

❖ Supercomputers today

❖ Numerical solution of Ordinary Differential Equations

- Euler method
- Leapfrog (symplectic 2nd order)
- RK4 algorithm
- Symplectic 4th order algorithm

❖ Programming and parallelism: C, Fortran, OpenMP (cont'd)

- ◆ Massively parallel processing of tetrahedrons (tetra-Dc) in Fortran [and tetra-Dg in CUDA Fortran]
- ◆ Analysis of the Laplace stencil program (art-1: ~/progD57)
ifor-laplace3-dp.f90, -sp.f90. cudafor-laplace-sp.f90

➤ Tutorial

Literature: see links on our course home page, the references page, and on the coding page.

PHYD57. Advanced Comp. Methods in Physics.

(c) Pawel Artymowicz UofT, 2024. Only for use by enrolled UTSC students



20 GB/s Infiniband switch present on art cluster

Modern supercomputing clusters compute in parallel. They consist of many nodes (workstations running Linux operating system), connected by a fast network in order to work in parallel on a single problem (or on many problems simultaneously). Hierarchical hardware inside as well.

Even though the modern networks (Ethernet, Infiniband) transfer from 10 to 50 GB/s between arbitrary two nodes using networking switches, data can flow up to 40x faster, at 250-790 GB/s, inside the Xeon Phi and GPU computational cards.

Node connectivity can therefore be a bottleneck for some (fortunately not all) parallel computer tasks



Picture shows some of the 28 nodes of the UTSC supercomputer designed and built in 2017 by your lecturer & a summer student.

- CPUs (6-core, 4GHz overclock),
- GPUs (graphics cards), and/or
- Intel Xeon Phi cards (57-core processors)

Each node contains CPU + 2 Nvidia graphics cards capable of combined ~10 TFLOPs in single precision.

This cluster can compute 10^{12} (trillion) times faster than ENIAC in 1946, and

~6 billion times faster than similarly sized PDP-11 (1970), using ~10 million times larger memory (RAM, disks) than PDP-11/45.

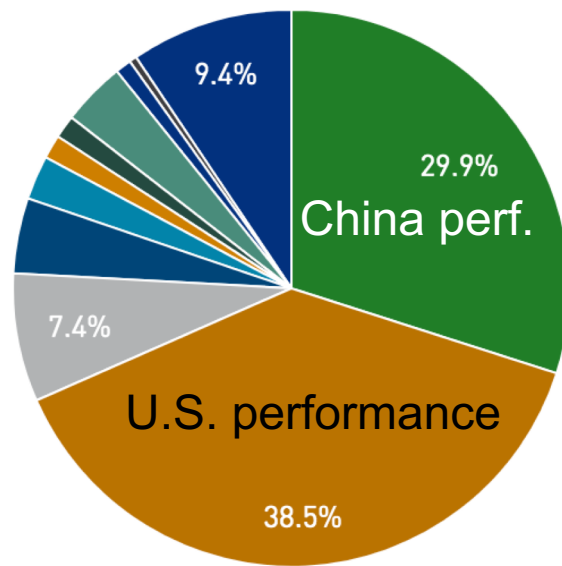
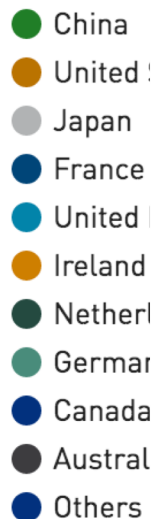
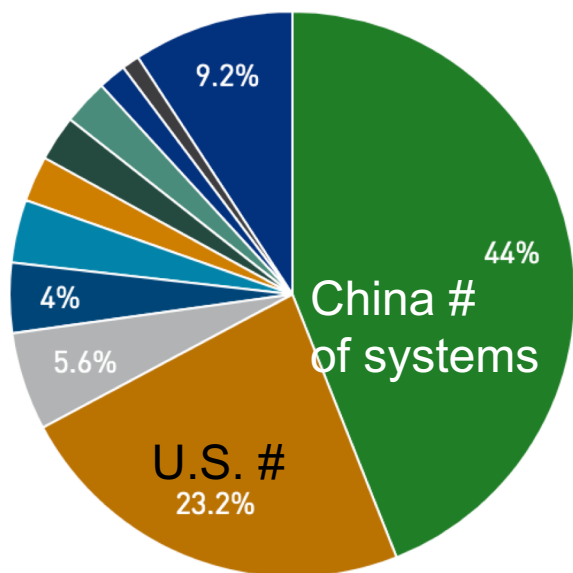
It costs 40x less than did PDP-11 (60k vs. 2400k, in today's \$)

Who has the fastest supercomputers today? USA again in 2024: it used to be the dominant player for 65 years, 1945...2010) Then China and Japan challenged U.S.

Countries System Share

2019

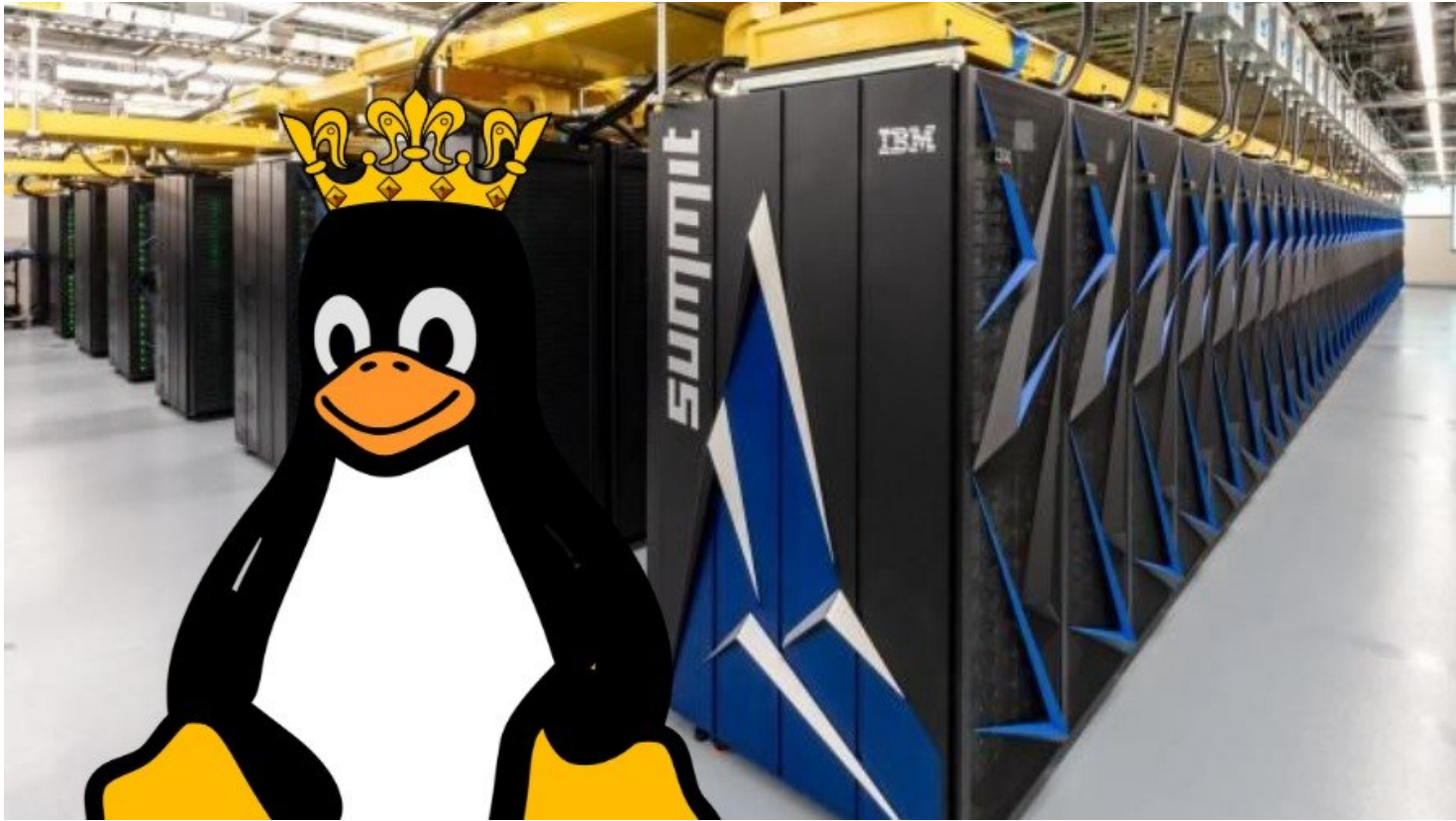
Countries Performance Share



Title of media report in November **2017**: “China overtakes U.S in the Top 500 Supercomputers List”. China had **two fastest** supercomputers in 2017 (Sunway & Tianhe-2), including one based on own 240-core processors and one on Intel Xeon Phi processors (60-core).

Up to date statistics available at top500.org

Year	China (%)	USA (%)
1999	0.2% (0.2%)	58% (61%)
2009	4% (3.5%)	58% (61%)
2019	44% (30%)	23% (39%)
2024	16% (4%)	34% (53%)



Title of article in June **2018**:

“Linux Powers ALL Top 500 Supercomputers in the World. U.S. beats China for #1”
Classified computer called *Summit* in the U.S. was the fastest machine in the world.

Linux followed Unix in its domination of HPC (High Performance Computing);
MacOS (Linux derivative) plays no role in HPC, Windows OS as well

<https://www.top500.org/statistics/sublist/>



Modern top supercomputers:
rows of 19"-wide racks filling basketball fields, 1000s of nodes (workstations), consume 18 to 23 MW of electric power and emit heat at that rate. Cost ~\$300 million.
Not environment-friendly (e.g. UofTs SciNet was warming/damaging envir., whose warming it was meant to study.)

However, let's put things into perspective! That's about the price of ONE Boeing 777 airplane, whose engines also produce 23 MW of power in cruise & much more while climbing. There are 1600 B777s, and 10000 Airbuses!
Each costs as much as a big supercomp center



Summit installed in 2018 as #1 in the world, had 4608 nodes, 9216 IBM POWER9 CPUs and 27648 Nvidia Tesla GPUs. Most computational power is from those Tesla V100 graphics cards.

The combined performance is measured at ~150 PFLOPS, out of the theoretical number of 200 PFLOPS (Summit could de facto perform 150,000,000,000,000,000 arithm.op./s)



Modern top supercomputer:

150,000,000,000,000,000 arithmetic operations/second.

Who'd ever need that?

No single user is allowed to hog the whole supercomputer, there are thousands of users at national supercomputing centers.

We will do a realistic estimate of how long one scientific model runs on Summit.

[Do not memorize it! It's informative, but not exact to better than on order of magnitude. Read this & the next 2 slides for fun.]

Suppose 1000 scientists want to do high-resolution simulations in 3D, simultaneously. Each of them divides simulated object or region of space into, say, $2000 \times 2000 \times 2000$ cells, or 8G cells [$(2K)^3 = 8G$]. Each cell must hold 5-10 floating point numbers of length 8 Bytes (double precision). For instance, in a fluid simulation those would be density and pressure, as well as 3 components of velocity vector in each cell. Storage of at least 40 to 80 B/cell is usually needed.

The total volume of simulated data may thus be $\sim 1000 * 8G * (40-80 \text{ B}) \sim 500 \text{ TB}$.



Modern top supercomputer:

150,000,000,000,000,000
arithmetic ops/s (0.15 ExaFLOP)

Is this not an overkill and a waste of money?!

If the total volume of simulated data is ~500 TB,
and ~30K graphics cards do simultaneous number crunching on Summit, then
~500GB/30 ~16 GB

of this data must be processed by each GPU in each time step. (Today, 16 GB can fit on the biggest RAM available on GPUs, avoiding transfer from/to CPU RAM, a good thing!)
Constantly shuffling data back and forth to RAM at a bandwidth of ~300 GB/s, the GPU card takes at least

$$\sim(16 \text{ GB}) / (300 \text{ GB/s}) = \sim 0.05 \text{ s}$$

to process its allotted data in a given time step of the simulation (inter-processor and inter-node communication takes additional time, sometimes majority of time, but let's assume this does not apply to CFD = Computational Fluid Dynamics.)

Next, how many time steps are needed?



Modern supercomputers:

Large-scale simulations take a lot of time & use all available resources! There is never “too much resolution” or “too much precision” in science or engineering.

Think of simulating a new passenger jet flying with complicated wing flaps. Is 2000x2000x2000 resolution sufficient? (Assume the length of aircraft is 70m. Estimate average cell size. Boundary layer of air is a few mm thick.)

A high resolution simulation may need ~1 million time steps to complete. (That’s because the computational cells are small and modeled physical signals propagating through the grid of cells cannot cross more than 1 cell per time step (CFL condition). In practice, a physical disturbance of some sort crossing 50 times a grid of 2000 cells in both directions needs a minimum of 500K steps, in agreement with the 1M estimate).

If so, then each of our hypothetical 1000 researchers have to wait a *minimum* of
 $1M * (0.05s) \sim 14 \text{ hours}$
for their simulations to complete, if everything works at 100% efficiency.

So the seemingly ‘ridiculously large’ number of arithmetic ops per second ($>10^{17}$), of which Summit is capable, is not so ridiculous after all when shared among the waiting scientists.

What about 2024?



- Taking into account *bandwidth limitations* (decisive for most computations!), Summit was not excessively fast. Neither did it have performance of 1 EFLOPS = Exaflop = 10^{18} FLOP/s in double precision (8B/floating point number). Such computers were supposed to arrive by 2020, it was a decade-long goal.
- They first arrived around 2022
- Summit fell to #9 in 2024 list of Top 500 computers
- the fastest machine of 2024 is Frontier (also at Oakridge National Lab); it uses
- AMD Optimized 3rd Gen EPYC 64Core 2GHz CPUs (8.7M cores)
- In 2022 did 1.2 EFLOP in tests, 1.7 EFLOP in theory
- Runs Cray OS

- Second is Aurora, 1.8 times more power (38 kW), at Argonne Lab. Aurora has theor. speed 2 EFLOP, in tests 1 EFLOP. It is Inter-CPU based



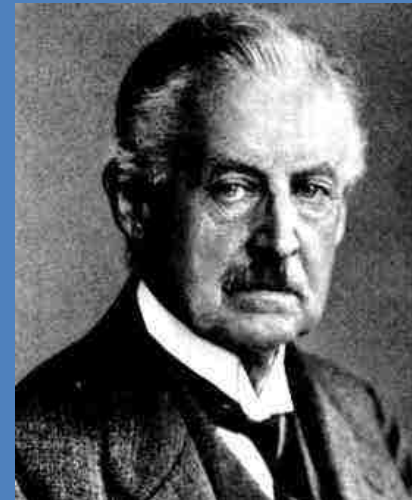
❖ Special theory of perturbations (old name for numerical calculations, Usually the solution of **ODEs**)

Most popular numerical integration methods for **differential eqs**
Euler method (1st order) & Runge-Kutta (2nd - 8th order)

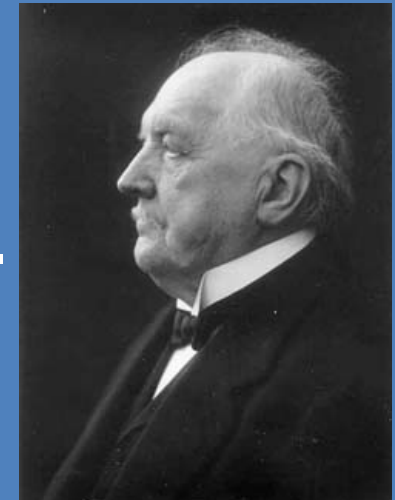
Leonard Euler

Carle Runge

Martin Kutta



1856-1927



1867-1944



The Euler method

We want to approximate the solution of the differential equation

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \quad (1)$$

For instance, the Kepler problem which is a 2nd-order equation, can be turned into the 1st order equations by introducing double the number of equations and variables: e.g., instead of handling the second derivative of variable x , as in the Newton's equations of motion, one can integrate the first-order (=first derivative only) equations using variables x and $v_x = dx/dt$ (that latter definition becomes an additional equation to be integrated).

Starting with the differential equation (1), we replace the derivative y' by the finite difference approximation, which yields the following formula

which yields

$$y'(t) \approx \frac{y(t+h) - y(t)}{h}, \quad (2)$$

$$y(t+h) \approx y(t) + hf(t, y(t)) \quad (3).$$

This formula is usually applied in the following way.

The Euler method (cont' d)

$$y(t + h) \approx y(t) + hf(t, y(t)) \quad (3).$$

This formula is usually applied in the following way.

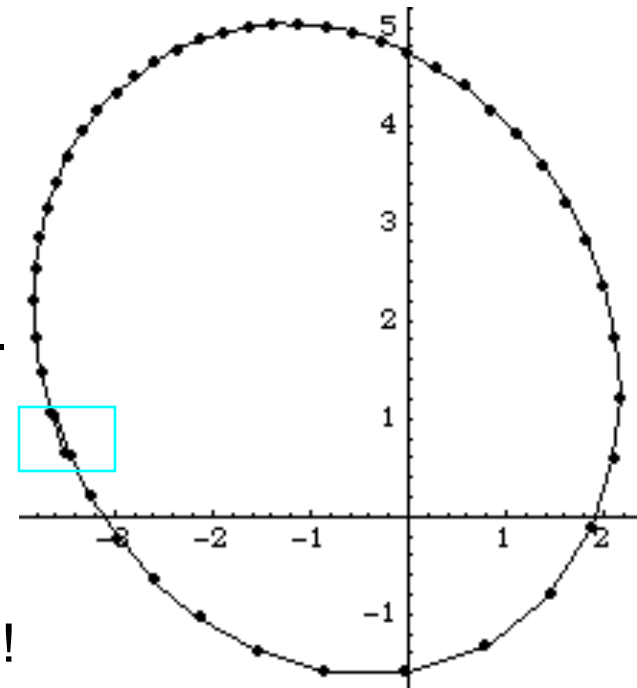
We choose a step size h , and we construct the sequence t_0 ,

$t_1 = t_0 + h$, $t_2 = t_0 + 2h$, ... We denote by \mathbf{y}_n a numerical estimate of the exact solution $y(t_n)$. Motivated by (3), we compute these estimates by the following recursive scheme

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \mathbf{f}(t_n, \mathbf{y}_n).$$

This is the *Euler method* (1768), most probably invented but not formalized earlier by Robert Hook.

It's a first (or second) order method, meaning that the total error is $\sim h^1$ (2) It requires small time steps & has only moderate accuracy, but it's very simple!



The classical fourth-order Runge-Kutta method

One member of the family of Runge-Kutta methods is so commonly used, that it is often referred to as "RK4" or simply as "*the* Runge-Kutta method".

The RK4 method for the problem

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \quad (1)$$

is given by the following equation:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$k_1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$$

$$k_4 = f(t_n + h, y_n + hk_3)$$

Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated slope.

Runge-Kutta 4th order (cont' d)

Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated 'slope' or 1st derivative (in fact, a space-time average of the r.h.s. of ODE)

$$\text{slope} = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}.$$

Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated slope. In RK4, the slope is a weighted average of derivatives:

- k_1 is the slope at the beginning of the interval;
- k_2 is the slope at the midpoint of the interval, using slope k_1 to determine the value of y at the point $t_n + h/2$ using Euler's method;
- k_3 is again the slope at the midpoint, but now using the slope k_2 to determine the y -value;
- k_4 is the slope at the end of the interval, with its y -value determined using k_3 .

When the four slopes are averaged, more weight is given to the midpoint.

The RK4 method is a fourth-order method, meaning that the total error after a fixed simulated time behaves like $\sim h^4$. It allows larger time steps & better accuracy than lower-order methods.

Symplectic integrators

In symplectic integration of Hamiltonian systems, the (x,p) volume of phase-space is preserved in time. This means that the total energy of the system does not drift in time, unlike in RK methods: a great advantage in very long astronomical simulations, for instance. Today RK integration is much less used in large-scale simulations of planetary systems, unless it is of very high order (RK78), or in applications *not* involving Hamiltonian dynamics of particles. (Heads up: not everything is Newtonian/Hamiltonian dynamics!)

Leapfrog method is a **2nd order symplectic** integrator – always use it for simple tasks in particle dynamics!

$f = f(x, v)$ # compute forces, usually just depending on position x
 $v = v + dt * f$ # it is implied that v & x are time-shifted (de-sync'ed) by $dt/2$
 $x = x + dt * v$ # so that x evaluations leapfrog over v evaluations.
(the order of updates is important!)

For Newtonian/Hamiltonian dynamics of particles, **4th order symplectic method** in one timestep evaluates r.h.s. of ODE only 3 times. (RK4 does it 4 times.) So you get better accuracy behavior at a smaller cost, than in RK4.

Algorithm: 4th Order Symplectic

Forest and Ruth (1990)

1. Push position: $x = x + c_1*v$
2. Calculate **force** (at an updated position)
3. Kick velocity: $v = v + d_1*a$

4. Push position: $x = x + c_2*v$
5. Calculate **force** (at an updated position)
6. Kick velocity: $v = v + d_2*a$

7. Push position: $x = x + c_3*v$
8. Calculate **force** (at an updated position)
9. Kick velocity: $v = v + d_3*a$

10. Push position: $x = x + c_4*v$

4th Order Symplectic integrator coefficients

Forest(1987), Forest and Ruth(1990), Candy and Rozmus(1991) obtained a 4th order integrator in a rather straightforward way with the result,

$$\begin{aligned}c_1 = c_4 &= \frac{1}{2(2 - 2^{1/3})}, & c_2 = c_3 &= \frac{1 - 2^{1/3}}{2(2 - 2^{1/3})}, \\d_1 = d_3 &= \frac{1}{2 - 2^{1/3}}, & d_2 &= \frac{-2^{1/3}}{2 - 2^{1/3}}, & d_4 &= 0.\end{aligned}\tag{37}$$

Yoshida (1993) paper about symplectic integration delves into more details, including some disadvantages w.r.t. RK4:

<https://planets.utsc.utoronto.ca/~pawel/PHYD57/Yoshida-paper-1993.pdf>

- **Tutorial** programs for Fortran and C are on our web page. Use the last link in the Languages section to review the usage of C.
- Programs are placed in /progD57 directory and in art-1 program directory
- Look at C-version of arithmetic/bandwidth demonstration program simpler-nb-3ra.c, which does not work (produces Segmentation fault, indicating a problem with memory management; as we say, this code 'segfaulted'). The reason is that the program tries to deposit results in a non-existing array element `tab[6][k]` . This needs to be changed to `tab[5][k]` because 6 indices of the 1st dimension are: 0,1,...,5. (C, like Numpy, has 0-base arrays!)
- simpler-nb-3ra.f90 is the corrected F90 program, and the version with swapped indices is called simpler-nb-3da.c
- simpler-nb-3da.f90 and simpler-nb-3ra.f90 are the Fortran versions

❖ Massively parallel integration on the newest HPC platforms: CPU, GPU and MIC

from a conference talk at UTSC, 2017,
discussing work by P. Artymowicz
and u/g student F. Horrobin

Concurrent simulation of 200 or 7000 planetary systems on
CPUs or MIC

Collisionless gigaparticle disks. Interaction with binary system.

Hybrid algorithm (4th order symplectic with collisions)
Implementation and optimization in Fortran90 on 1..32 MIC (Φ)
Migration problem
Tests and preliminary results
Fast migration in particle disks as type III CR-driven migration

1990s and 2000s was the era of clusters



MPI for parallelization.

Later, in 2000s, coprocessors
appeared.... like

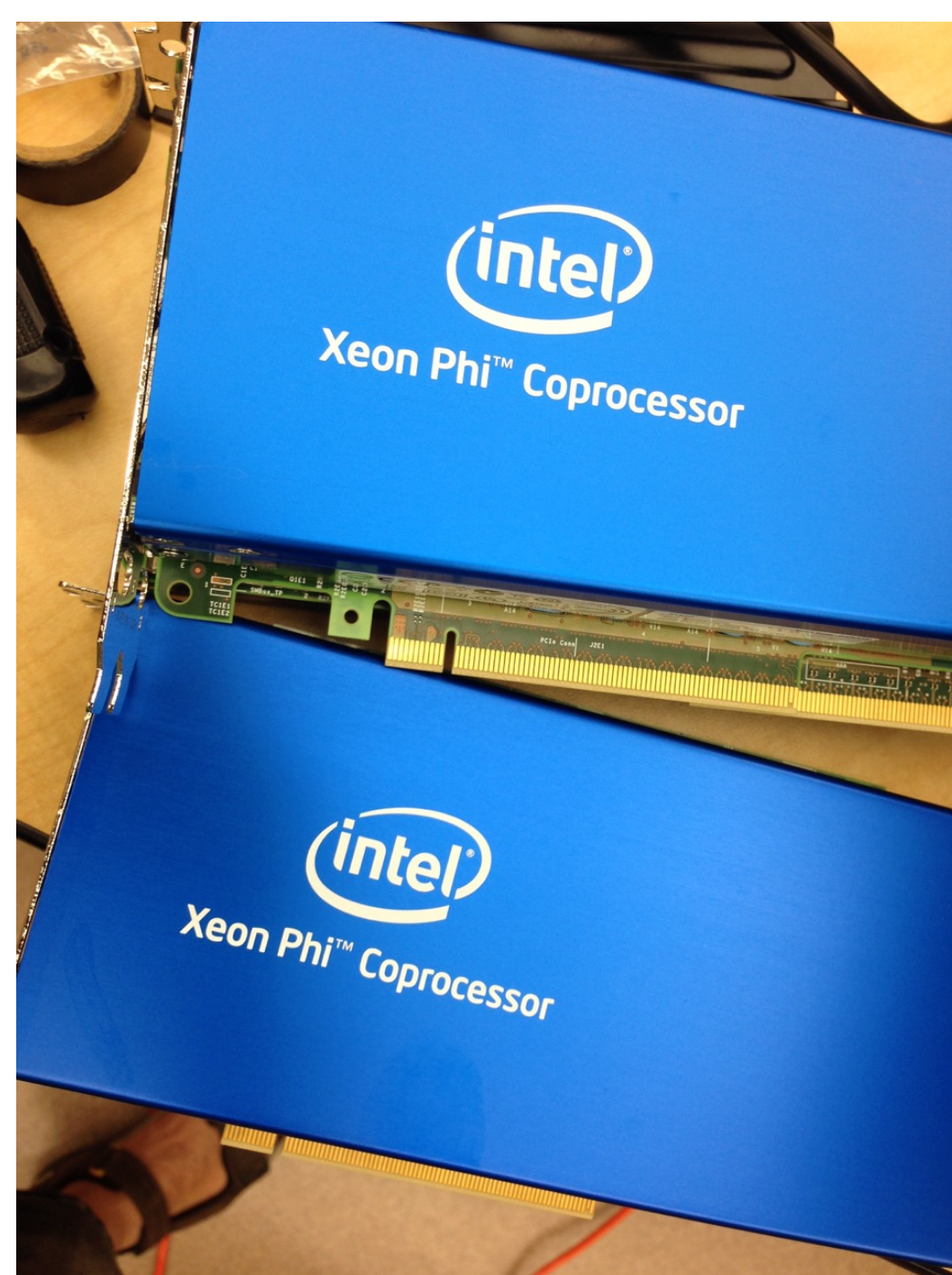
MIC

MIC = many integrated cores
(Intel's term for many-core, massively parallel, CPU-like
processors)

and GPU

GPU = Graphics Processing Unit (processor inside graphics
card, actually more capable of quick computation than CPU).

It seemed that the we won't bother to build clusters any more, but it wasn't
true.



MIC = many integrated CPU-like cores (~60)

Intel Xeon Phi accelerators

Knights Corner:

~1 TFLOP dp

~2 TFLOP sp

Knights Landing: ~3x more TFLOPs

TFLOP = 1 T FLOP/s.

1 T = $\sim 10^{18}$ exa

1 P = $\sim 10^{15}$ peta

1 T = $\sim 10^{12}$ tera

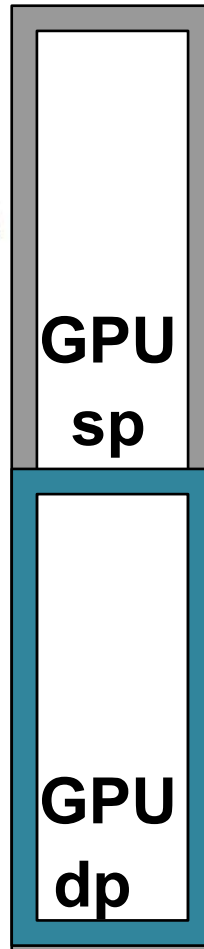
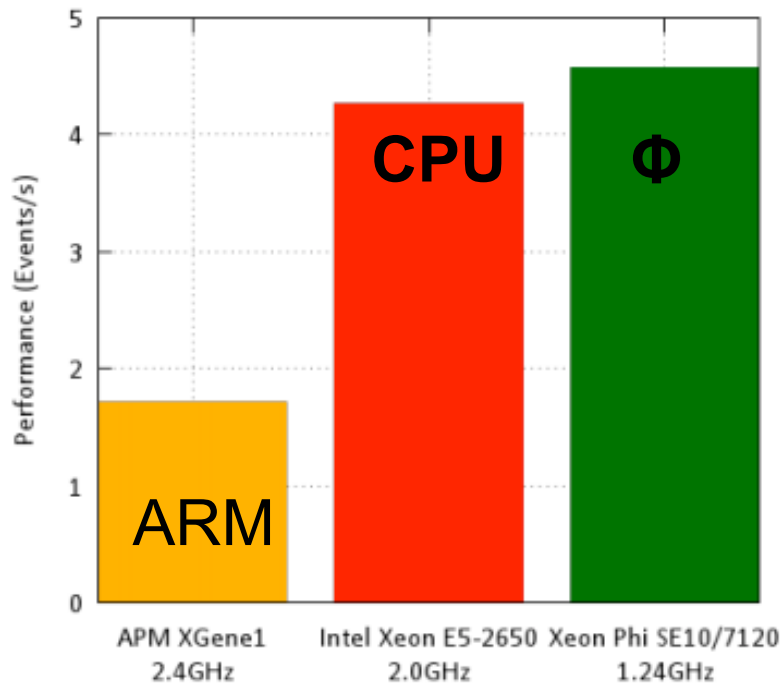
1 G = $\sim 10^9$ giga

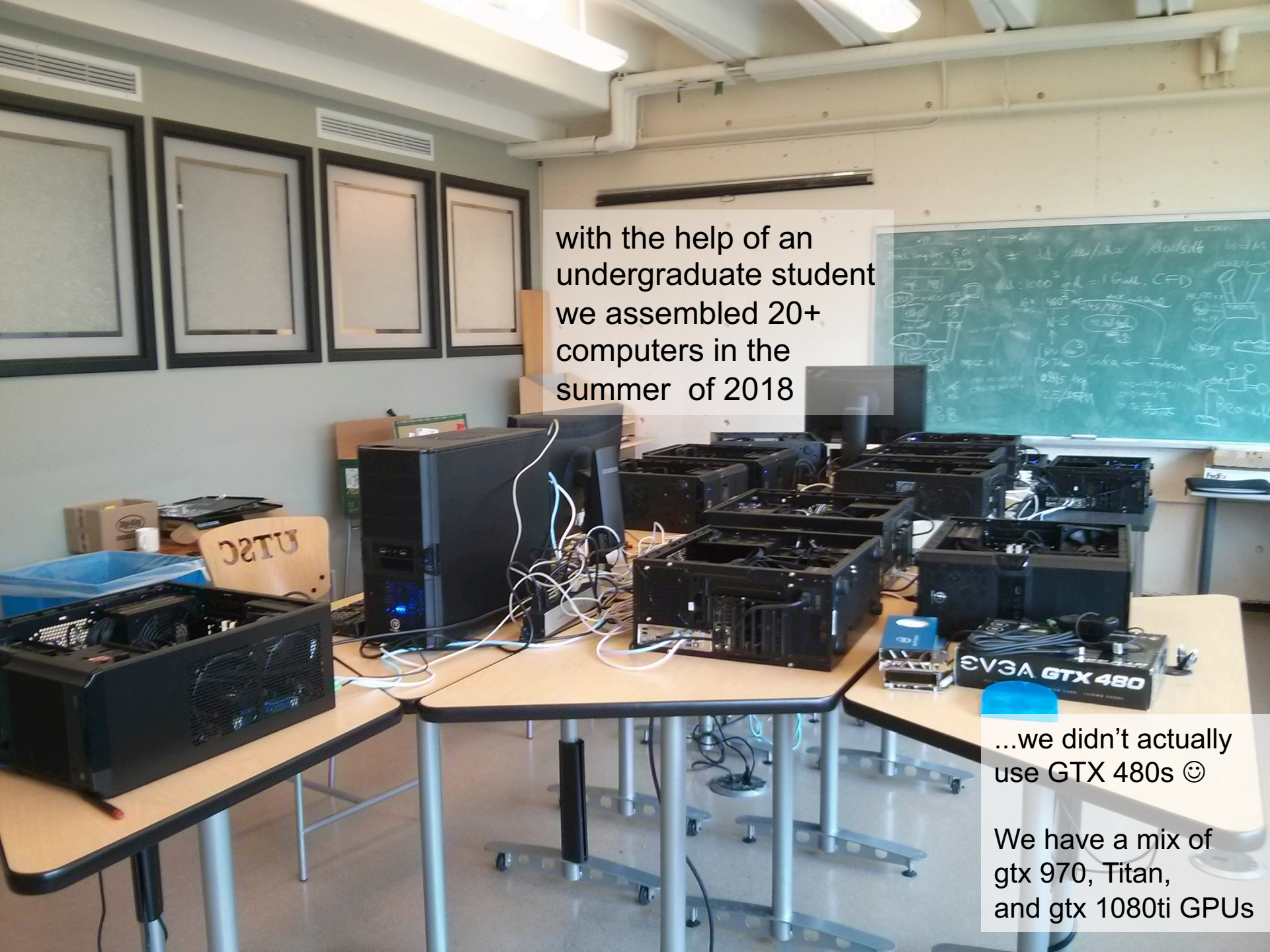
1 M = $\sim 10^6$ mega

1 K = $\sim 10^3$ kilo

In 2014, CERN Researchers considered which of the platforms makes the most sense for distributed Worldwide LHC Computing Grid, processing data for Large Hadron Collider experiments in 170 computing centers, in 40 countries (incl. UofT). The height of the bar is proportional to the estimated speed in CERN simulations with then-current hardware. Nowadays GPU have somewhat more advantage over CPU & MIC

*[dp = double precision (8B/float like in Python, 15 accurate decimal places),
sp = single precision (4B/float, 7 decimal places accuracy)]*



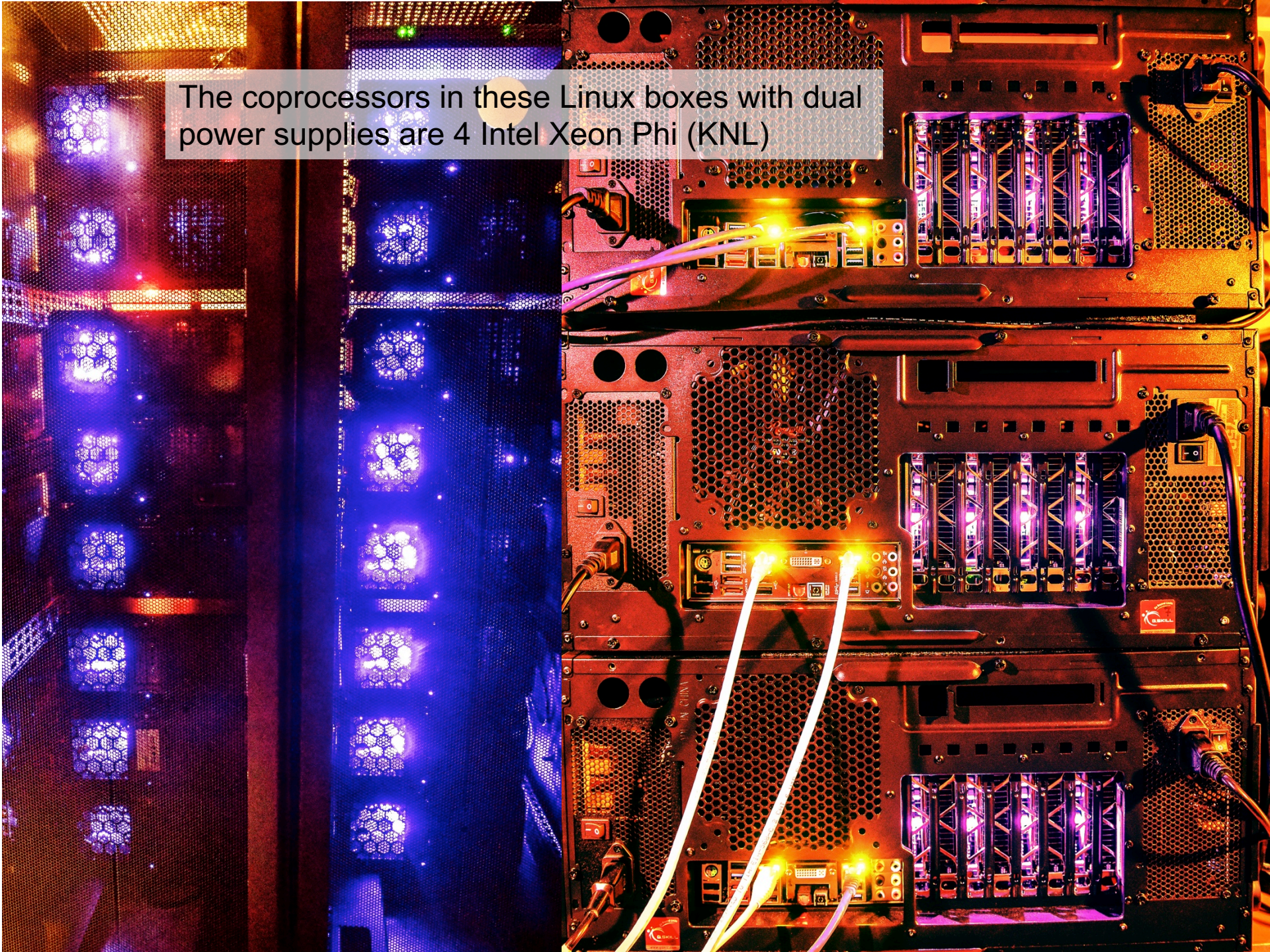


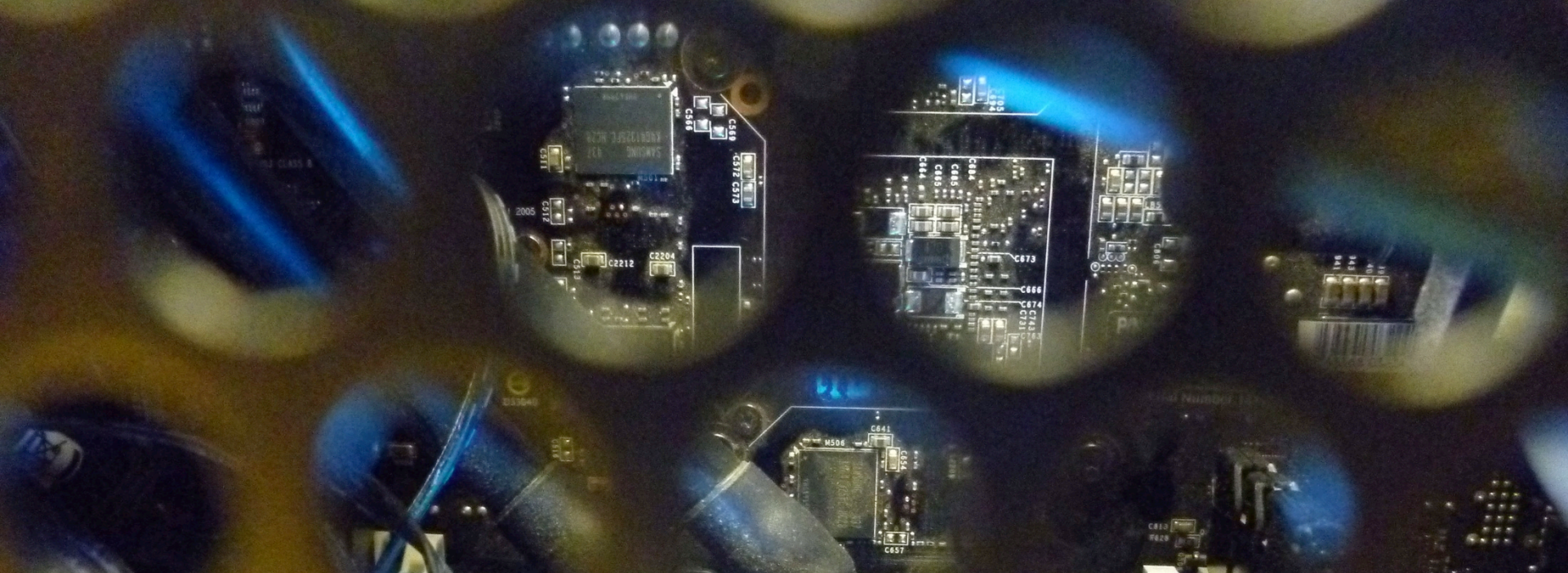
with the help of an
undergraduate student
we assembled 20+
computers in the
summer of 2018

...we didn't actually
use GTX 480s 😊

We have a mix of
gtx 970, Titan,
and gtx 1080ti GPUs

The coprocessors in these Linux boxes with dual power supplies are 4 Intel Xeon Phi (KNL)





In a small cluster, CPU, MIC and GPU are combined. They can work together or separately. We can simulate a galaxy's inner part star by star (~4 G stars), and/or its gas disk at high resolution (2 Gcell) by exchanging data between linux nodes in every time step.

We can run 200 8-planet simulations very fast (1G periods simulated in a day at 360+ steps per orbit), or 7000 simulations, 5 times slower

Large N-body systems by direct summation

20 arithmetic operations per one pairwise grav. interaction

leapfrog (**Fortran90**)

leapfrog (**CUDA C**)

N = 10 K ... 1 M

CPU (i7-5820K 4GHz)

MIC (KNC)

GPU (gtx 980, Titan)

0.28 TFLOP sp
14 G interac/s

1.33 TFLOP sp
67 G interac/s

3.5 TFLOP sp (gtx980)
190 G interac/s

0.09 TFLOP dp
4.5 G interac/s

0.51 TFLOP dp
25 G interac/s

0.81 TFLOP dp (Titan)
40 G interac/s

! on MIC the calculation is **2.8 times slower** than on GPU (sp)

! **1.6 times slower** than on GPU (dp)

! CPU (6c.) is **9..13 times slower** than GPU

!
note: this is a rare fully compute-bound calculation!

Concurrent 8-body systems by 4th order symplectic code

n8b-aug14.3.f90

Same double precision program. Compiled with ifort

platform	CPU	MIC
compiler flag	-xhost	-mmic
number of N-body systems per processor	12	224
N [#threads per sys.]	8 [1]	8 [1]
exec. time per step	0.871 μ s	4.58 μ s
steps per orbit	360	360
exec. time of 1 orbit	0.313 ms	1.65 ms
exec. time (1G orbits)	3.63 days	19.1 days
system clock	4 GHz	1.1 GHz
throughput	13.8 M sys-step/s	49 M sys-step/s
# concurrent systems (SciPhi cluster UTSC)	192	10752

Practical capabilities of processor platforms for dynamical astro-calculations. Single (co)processors

CPU ~ E5 and i7 ser. (Intel), MIC = Knights Corner (Intel 2013),

GPU = Nvidia GTX970..1080 (sp) and Titan (dp) run:

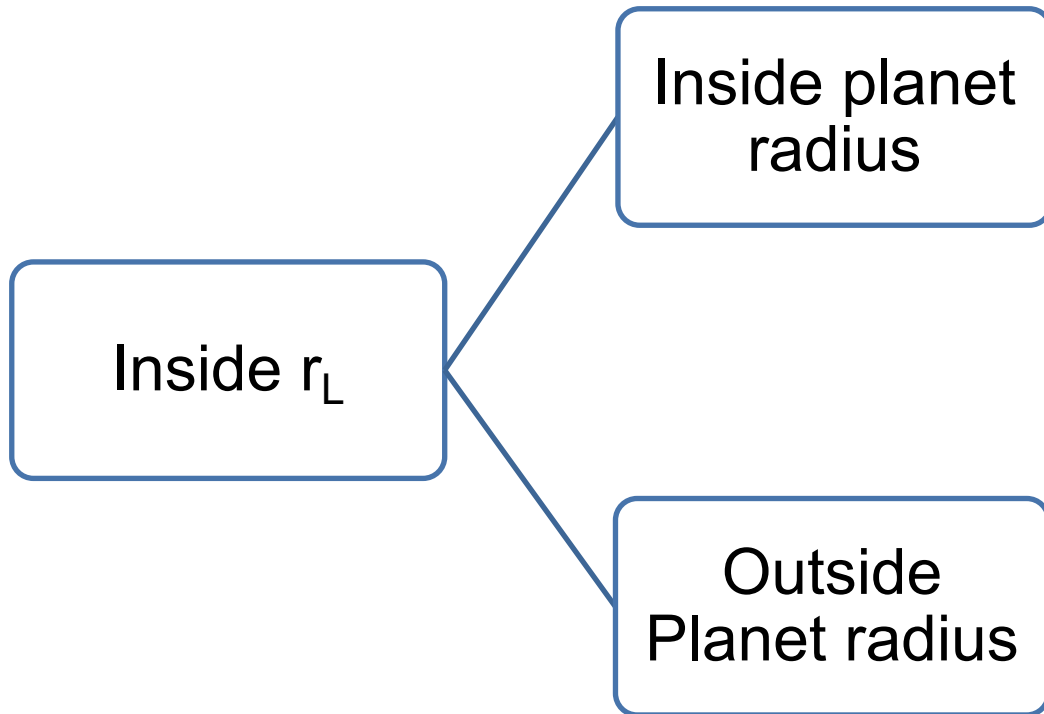
1. Gravit. **N-body problem** $O(\sim N^2)$. $N \sim 10^6$ real-time (~ 1 fps)
GPU > MIC ~ CPU (mostly comput. limited, > TFLOP)
2. Disks of **particles** (stars; asteroids, planetesimals, meteoroids and dust).
 $\sim 10^9/s$, $\sim 10^8$ in RAM, (~ 10 fps)
GPU ~ MIC > CPU (bandwidth-limited to 150 GB/s)
3. Pure CFD = **fluids**, cells: $\sim 10^8/s$, $\sim 10^8$ in RAM
GPU ~ MIC ~ CPU (mostly bandwidth limited) , (~ 1 fps)

GPU – some have decent double precision, most don't.

Somewhat difficult to program and optimize, compared to x86 platforms. Very fast on direct summation.

Collisionless gigaparticle disks can be simulated with the 4th order symplectic algorithm described earlier in this lecture.

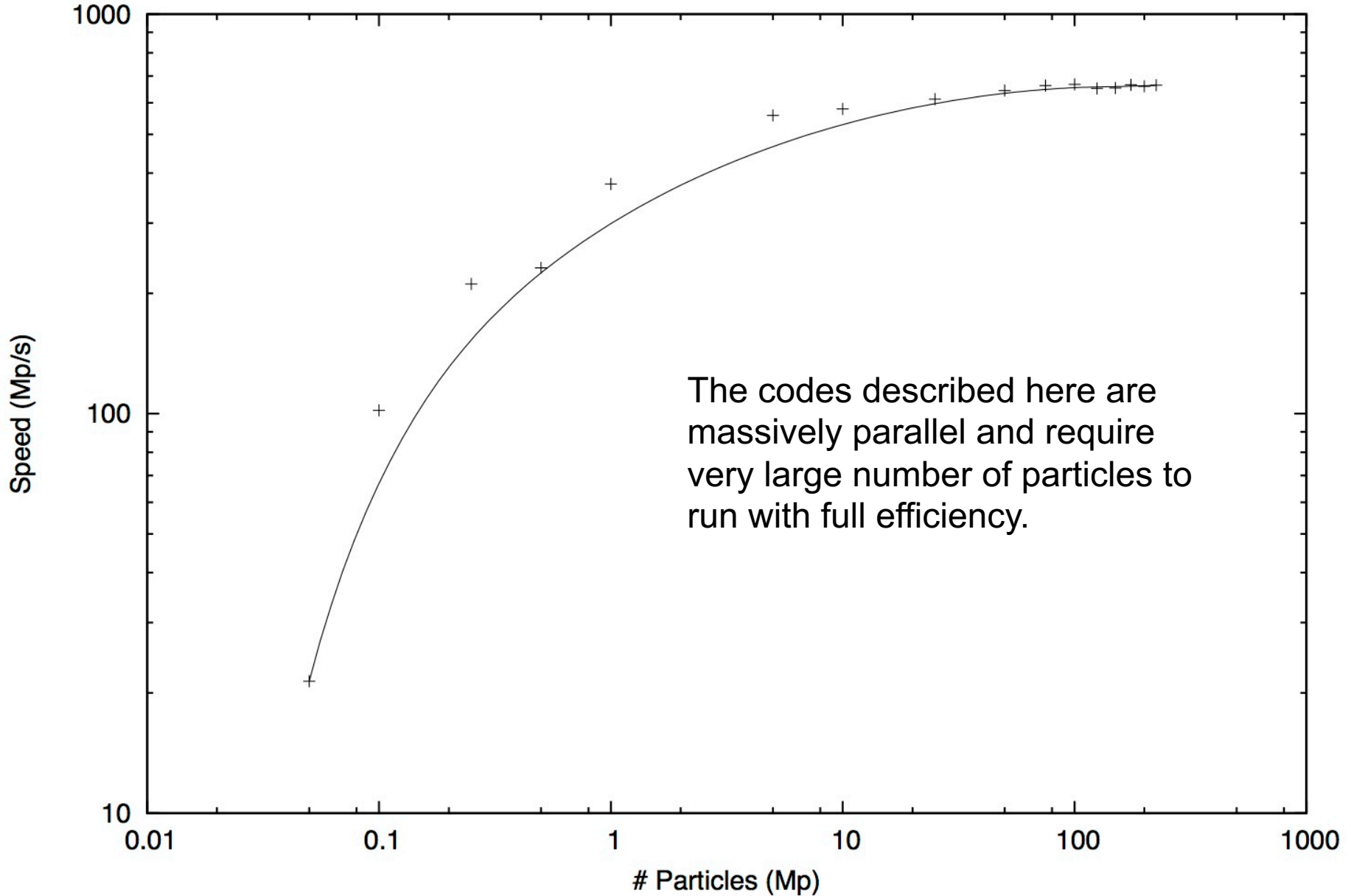
Collision with a Binary and Variable dt



- Store particle and set to large r in main array
- Remove from array
- Transfer momentum and cm position
- Increase mass and spin

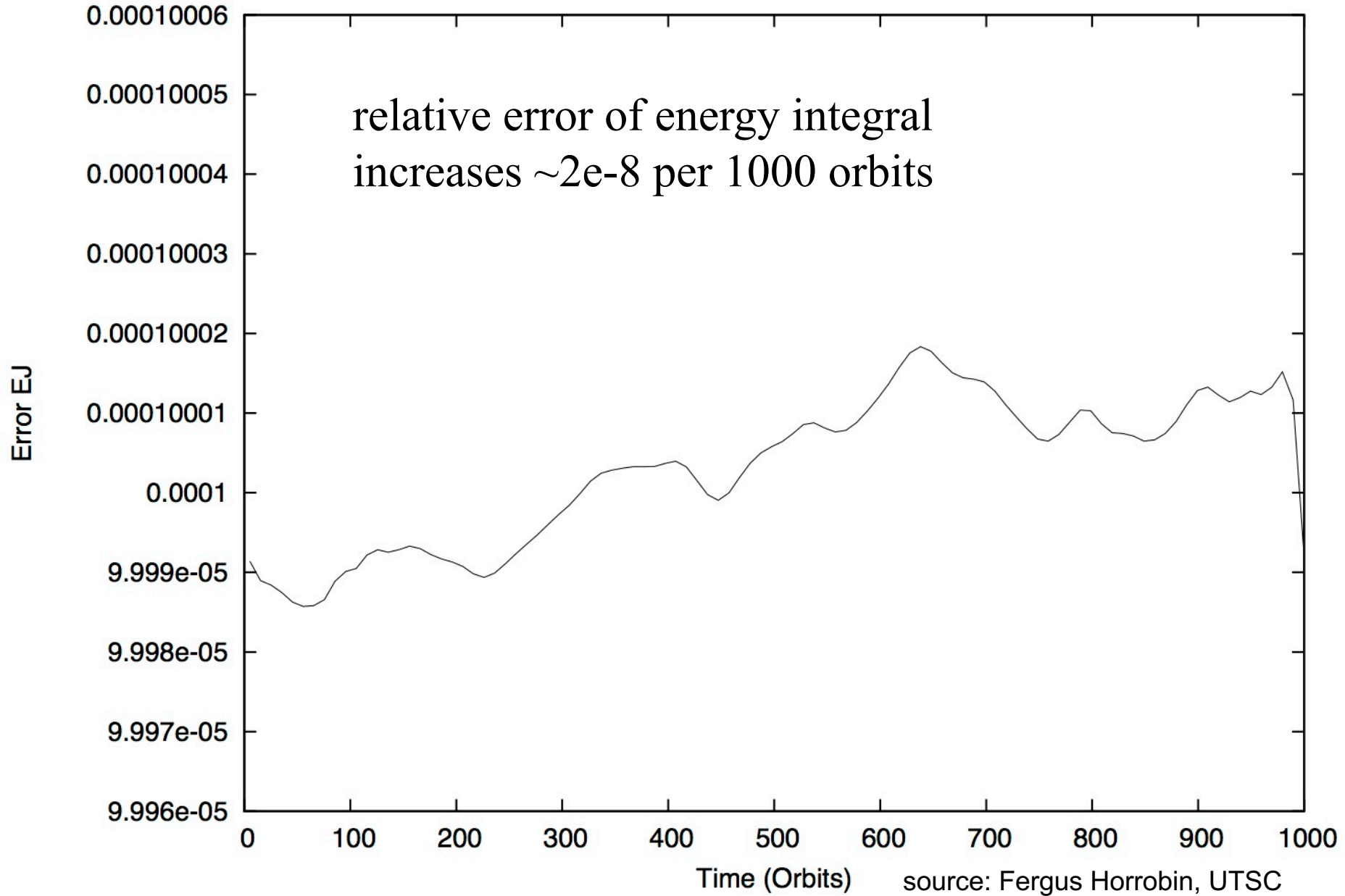
- Store particle and set to larger r in main array
- Perform same scheme but with variable dt
- Range $1e-8 - dt_1(0.004)$

Speed vs Number of Particles



The codes described here are massively parallel and require very large number of particles to run with full efficiency.

Jacobi Constant Error vs Time, MP = 0.001, MD = 10e-5

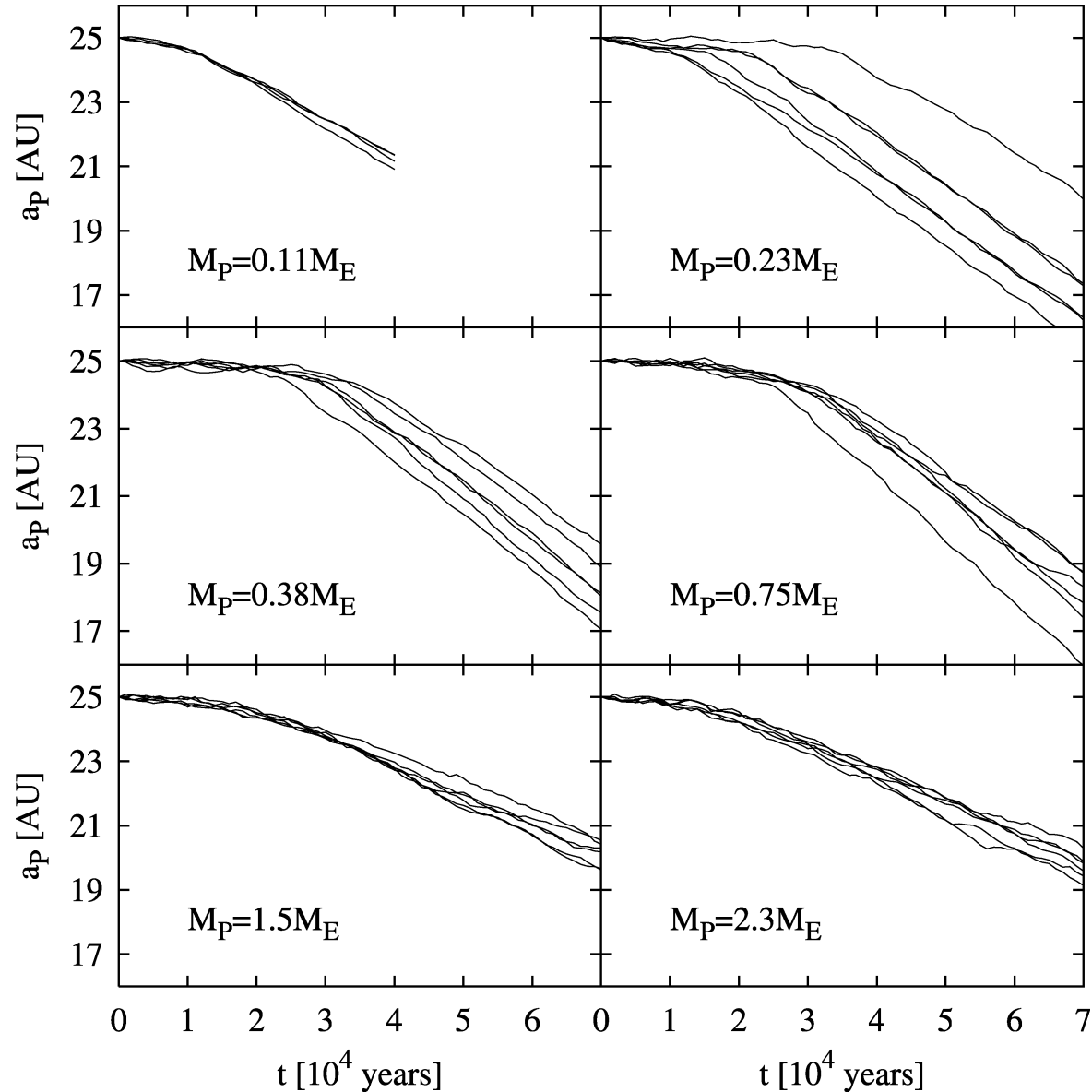


We study type III migration in Disks

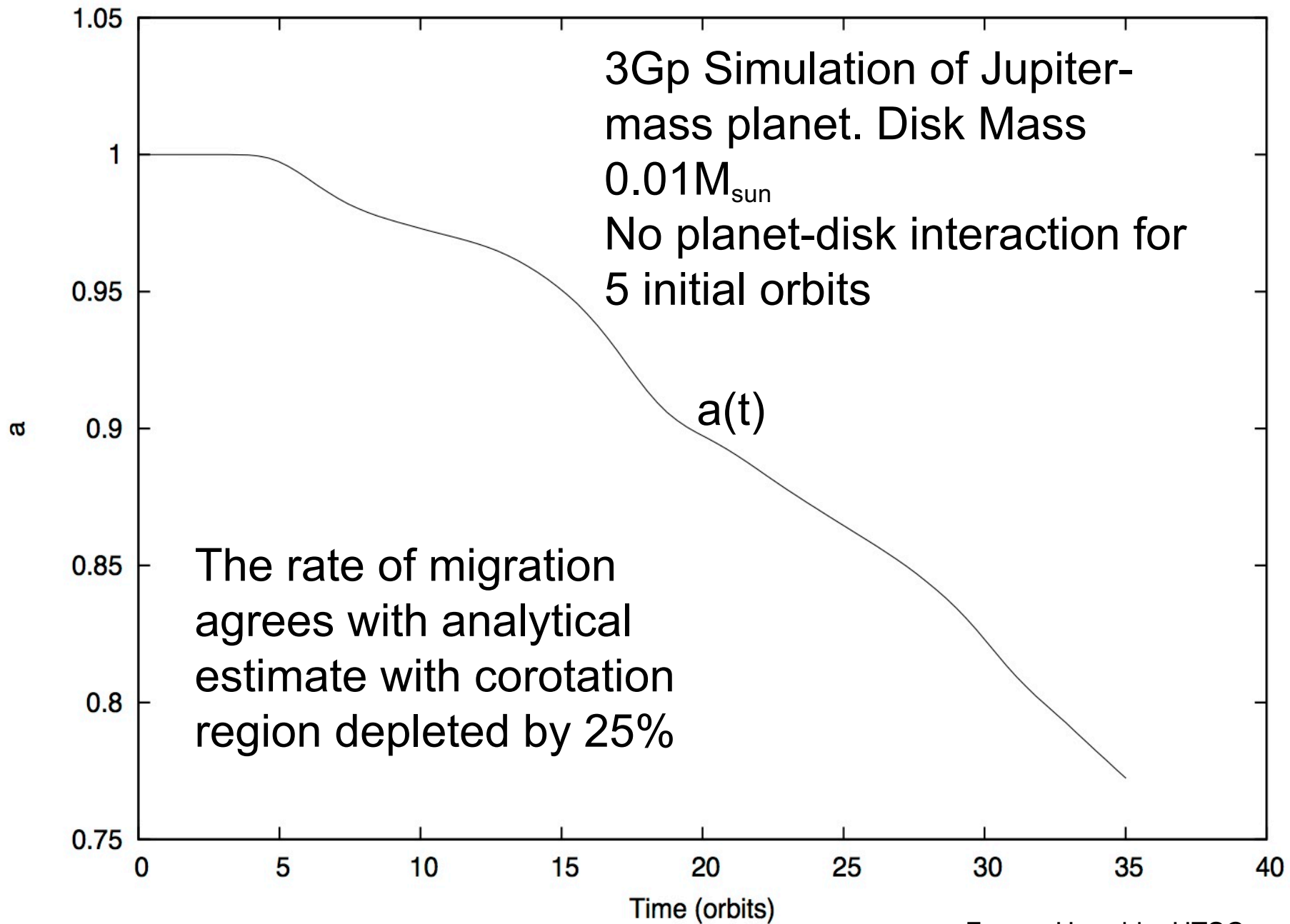
- Very rapid migration in *gas* disks: 40-50 orbits timescale for Jupiter-mass planet in a solar nebula disk
(Papaloizou et al. in Protostars and Planets V, 2005)
- Rate does not depend on mass of planet
- Criterion compares disk (in CR = corotation region) and planet masses:
 - $M_p < M_{\text{deficit}}$. Difficult to satisfy by planetesimals...

Previous results: Kirsh et al. 2009 identified the fast migration and offered an explanation [without noticing a connection with type III migration, e.g. as reviewed by Papaloizou et al. 2006, PP V]

Much slower migration by mean-motion resonant scattering (w/similarly v. massive disks) proposed by Murray et al (1998).



Semi Major Axis vs Time

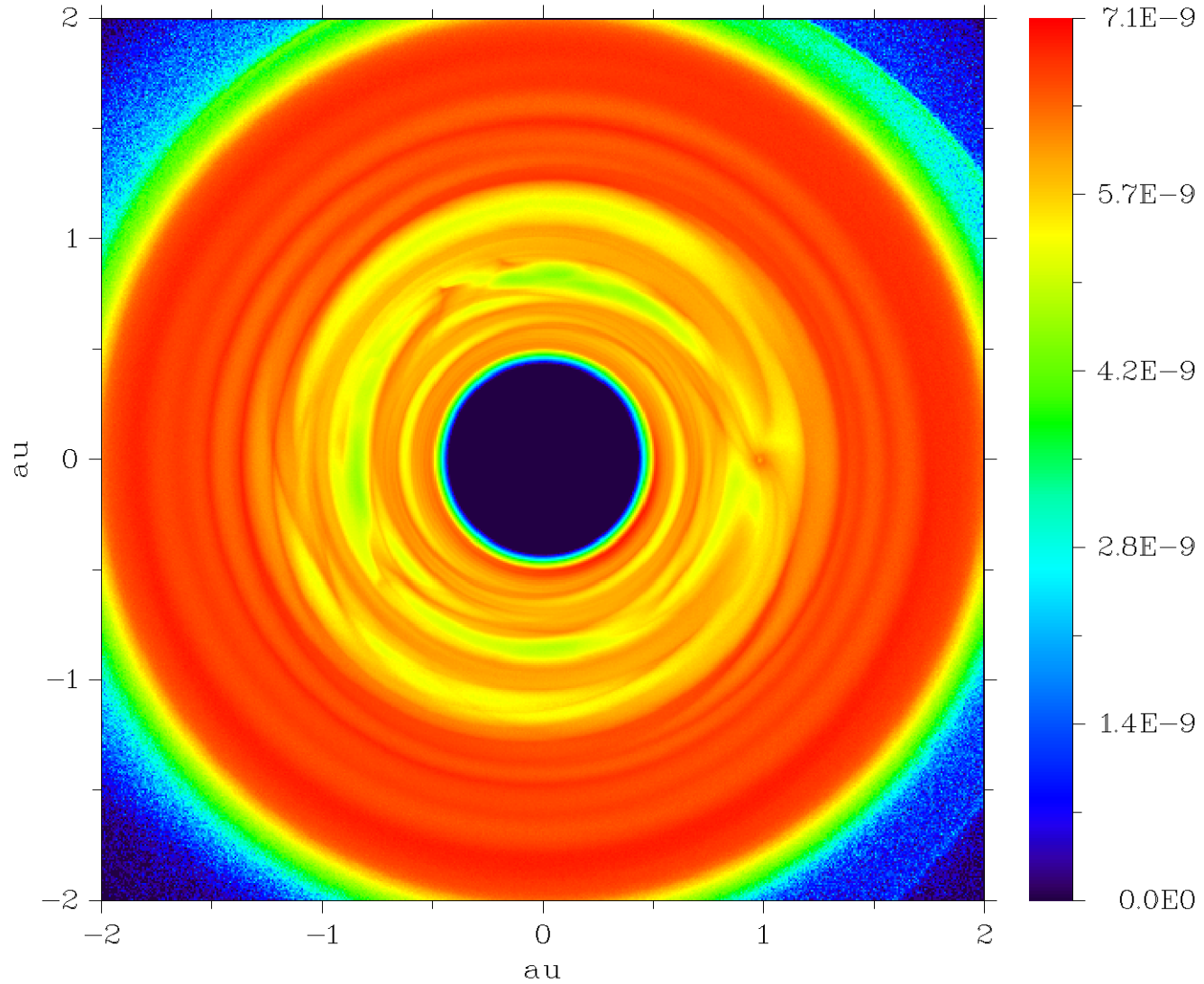


3Gp Simulation of Jupiter-mass planet. Disk Mass $0.01M_{\text{sun}}$
No planet-disk interaction for 5 initial orbits

The rate of migration agrees with analytical estimate with corotation region depleted by 25%

Density Plot of X, Y Plane

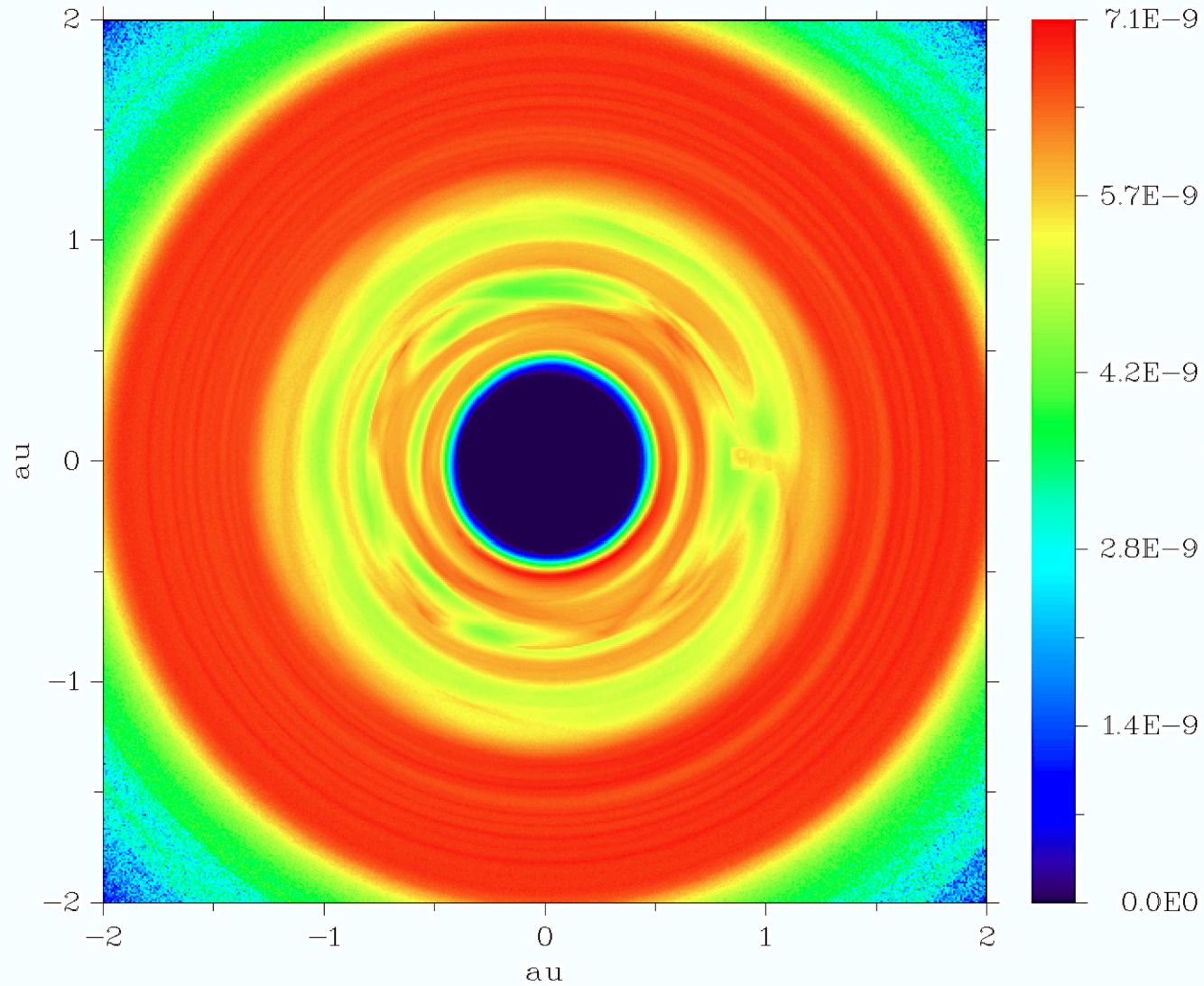
Time: 8.0 Orbits



source: Fergus Horrobin, UTSC

Density Plot of X, Y Plane

Time: 24.9 Orbits



source: Fergus Horrobin, UTSC

Conclusions of Fergus Horrobin's summer research in 2017

For large-scale particle integrations in non-collisional disks, codes can run v. fast on MIC cluster (Xeon Phi)

- 3+ billion particles (150M per MIC), timestep ~ 0.2 s
- Hybrid parallelization method combining OpenMP and MPI seems best for this type of platform
- We've implemented 4th order symplectic integrator.
- Though deeper analysis must be made, we see similarities between gas and particle disks in the context of rapid migrations

N-body simulations of the Universe

- <https://www.youtube.com/watch?v=YjUICiYICYE>
- Millenium – 10+G particles Gadget code,
- kept the main supercomp at MPI Inst of Astronomy in Garching, Germany, busy for a month in 2004
- $(700 \text{ MPc})^3$

- <https://www.youtube.com/watch?v=32qqEzBG9OI>
- $(350 \text{ Mpc})^3$, $5e4$ galaxies, 12G particles, 8k CPUs
- Millenium XXL



N-body simulations of the Universe

- <https://www.youtube.com/watch?v=YjUICiYICYE>
 - <https://www.youtube.com/watch?v=32qqEzBG9OI>
- $(350\text{Mpc})^3 = (1 \text{ billion ly})^3$, 50K galaxies, 12G particles
- Simulation name: Bolshoi
 - Run on Pleiades cluster (supercomputer) at NASA Ames Research Center in Mountainview, California.



N-body simulations of the Universe

12G particles create 50000 galaxies, gas: Adapt. Mesh Refinement grid, 8k CPUs used for Bolshoi-Planck simulation



Pleiades has theor. peak performance 7.3 PFLOPS

1. Astrophysical problems for CPU and GPU calc's:

Disk-planet interaction and migration

Disks with structure: IRI (irradiation instability in particle and gas disks)

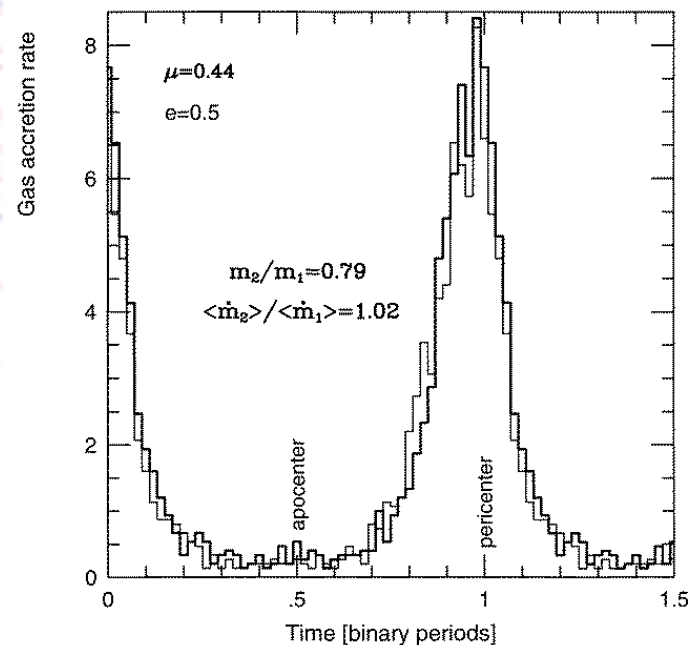
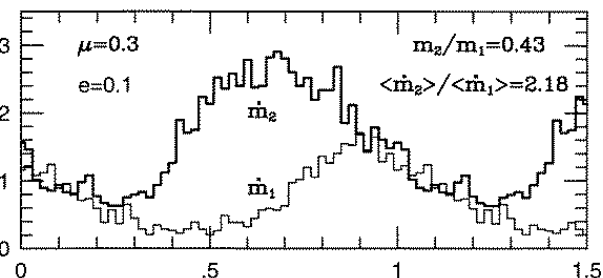
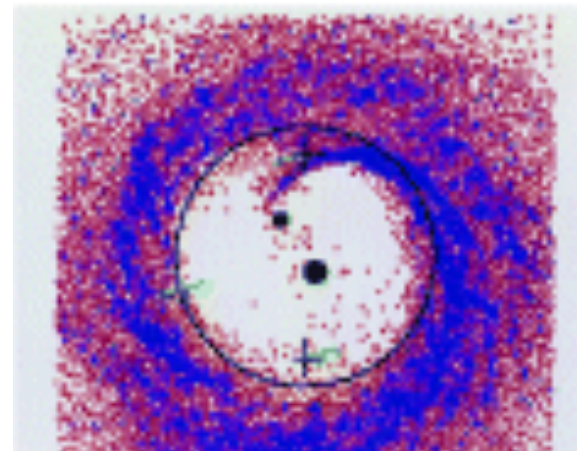
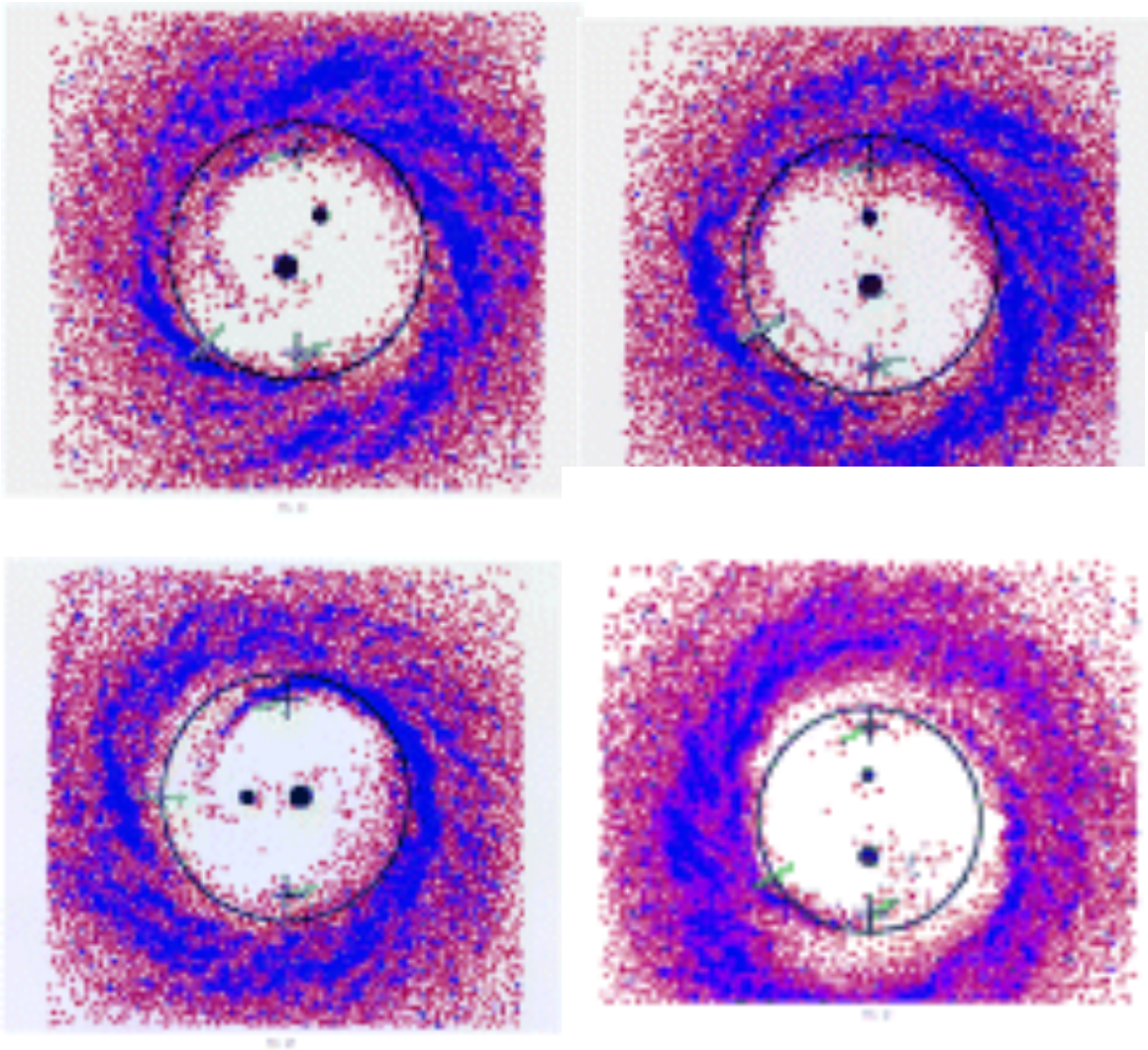
Flow of gas around Super-Earth ($5 M_E$)

2. Massively parallel numerics on mini-supercomputers:

Comparison of HPC platforms: CPU, GPU, and MIC (Φ)

UTSC clusters

Binary-disk interaction

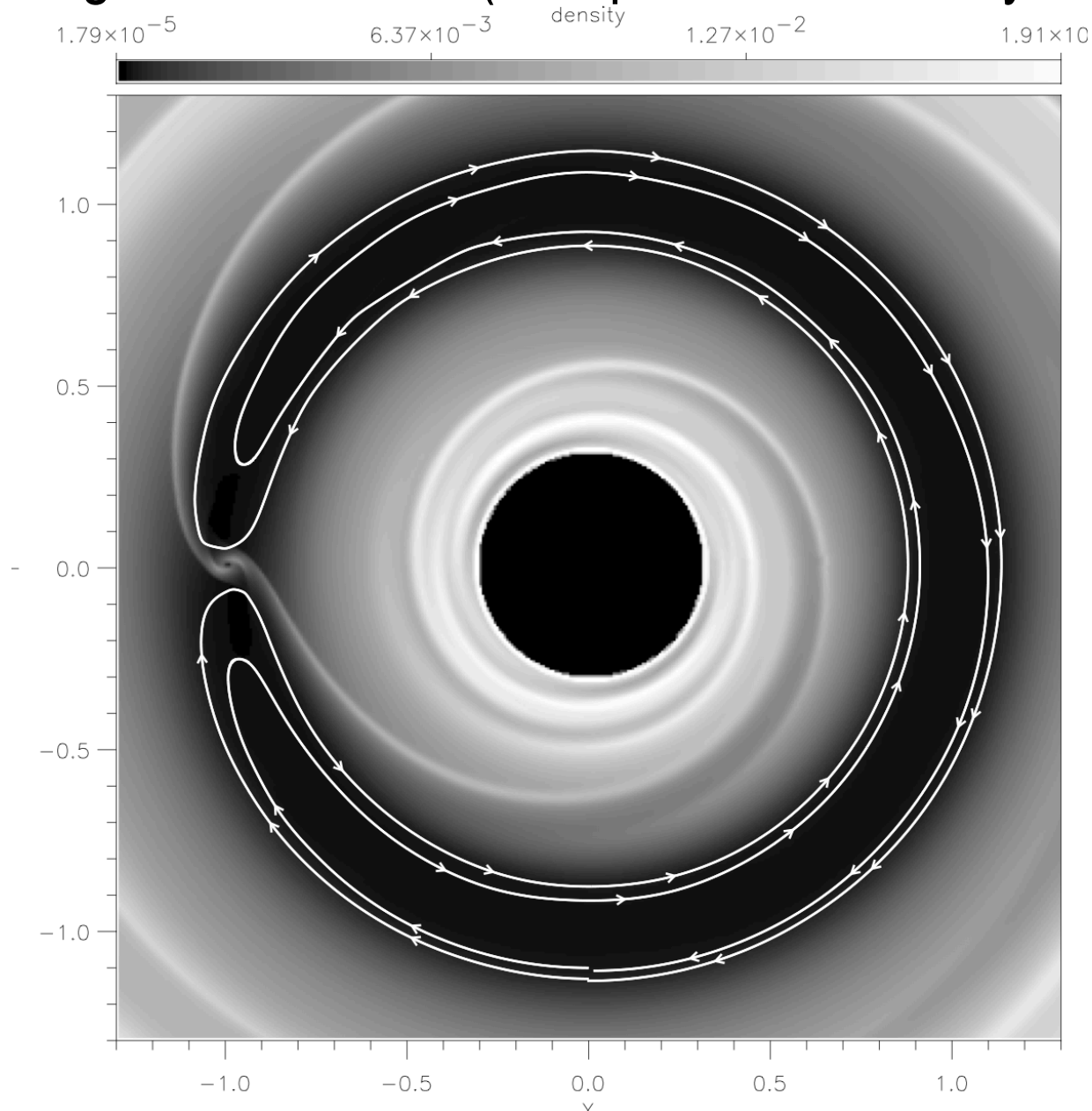


SPH = smoothed hydrodynamics: cf. wiki

Artymowicz and Lubow (1996)

Binary-disk interaction

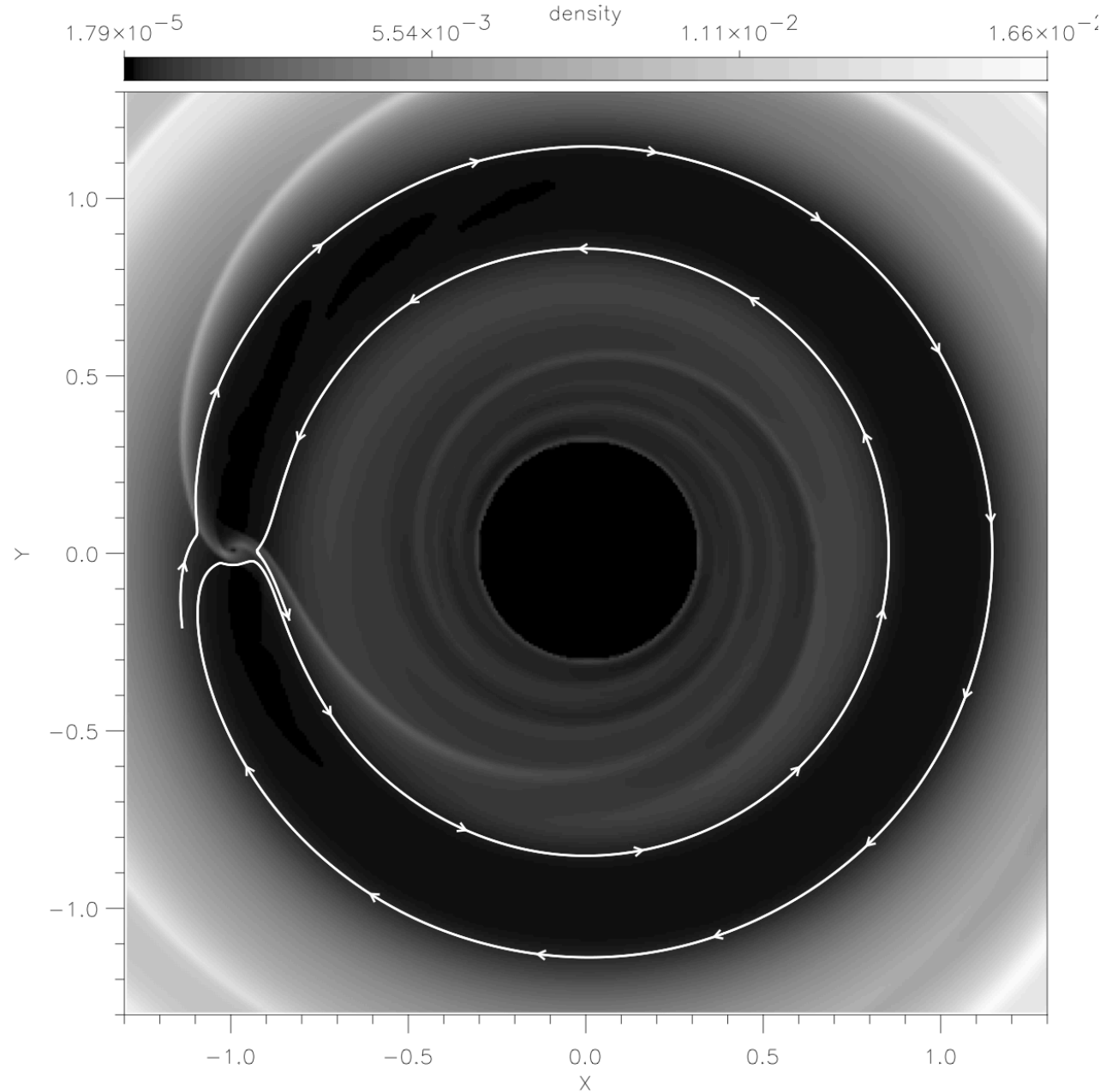
method: grid-based CFD (Computational fluid dynamics)



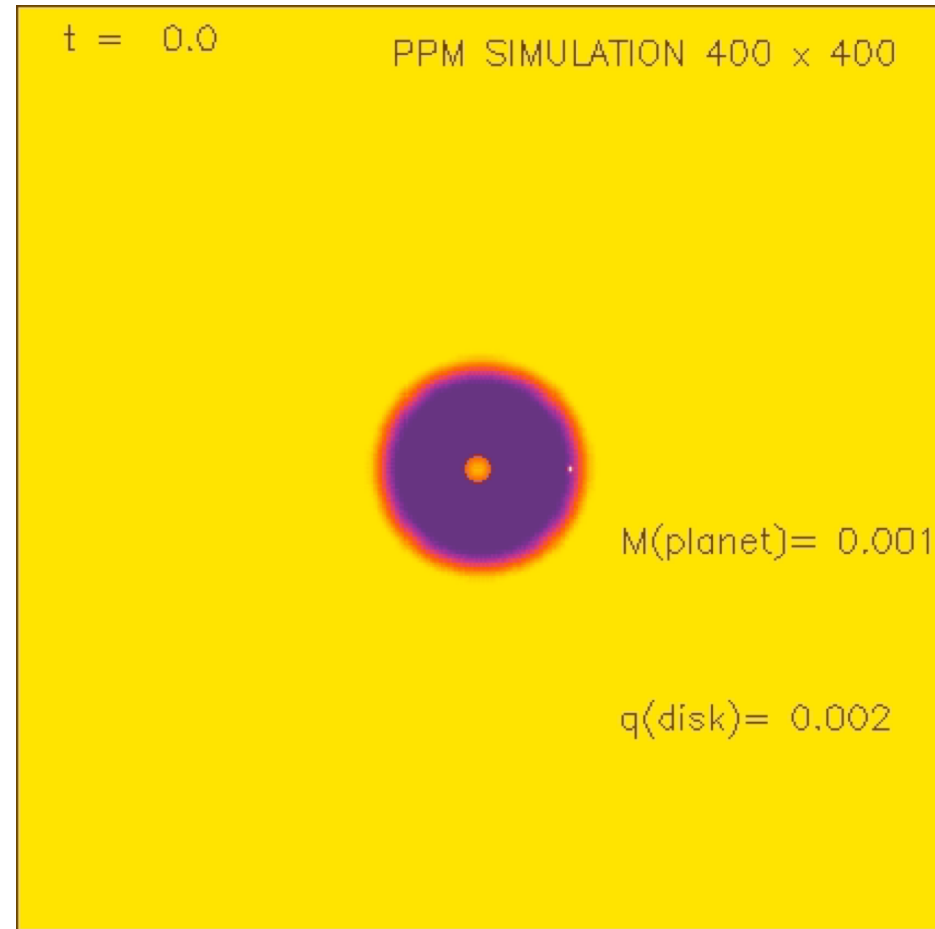
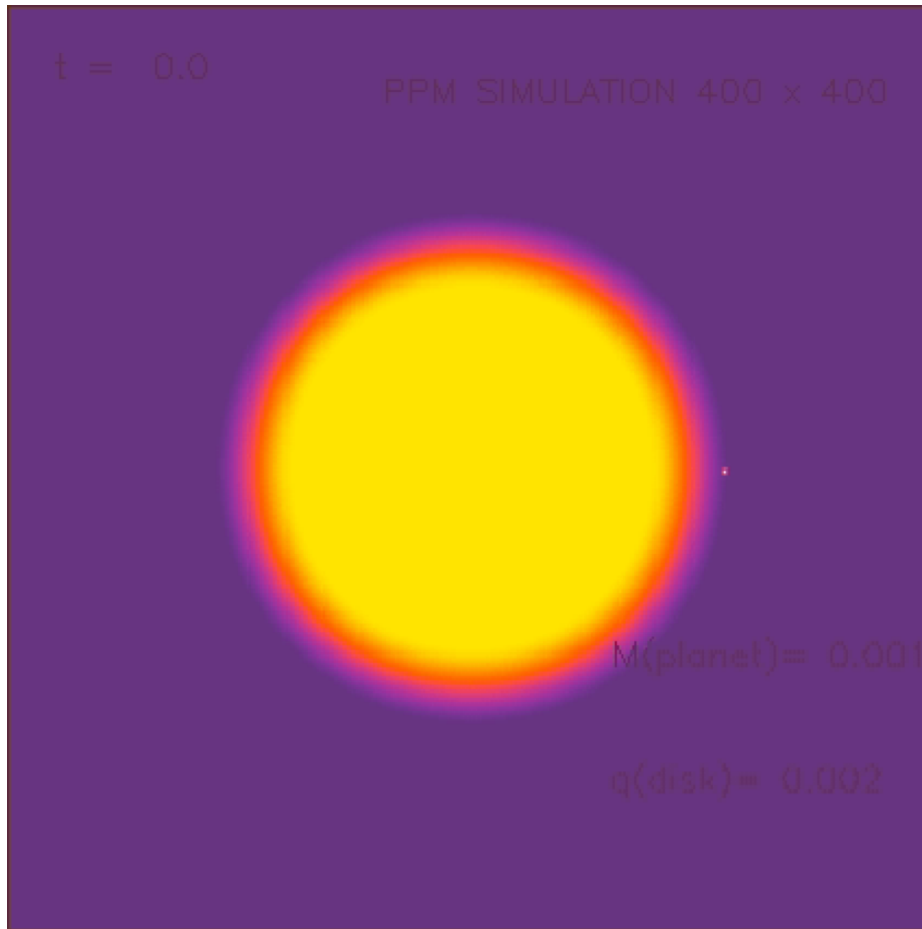
CPU 2-d

2nd order
ZEUS
hydro

notice mass
flow through
gap



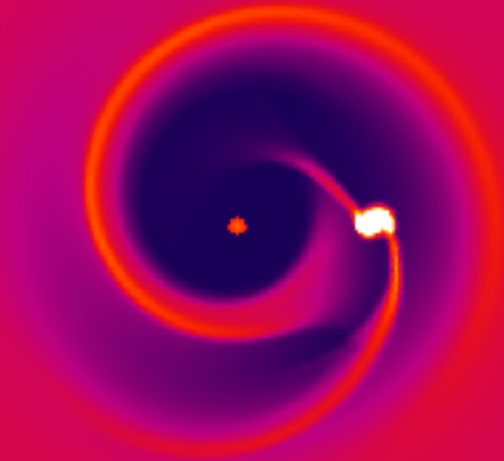
One-sided disk (**inner/outer disk only**). The rapid inward migration is **OPPOSITE** to the expectation based on shepherding (Lindblad resonances).



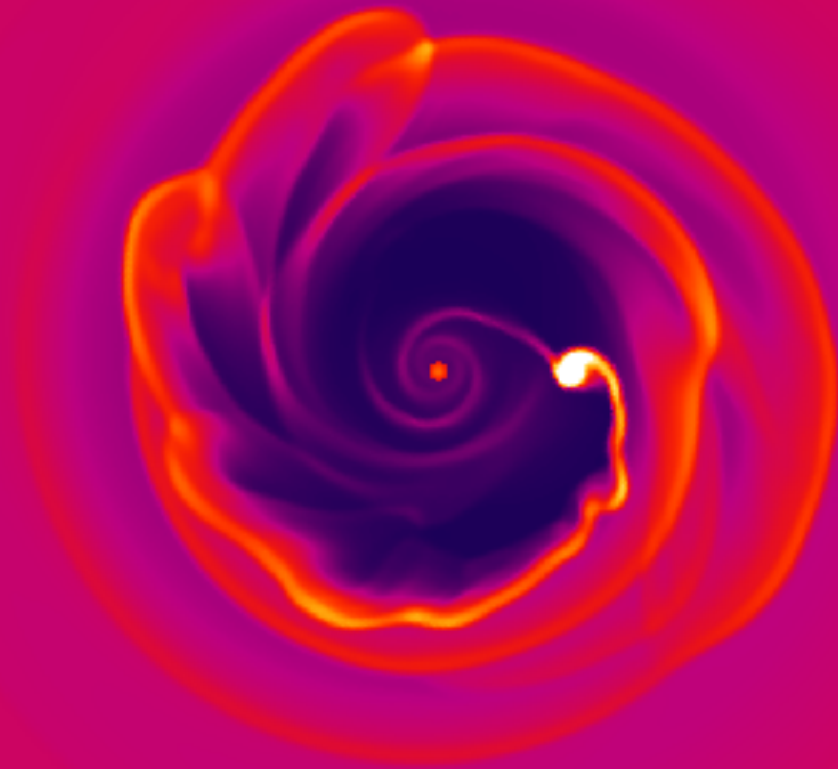
Like in the well-known problem of “sinking satellites” (small satellite galaxies merging with the target disk galaxies), **Corotational torques** cause rapid inward sinking.

A few snapshots from a 2-D simulation of a brown dwarf circling a star interacting with the circum-binary disk. Density of gas is color-coded.

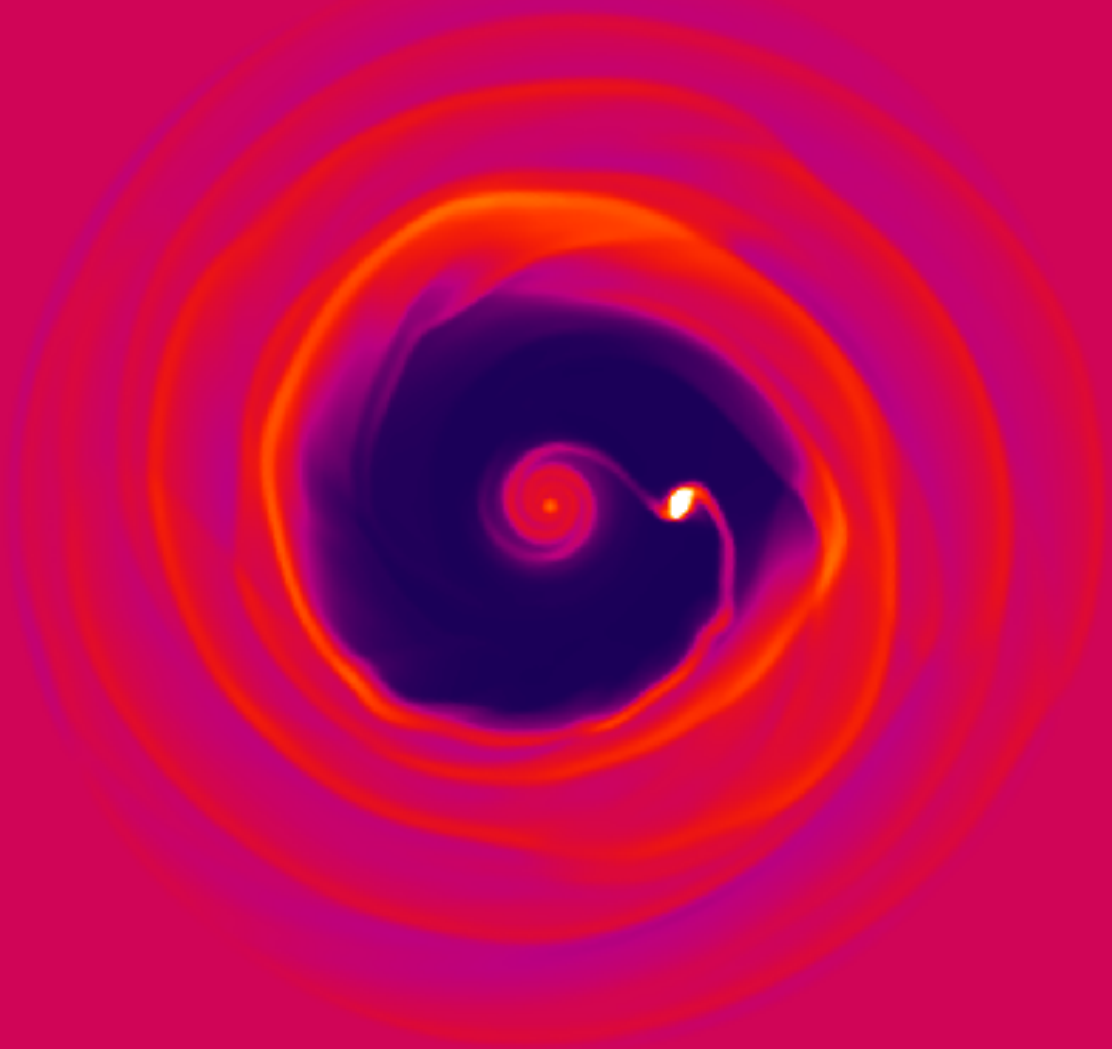
t = 0.8



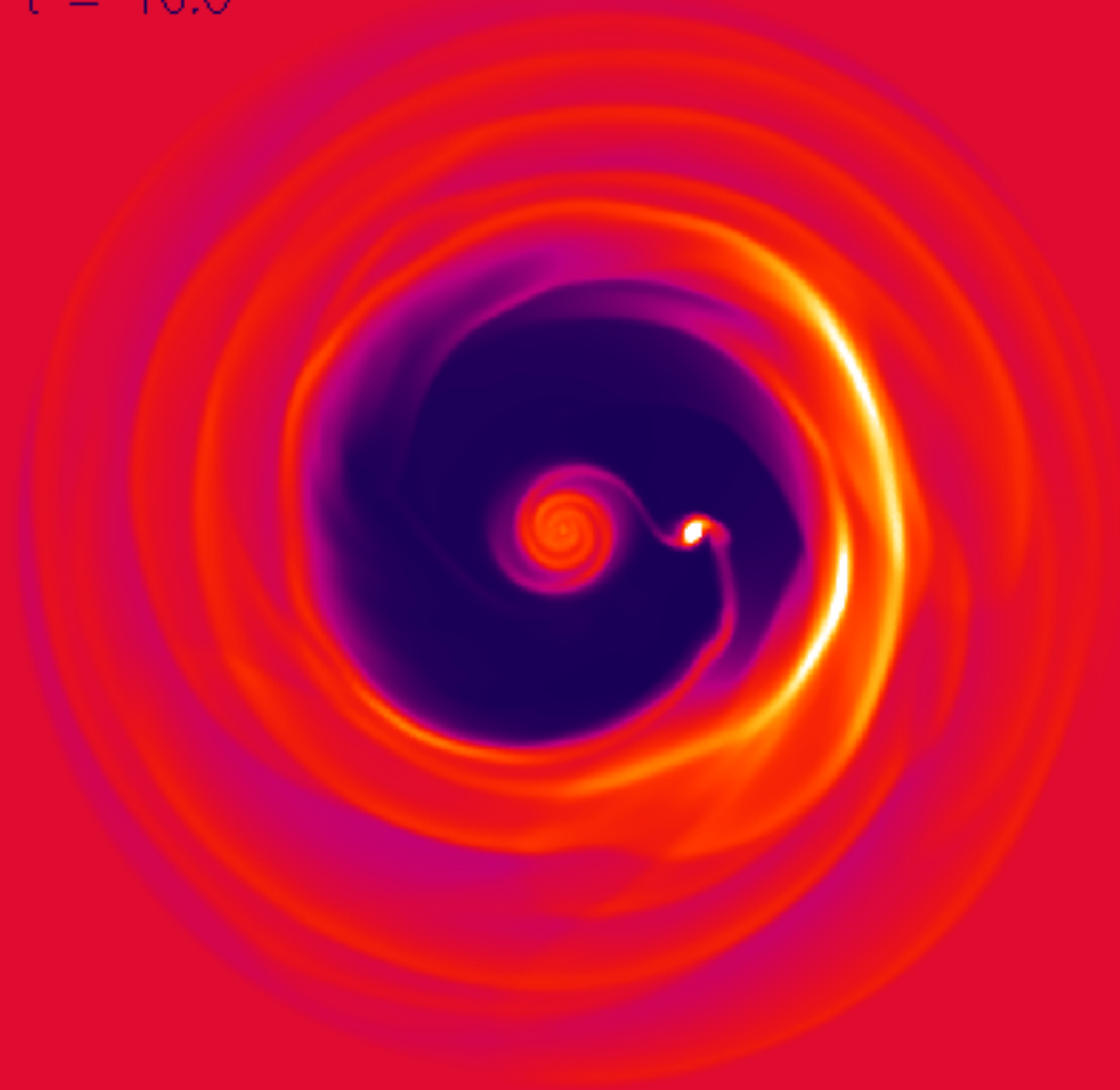
$t = 2.8$



$t = 8.6$

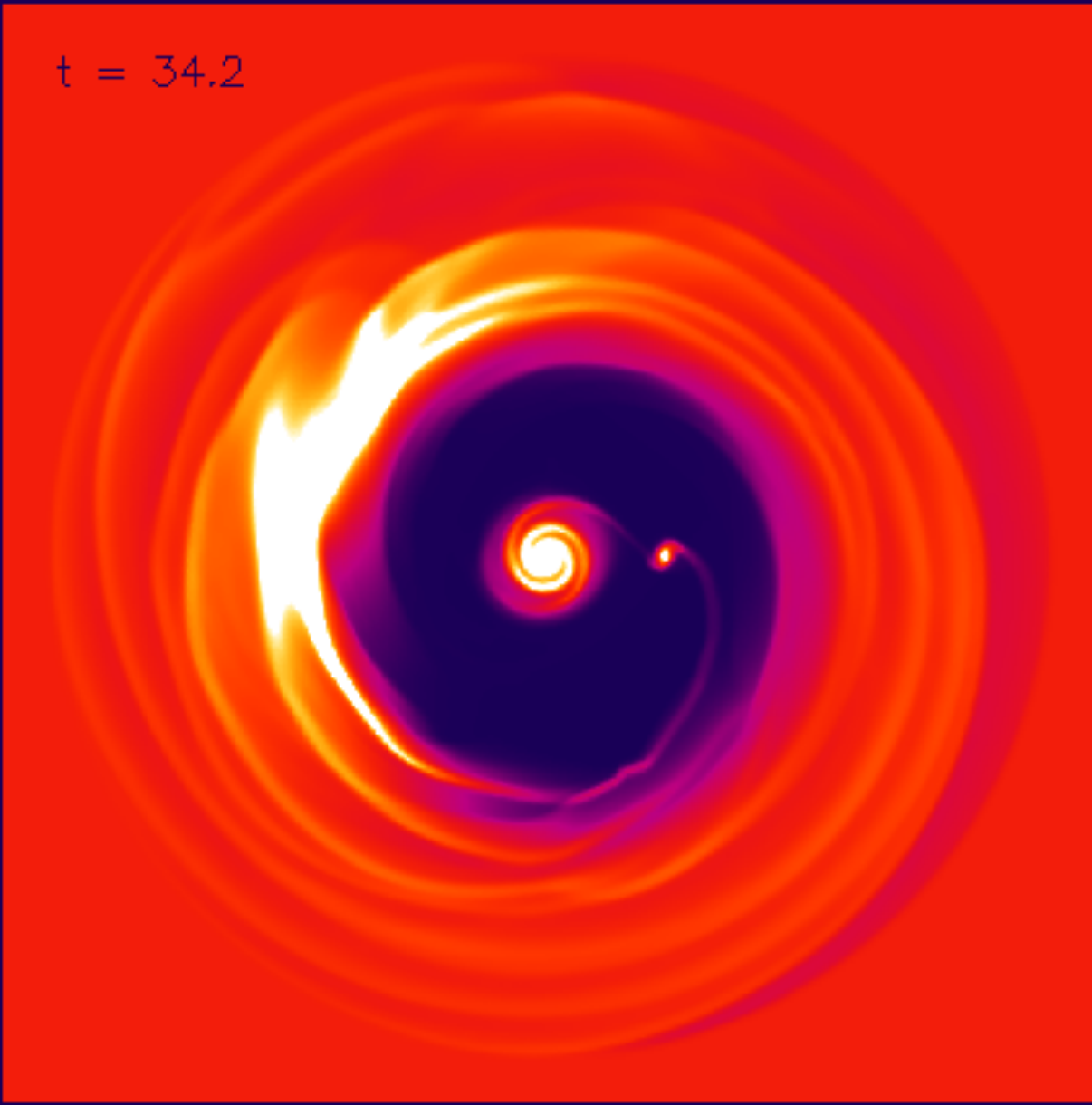


$t = 16.0$



An edge mode of spiral density waves appears, grows non-linear, and forms a vortex-like structure in disk. Density of gas is color-coded.

$t = 34.2$



**This computation used C
Fortran hydrocode algo
PPM (Piecewise Parabo**

GPUs



The 3-D flow around a small, embedded planet

J. Fung, P. Artymowicz and Y. Wu (ApJ, Nov 2015)

Code: PenGUIn.

CUDA C++. Processes up to ~20 Mcells/s (dp), ~40 Mcell/s (sp)

for comparison, Xeon Phi can run the same size problems at ~30 Mcell/s (sp)

and a modern 6-core CPU does ~28 Mcell/s.

These codes are bandwidth-bound. GPU > MIC ~ CPU

We have found big differences between 2-D and 3-D flow pattern of gas from a protoplanetary disk around a planet. These differences may influence the way planets form and migrate in disks.

2-D

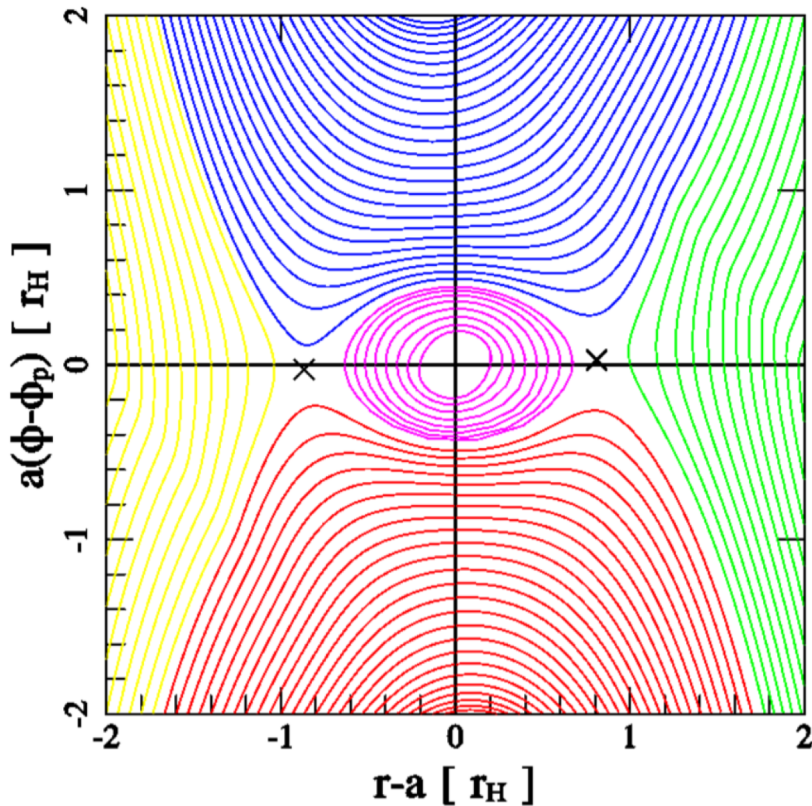


FIG. 1.— Streamlines around a planet in 2D, plotted in the corotating frame

3-D

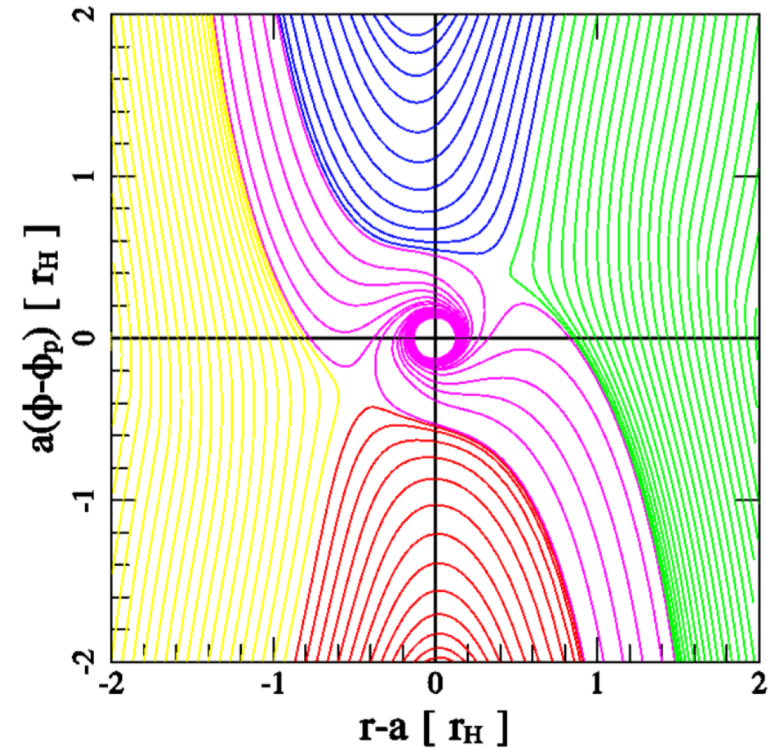
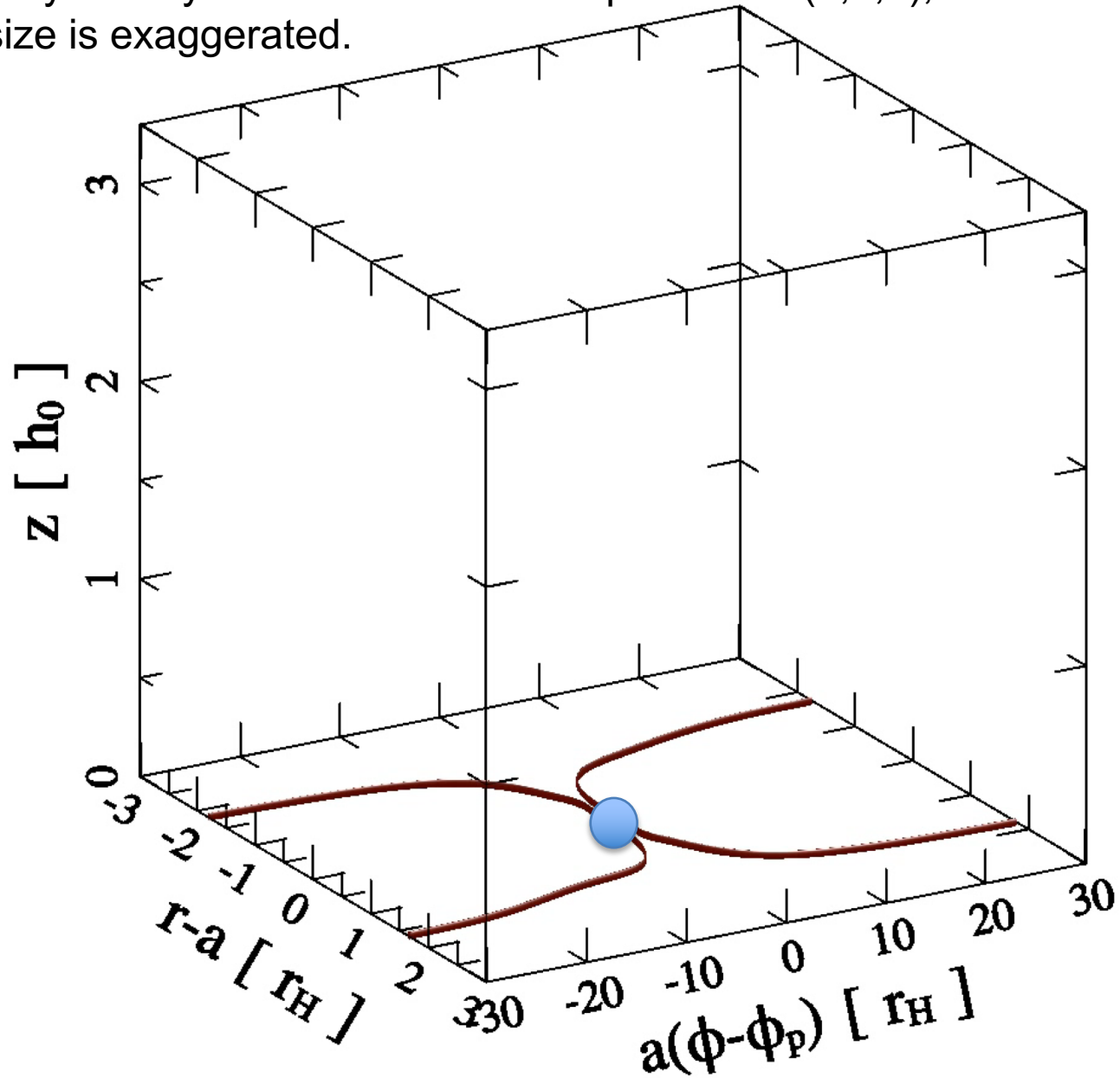
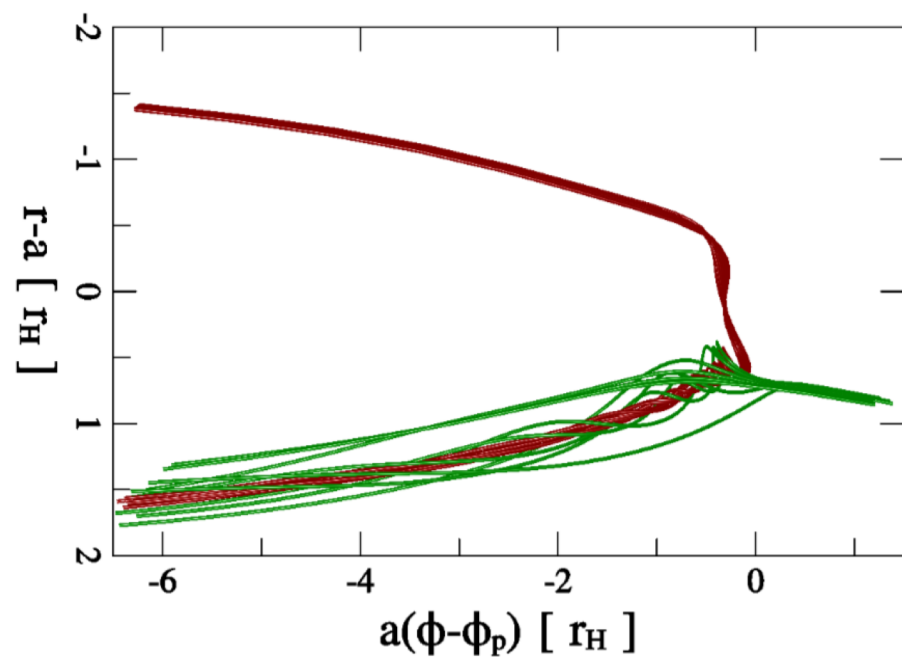
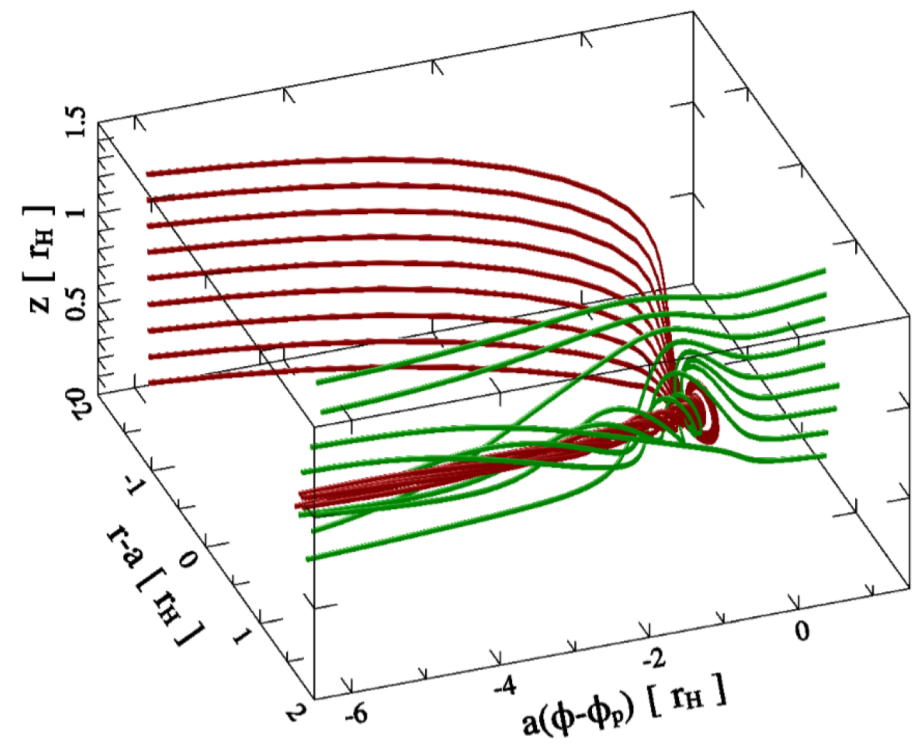


FIG. 8.— Streamlines in the disk midplane. Compare with Figure 1 for differences between 2D and 3D flow. Yellow, red, green, and blue streamlines are assigned in the same manner as Figure 1. Unlike Figure 1, magenta lines are outflows away from the planet, pulled down from initially higher altitudes. They reach as close as $1.5r_s$ from the planet and are unbound.

Computational box, the bottom part of which lies in the disk midplane. Up-down symmetry is assumed. Planet's position is $(0,0,0)$, indicated by a circle) but the size is exaggerated.

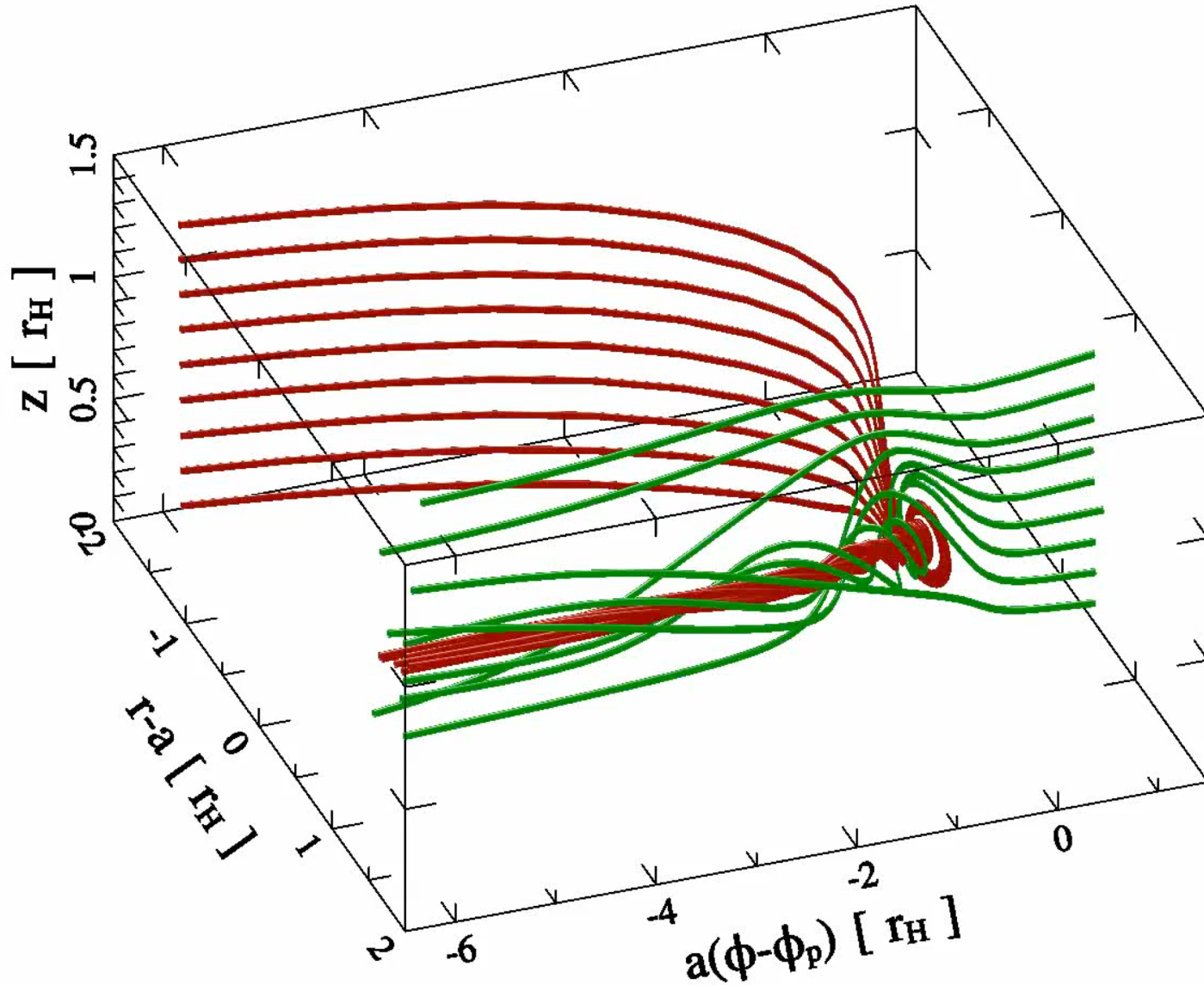


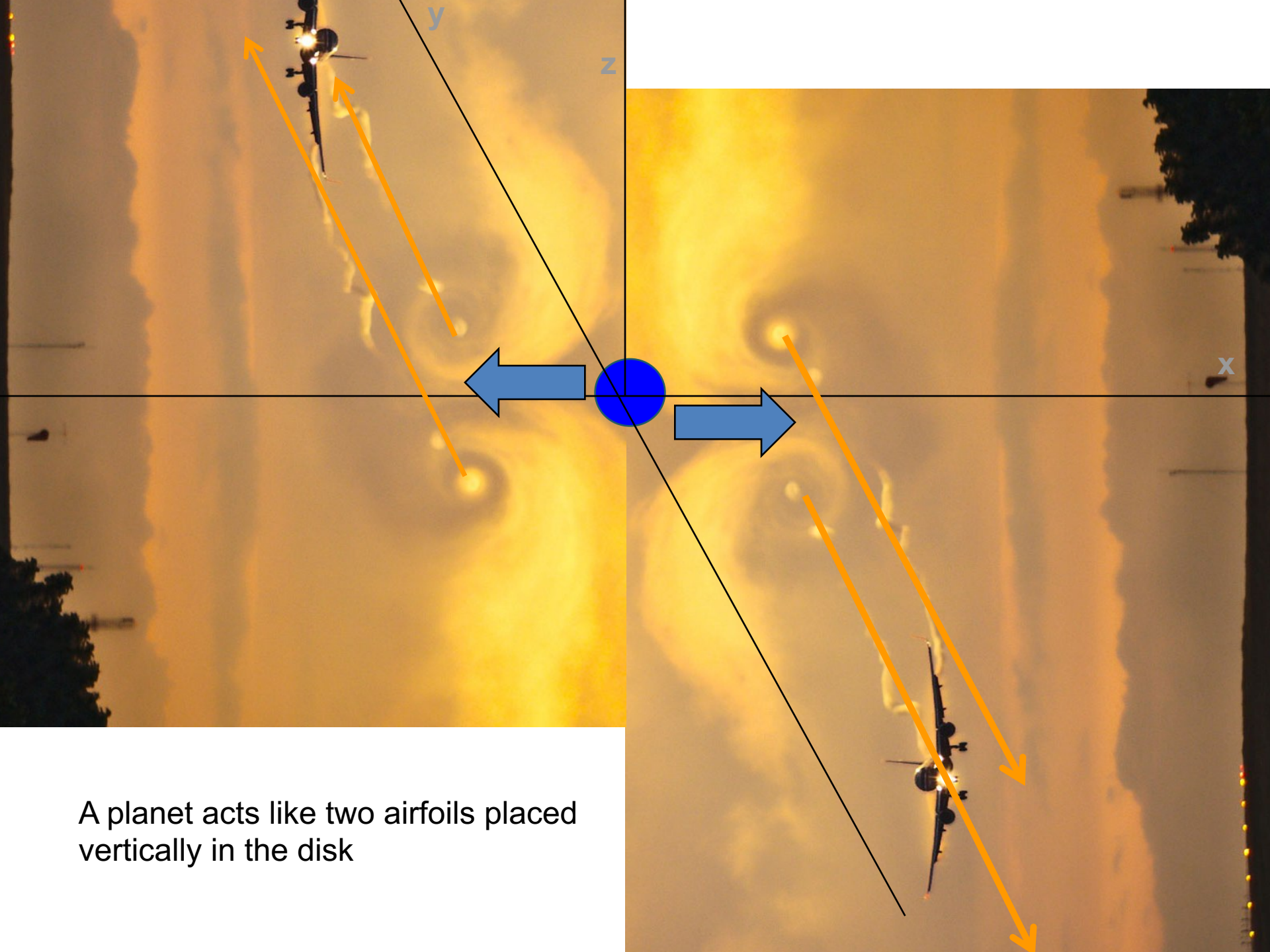
Gas flow approaches planet on one side of the disk (say, further from the star than the orbit of the planet) and after curious vertical compression (into a vortex) departs on the other side (closer to the star than planet).



top view

Such vortex, because of up-down and far-near symmetry is found near the protoplanet in 4 copies (2 counterrotating pairs). A planet sheds vortices familiar to flows patterns in aerodynamics, where there are called wingtip vortices.





A planet acts like two airfoils placed vertically in the disk

Novel results from 3D simulations

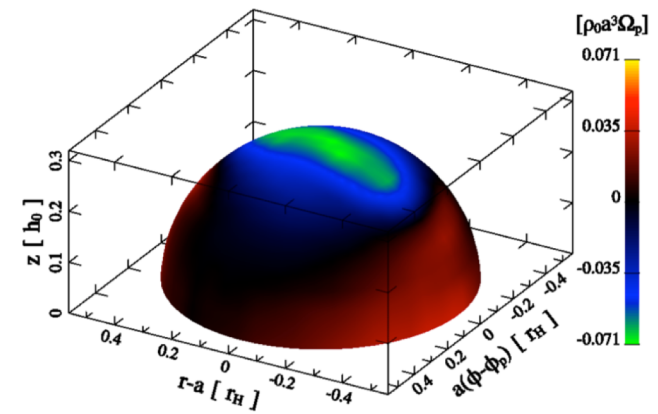


FIG. 9.— Mass flux across the surface of a sphere centered on the planet. The sphere has a radius of $0.5r_B$. Blue and green indicate influx; red and yellow are outflux. The speed of the downward flow is about $0.7c_s$ in this plot, while the two radial outward flows in the midplane (one not visible from this viewing angle) each has a speed of $\sim 0.2c_s$, as is explained in Appendix A. Match this figure with Figure 8 for a more complete view of the flow topology near the midplane.

New 3D phenomena, absent in 2D flows, including new columnar topology

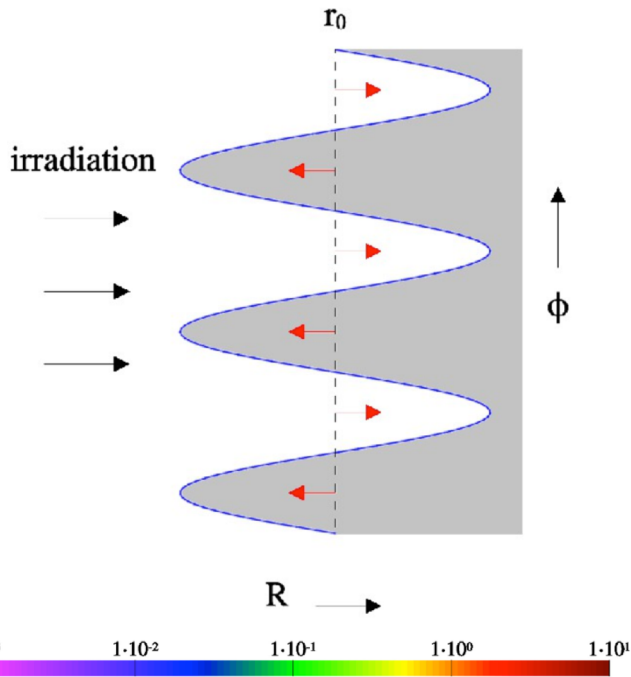
vorticity generation mechanism around a small planet, have a potential to resolve the long-standing problems in planet formation theory:

migration and cooling/contraction of the growing planet, occasional transmutation into a giant gaseous planet.

DUST/RADIATION PRESSURE-RELATED INSTABILITIES

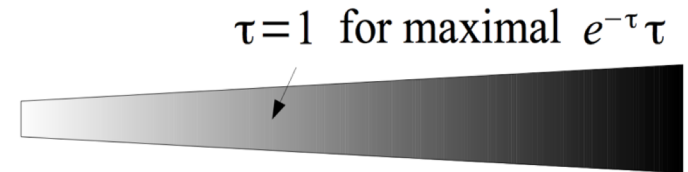
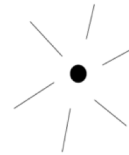
including the IRI = **I**r**R**adiation **I**nstability

IRI in 2D



Jeffrey Fung (UC Berkeley)
used workstations at UofT with 3 GPUs
for parallel computations

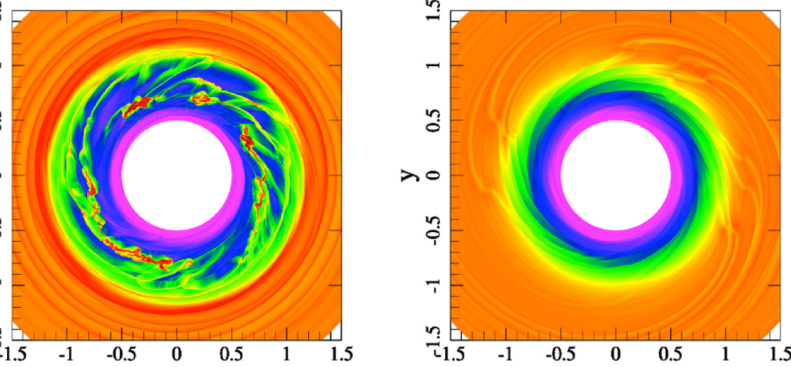
IRI in 2D



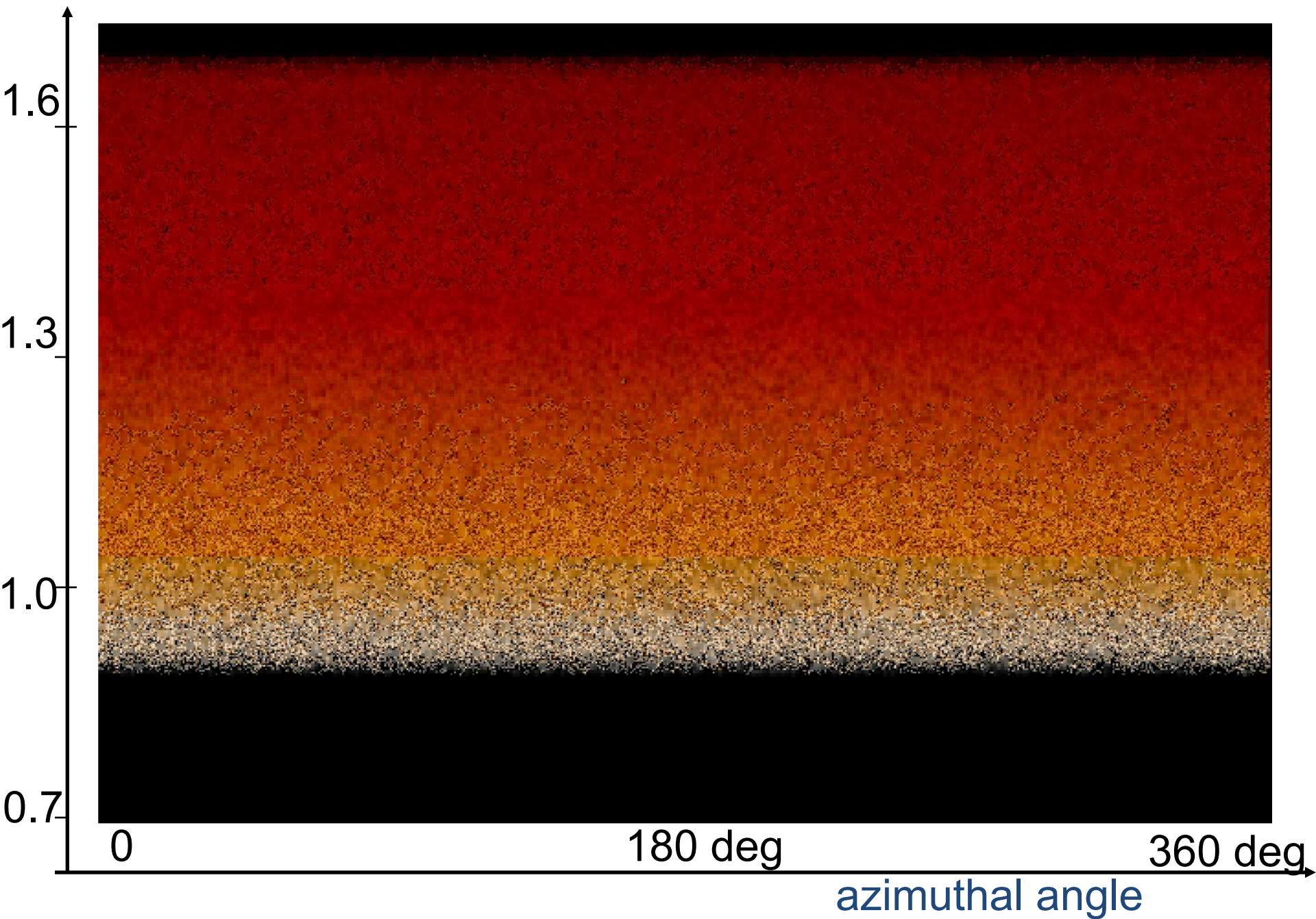
Criterion for instability: $\beta e^{-\tau} \tau \frac{d \ln[r \mathcal{R}]}{d \ln r} > 1$

$$\mathcal{R} \equiv \frac{\Sigma \Omega_k \beta e^{-\tau}}{\kappa^2}$$

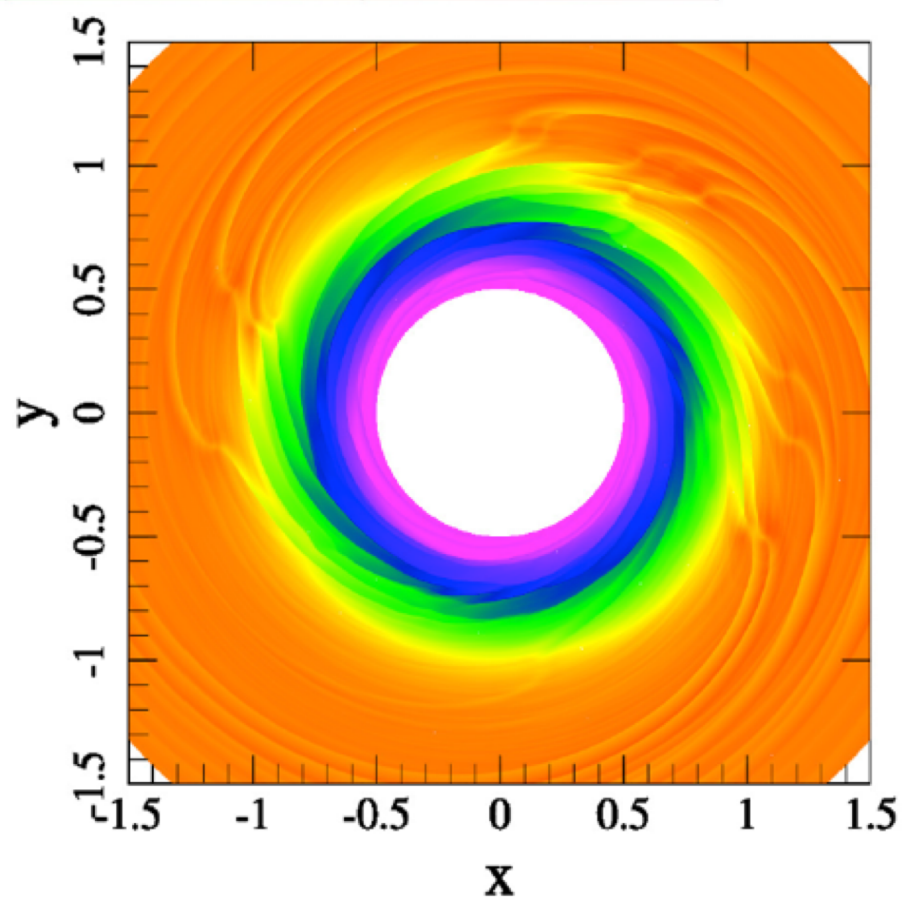
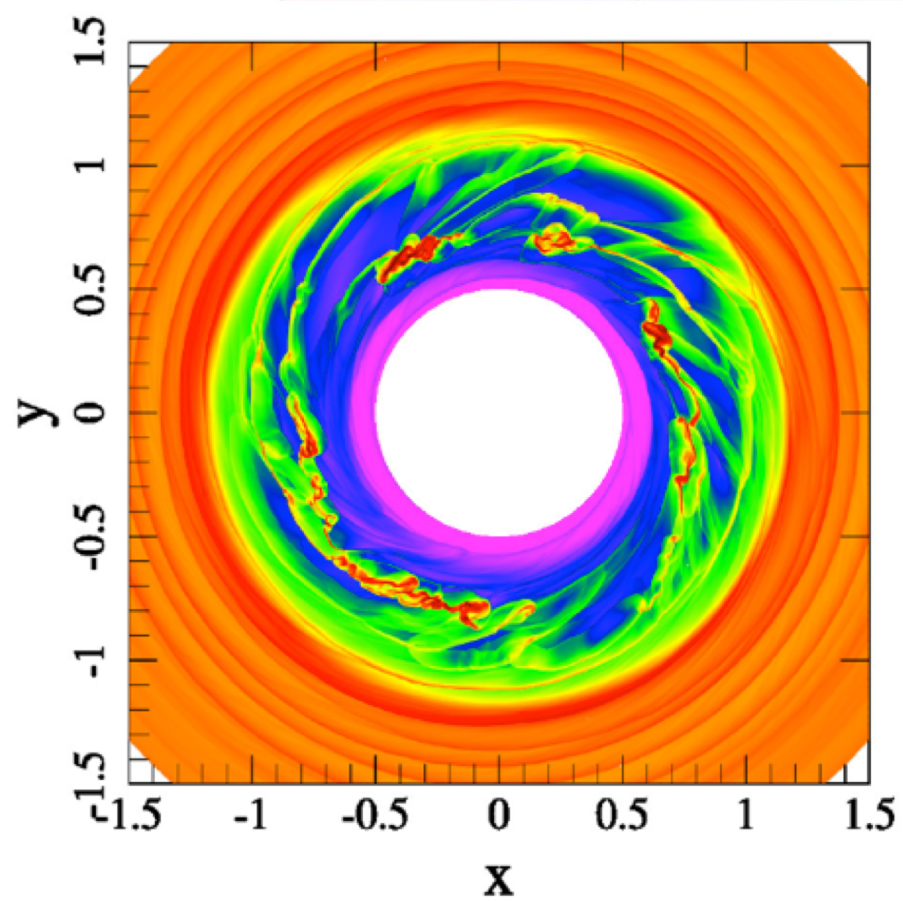
See Fung & Artymowicz (2014) for details



radius *Particle disks have IRI instab. too!* $\tau = 4$, $\beta = 0.2$



IRI



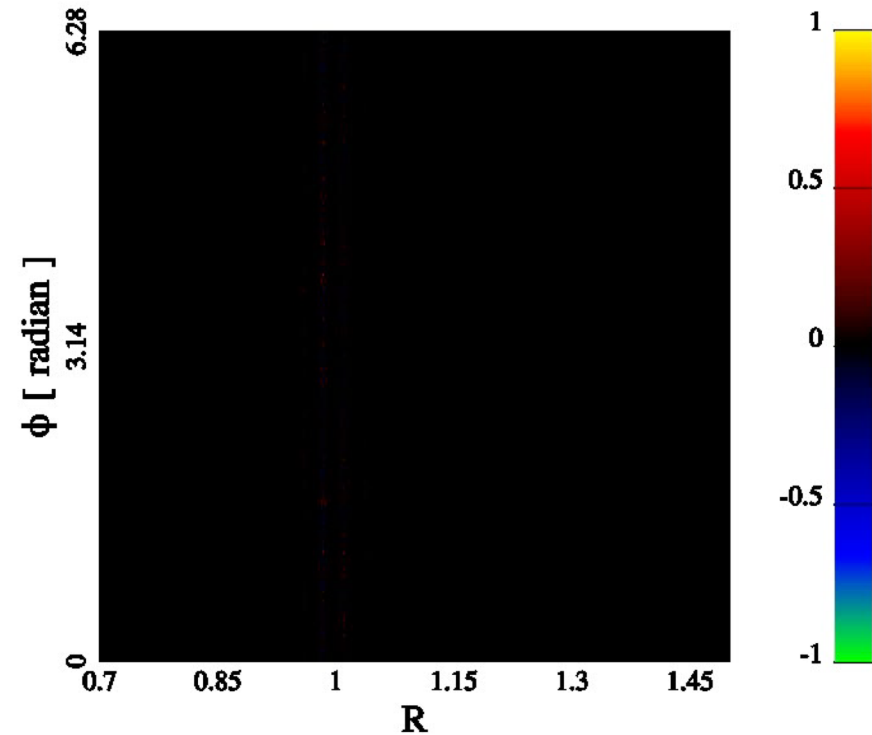
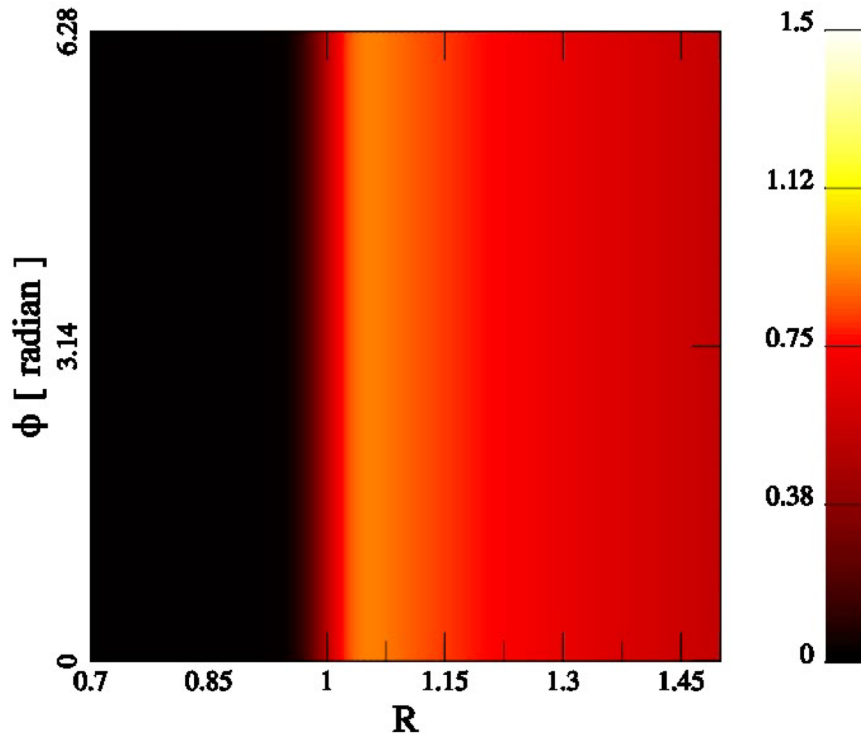
GAS DISK HYDRODYNAMICAL SIMULATION (PPM method, 2-D)

R.h.s. shows a background-removed picture of density of growing modes.

Analytical predictions are in agreement with calculations.

Models of disks were running faster on 3 GPUs than on UCB 128-cpu cluster.

$t = 00.09$ orbits



Opaque disks are unstable under illumination by the central object

Program tau-nopgpl.f that Jeffrey Fung used
to calculate IRI:

240 essential lines
+ 200 lines of plotting routines