R, Rcpp and Parallel Computing

Notes from our Rcpp Experience

Dirk Eddelbuettel and JJ Allaire

Jan 26-27, 2015
Workshop for Distributed Computing in R
HP Research, Palo Alto, CA

Outline

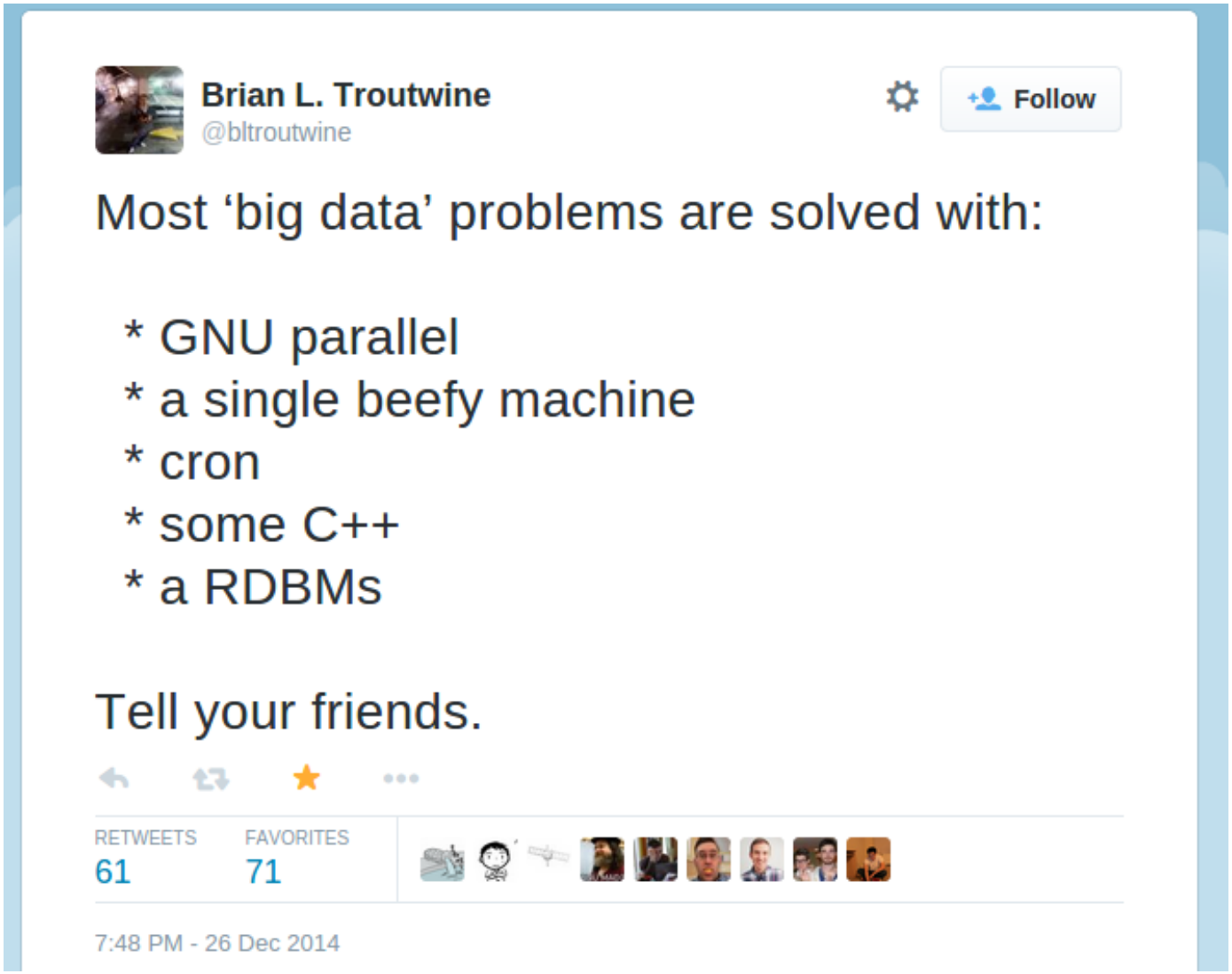
One View on Parallel Computing

The whole "let's parallelize" thing is a huge waste of everybody's time. There's this huge body of "knowledge" that parallel is somehow more efficient, and that whole huge body is pure and utter garbage. Big caches are efficient. Parallel stupid small cores without caches are horrible unless you have a very specific load that is hugely regular (ie graphics).
[...]
Give it up. The whole "parallel computing is the future" is a bunch of crock.

Linus Torvalds, Dec 2014

Another View on Big Data

Imagine a `gsub("DBMs", "", tweet)` to complement further...



Outline

CRAN Task View on HPC

http://cran.r-project.org/web/views/HighPerformanceComputing.html

Things R does well:

- Package snow by Tierney et al a trailblazer
- Package Rmpi by Yu equally important
- multicore / snow / parallel even work on Windows
- Hundreds of applications
- *It just works* for data-parallel tasks

Outline

Rcpp: Early Days

In the fairly early days of Rcpp, we also put out RInside as a simple C++ class wrapper around the R-embedding API.

It got one clever patch taking this (ie: R wrapped in C++ with its own `main()` function) and encapsulating it within MPI.

HP Vertica also uses Rcpp and RInside in DistributedR.

Rcpp: More recently

Rcpp is now easy to deploy; Rcpp Attributes played a key role:

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

// [[Rcpp::export]]
double piSugar(const int N) {
    NumericVector x = runif(N);
    NumericVector y = runif(N);
    NumericVector d = sqrt(x*x + y*y);
    return 4.0 * sum(d < 1.0) / N;
}
```

Rcpp: Extensions

Rcpp Attributes also support "plugins"

OpenMP is easy to use and widely supported (on suitable OS / compiler combinations).

So we added support via a plugin. Use is still not as wide-spread.

Errors have commonality: calling back into R.

Outline

Parallel Programming for Rcpp Users

NOT like this...

```
using namespace boost;

void task()
{
    lock_guard<boost::mutex> lock(mutex);
    // etc...
}

threadpool::pool tp(thread::hardware_concurrency());
for (int i=0; i<slices; i++)
    tp.schedule(&task);
```

Parallel Programming for Rcpp Users

Goals:

- Encapsulate threading and locking
- Provide high level constructs for common parallel tasks
- High performance: locality, work stealing, etc.
- Safe access to R data structures on multiple threads

Parallel Programming Alternatives

	TBB	OMP	RAW
Task level parallelism	■	■	
Data decomposition support	■	■	
Non loop parallel patterns	■		
Generic parallel patterns	■		
Nested parallelism support	■		
Built in load balancing	■	■	
Affinity support		■	■
Static scheduling		■	
Concurrent data structures	■		
Scalable memory allocator	■		

TBB vs. OpenMP vs. Threads

- Raw threads shift too much burden for parallelization onto the developer (error prone and not performant)
- OpenMP is excellent for parallelizing existing loops where the iterations are independent (R already has some support for OpenMP)
- TBB fares better when there is more potential interaction between threads (e.g. more complex loops, simulations, or where concurrent containers are required).
- RcppParallel: Enable use of TBB with R to complement existing OpenMP stack.

Win32 Platform Complications

- TBB supports mingw on Win32 however we haven't (yet) sorted out how to build it with Rtools
- As a result we use TinyThread on Win32
- This requires that we create a layer to abstract over TBB and TinyThread (thus limiting the expressiveness of code that wants to be portable to Windows).
- Developers are still free to use all of TBB if they are content targeting only Linux and OSX
- Would love to see TBB working on Win32 (pull requests welcome!)

R Concurrency Complications

R is single-threaded and includes this warning in Writing R Extensions when discussing the use of OpenMP:

> *Calling any of the R API from threaded code is 'for experts only': they will need to read the source code to determine if it is thread-safe. In particular, code which makes use of the stack-checking mechanism must not be called from threaded code.*

However we don't really want to force Rcpp users to resort to reading the Rcpp and R source code to assess thread safety issues.

RcppParallel Threadsafe Accessors

Since R vectors and matrices are just raw contiguous arrays it's easy to create threadsafe C++ wrappers for them:

- `RVector<T>` is a very thin wrapper over a C array.
- `RMatrix<T>` is the same but also provides `Row<T>` and `Column<T>` accessors/iterators.

The implementions of these classes are extremely lightweight and never call into Rcpp or the R API (so are always threadsafe).

RcppParallel Operations

Two high-level operations are provided (with TBB and TinyThread implementations of each):

- `parallelFor` – Convert the work of a standard serial "for" loop into a parallel one
- `parallelReduce` – Used for accumulating aggregate or other values.

Not surprisingly the TBB versions of these operations perform ~ 50% better than the "naive" parallel implementation provided by TinyThread.

Basic Mechanics: Create a Worker

Create a `Worker` class with `operator()` that RcppParallel uses to operate on discrete slices of the input data on different threads:

```
class MyWorker : public RcppParallel::Worker {

   void operator()(size_t begin, size_t end) {
      // do some work from begin to end
      // within the input data
   }

}
```

Basic Mechanics: Call the Worker

Worker would typically take input and output data in it's constructor then save them as members (for reading/writing within `operator()`):

```
NumericMatrix matrixSqrt(NumericMatrix x) {

  NumericMatrix output(x.nrow(), x.ncol());

  SquareRootWorker worker(x, output);

  parallelFor(0, x.length(), worker);

  return output;
}
```

Basic Mechanics: Join Function

For `parallelReduce` you need to specify how data is to be combined. Typically you save data in a member within `operator()` then fuse it with another `Worker` instance in the `join` function.

```
class SumWorker : public RcppParallel::Worker

   // join my value with that of another SumWorker
   void join(const SumWorker& rhs) {
      value += rhs.value;
   }
}
```

What does all of this buy us?

- Developers just write pieces of code that are called at the correct time by an intelligent parallel supervisor
- In most cases no locking or explicit thread management required!
- Supervisor does some intelligent optimization around:
  - Grain size (which affects locality of reference and therefore cache hit rates). Note that grain size can also be tuned directly per-application.
  - Work stealing (detecting idle threads and pushing work to them)
- In the case of TBB, high performance concurrent containers are available if necessary

Examples

- All available on the Rcpp Gallery
  http://gallery.rcpp.org
- Tested with 4 cores on a 2.6GHz Haswell MacBook Pro
- Note that benchmarks will be 30-50% slower on Windows because we aren't using the more sophisticated scheduling of TBB

Example: Transforming a Matrix in Parallel

http://gallery.rcpp.org/articles/parallel-matrix-transform

```
void operator()(size_t begin, size_t end) {
    std::transform(input.begin() + begin,
                   input.begin() + end,
                   output.begin() + begin,
                   ::sqrt);
  }
```

	test	replications	elapsed	relative
2	parallelMatrixSqrt(m)	100	0.294	1.000
1	matrixSqrt(m)	100	0.755	2.568

Example: Summing a Vector in Parallel

http://gallery.rcpp.org/articles/parallel-vector-sum

```
void operator()(size_t begin, size_t end) {
   value += std::accumulate(input.begin() + begin,
                            input.begin() + end,
                            0.0);
}
void join(const Sum& rhs) {
   value += rhs.value;
}
```

	test	replications	elapsed	relative
2	parallelVectorSum(v)	100	0.182	1.000
1	vectorSum(v)	100	0.857	4.709

Example: Parallel Distance Matrix Calculation

http://gallery.rcpp.org/articles/parallel-distance-matrix

	test	reps	elapsed	relative
3	rcpp_parallel_distance(m)	3	0.110	1.000
2	rcpp_distance(m)	3	0.618	5.618
1	distance(m)	3	35.560	323.273

- Rcpp + RcppParallel = 323x over R implementation!
- Unbalanced workload benefits from work stealing

The Rest of TBB

- Advanced algorithms: `parallel_scan`, `parallel_while`, `parallel_do`, `parallel_pipeline`, `parallel_sort`
- Containers: `concurrent_queue`, `concurrent_priority_queue`, `concurrent_vector`, `concurrent_hash_map`
- Mutual exclusion: `mutex`, `spin_mutex`, `queuing_mutex`, `spin_rw_mutex`, `queuing_rw_mutex`, `recursive_mutex`
- Atomic operations: `fetch_and_add`, `fetch_and_increment`, `fetch_and_decrement`, `compare_and_swap`, `fetch_and_store`
- Timing: portable fine grained global time stamp
- Task Scheduler: direct access to control the creation and activation of tasks

Open Issues

- Additional (portable to Win32 via TinyThread) wrappers for other TBB constructs?
- Alternatively, sort out Rtools configuration issues required to get TBB working on Windows.
- Education: Parallel Programming is *hard*.
- Simple `parallelFor` and `parallelReduce` are reasonably easy to grasp, but more advanced idioms aren't trivial to learn and use (but for some applications have lots of upside so are worth the effort).