Five point One:
The Eilmer Performance and Optimisation Patch

Dr. Nick N. Gibbons,
The University of Queensland, Brisbane, Queensland 4072, Australia

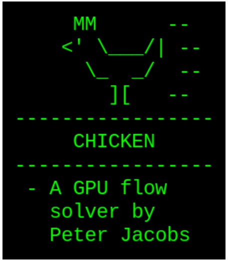
November 16, 2023

About Me: Hi!

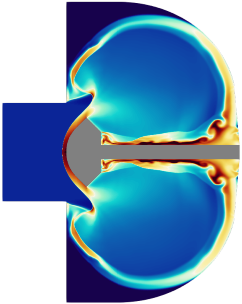
I am Nick Gibbons:

- Centre for Hypersonics Alumnus (2019)
- Currently PostDoc Fellow and developer of GDTk
- Supported by Discovery Project 220102767 (Vince Wheatley et al.) + DST

Code:



Sims:



Chemical Thermodynamics:

$$N_S = \sum_{k=0}^{M} (-1)^k \frac{12!}{k!(12-k)!} \frac{(S-6k-1)!}{(12-1)!(S-6k-12)!}$$

Space News: Webb, Webb, and More Webb

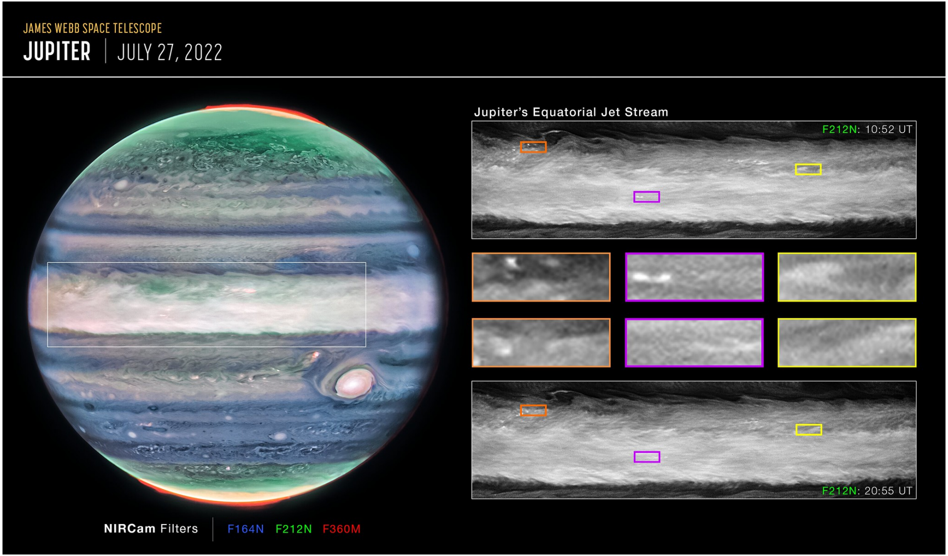


Image Credit:
NASA

Space News: Webb, Webb, and More Webb

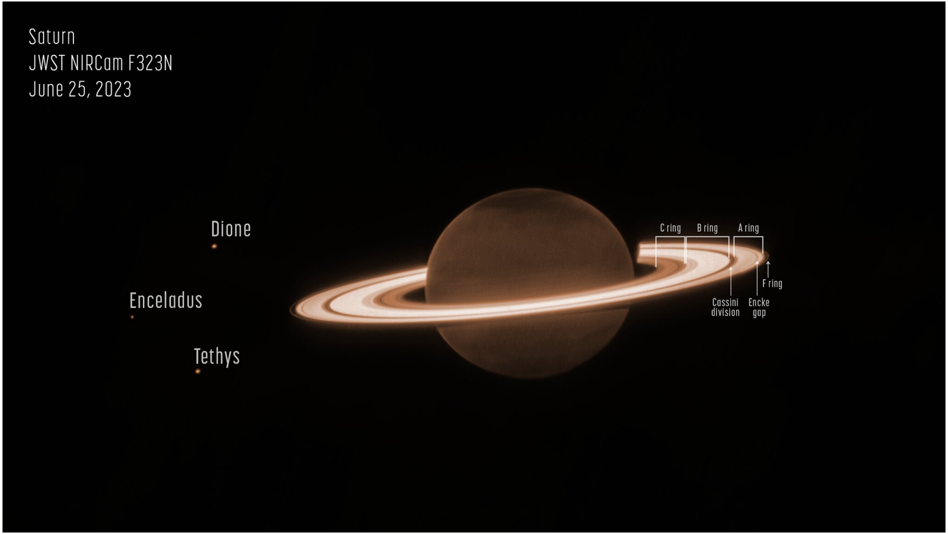


Image Credit: NASA

Space News: Webb, Webb, and More Webb

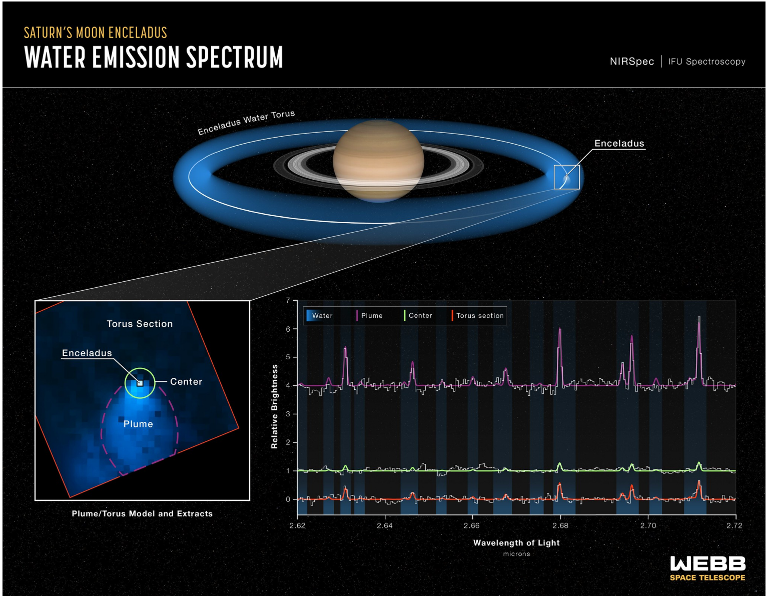


Image Credit:
NASA

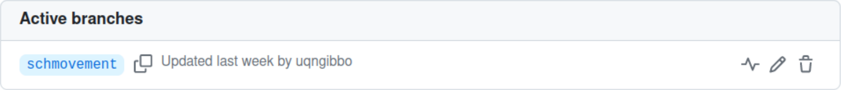
Space News: Webb, Webb, and More Webb

Image Credit: NASA

Today: I've been working on a performance patch for Eilmer

Key features:

- Major reorganisation of the code's core datastructures
- Detailed profiling of every major routine
- Some parallel efficiency improvements



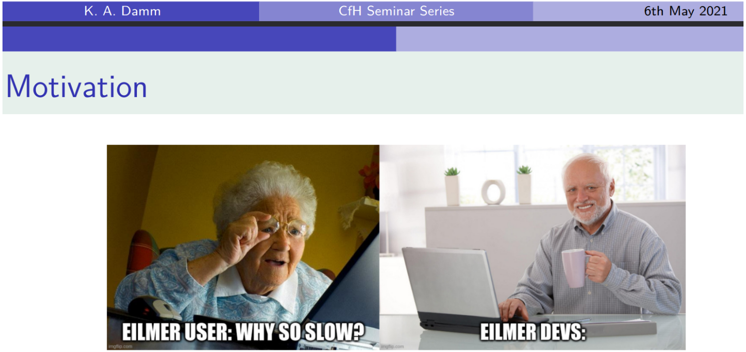
Why performance?

Code execution speed matters to just about everyone:

- Large supercomputers bill for their time
- Even in small scale, compute is often a research productivity bottleneck
- High impact effort to help ratio, assuming good speedup can be found...

A Winter of Discontent

Mid-last year testing revealed we were WAY behind other codes in speed:



1. JFNK too expensive for engineering work?
2. Is the D language just slow?
3. Some kind of fixable problem with the code's layout?

Possibility 1: Kyle's Testing of other Implicit Methods

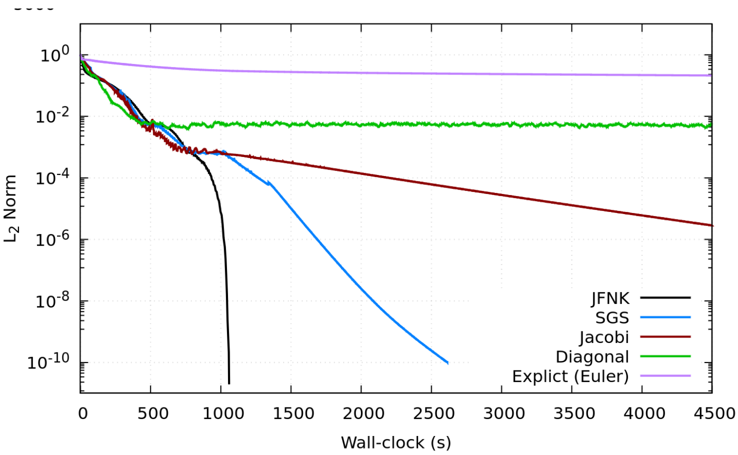
Many other hypersonics codes use "Defect Correction" methods for time advancement:

- When solving the $Ax = b$ problem, use an approximate A
- Economise on space and time
- Hurts deep convergence, but maybe who cares?

We could try implementing one of these...

Possibility 1: Kyle's Testing of other Implicit Methods

Kyle's 2022 CfH seminar pretty much showed that the JFNK rules:



Possibility 2: Testing of the D language

My spring 2022 Lightning Talk compared different languages on a simple task:

- Take 100,000 Conserved Quantities and compute 100,000 Fluxes in x and y
- Then scale the Us and do it again, 10,000 times

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix} \qquad \mathbf{F}_x = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho v u \\ \rho w u \\ \rho E u + p u \end{bmatrix} \qquad \mathbf{F}_y = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho w v \\ \rho E v + p v \end{bmatrix}$$

Possibility 2: Testing of the D language

Different languages and different implementations:

	LLVM D (ldc2 v1.24)	Plain C (gcc v9.4)	Fortran 90 (gfortran v9.4)
Classes			
Arrays			
Arrays2			
Structs			

Possibility 2: Testing of the D language

Different languages and different implementations:

	LLVM D (ldc2 v1.24)	Plain C (gcc v9.4)	Fortran 90 (gfortran v9.4)
Classes	14.7s (1.18 ipc)		
Arrays	12.0s (1.15 ipc)	11.7s (1.22 ipc)	6.84s (1.88 ipc)
Arrays2	3.74s (2.88 ipc)	5.14s (4.05 ipc)	4.92s (4.20 ipc)
Structs	3.27s (3.50 ipc)	3.51s (3.41 ipc)	3.45s (3.45 ipc)

This seems crazy. What's going on?

	LLVM D (ldc2 v1.24)	Plain C (gcc v9.4)	Fortran 90 (gfortran v9.4)
Classes	14.7s (1.18 ipc)		
Arrays	12.0s (1.15 ipc)	11.7s (1.22 ipc)	6.84s (1.88 ipc)
Arrays2	3.74s (2.88 ipc)	5.14s (4.05 ipc)	4.92s (4.20 ipc)
Structs	3.27s (3.50 ipc)	3.51s (3.41 ipc)	3.45s (3.45 ipc)

```
uqngibbo@continuity:~/.../nng_bench$ diff objects2.d structs2.d

11c11
< class FlowState{
---
> struct FlowState{
23c23
< class Flux{
---
> struct Flux{
```

Optimisation for Modern Processors 101

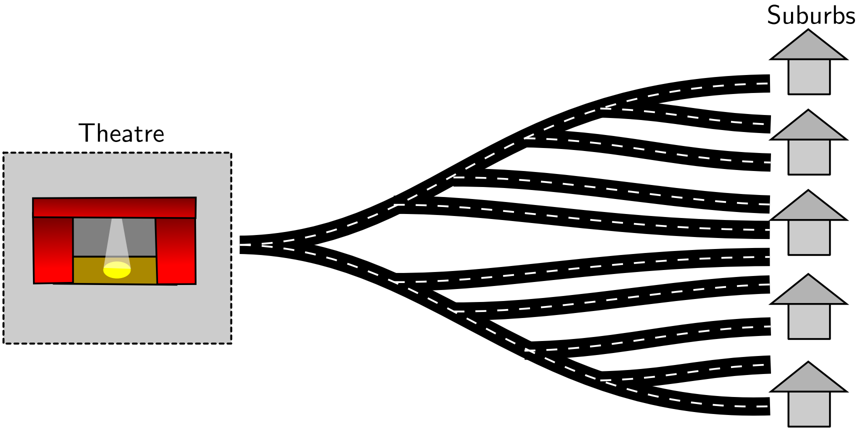
It's actually not that crazy:

- Modern processors are much faster at processing data than accessing it
- This means how you write your code impacts its speed a LOT

Optimisation for Modern Processors 101

Imagine rehearsing a stage play with 2000 actors:

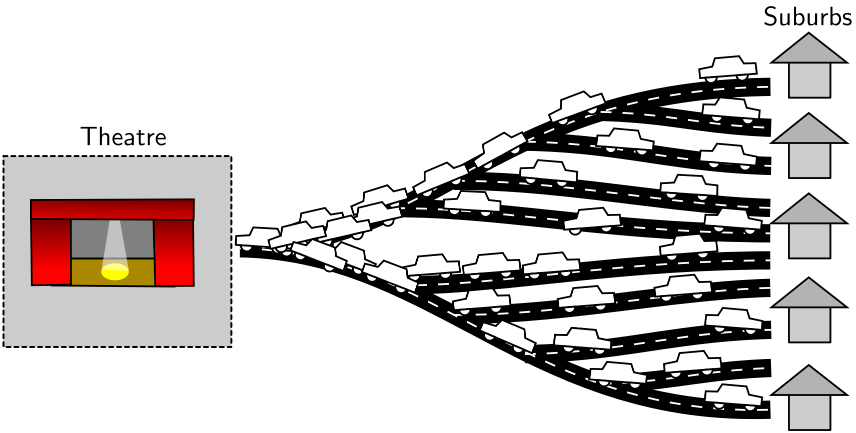
- There's not enough room in the theatre for everyone
- People have to come in when its time for their scene and then leave
- Imagine everyone lives in the suburbs far away from the theatre



Optimisation for Modern Processors 101

Imagine rehearsing a stage play with 2000 actors:

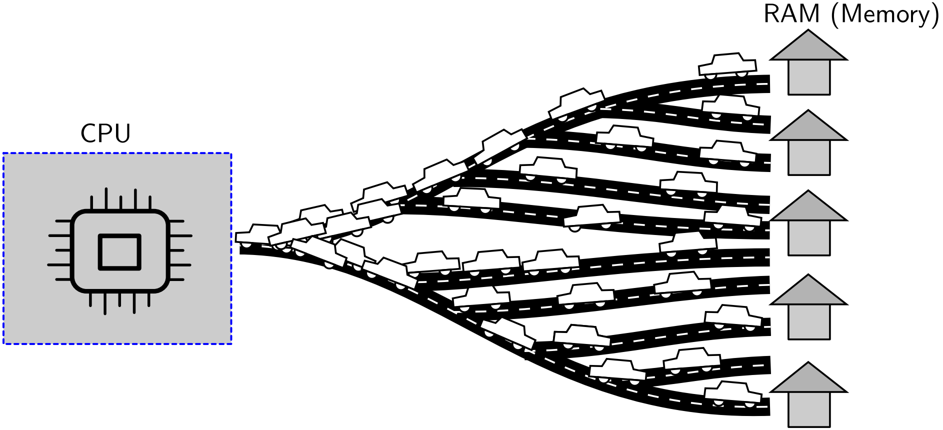
- There's not enough room in the theatre for everyone
- People have to come in when its time for their scene and then leave
- Imagine everyone lives in the suburbs far away from the theatre



Optimisation for Modern Processors 101

Modern CPUs are kind of like this:

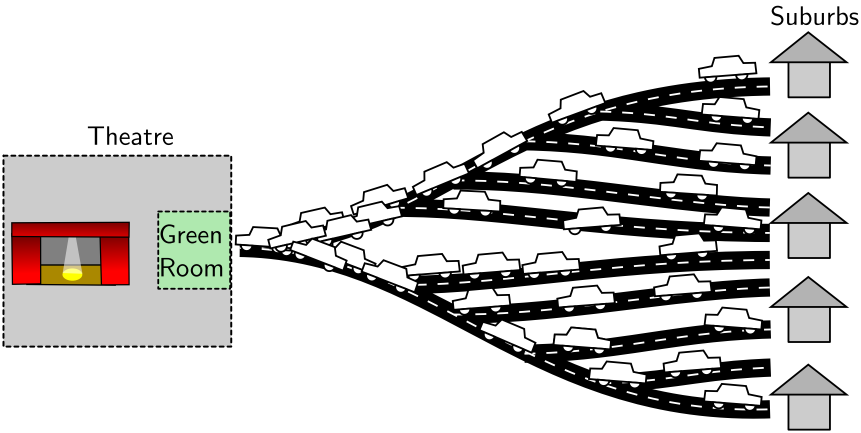
- RAM is where the data you are working with is stored
- Shuttling this data to the CPU takes a long time
- Once it's there the actual math is fairly quick



Optimisation for Modern Processors 101

CPUs developers mitigate this problem with tricks and workarounds

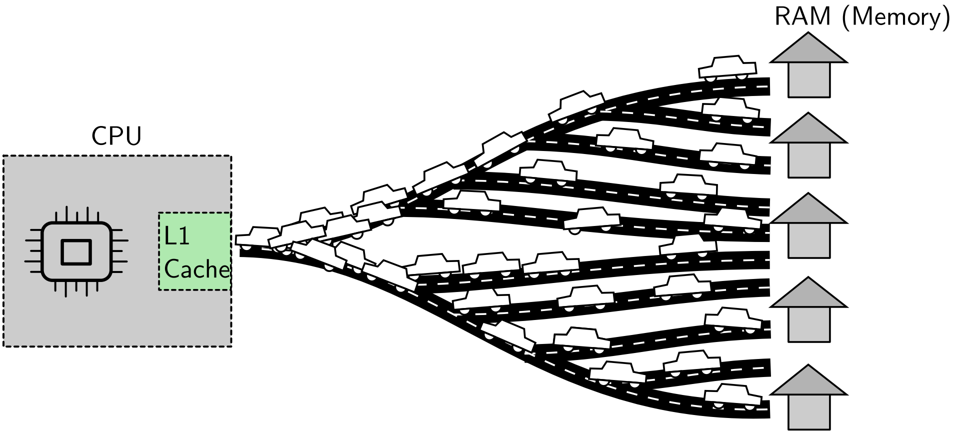
1. Caches



Optimisation for Modern Processors 101

CPUs developers mitigate this problem with tricks and workarounds

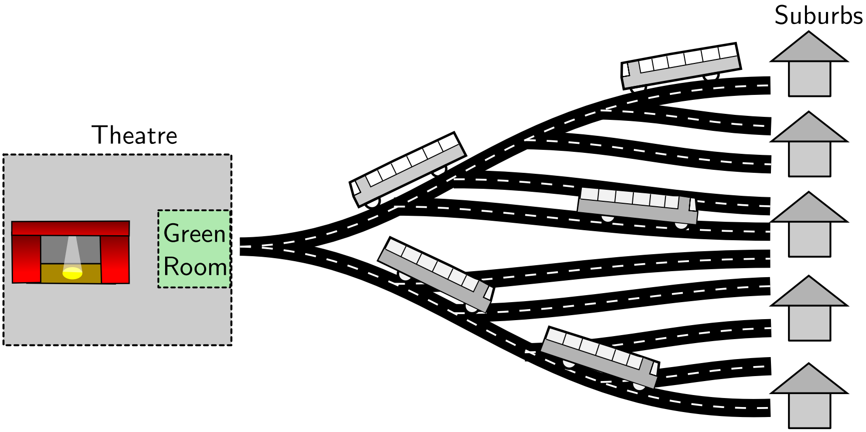
1. Caches



Optimisation for Modern Processors 101

CPUs developers mitigate this problem with tricks and workarounds

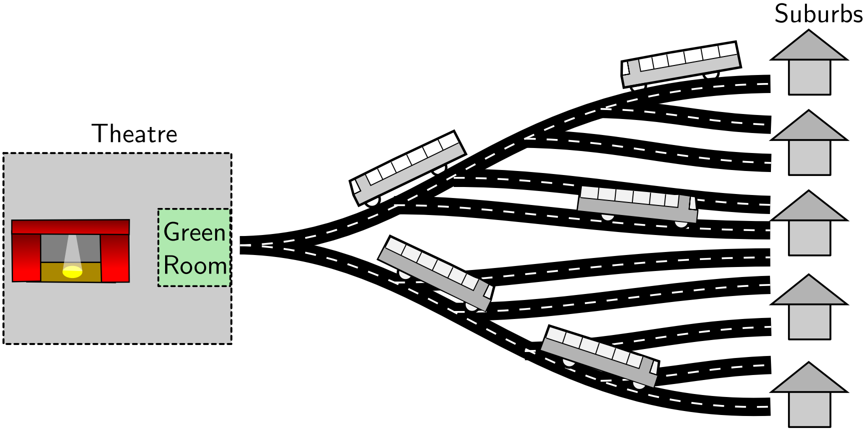
1. Caches
2. Memory Coalescing



Optimisation for Modern Processors 101

CPUs developers mitigate this problem with tricks and workarounds

1. Caches
2. Memory Coalescing
3. Branch prediction



Tips for Writing Fast Code

- Operate on one thing multiple times
- Keep related data together in memory
- Be careful with abstractions...

If in doubt make it look like C!

Interlude: What is Perf?

- perf (formerly Performance Counters for Linux) is command line profiling tool
- perf hooks into the hardware performance counters for very deep and detailed profiling
- No need to compile anything into your code, just compile with -g and go!

```
$ sudo perf stat ./objects

Performance counter stats for './objects':

      14,728.08 msec task-clock                #    0.996 CPUs utilized
 61,939,259,259      cycles                    #    4.206 GHz
     60,176,529      stalled-cycles-frontend   #    0.10% frontend cycles idle
 51,991,892,750      stalled-cycles-backend    #   83.94% backend cycles idle
 73,177,964,238      instructions              #    1.18  insn per cycle
                                               #    0.71  stalled cycles per insn
  3,040,637,903      branches                  #  206.452 M/sec
        798,514      branch-misses             #    0.03% of all branches

   14.780695489 seconds time elapsed
```

schmovement: Experimental Eilmer branch

- Started in April this year and wrapped up a few weeks ago
- 110 commits in a separate branch to the main repo

```
$ git diff --stat c7264003 HEAD
117 files changed, 5878 insertions(+), 2671 deletions(-)
```

Changelog Highlights

- Core datastructures packed together in dense arrays
- Cell,Face,Vertex classes still exists: have pointers to their flow data

```
// New                                           // Old
struct FVCellData{                             | class FVCell {
    size_t[] nfaces;                           |     int id;
    int[][] outsigns;                          |     int[] outsign;
    bool[] data_is_bad;                        |     bool data_is_bad;
    size_t[][] halo_cell_ids;                  |     FVCell[] cell_cloud;
    size_t[][] halo_face_ids;                  |     FVInterface[] iface;
    number[] areas;                            |     number area;
    number[] volumes;                          |     number volume;
    Vector3[] positions;                       |     Vector3 pos;
    FlowState[] flowstates;                    |     FlowState fs;
    FlowGradients[] gradients;                 |     FlowGradients grad;
    ConservedQuantities U0, U1, U2;            |     ConservedQuantities[] U;
```

Changelog Highlights

- Major RHS routines all use the dense datastructures for their work
- Below computed with `perf record` at 75,000 Hz
- Units are billions of CPU cycles, 1 second is $\approx$ 6B cycles

```
                              Old       New    Improvement
----------------------------------------------------------------
convective_flux_phase2       51.1  ->  26.7   (x1.9)
rpcGMRES_solve               92.7  ->  16.6   (x5.6)
gradients_leastsq            23.1  ->  11.4   (x2.0)
convective_flux_phase0       32.3  ->  19.2   (x1.7)
average_lsq_cell_derivs      36.8  ->   9.3   (x4.0)
compute_gradients_at_cells   37.1  ->   8.9   (x4.1)
fluxcalc.ausmdv              18.6  ->   7.9   (x2.4)
time_derivatives             34.7  ->   5.7   (x6.0)
viscous_flux                 34.8  ->   4.9   (x7.1)
decode_conserved              5.3  ->   1.8   (x2.9)
```

Changelog Highlights 2

Mathematical functions are expensive:

- many cases of `log`, `pow`, and `tanh` removed
- Lots of improvement in thermo tables and transport properties
- `pow(2)` versus `pow(2.0)`

```
// Old therm_perf_gas_mix.d                 | // New
override number enthalpy(in GasState gs)    | override number enthalpy(in GasState gs)
{                                           | {
                                            |     number logT = log(gs.T);
                                            |
    foreach (isp, ref h; mVals) {           |     foreach (isp, ref h; mVals) {
        h = mCurves[isp].eval_h(gs.T);      |         h = mCurves[isp].eval_h(gs.T, logT);
    }                                       |     }
                                            |
    number h = mass_average(gs, mVals);     |     number h = mass_average(gs, mVals);
                                            |
    return h;                               |     return h;
}                                           | }
```

Changelog Highlights 3

Sparse matrix and linear algebra improvements in the JFNK:

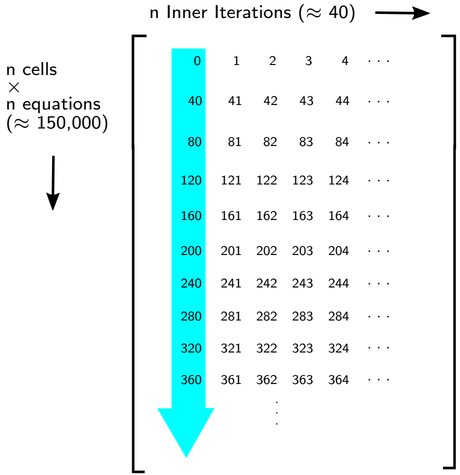
- Memory alignment of GMRes V vector
- Sparse matrix solve relies on index function

```
# Samples: 10M of event 'cycles'
# Event count (approx.): 533422083628
#
# Overhead  Command         Object                Symbol
# .............................................................................
 8.10%  e4-nk-shared-real [.] const @nogc double Matrix.opIndex(ulong, ulong)
```

Changelog Highlights 3

Sparse matrix and linear algebra improvements in the JFNK:

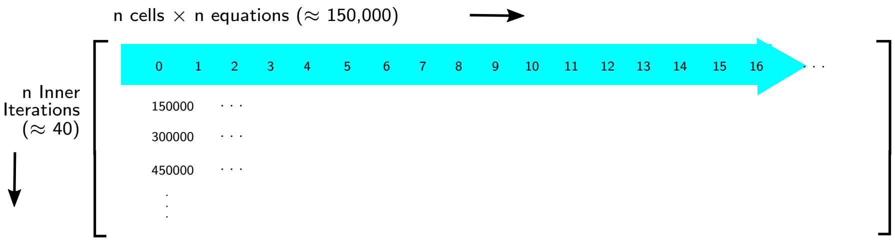
- Memory alignment of GMRes V vector
- Blue arrow indicates loop direction



Changelog Highlights 3

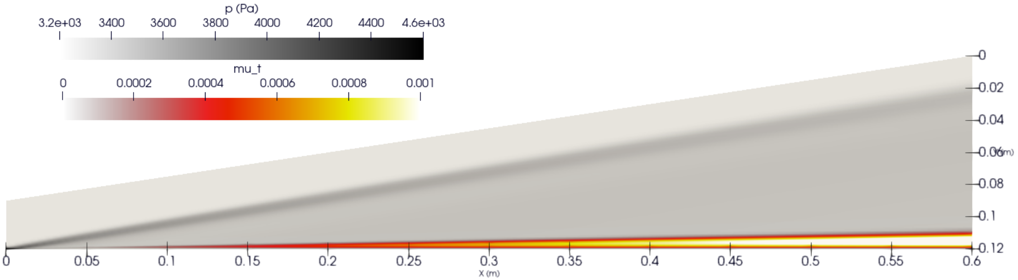
Sparse matrix and linear algebra improvements in the JFNK:

- Memory alignment of GMRes V vector
- Blue arrow indicates loop direction



Testing: JFNK Steady-state Solver

- 128x64 turbulent flat plate @ Mach ??, ideal gas, 1 core



Testing: JFNK Steady-state Solver

- 128x64 turbulent flat plate @ Mach 8, ideal gas, 1 core

```
Reference commit f3265a1d:
STOPPING: The relative global residual is below target value.
          current value= 1.538020779225e-09   target value= 1.000000000000e-08
          STEP=     623  pseudo-time= 2.250e-04 dt= 6.483e-05 cfl= 1.000e+04  WC=346.1

New commit 598876b2:
STOPPING: The relative global residual is below target value.
          current value= 5.879533649272e-09   target value= 1.000000000000e-08
          STEP=     622  pseudo-time= 1.942e-04 dt= 6.476e-05 cfl= 1.000e+04  WC=126.6

Improvement: x2.7
```

Testing: JFNK Steady-state Solver, Reacting

- 128x64 turbulent flat plate @ Mach 8, Premixed H2/air, 8 core

```
Reference commit f3265a1d:
STOPPING: The relative global residual is below target value.
          current value= 4.347608433253e-09   target value= 1.000000000000e-08
          STEP=     946  pseudo-time= 2.132e-04 dt= 6.350e-05 cfl= 1.000e+04  WC=503.1

New commit 598876b2:
STOPPING: The relative global residual is below target value.
          current value= 2.956551365153e-09   target value= 1.000000000000e-08
          STEP=     953  pseudo-time= 2.293e-04 dt= 6.350e-05 cfl= 1.000e+04  WC=127.9


Improvement: x3.9
```

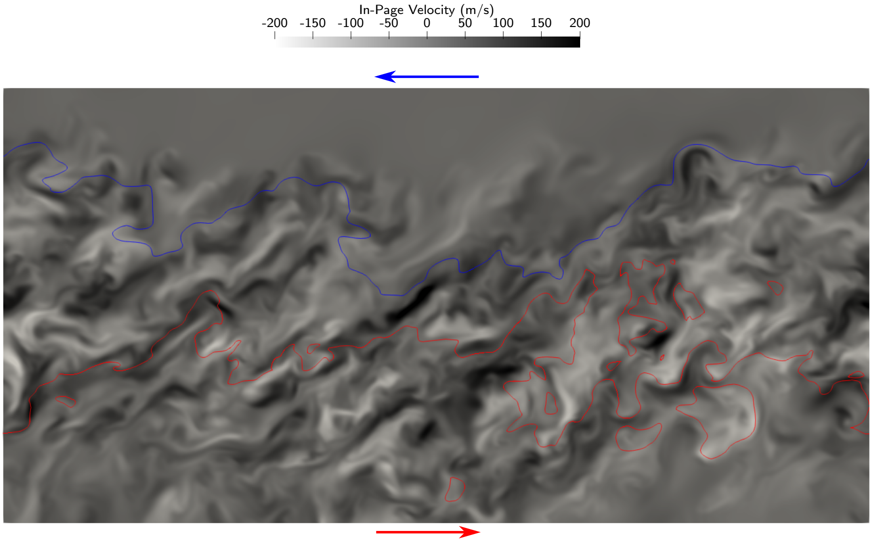
Testing: Transient Solver, Non-Reacting

- 128x64 turbulent flat plate @ Mach 8, ideal gas, 8 core

```
Reference commit f3265a1d:
Step=  19200 t= 1.993e-04 dt= 1.041e-08 cfl=0.50 WC=273.9 WCtFT=0.9 WCtMS=28255.1
Integration stopped: Reached target simulation time of 0.0002 seconds.

New commit 598876b2:
Step=  19200 t= 1.998e-04 dt= 1.043e-08 cfl=0.50 WC=143.6 WCtFT=0.2 WCtMS=14814.6
Integration stopped: Reached target simulation time of 0.0002 seconds.



Improvement: x1.9
```

Testing: Transient Solver, Reacting DNS

- 20 Million Cell Mixing Layer C2H4/Air, 640 cores on Setonix



Testing: Transient Solver, Reacting DNS

- 20 Million Cell Mixing Layer C2H4/Air, 640 cores on Setonix

```
Reference commit f3265a1d:
Step=    500 t= 3.823e-06 dt= 1.273e-08 cfl=0.50 WC=8844.0 WCtFT=22483.0 WCtMS=0.0
Est. time 30.12 Hours

New commit 598876b2:
Step=  14340 t= 1.999e-04 dt= 1.462e-08 cfl=0.50 WC=48130.5 WCtFT=28.1 WCtMS=455326.2
Total time 13.37 Hours

Improvement: x2.25
```

Thanks!

Acknowledgements:

- Vince Wheatley
- Defence and Technology Group
- The rest of GDTk: Kyle, PJ, Rowan, Lachlan and others

Also here's a funny tweet because reasons: