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
- Excellent Fortran bindings
  (Fortran and C are reference languages)
- Standard in high performance computing
- Excellent 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:
  \[
  x \leftarrow \text{as.double}(\text{runif}(100))
  \]
  \[
  y \leftarrow \text{as.double}(\text{runif}(100))
  \]
  \[
  \text{for}(i \in 1:100) \quad \text{res} \leftarrow \text{res} + (x[i] - y[i])^2
  \]

- Using ROMP calls:
  \[
  \text{sum.mp}(\text{dosum}, (x[i] - y[i])^2, \text{dbl}(), i=1:100)
  \]
  \[
  \text{dosum.f} \leftarrow \text{compile.mp}(\text{dosum}(), \text{dbl}(), x=\text{dbl}(100), y=\text{dbl}(100))
  \]
  \[
  \text{dosum.f}(\text{res}=\text{res}, x=x, y=y)
  \]
Non-trivial Example: Pointwise Fractal Dimension

Compute pointwise dimension of a cloud of points

\[ N(x_i, r) = \sum_j \Theta(r - |x_j - x_i|). \]

where each point is smoothed with radius \( r \)

The fractal pointwise dimension is then defined as:

\[ \alpha_i = \frac{\log N(x_i, r_2) - \log N(x_i, r_1)}{\log r_2 - \log r_1} \]

Pure R style (verbose)

- Compute local density of point set:
  
  ```R
  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:

```r
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())
```

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

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

```r
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”

![Graph showing time vs. nproc]
Rmpi

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

```r
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

- download the latest version from:

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

  and find more information at:

  http://romp.r-forge.r-project.org