So You Think You're Fast: Performance Evaluation of Two High-Performance Software Libraries

Malcolm Roberts

University of Strasbourg

Journée du Labex, 2015-12-14

Outline

‘

- schnaps
  - A discontinuous Galerkin solver in OpenCL
  - How can we evaluate it's performance?
    - Roofline
    - Profiling
    - Benchmarking
- fftw++
  - A wrapper for FFTW
  - Implicitly dealiased convolutions
  - How can we compare two different algorithms?

schnaps

Solveur pour les lois de Conservation Hyperboliques Non-linéaires Appliqué aux PlasmaS

Solver for Conservative Hyperbolic Non-linear systems for PlasmaS

- Discontinuous Galerkin solver for hyperbolic equations
- Written in `OpenCL` for `GPUs`, `CPUs`, etc.
- `gforge.inria.fr/projects/schnaps`

Collaborators: Philippe Helluy, Emmanuel Franck, Michel Massaro, Bruno Weber, ...

Consider the general hyperbolic equation

$$\partial_t w + \sum_{k=1}^{k=d} \partial_k F^k(w) = S, \qquad (1)$$

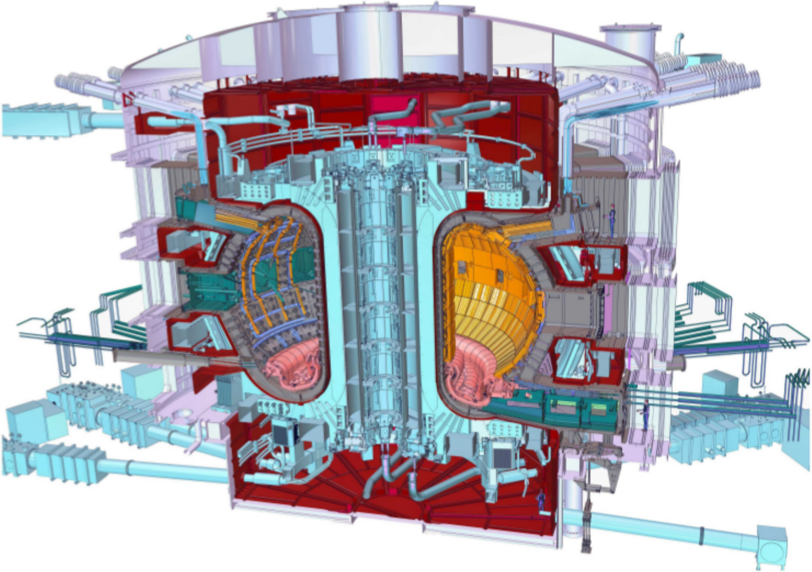
in $d =$ dimensions. F is the flux and S the source term.

Examples:

- Navier–Stokes equations
- Maxwell's equations
- MHD
- Vlasov equations

We would like to numerically solve such equations in complex geometries with as general boundary conditions as possible.

schnaps example: Vlasov equations for fusion



schnaps: Discontinuous Galerkin Method

The physical domain is divided into cells.

In each cell L, w is projected onto a finite set of basis functions $\psi_i^L(x)$:

$$w(x,t) \approx \sum_{i \in L} w_L^i(t)\psi_i^L(x). \tag{2}$$

The evolution equation is approximated by

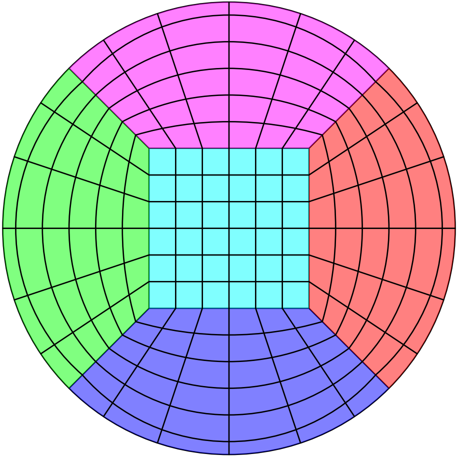
$$\int_L \partial_t w \psi_i^L - \int_L F(w, w, \nabla \psi_i^L) + \int_{\partial L} F\left(w_L, w_R, \boldsymbol{n}_{LR}\right)\psi_i^L = S_i^L, \tag{3}$$

where $\boldsymbol{n}_{LR}$ is the normal vector from cell L to cell R.

The DG formulation is good for conserving invariants.

Elements can be curved and meshes can be non-conformal.

schnaps: Macrocell/subcell

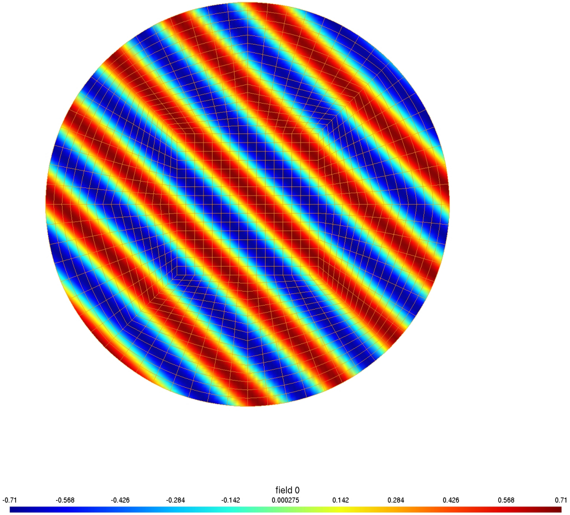


Malcolm Roberts

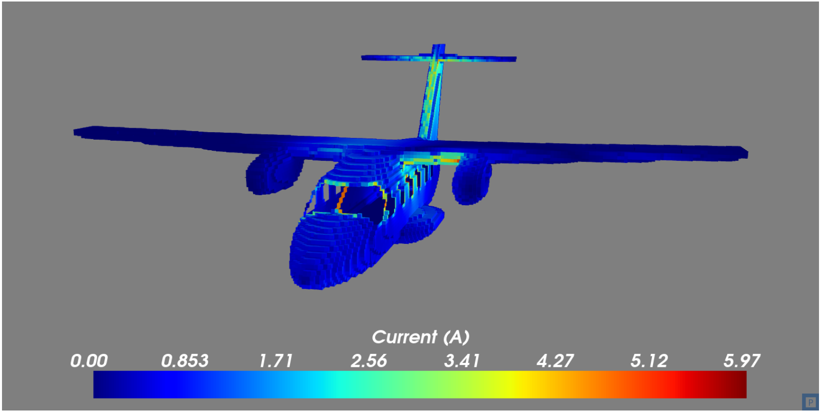
There computational stages are:

1. Initialize the buffer.
2. Compute boundary flux.
3. Compute macrocell interface flux.
4. Compute subcell interfaces flux.
5. Compute volumic flux.
6. Compute source term.
7. Apply the mass division.
8. Time-step.

schnaps example simulation: Maxwell's equations



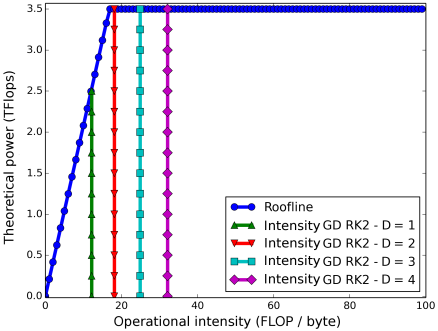
schnaps example simulation: Maxwell's equations



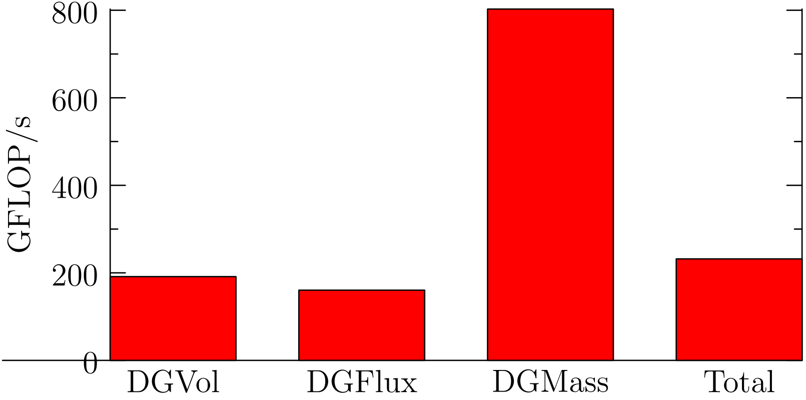
schnaps: roofline analysis

Roofline: Count FLOPs, count i/o, compare with manufacturer specs.

- Compare the achieved vs theoretical FLOPs and i/o.
- Manufacturer specifications are available for most GPUs.



schnaps: roofline analysis



`schnaps`: profiling

`OpenCL` provides profiling tools.

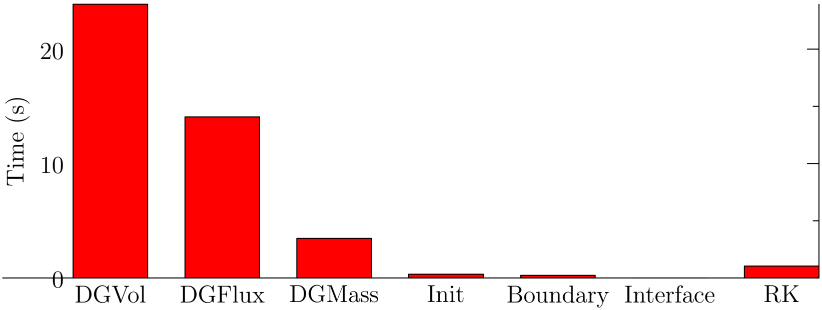
- We can find the start and end times for a kernel
- Reports ns time (accurate to $\approx$ 50ns).

Profiling is complicated in an `OpenCL` environment:

- Kernel execution might be overlapped
- Kernels are launched into a queue, and we need to wait until it's done before asking for the execution time.

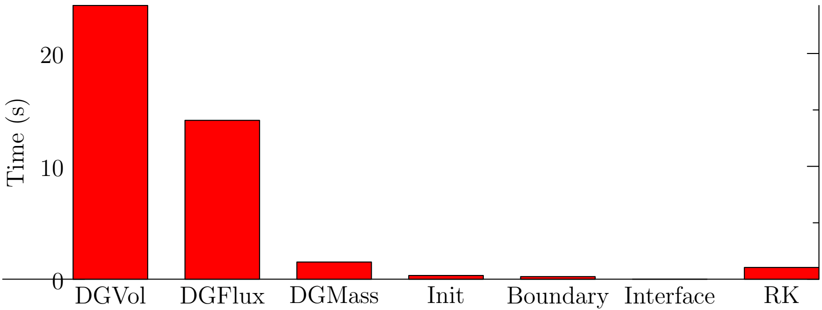
schnaps: profiling

Total execution times for all kernels:



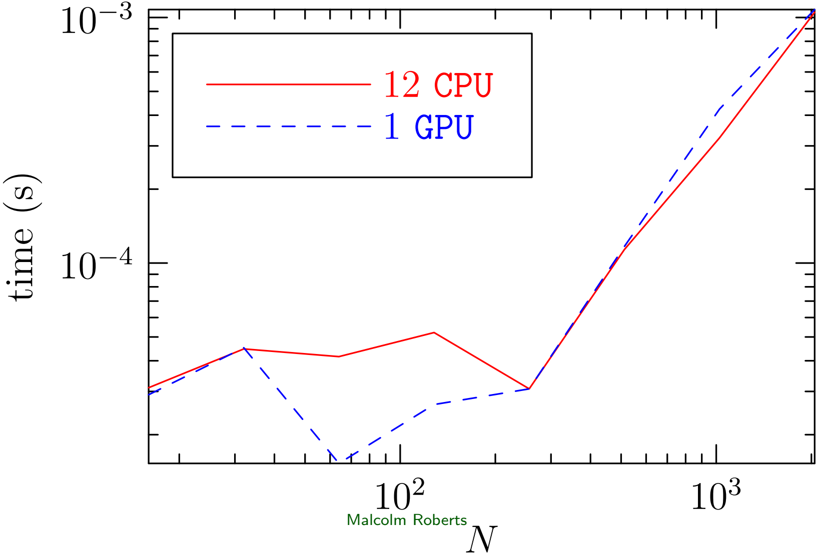
schnaps: profiling

Total execution times with mass pre-computed:

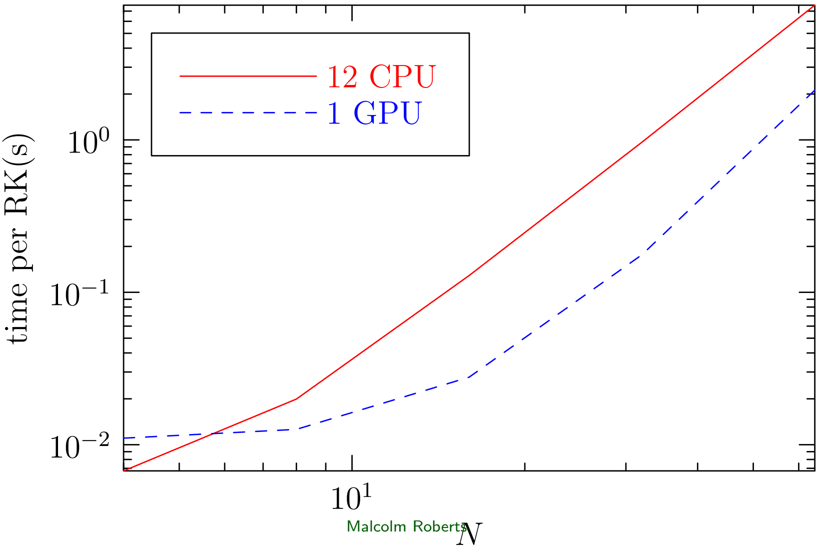


schnaps: benchmarking the CPU and the GPU

Comparison of CPU and the GPU speeds using clFFT:



Comparison of `schnaps` on CPU and the GPU:



schnaps: other geometries

The timing examples were for a mesh with one macrocell.
The macrocell interfaces are complicated:

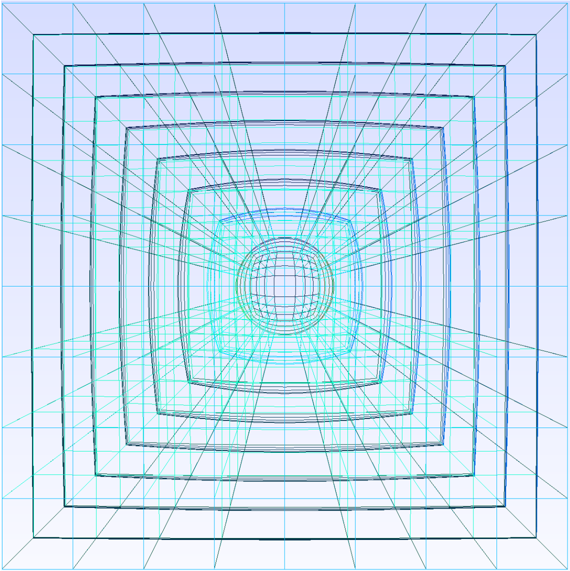
- Non-coalescent memory access
- Changes of geometry
- Parallel computation not straightforward.

Two options for parallelizing macrocell interfaces:

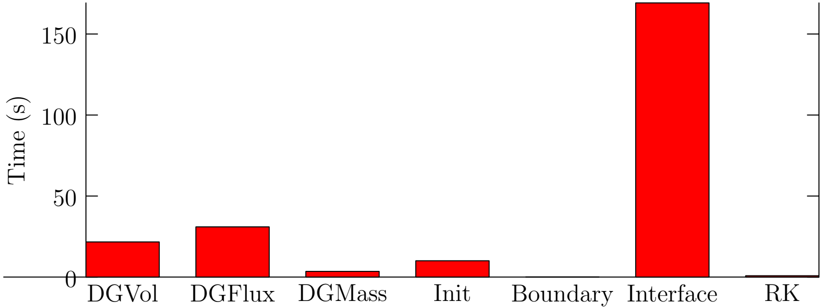
- In-place, use map-colouring for parallelism
- Extraction, computation in parallel, insertion.

Also reducing computational complexity.

schnaps: other geometries



schnaps: other geometries



schnaps: summary

- Roofline predicts how fast your code will be.
- Profiling tells you where you need to work.
- Benchmarking tells you your speed.
- Intra-macrocell code looks ok.
- Inter-macrocell code needs work.

fftw++

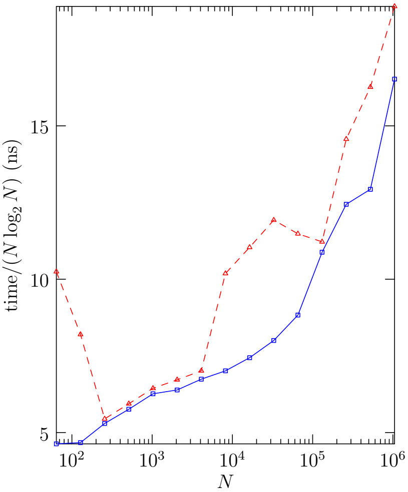
fftw++ offers:

- C++ wrappers for FFTW
- Implementations of implicitly dealiased convolutions.
- New MPI routines in 2.0:
  - Adaptive recursive non-blocking transpose
  - MPI version of convolutions and FFTs
  - 2D data decomposition
- fftwpp.sf.net.

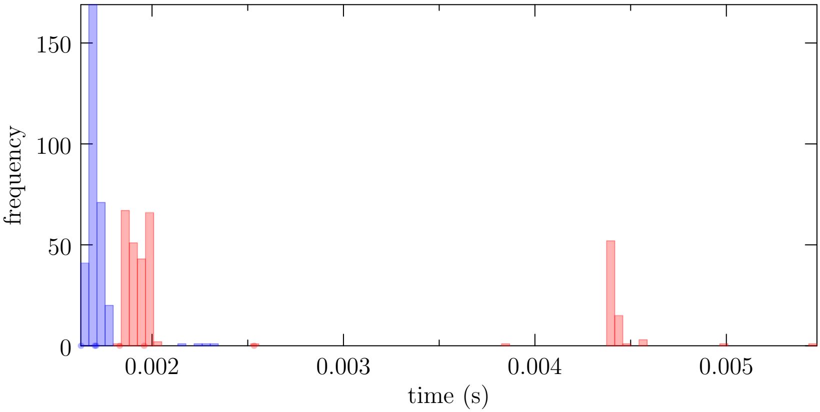
Collaborator: John Bowman.

We need to test the speed of these routines.

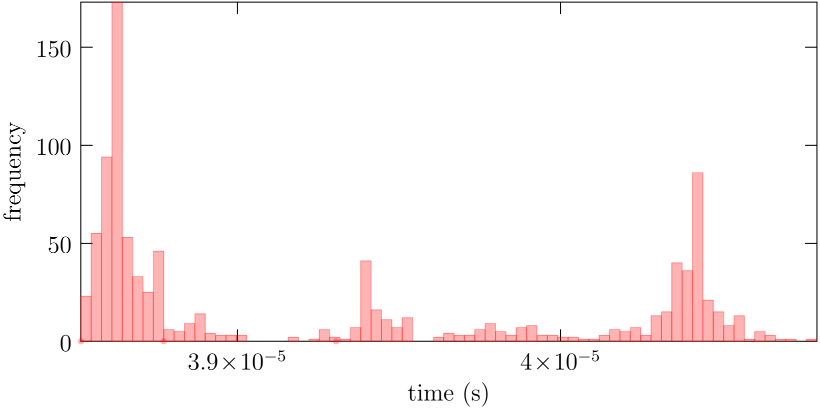
Timing 1D convolution on Atlas:



Histogram of execution times on Atlas:



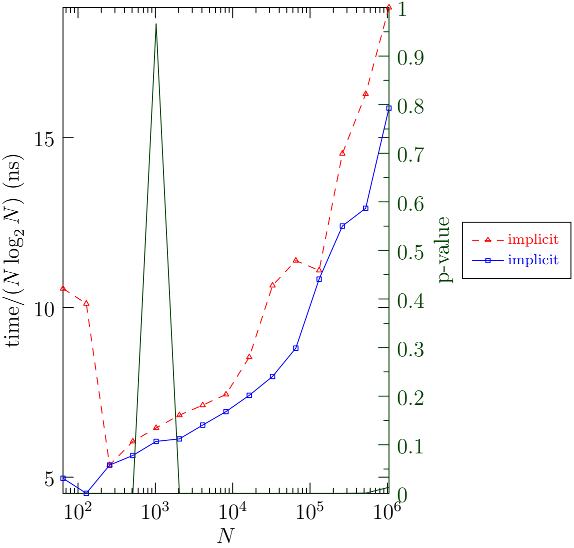
Histogram of clFFT execution times on a K40:



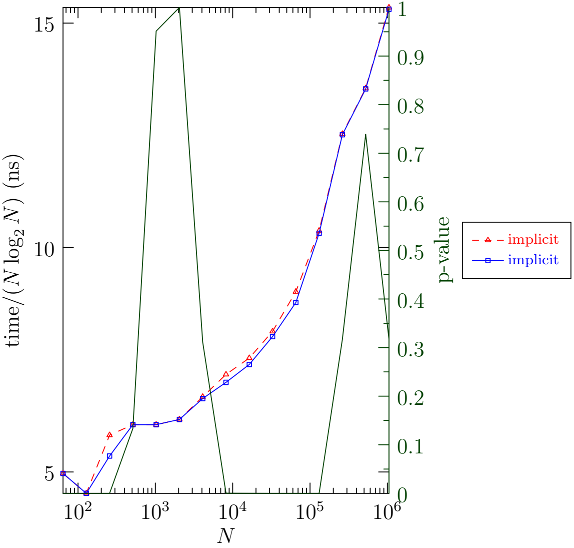
We need stats!

- The mean is not a good measure.
- We can't predict the confidence for the minimum.
- So, we should look at the median time.
- We need statistical tests; look at Mood's Median test.
  - Or the Wilcoxon signed-rank test, etc.
- Applications: publishing papers, self-tuning software libraries.

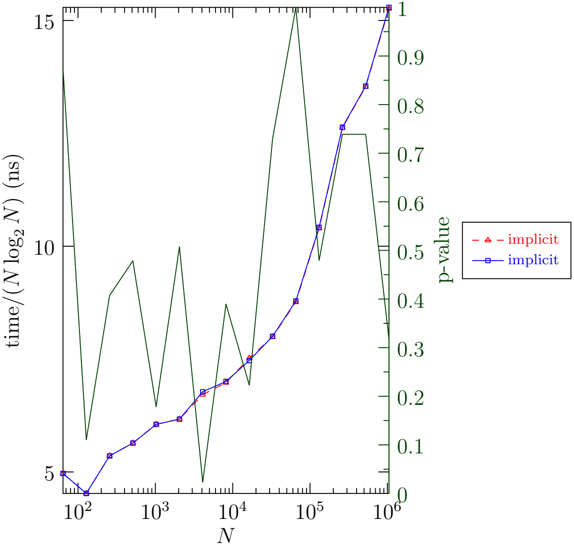
fftw++: sequential tests



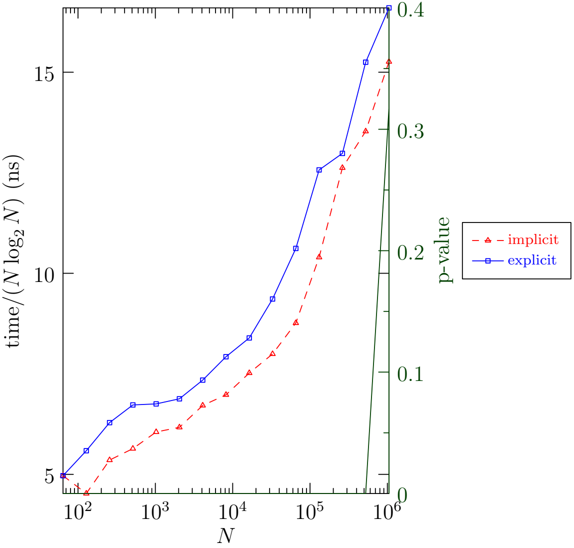
fftw++: alternating tests



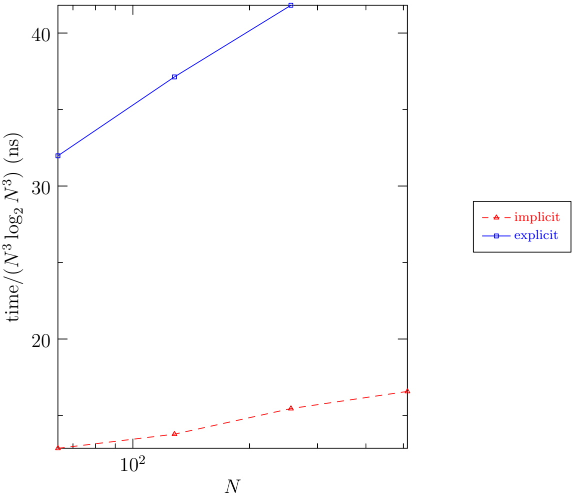
fftw++: randomizing tests



fftw++: randomizing tests, different algorithms



fftw++: Comparison of 3D routine



fftw++: conclusion

- Noise can have a large affect on timing algorithms!
- Running algorithms in series gives noisy results.
- Randomizing the algorithm order gives the best results.
- We can (and probably should) use some stats.

Conclusion

I presented the analysis of two software packages.

`schnaps`:

- Looked at: roofline, profiling, and benchmarks.
- It's fast, but comparison is difficult.
- The macrocell interfaces will be faster soon.

`fftw++`:

- We found a way to deal with noise when timing.
- Now we re-write a bunch of scripts. :(
- But the results look good so far!

Thank you for your attention!
Merci pour votre attention!