LEIBNIZ
RECHEN
ZENTRUM

R/openMP binding

F. Jamitzky

Leibniz Supercomputing Centre, Garching

jamitzky@lrz.de

Why ROMP?

- Put R on the Supercomputer (1000s of cores)
- Start R on each core? slow!
- Lightweight approach: openMP
- R Syntax to Fortran Converter
- Accelerate R code by compilation
- Parallelize R code by vectorization
- Speedup by Compilation: ~100
- Speedup by Vectorization: ~100
- Total Speedup: ~10000

Why R?

- Very high abstraction level
- Lisp roots – “code that writes code”
- Interactivity – “Instant gratification”
- Fast prototyping language
- Huge Libraries – “Batteries included”
- Graphics and Plots – “nice and shiny”

Why Fortran?

- Well suited for numerical programming (very fast)
- Array arithmetics (syntax similar to R)
- Excellent R bindings (parts of R are written in Fortran)

Why openMP?

- Abstraction for vector processing
- Excelent Fortran bindings
  (Fortran and C are reference languages)
- Standard in high performance computing
- Excelent implementations, Fortran/openMP compiler from:
  GNU, Intel, IBM, NAG, Microsoft (no more),
- Generated code for many different CPUs and OSs.

Philosophy

- Use functional programming style
- Use closures
- R functions to Fortran functions in the "`contains`" part.
- Higher order functions: map/reduce
- Translate map/reduce to openMP for/reduce pragmas (uses the gsubfn package from http://code.google.com/p/gsubfn)

Abstractions

- R functions are translated to “pure” functions in Fortran
- R “`sum`” is replaced by “`sum.mp`”
- R “`apply`” is replaced by “`apply.mp`”
- Typing required, implemented types: `int, double`

Example

- Compute distance of two vectors:

```
x <- as.double(runif(100))
y <- as.double(runif(100))
for(i in 1:100) res <- res+(x[i]-y[i])**2
```

- Using ROMP calls:

```
sum.mp(dosum,(x[i]-y[i])**2, dbl(),i=1:100)
dosum.f <- compile.mp(dosum(),
dbl(),x=dbl(100),y=dbl(100))
dosum.f(res=res, x=x, y=y)
```

Non-trivial Example: Pointwise Fractal Dimension

Compute pointwise dimension
of a cloud of points

Let N be the density of points at location x

$$N(\boldsymbol{x}_i, r) = \sum_j \Theta(r - |\boldsymbol{x}_j - \boldsymbol{x}_i|).$$

where each point is smoothed with radius r

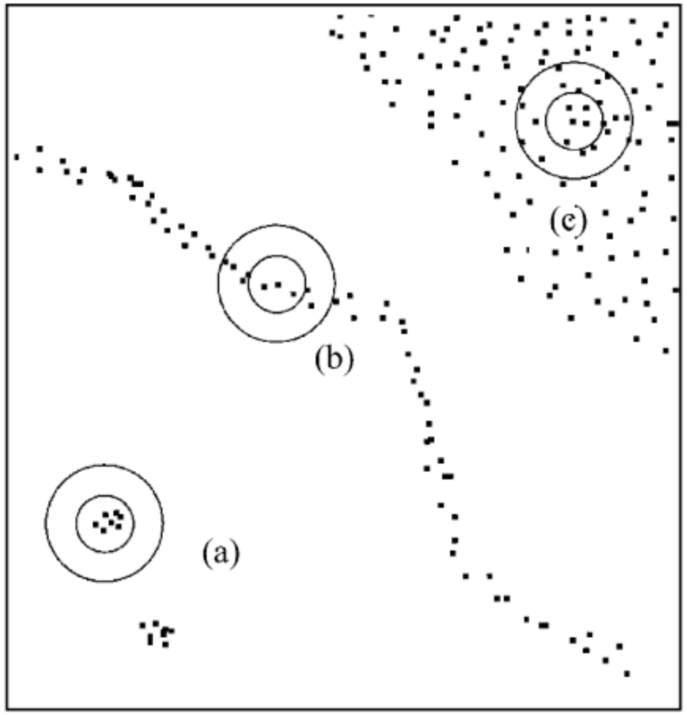


Fig. 1. Scheme illustrating the different dimensionality of point distributions. (a) A point-like structure. (b) A line-like structure. (c) An area-like structure.

The fractal pointwise dimension is then defined as:

$$\alpha_i = (\log N(\boldsymbol{x}_i, r_2) - \log N(\boldsymbol{x}_i, r_1))/(\log r_2 - \log r_1)$$

Ref: Local Scaling Properties for Diagnostic Purposes by W. Bunk, F. Jamitzky, R. Pompl, C. Rath and G. Morfill, Springer 2002

Pure R style (verbose)

- Compute local density of point set:

```
dist <-
   function(i,j,x,r)
   ifelse(sum((x[i,1:ndim]-x[j,1:ndim])**2)>r**2,0,1)

dens_one <-
   function(j,x,r)
   sum(sapply(1:np, function(i) dist(i,j,x,r)))

comp.dens <-
   function(x,r)
   sapply(1:np, function(j) dens_one(j,x,r))

comp.dens(x, r=0.1)
```

“ROMP in style”

- Compute local density of point set:

```
sum.mp(dens_one,
ifelse(sum((x[i,1:ndim]-x[j,1:ndim])**2)>r**2,0,1),
int(), i=1:np, j=int())

apply.mp(dens, dens_one(j), int(np), j=1:np)

comp.dens <-compile.mp( dens(),
int(np),x=dbl(np,ndim),r=dbl(),ndim=int(),np=int())

comp.dens(x, r=0.1, ndim=3, np=100000)
```

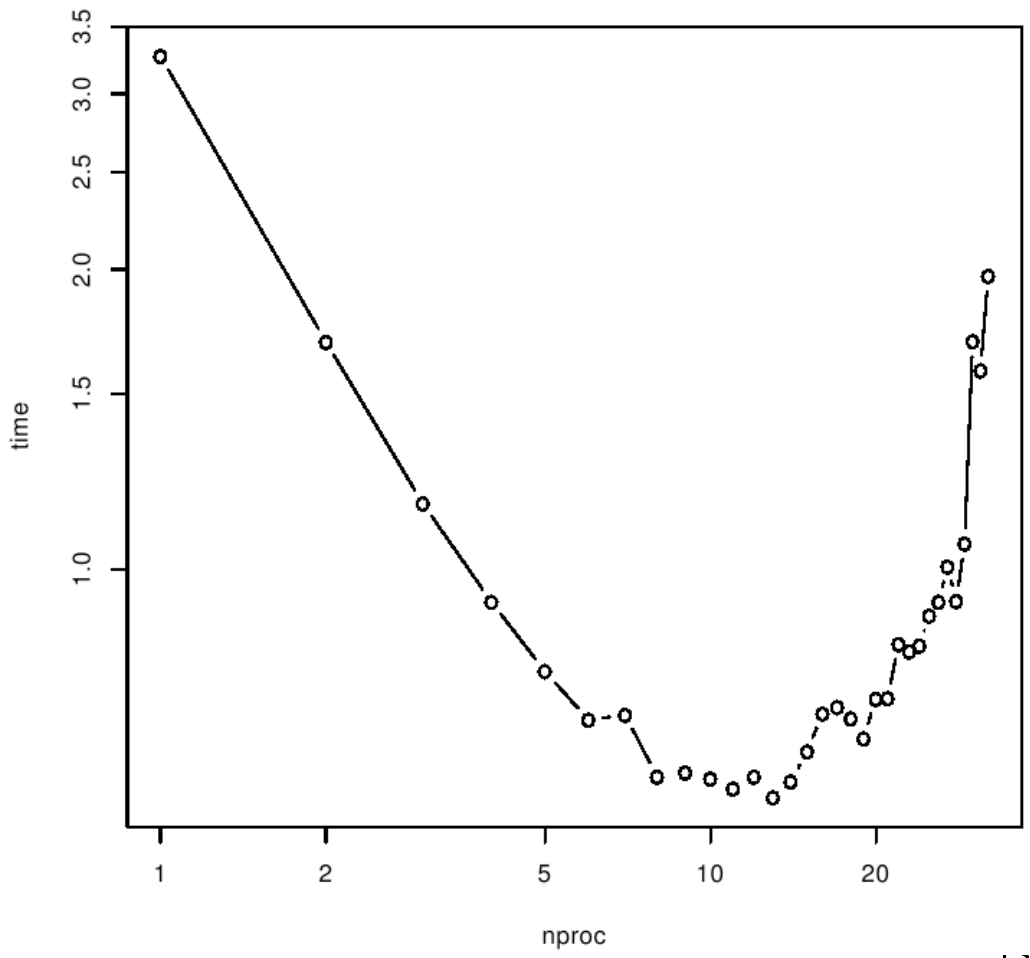
Benchmarks

- openmp on SGI Altix 4700 / 512 cores
- np=10000
- Pure R: time = 21800s = 6h!!
- ROMP: nproc=1 time = 3.2s
- ROMP: nproc=8 time = 0.6s

Acceleration factor: >30000 !!

Benchmarks ROMP

- ROMP on HLRBII
- npoints=10000
- nproc < 32
- scaling up to 10 cores
- due to small problem size
- use “first touch”



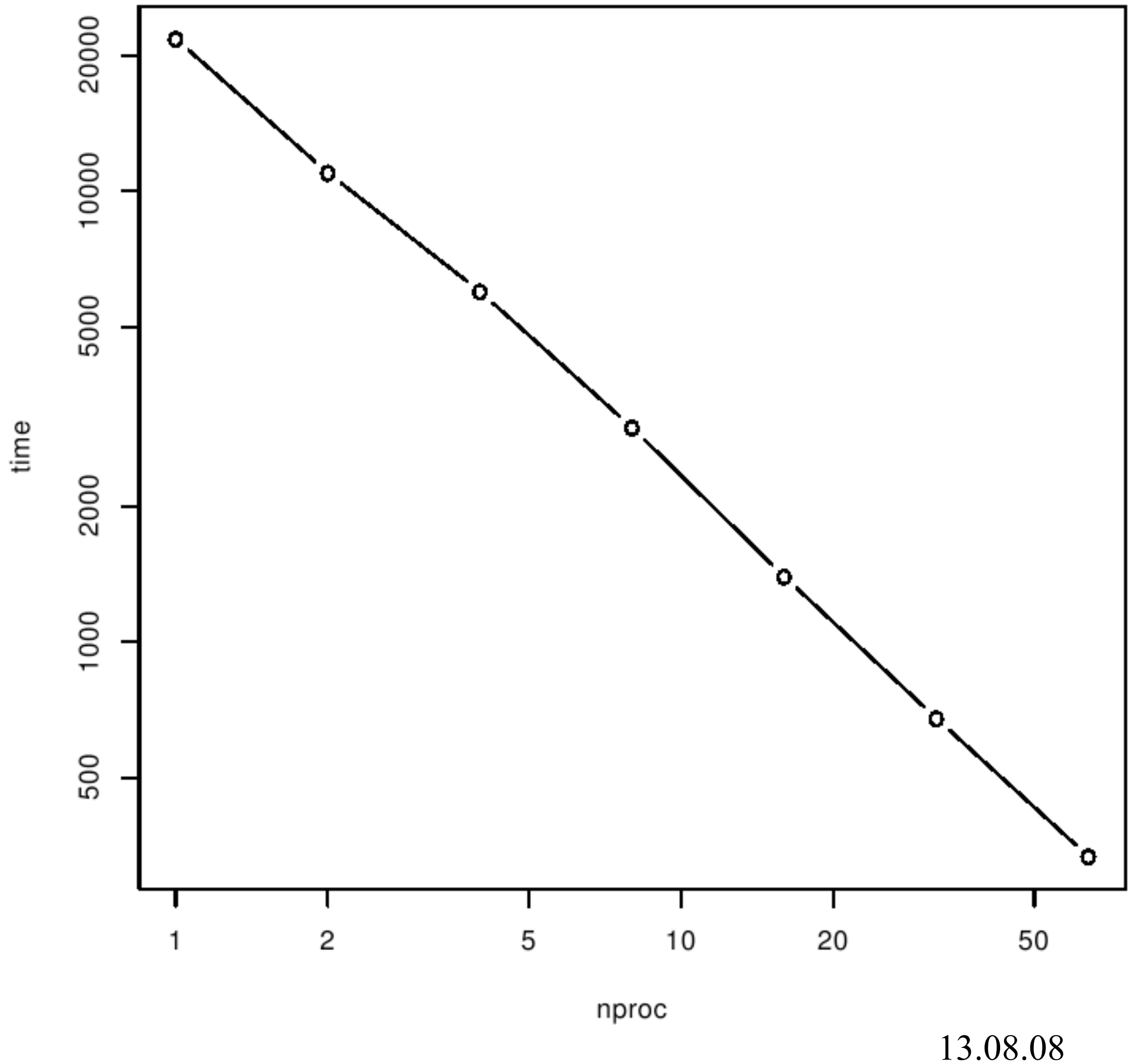
Rmpi

- Rmpi `http://www.stats.uwo.ca/faculty/yu/Rmpi`
- Rmpi: spawn R interpreter on each core
- `applyLB` MPI with load balancing

```
library(Rmpi)
mpi.bcast.Robj2slave(x)
mpi.applyLB(1:np,
 function(i)
 sum(sapply(1:np,
  function(j)
  ifelse(sum((x[i,1:ndim]-x[j,1:ndim])**2)>r**2,0,1)
  ))))
```

Benchmark Rmpi

- Rmpi on HLRBII
- SGI MPI
- npoints=10000
- nproc < 64
- strong scaling
  up to 100s cores



Summary and Outlook

- ROMP scales up to ~100 cores (SMP)
- Acceleration factor up to 10000
- Pre Alpha Version
- Combination Rmpi+ROMP?
- Extending map/reduce: Use monads?
- Type inference aka automatic typing?

Download from:

http://code.google.com/p/romp