 8, 3-30.
  The 'state' is a 624-dimensional set of 32-bit integers plus a current position in that set.

Quality of Random Number Generators

- "Random numbers should not be generated with a method chosen at random" (Knuth, 1981, p.5)
  Some old implementations were unreliable!
- Desirable properties of random number generators:
  - Statistical uniformity and unpredictability
  - Period Length
  - Efficiency
  - Theoretical Support
  - Repeatability, portability, jumping ahead, ease of implementation

  (more on this see e.g. Gentle (2003), L'Ecuyer (2004), L'Ecuyer (2006), Knuth (1998))
- Usually you will do well with generators in modern software (e.g. the default generators in R).
  Don't try to implement your own generator!
  (unless you have very good reasons)

Nonuniform Random Number Generation

- How to generate nonuniform random variables?
- Basic idea:

  Apply transformations to a stream of iid U[0,1] random variables

Inversion Method

- Let F be a cdf.
- Quantile function (essentially the inverse of the cdf):

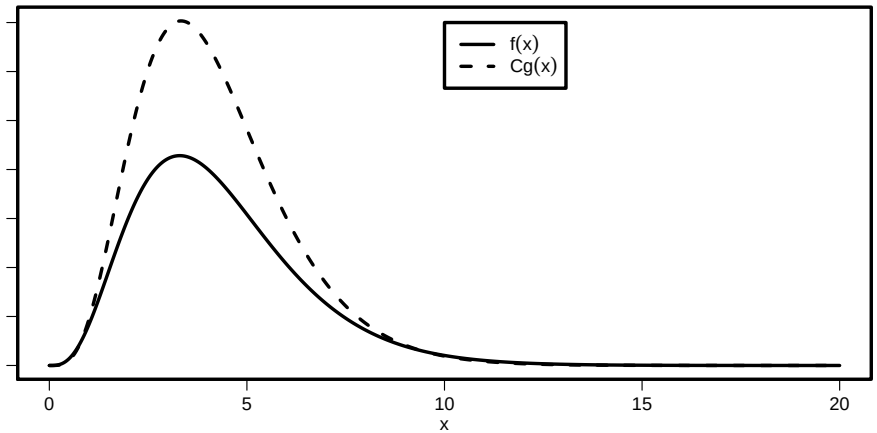
$$F^{-1}(u) = \inf\{x : F(x) \geq u\}$$

- If U is uniform on [0,1] then $F^{-1}(U) \sim F$. Indeed, assuming F is strictly increasing,

$$\mathrm{P}(X \leq x) = \mathrm{P}(F^{-1}(U) \leq x) = \mathrm{P}(U \leq F(x)) = F(x)$$

- Only works if F^{-1} (or a good approximation of it) is available. Numerical approximation if only F is available: Solve F(x)=U for x using your favourite numerical root-finder.

Acceptance-Rejection Method



- target density f
- Proposal density g (easy to generate from) such that for some $C < \infty$:

$$f(x) \leq Cg(x) \forall x$$

- Algorithm:
  1. Generate X from g.
  2. With probability $\frac{f(X)}{Cg(X)}$ return X - otherwise goto 1.
- $\frac{1}{C}=$ probability of acceptance - want it to be as close to 1 as possible.

Further Algorithms

Examples:

- Ratio-of-Uniforms
- Use of the characteristic function
- MCMC

For many of those techniques and techniques to simulate specific distributions see e.g. Gentle (2003).

Outline

Evaluation of an Integral

- Want to evaluate

$$I := \int_{[0,1]^d} g(x)dx$$

- Importance for statistics: computation of expected values (posterior means), probabilities (p-values), variances, normalising constants, ... For example, let X be a r.v. with pdf f. Then

$$\mathrm{E}(X) = \int xf(x)dx \quad \text{and} \quad \mathrm{P}(x \in A) = \int \mathrm{I}(x \in A)f(x)dx$$

- Often, d is large. In a random sample, often d =sample size.
- How to solve it?
  - Symbolical (programs such as Maple, Mathematica may help)
  - Numerical Integration
  - Quasi Monte Carlo
  - Monte Carlo Integration

Numerical Integration/Quadrature

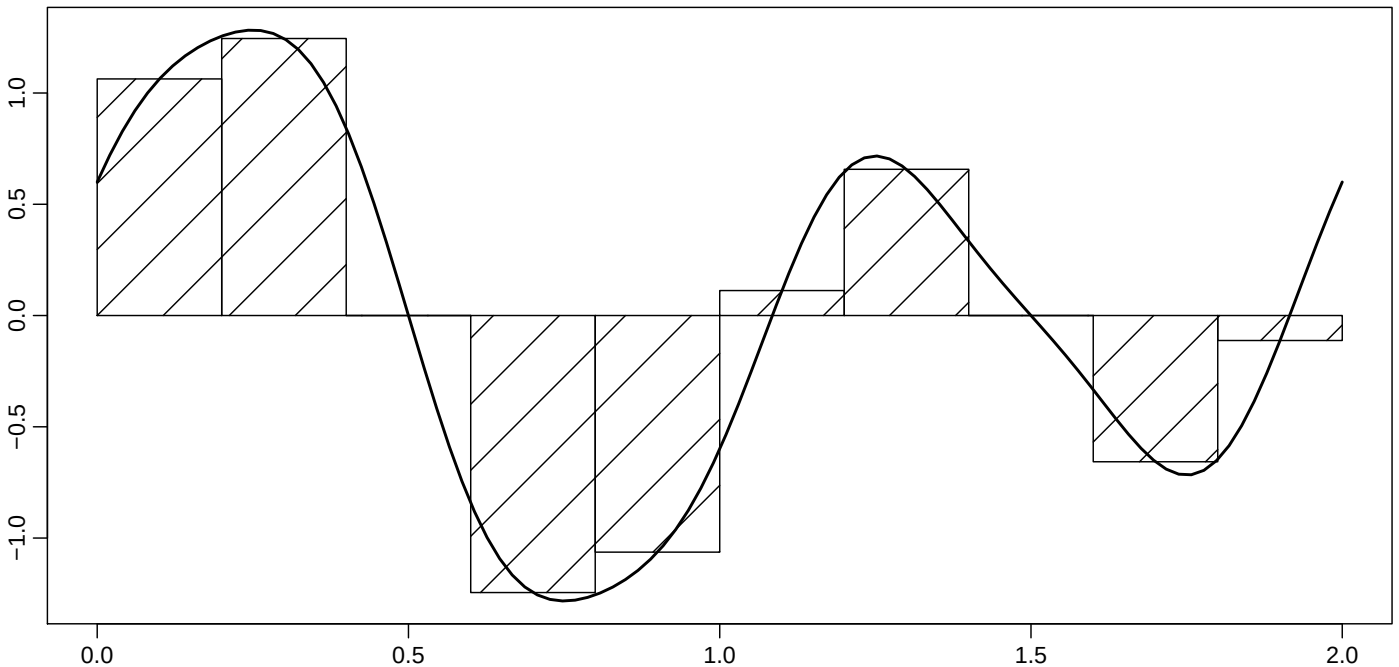
- Main idea: approximate the function locally with simple function/polynomials
- Advantage: good convergence rate
- Not useful for high dimensions - curse of dimensionality

Midpoint Formula

- Basic:

$$\int_0^1 f(x)dx \approx f\left(\frac{1}{2}\right)(1-0)$$

- Composite: apply the rule in n subintervals (similar to Riemann Sums) ▸ R-Demo



Error: $O(\frac{1}{n^2})$ if f is twice continuously differentiable.

Approximating $\int_a^b f(x)\,dx$ by $(b-a) f\left(\frac{a+b}{2}\right)$

If $f \in C_2[a,b]$ then

$$\left| \int_a^b f(x)dx - f\left(\frac{a+b}{2}\right)(b-a) \right| = \left| \int_a^b \left[f(x) - f\left(\frac{a+b}{2}\right) \right] dx \right|$$

Taylor

$$= \left| \underbrace{\int_a^b f'\left(\frac{a+b}{2}\right)\left(x-\frac{a+b}{2}\right) dx}_{=0} + \int_a^b f''(\xi_x) \frac{\left(x-\frac{a+b}{2}\right)^2}{2} dx \right|$$

$$\leq \sup_{x\in[a,b]} |f''(x)| \int_a^b \left(x-\frac{a+b}{2}\right)^2/2\, dx$$

$$\leq \frac{1}{24}(b-a)^3 \sup_{x\in[a,b]} |f''(x)|$$

Approximating $\int_0^1 f(x)\,dx$ by $\frac{1}{n}\sum_{i=1}^{n} f\left(\frac{2i-1}{2n}\right)$

$$\left| \int_0^1 f(x)\,dx - \frac{1}{n}\sum_{i=1}^{n} f\left(\frac{2i-1}{2n}\right) \right| \le \sum_{i=1}^{n} \left| \int_{\frac{i-1}{n}}^{i/n} f(x)\,dx - \frac{1}{n} f\left(\frac{2i-1}{2n}\right) \right|$$

$$\le \sum_{i=1}^{n} \frac{1}{24}\frac{1}{n^3} \sup_{x\in\left[\frac{i-1}{n},\,\frac{i}{n}\right]} |f''(x)|$$

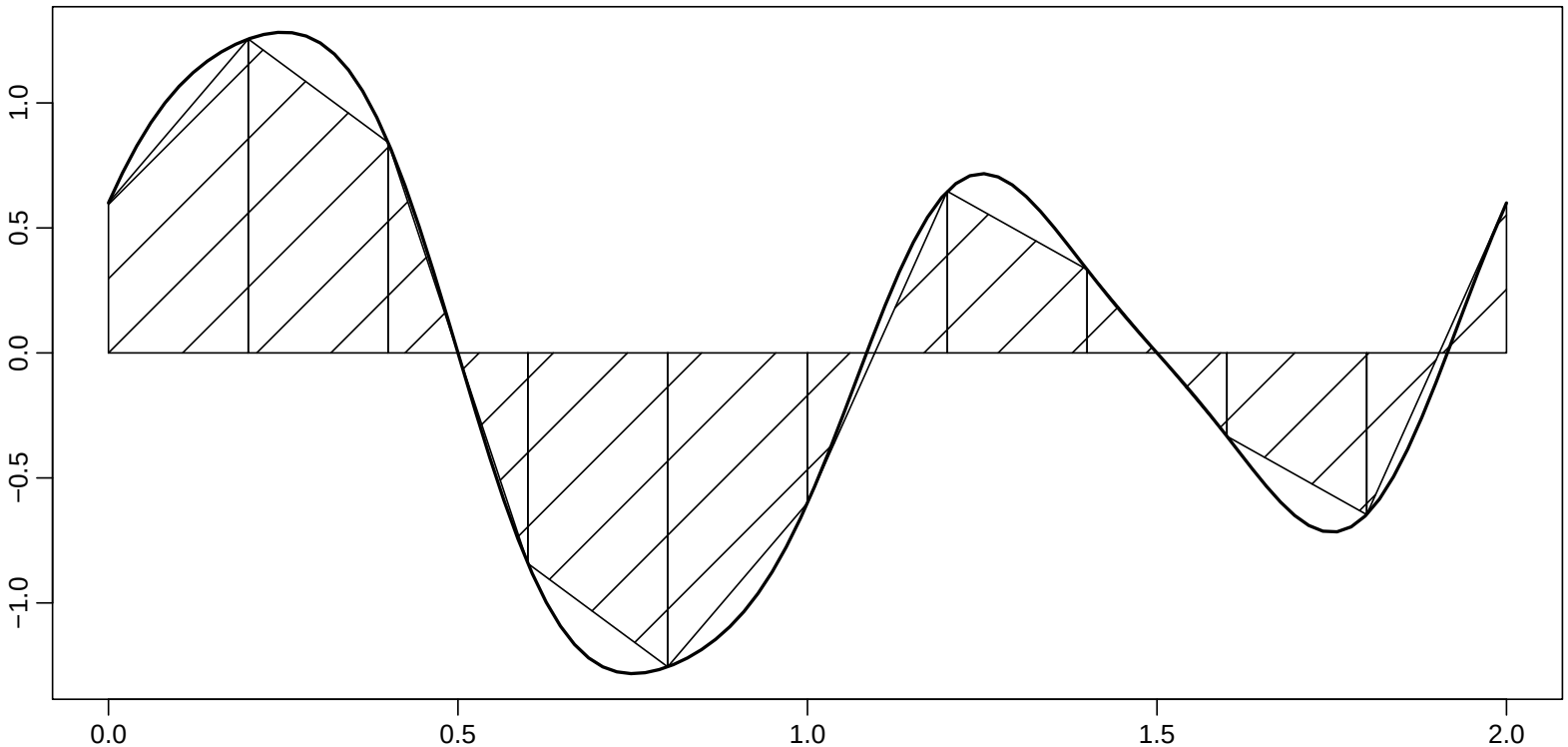
$$\le \frac{1}{24}\frac{1}{n^2} \sup_{x\in[0,1]} |f''(x)| = O\left(\frac{1}{n^2}\right)$$

Trapezoidal Formula

- Basic:

$$\int_0^1 f(x)dx \approx \frac{1}{2}(f(0) + f(1))$$

- Composite:



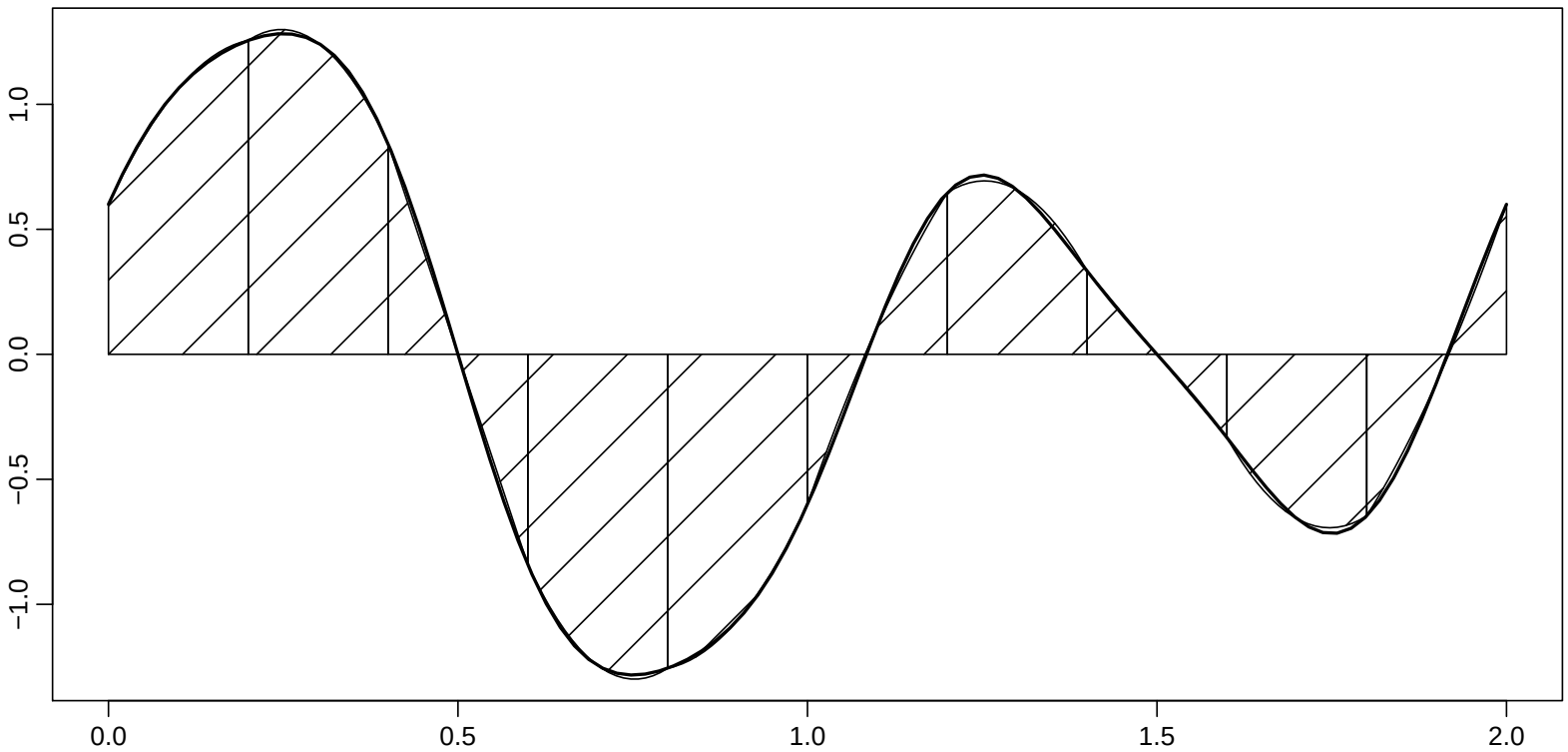
Error: $O(\frac{1}{n^2})$.

Simpson's rule

- Approximate the integrand by a quadratic function

$$\int_0^1 f(x)dx \approx \frac{1}{6}[f(0) + 4f(\frac{1}{2}) + f(1)]$$

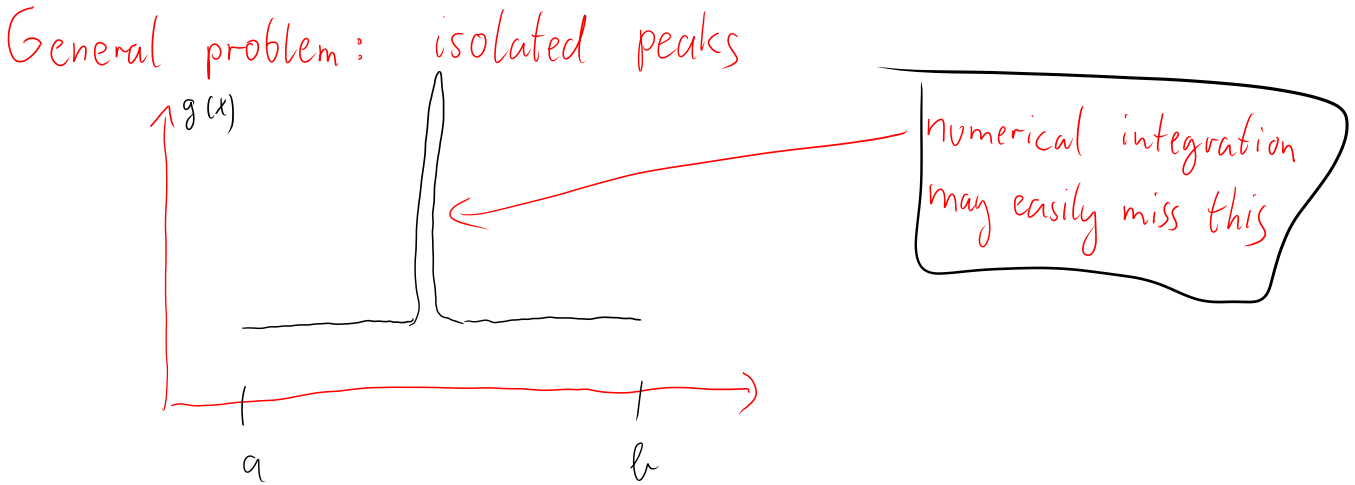
- Composite Simpson:



Error: $O(\frac{1}{n^4})$.

Advanced Numerical Integration Methods

- Newton Cotes formulas (assume existence of higher derivatives)
- Adaptive methods (more points in areas where function changes quickly)
- Unbounded integration interval: transformations
  (use substitution to transform unbounded interval to bounded interval)



▸ R-Demo

Curse of dimensionality - Numerical Integration in Higher Dimensions

$$I := \int_{[0,1]^d} g(x)dx$$

- Naive approach:
  - write as iterated integral
$$I := \int_0^1 \dots \int_0^1 g(x)dx_n \dots dx_1$$
  - use 1D scheme for each integral with, say g points .
  - $n = g^d$ function evaluations needed
    for $d = 100$ (a moderate sample size) and $g = 10$ (which is not a lot):
    $n >$ estimated number of atoms in the universe!
  - Suppose we use the trapezoidal rule, then the error $= O(\frac{1}{n^{2/d}})$
- More advanced schemes are not doing much better!

Monte Carlo Integration

$$\int_{[0,1]^d} g(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} g(X_i),$$

where $X_1, X_2, \cdots \sim U([0,1]^d)$ iid.

- SLLN:
$$\frac{1}{n}\sum_{i=1}^{n} g(X_i) \to \int_{[0,1]^d} g(x)dx \quad (n \to \infty)$$
- CLT: error is bounded by $O_P(\frac{1}{\sqrt{n}})$.
  independent of d
- Can easily compute asymptotic confidence intervals.
- Note: Trapezoidal rule has faster convergence for $d \leq 3$.

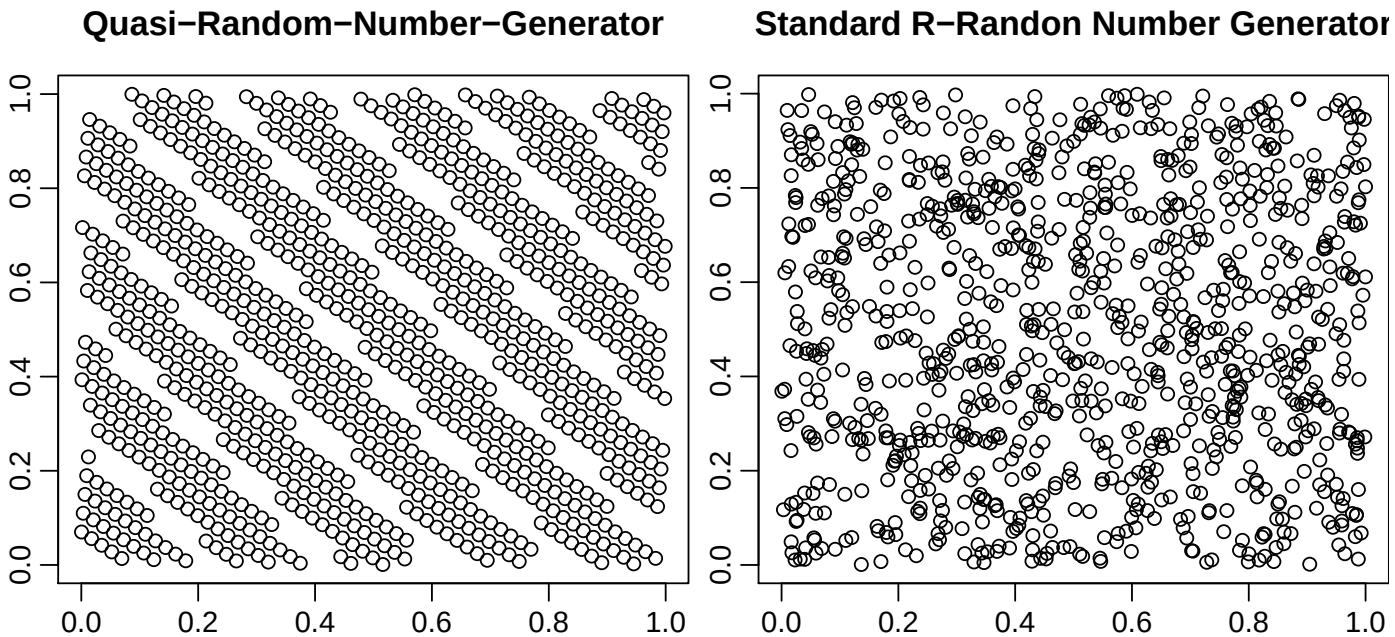
Quasi-Monte-Carlo

- Similar to MC, but instead of random X_i: Use deterministic x_i that fill $[0,1]^d$ evenly.
  so-called "low-discrepancy sequences".

R-package randtoolbox

Comparison between Quasi-Monte-Carlo and Monte Carlo - 2D

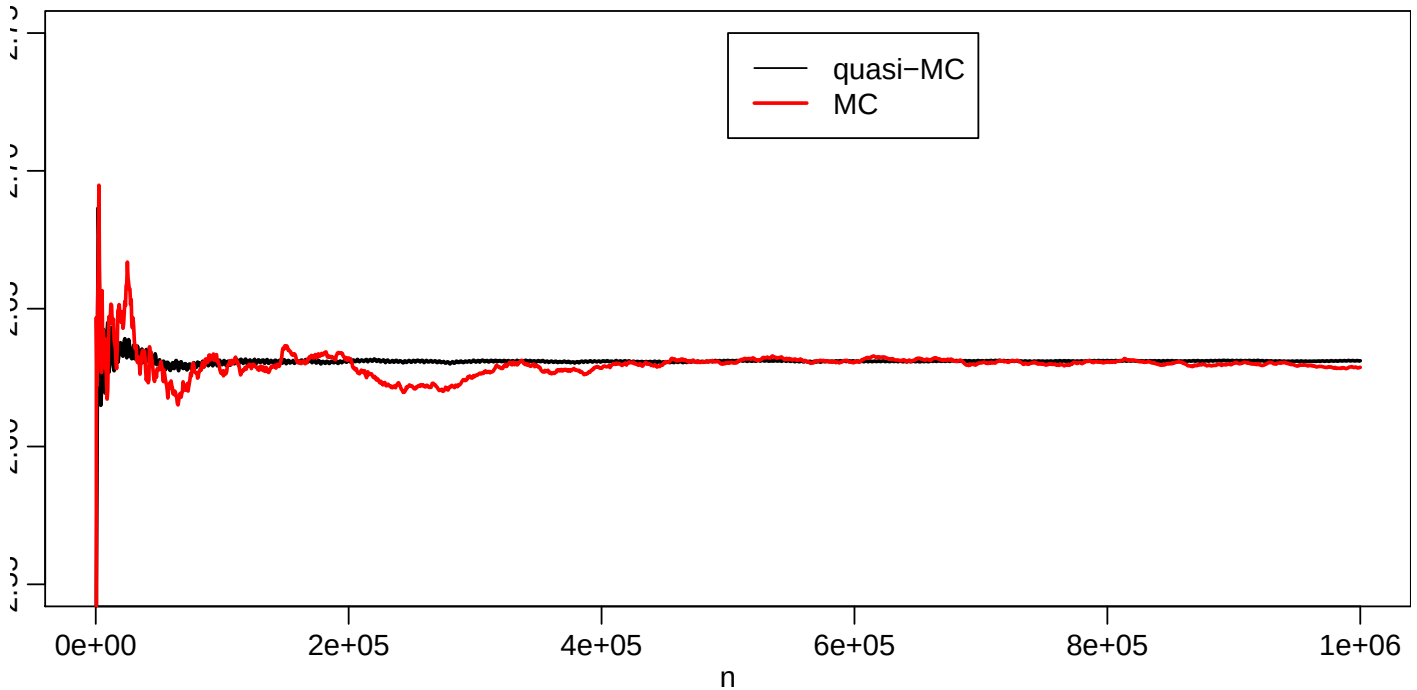
1000 Points in $[0,1]^2$ generated by a quasi-RNG and a Pseudo-RNG



Comparison between Quasi-Monte-Carlo and Monte Carlo

$$\int_{[0,1]^4} (x_1 + x_2)(x_2 + x_3)^2(x_3 + x_4)^3 dx$$

Using Monte-Carlo and Quasi-MC



Bounds on the Quasi-MC error

- Koksma-Hlawka inequality (Niederreiter, 1992, Theorem 2.11)

$$\|\frac{1}{n}\sum g(x_i) - \int_{[0,1]^d} g(x)dx\| \leq V_d(g)D_n^*,$$

where

$V_d(g)$ is the so-called Hardy and Krause variation of g and

D_n is the discrepancy of the points $x_1, \ldots, x_n$ in $[0,1]^d$ given by

$$D_n^* = \sup_{A \in \mathcal{A}} |\#\{x_i \in A : i = 1, \ldots, n\} - \lambda(A)|$$

where

- λ is Lebesgue measure
- $\mathcal{A}$ is the set of all subrectangles of $[0,1]^d$ of the form $\prod_{i=1}^d [0, a_i]^d$

- Many sequences have been suggested, e.g. the Halton sequence (other sequences: Faure, Sobol, ...) with:

Bounds on the Quasi-MC error

- Koksma-Hlawka inequality (Niederreiter, 1992, Theorem 2.11)

$$\|\frac{1}{n}\sum g(x_i) - \int_{[0,1]^d} g(x)dx\| \leq V_d(g)D_n^*,$$

- Many sequences have been suggested, e.g. the Halton sequence (other sequences: Faure, Sobol, ...) with:

$$D_n^* = O(\frac{\log(n)^{d-1}}{n})$$

$\rightarrow$ better convergence rate than MC integration!
However, it does depend on d

- Conjecture: for all sets of points D_n

$$D_n^* \geq O(\frac{\log(n)^{d-1}}{n})$$

(Niederreiter, 1992, p.32)

Comparison

▸ R-Demo

The consensus in the literature seems to be:

- use numerical integration for small d
- Quasi-MC useful for medium d
- use Monte Carlo integration for large d

Outline

Importance Sampling

- Main idea: Change the density we are sampling from.
- Interested in $\mathrm{E}(\phi(X)) = \int \phi(x) f(x) dx$
- For any density g,

$$\mathrm{E}(\phi(X)) = \int \phi(x) \frac{f(x)}{g(x)} g(x) dx$$

- Thus an unbiased estimator of $\mathrm{E}(\phi(X))$ is

$$\hat{I} = \frac{1}{n} \sum_{i=1}^{n} \phi(X_i) \frac{f(X_i)}{g(X_i)},$$

  where $X_1, \ldots, X_n \sim g$ iid.
- How to choose g?
  - Suppose $g \propto \phi f$ then $\mathrm{Var}(\hat{I}) = 0$.
    However, the corresponding normalizing constant is $\mathrm{E}(\phi(X))$, the quantity we want to estimate!
  - A lot of theoretical work is based on large deviation theory.

Importance sampling - Comments

- Importance sampling can greatly reduce the variance for estimating the probability of rare events, i.e. $\phi(x) = \mathtt{I}(x \in A)$ and $\mathrm{E}(\phi(X)) = \mathrm{P}(X \in A)$ small.
- It is not just useful for variance reduction - it can also be very useful to generate random variables.
- Be careful with support/tails!

Control Variates

- Interested in $I = \mathrm{E}\, X$
- Suppose we can also observe Y and know $\mathrm{E}\, Y$.
- Consider $T = X + a(Y - \mathrm{E}(Y))$
- Then $\mathrm{E}\, T = I$ and

$$\mathrm{Var}\, T = \mathrm{Var}\, X + 2a\, \mathrm{Cov}(X, Y) + a^2\, \mathrm{Var}\, Y$$

  Minimized for $a = -\frac{\mathrm{Cov}(X,Y)}{\mathrm{Var}\, Y}$.
- usually, a not known $\rightarrow$ estimate
- For Monte Carlo sampling:
  - generate iid sample $(X_1, Y_1), \ldots, (X_n, Y_n)$
  - estimate $\mathrm{Cov}(X, Y)$, $\mathrm{Var}\, Y$ based on this sample $\rightarrow \hat{a}$
  - $\hat{I} = \frac{1}{n} \sum_{i=1}^{n} [X_i + \hat{a}(Y_i - \mathrm{E}(Y))]$
- $\hat{I}$ can be computed via standard regression analysis.
  Hence the term"regression-adjusted control variates".
- Can be easily generalised to several control variates.

Further Techniques

- If density symmetric around point: Antithetic Sampling
  Use X and $-X$ if symmetric around 0.
- Conditional Monte Carlo
  Evaluate parts explicitly
- Common Random Numbers
  For comparing two procedures - use the same sequence of random numbers.
- Stratification
  - Divide sample space Ω into strata $\Omega_1, \ldots, \Omega_s$
  - In each strata, generate R_i replicates conditional on Ω_i and obtain an estimates $\hat{I}_i$
  - Combine using the law of total probability:

$$\hat{I} = p_1 \hat{I}_1 + \cdots + p_s \hat{I}_s$$

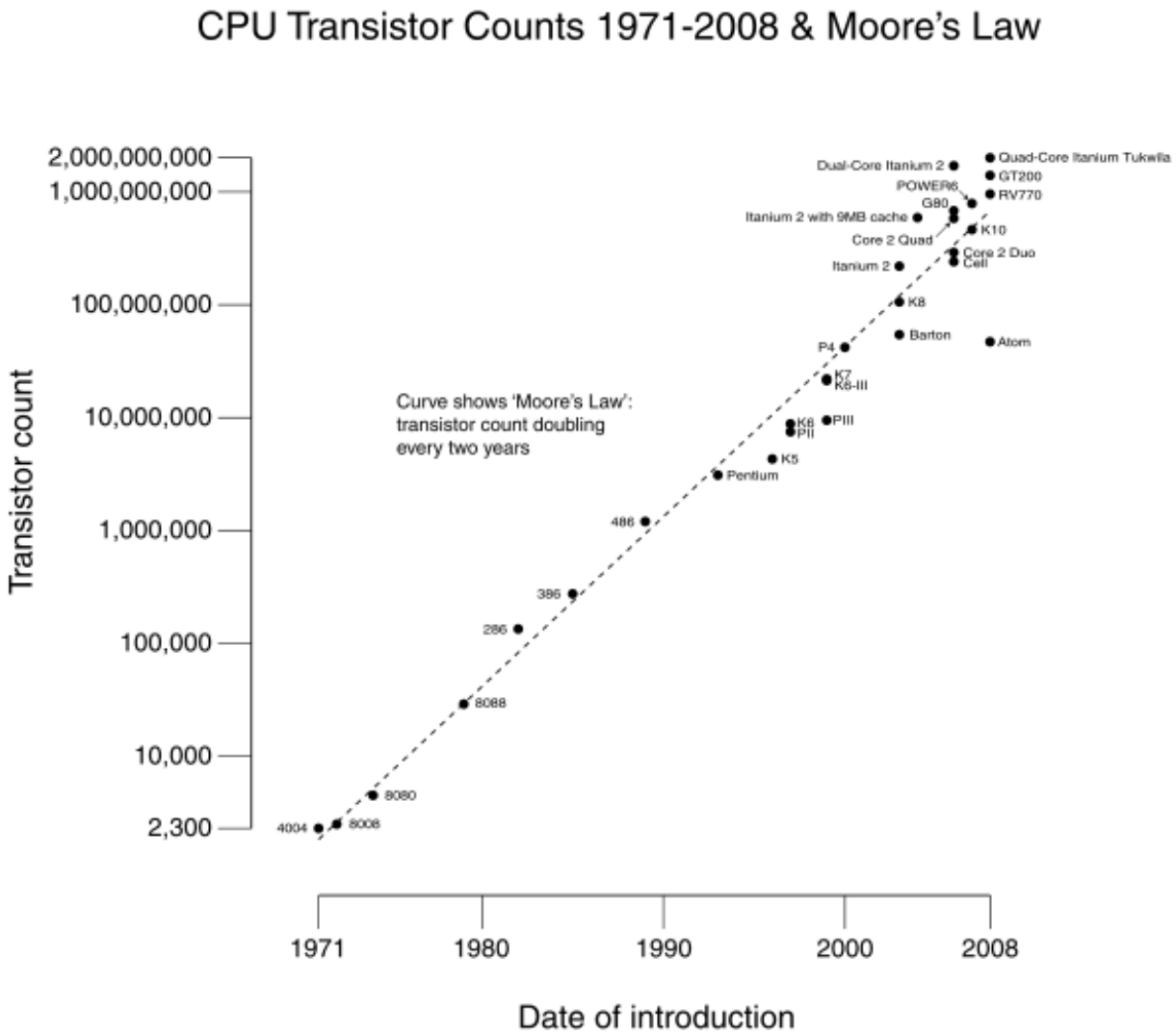
  - Need to know $p_i = \mathrm{P}(\Omega_i)$ for all i

Part II

Parallel Computing

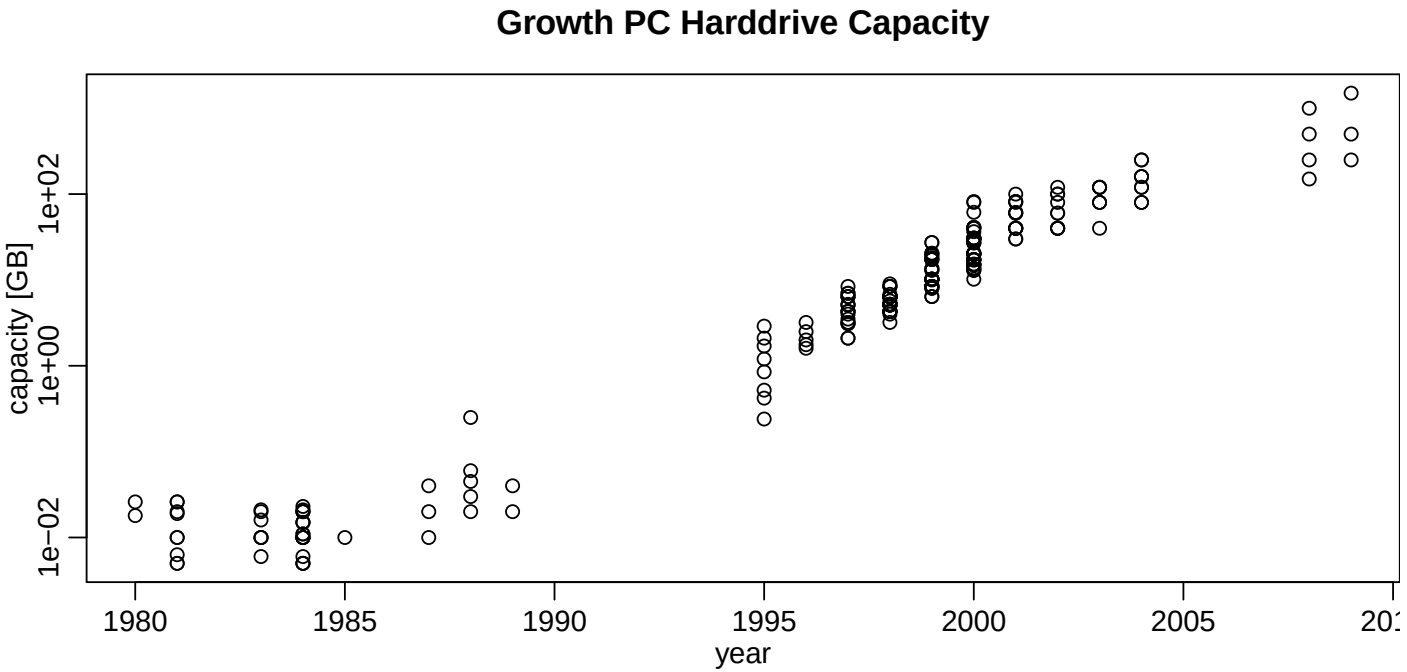
Outline

Moore's Law



(Source: Wikipedia, Creative Commons Attribution ShareAlike 3.0 License)

Growth of Data Storage



- Not only the computer speed but also the data size is increasing exponentially!
- The increase in the available storage is at least as fast as the increase in computing power.

Introduction

- Recently: Less increase in CPU clock speed
- $\rightarrow$ multi core CPUs are available (eight cores readily available - 80 cores in labs)
- $\rightarrow$ software needs to be adapted to exploit this

- Traditional computing:
  Problem is broken into small steps that are executed sequentially
- Parallel computing:
  Steps are being executed in parallel

von Neumann Architecture

- CPU executes a stored program that specifies a sequence of read and write operations on the memory.
- Memory is used to store both program and data instructions
- Program instructions are coded data which tell the computer to do something
- Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then sequentially performs them.

Different Architectures

- Multicore computing
- Symmetric multiprocessing
- Distributed Computing
  - Cluster computing
  - Massive Parallel processor
  - Grid Computing

List of top 500 supercomputers at `http://www.top500.org/`

Flynn's taxonomy

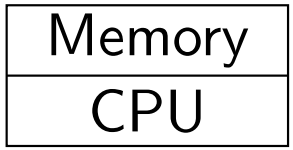
	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

Examples:

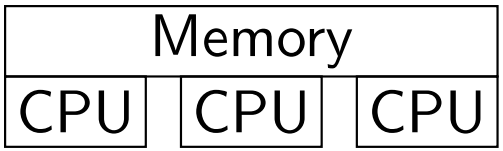
- SIMD: GPUs

Memory Architectures of Parallel Computers

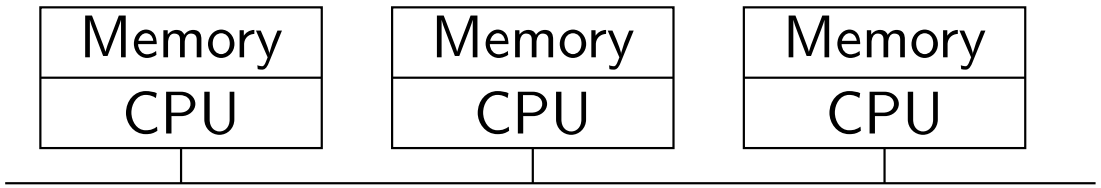
- Traditional System



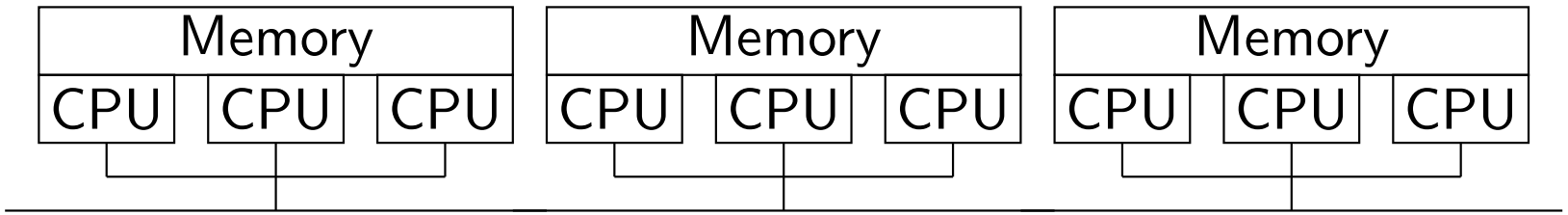
- Shared Memory System



- Distributed Memory System



- Distributed Shared Memory System



Embarrassingly Parallel Computations

Task can be divided into parts that can be executed separatedly.
Examples:

- Monte Carlo Integration
- Bootstrap
- Cross-Validation

Note: MCMC methods do not fall (easily) into this category.

Speedup

Ideally: computational time reduced linearly in the number of CPUs

- Suppose only a fraction p of the total tasks can be parallelized.
- Supposing we have n parallel CPUs, the speedup is

$$\frac{1}{(1-p)+p/n} \quad \text{(Amdahl's Law)}$$

$\rightarrow$ no infinite speedup possible.

Example

$p = 90\%$, maximum speed up by a factor of 10.

Speedup

Ideally: computational time reduced linearly in the number of CPUs

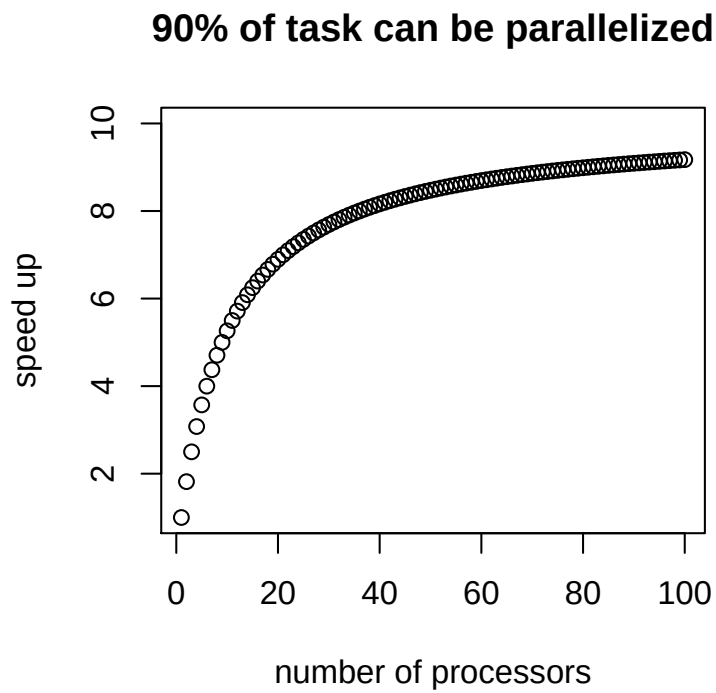
- Suppose only a fraction p of the total tasks can be parallelized.
- Supposing we have n parallel CPUs, the speedup is

$$\frac{1}{(1-p)+p/n} \quad \text{(Amdahl's Law)}$$

$\rightarrow$ no infinite speedup possible.

Example

$p = 90\%$, maximum speed up by a factor of 10.



Communication between processes

- Forking
- Threading
- OpenMP (good for multicore machines)
  shared memory multiprocessing
- PVM (Parallel Virtual Machine)
- MPI (Message Passing Interface; de facto standard for large scale parallel computations)
- For "Big Data": Hadoop and related approaches.

How to divide tasks? e.g. Master/Slave concept

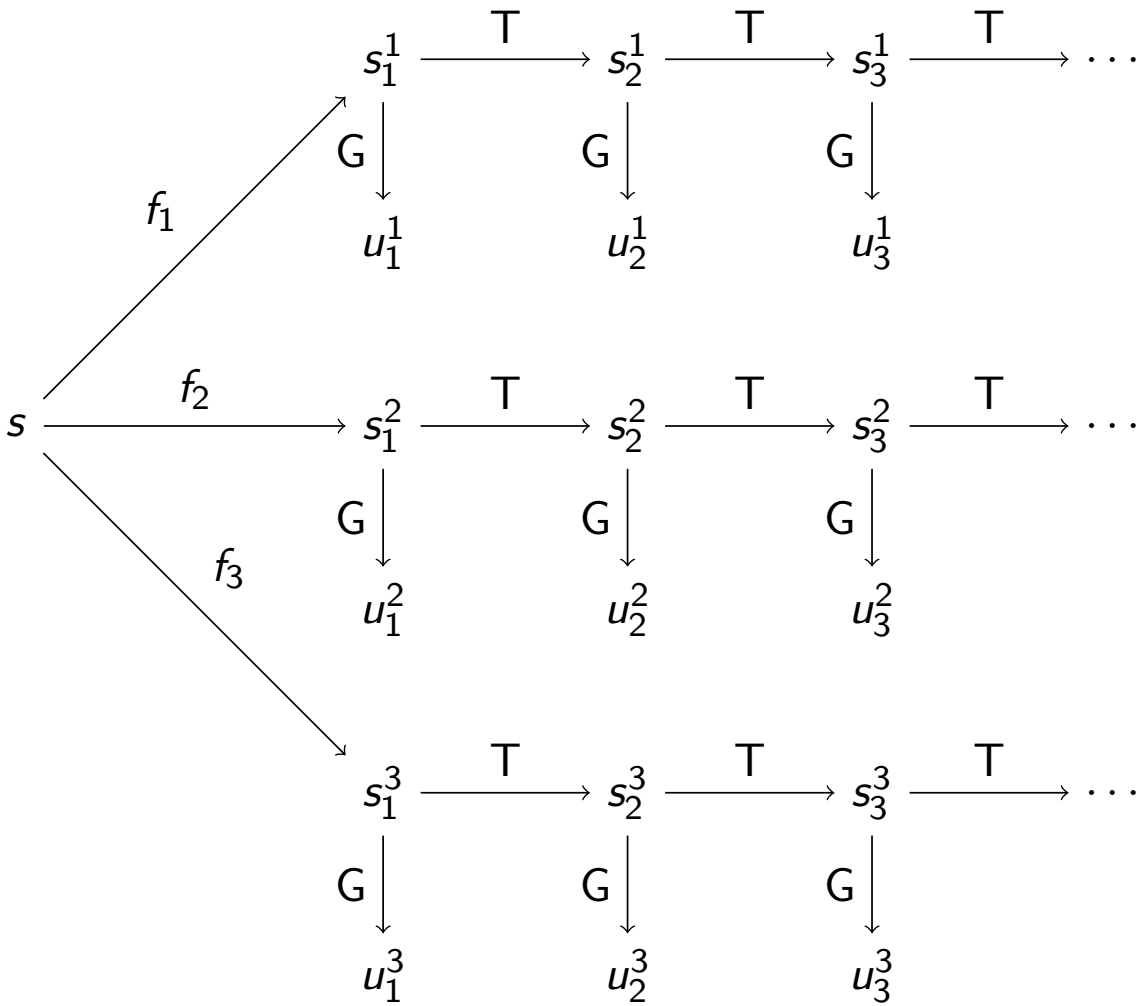
Outline

Parallel Random Number Generation

Problems with RNG on parallel computers

- Cannot use identical streams
- Sharing a single stream: a lot of overhead.
- Starting from different seeds: danger of overlapping streams (in particular if seeding is not sophisticated or simulation is large)
- Need independent streams on each processor...

Parallel Random Number Generation - sketch of general approach



Packages in R for Parallen random Number Generation

rsprng Interface to the scalable parallel random number generators library (SPRNG) `http://sprng.cs.fsu.edu/`

rlecuyer Essentially starts with one random stream and partitions it into long substreams by jumping ahead. L'Ecuyer et al. (2002)

Outline

Profile

- Determine what part of the programme uses most time with a profiler
- Improve the important parts (usually the innermost loop)
- R has a built-in profiler (see Rprof, Rprof.summary, package profr)

Use Vectorization instead of Loops

```
> a <- rnorm(1e7);
> system.time({x <- 0; for (i in 1:length(a)) x <- x+a[i]})
```

```
elapsed
    8.5
```

```
> system.time(sum(a))[3]
```

```
elapsed
   0.02
```

Just in time compilation - the compiler package I

- compiler package: can pre-compile R code into "Byte Code"
- R core - the base and recommended packages are now byte-compiled by default.

```
> library(compiler)
> f <- function(i){j <- 0;for (i in 1:10000) j <- j+i;j}
> system.time(replicate(1000,f()))
   user  system elapsed
   4.98    0.00    4.98
> fc <- cmpfun(f)
> system.time(replicate(1000,fc()))
   user  system elapsed
   0.44    0.00    0.43
```

Just in time compilation - the compiler package II

This is an extreme example; the speedup will usually be not as big. Can enable this functionality automatically via

```
require(compiler)
enableJIT(3)
```

- Other JIT implementation: RA; Not just a package - central parts are reimplemented. (`http://www.milbo.users.sonic.net/ra/`); need to install Ra instead of R (Ra has not been updated for some time).
- Bill Venables (on R help archive): "if you really want to write R code as you might C code, then jit can help make it practical in terms of time. On the other hand, if you want to write R code using as much of the inbuilt operators as you have, then you can possibly still do things better."

Use Compiled Code

- R is an interpreted language.
- Can include C, C++ and Fortran code.
- Can dramaticallly speed up computationally intensive parts (a factor of 100 is possible)
- No speedup if the computationally part is a vector/matrix operation.
- Downside: decreased portability, longer programming time
- Helpful libraries: Rcpp

R-package: parallel

- Part of R core since R version 2.14.
- Mostly for "embarassingly parallel" computations
- Extends the "apply"-style function to a cluster of machines

```
> detectCores()
[1] 4
> cl <- makeCluster(4)
> f <-  function(i) mean(replicate(10000,mean(rnorm(10000,me
> parSapply(cl,1:4,FUN=f)
[1] 1.000165 2.000036 3.000067 3.999995
> system.time(parSapply(cl,1:4,FUN=f))
   user  system elapsed
   0.00    0.00   16.24
> system.time(sapply(1:4,FUN=f))
   user  system elapsed
  44.32    0.01   44.63
> stopCluster(cl)
```

Random number generator

parallel RNG is being set up automatically

```
> cl<-makeCluster(2)
> parLapply(cl,1:2,function(i) rnorm(3))
[[1]]
[1] -0.1490175  1.4870953 -0.4753602

[[2]]
[1] -1.139051  0.202475  1.184057

> stopCluster(cl)
```

Rmpi

For more complicated parallel algorithms that are not embarassingly parallel.
Tutorial under http://math.acadiau.ca/ACMMaC/Rmpi/
Hello world from this tutorial

```
# Load the R MPI package if it is not already loaded.
if (!is.loaded("mpi_initialize")) {
library("Rmpi") }

# Spawn as many slaves as possible
mpi.spawn.Rslaves()

# In case R exits unexpectedly, have it automatically clean up
# resources taken up by Rmpi (slaves, memory, etc...)
.Last <- function(){
if (is.loaded("mpi_initialize")){
if (mpi.comm.size(1) > 0){
print("Please use mpi.close.Rslaves() to close slaves.")
mpi.close.Rslaves()
}
 print("Please use mpi.quit() to quit R")
.Call("mpi_finalize") } }

# Tell all slaves to return a message identifying themselves
mpi.remote.exec(paste("I am",mpi.comm.rank(),"of",mpi.comm.size()))
# Tell all slaves to close down, and exit the program
mpi.close.Rslaves()
mpi.quit()
```

(not able to install under win from CRAN - install from http://www.stats.uwo.ca/faculty/yu/Rmpi/)

Some other Packages

multicore Use of parallel computing on a single machine via fork (Unix, MacOS) - very fast and easy to use.

GridR `http://cran.r-project.org/web/packages/GridR/` Wegener et al. (2009, Future Generation Computer Systems)

rparallel `http://www.rparallel.org/`

GPUs

- graphical processing units - in graphics cards
- very good at parallel processing
- need to taylor to specific GPU.
- Packages in R:

  gputools several basic routines.

  cudaBayesreg Bayesian multilevel modeling for fMRI.

Further Reading

- A tutorial on Parallel Computing: `https://computing.llnl.gov/tutorials/parallel_comp/`
- High Performance Computing task view on CRAN `http://cran.r-project.org/web/views/HighPerformanceComputing.html`
- A talk on high performance comuting with R: `http://dirk.eddelbuettel.com/papers/useR2010hpcTutorial.pdf`

Part III

Appendix

Topics in the coming lectures:

- Optimisation
- MCMC methods
- Bootstrap
- Particle Filtering

References I

Gentle, J. (2003). *Random Number Generation and Monte Carlo Methods*. Springer.

Knuth, D. (1981). *The art of computer programming. Vol. 2: Seminumerical algorithms*. Addison-Wesley.

Knuth, G. (1998). *The Art of Computer Programming, Seminumerical Algorithms, Vol.2*.

L'Ecuyer, P. (1994). Uniform random number generation. *Annals of Operations Research* **53**, 77–120.

L'Ecuyer, P. (2004). Random number generation. In *Handbook of Computational Statistics: Concepts and Methods*, Springer.

L'Ecuyer, P. (2006). Uniform random number generation. In *Handbooks in Operations Research and Management Science*, Elsevier.

L'Ecuyer, P., Simard, R., Chen, E. J. & Kelton, W. D. (2002). An objected-oriented random-number package with many long streams and substreams. *Operations Research* **50**, 1073–1075. The code in C, C++, Java, and FORTRAN is available.

References II

Niederreiter, H. (1992). *Random number generation and quasi-Monte Carlo methods*. SIAM.