# Particles from viscous hydrodynamics - with GPUs

**Denes Molnar, Purdue University**

**GPU Day 2017 - The Future of Many-Core Computing in Science**

**June 22-23, Wigner RCP, Budapest, Hungary**

**with Mridula Damodaran (Ph.D. student)**

# Outline

**I. Physics question**

**II. Math/computing problem - single-threaded solution**

**III. Getting to a multi-threaded solution**

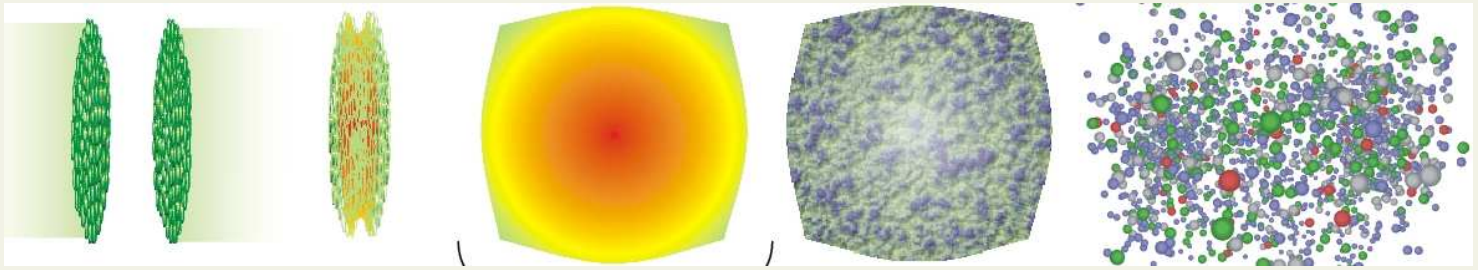
**IV. Some results, future steps**

# Heavy ion physics

**bang two heavy nuclei together to study the quark-gluon plasma**

**e.g., at Large Hadron Collider (LHC) or Relativistic Heavy Ion Collider (RHIC)**

**initial nuclei** **quark-gluon plasma** **hadron gas**



[image: S. Bass]

$\uparrow$

$\mathcal{O}(10\ \mathrm{fm})$

$\downarrow$

$<\!\!---\ \mathcal{O}(10\ \mathrm{fm}/c)\ ---\!\!>$

**Initconds**

**Hydrodynamics**

- **hydro fields, e.g,** $e(\vec{r}, t)$
- **equation of state,**
- **viscosities, relax. times**

**Kinetic theory / flight to detectors**

# The $\delta f$ problem (hydro $\rightarrow$ particles)

**hydro gives $N^\mu$ & $T^{\mu\nu}$, but experiments measure particles**

$$N^\mu(\vec{r},t) \equiv \sum_i \int \frac{d^3p}{E} p^\mu f_i(p,\vec{r},t)$$

$$T^{\mu\nu}(\vec{r},t) \equiv \sum_i \int \frac{d^3p}{E} p^\mu p^\nu f_i(\vec{p},\vec{r},t)$$

- **in local equilibrium (ideal hydro) - 1-to-1 map to thermal distributions**

$$T^{\mu\nu}_{LR}(x) = diag(e,p,p,p) \quad \Leftrightarrow \quad f_{eq,i}(x,p) = \frac{g_i}{(2\pi)^3} \frac{1}{e^{(p^\mu u_\mu - \mu_i)/T} + a}$$

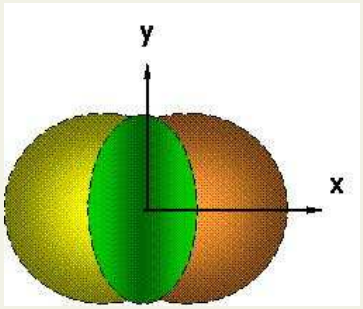
- **near local equilibrium (viscous hydro) - "few to many"**

$$\begin{aligned} T^{\mu\nu}(x) &= T^{\mu\nu}_{ideal}(x) + \delta T^{\mu\nu}(x) \\ N^\mu(x) &= N^\mu_{ideal(x)} + \delta N^\mu(x) \end{aligned} \quad \Leftarrow \quad f_i(x,p) = f_{eq,i}(x,p) + \delta f_i(x,p)$$

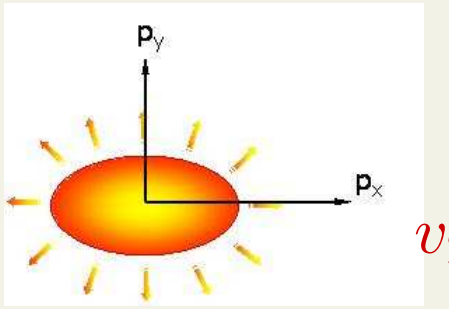
$\Rightarrow$ **question of $\delta f$ (even for single-species systems!)**

# Elliptic flow ($v_2$) and viscosity

**initial spatial anisotropy converts to final momentum space anisotropy**



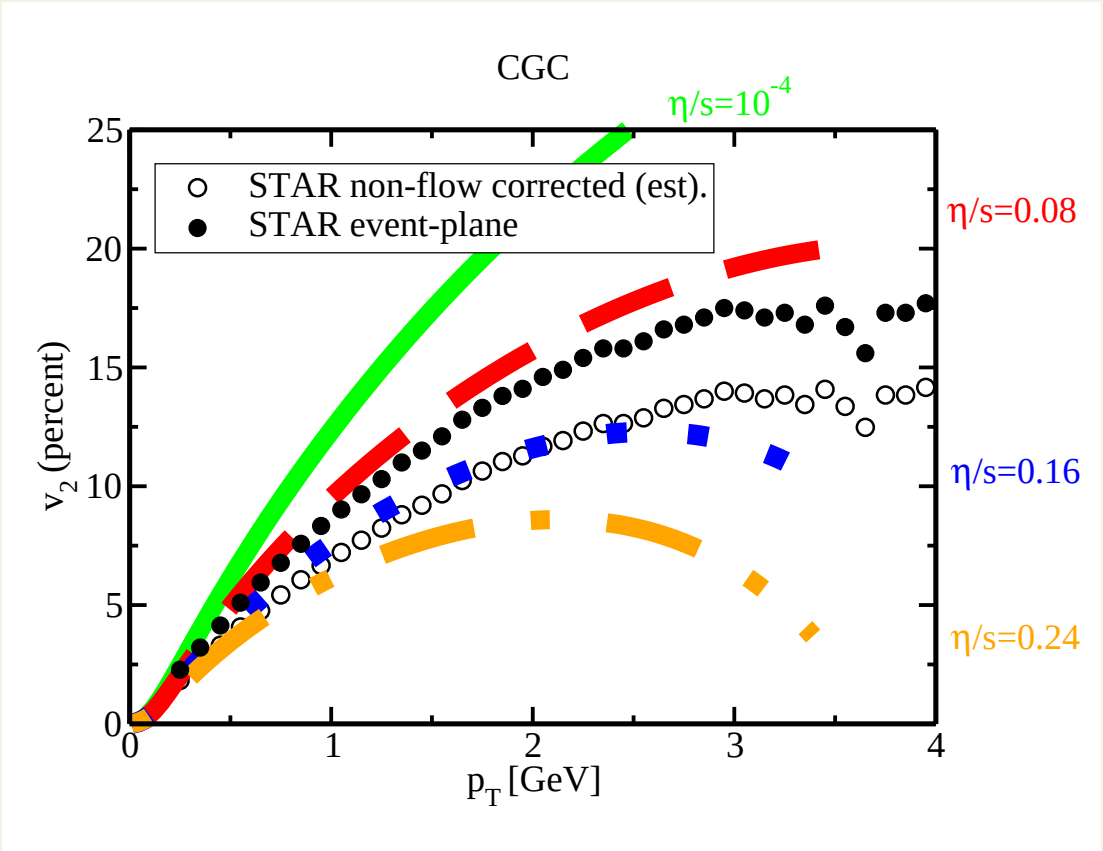

$$\varepsilon \equiv \frac{\langle x^2 - y^2 \rangle}{\langle x^2 + y^2 \rangle}$$




$$v_2 \equiv \frac{\langle p_x^2 - p_y^2 \rangle}{\langle p_x^2 + p_y^2 \rangle} \equiv \langle \cos 2\phi_p \rangle$$

**can be used to measure viscosity**

e.g., Romatschke & Luzum, PRC78 ('08):




**however, result sensitive to $\delta f$**

DM & Wolff, PRC95 ('17); arXiv:0611.09185

# What $\delta f$ to take?

**common: ad-hoc parametrizations - e.g., Grad's ansatz**

$$\delta f = const \times p_\mu p_\nu \delta T^{\mu\nu} f_{eq}$$

$\rightarrow$ **not based on dynamics**

**better: calculate from kinetic theory**

Dusling, Moore, Teaney, PRC81 ('09); DM, JPG38 ('12); DM & Wolff, PRC95 ('17)

$\rightarrow$ **leads to integral equations for $\chi(p) \propto \delta f / f_{eq}$ of the form**

$$S(\vec{p}_1) = \int d^3p_2\, d^3p_3\, d^3p_4\, \left[K(\vec{p}_1,\vec{p}_2,\vec{p}_3,\vec{p}_4)\chi(|\vec{p}_2|) + K'(\vec{p}_1,\vec{p}_2,\vec{p}_3,\vec{p}_4)\chi(|\vec{p}_1|)\right]$$

$\rightarrow$ **source $S$ contains gradients of local equilibrium distribution**

$\rightarrow$ **kernels depend on microscopic scattering rates**

# Variational solutions

**the integral eqns can be turned into a maximization problem for**

$$Q[\chi] = -\int d^3p_1\, S\, \chi(|\vec{p}_1|) + \frac{1}{2}\int d^3p_1 d^3p_2 d^3p_3 d^3p_4 \left[K\, \chi(\vec{p}_2)\chi(\vec{p}_1) + K'\, \chi^2(\vec{p}_1)\right]$$

**Approx solution using finite basis:** $\chi(p) \approx \sum_{k=0}^{n} c_k \phi_k(p)$

$$\Rightarrow \qquad Q \approx -\sum_k c_k S_k + \frac{1}{2}\sum_{kl} c_k A_{kl} c_l \quad \rightarrow \quad \text{maximal for} \quad c_k = \sum_l A^{-1}_{kl} S_l$$

**where**

$$S_k \equiv \int d^3p_1\, S\, \phi_k(p_1)\ , \quad A_{kl} \equiv \int \prod_{i=1}^{4} d^3p_i \left[K\phi_k(p_2)\phi_l(p_1) + K'\phi_k(p_1)\phi_l(p_1)\right]$$

**$\rightarrow$ in practice, 4 numerical integrals to do (for isotropic cross sections)**

# Single-threaded computation

**4 nested integrals, use adaptive 1D routines from GNU Scientific Lib (GSL)**

**61-point Gauss-Kronrod:**

$$\int_a^b dx\, f(x) \approx \sum_{n=0}^{60} w_i\, f(x_i)$$

**for error estimate take only half the points:** $\int_a^b dx\, f(x) \approx \sum_{n=even}^{60} w'_i\, f(x_i)$

**If error large, bisect** $[a, b]$ **and its bisections, until total error small enough**

**→ always bisect interval that has largest error next**

$$A_{kl} = \int_0^\infty dx \int_0^\infty dy \int_{-1}^1 dt \int_{-1}^1 dz\, (...)$$

$$\downarrow$$

$$f_x(x_i) \hookrightarrow f_y(y_j) \hookrightarrow f_t(t_k) \hookrightarrow f_z(z_l)$$

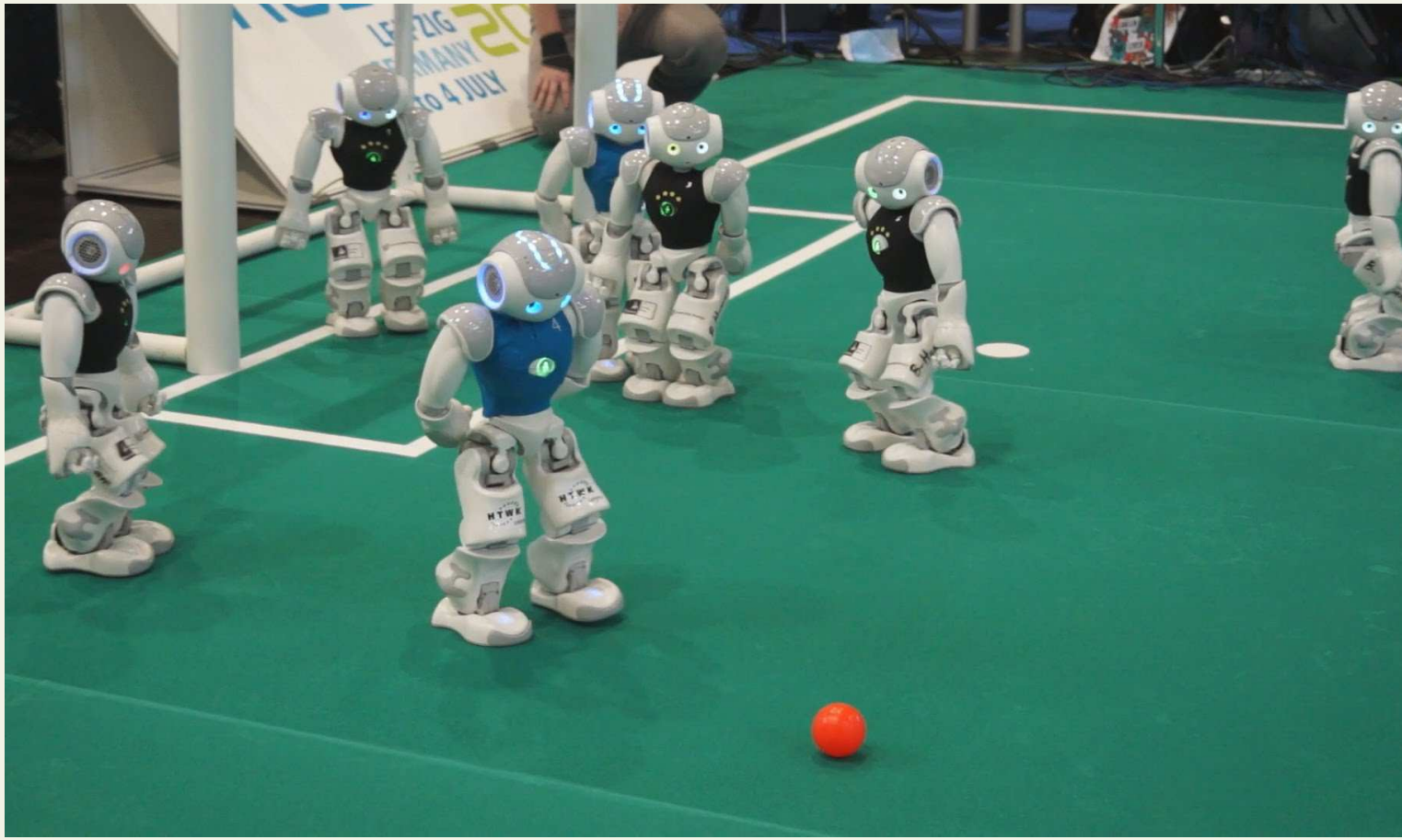
**→ dependency tree**

# Moving to GPUs... sorta like

[Amazon.com]

**can we win with team of so-so players?**
**(plus other constraints)**

**perhaps, if we have many so-so players...**



# Progression of ideas

**Idea 0:** **61 parallel evaluations in G-K $\rightarrow$ doable, but not very parallel**

**Idea 1:** **do innermost two integrals in parallel via 2D Simpson**

$$\int_{-1}^{1} dt \int_{-1}^{1} dz\, F(t,z) \approx \sum_{k,l=0}^{N} w_{k,l}\, F(t_k, z_l)$$

**weight pattern:**

$$\begin{array}{ccccc} 1 & - & 4 & - & 1 \\ | & & | & & | \\ 4 & - & 16 & - & 4 \\ | & & | & & | \\ 1 & - & 4 & - & 1 \end{array} \quad \rightarrow \quad \begin{array}{ccccccccc} 1 & - & 4 & - & 2 & - & 4 & - & 2 \cdots \\ | & & | & & | & & | & & | \\ 4 & - & 16 & - & 8 & - & 16 & - & 8 - \\ | & & | & & | & & | & & | \\ 2 & - & 8 & - & 4 & - & 8 & - & 4 \cdots \\ \vdots & & | & & \vdots & & | & & \vdots \end{array}$$

$\rightarrow$ **for error estimate: use only even-even sites**

$\rightarrow$ **completely parallel - no dependencies, no conditionals**

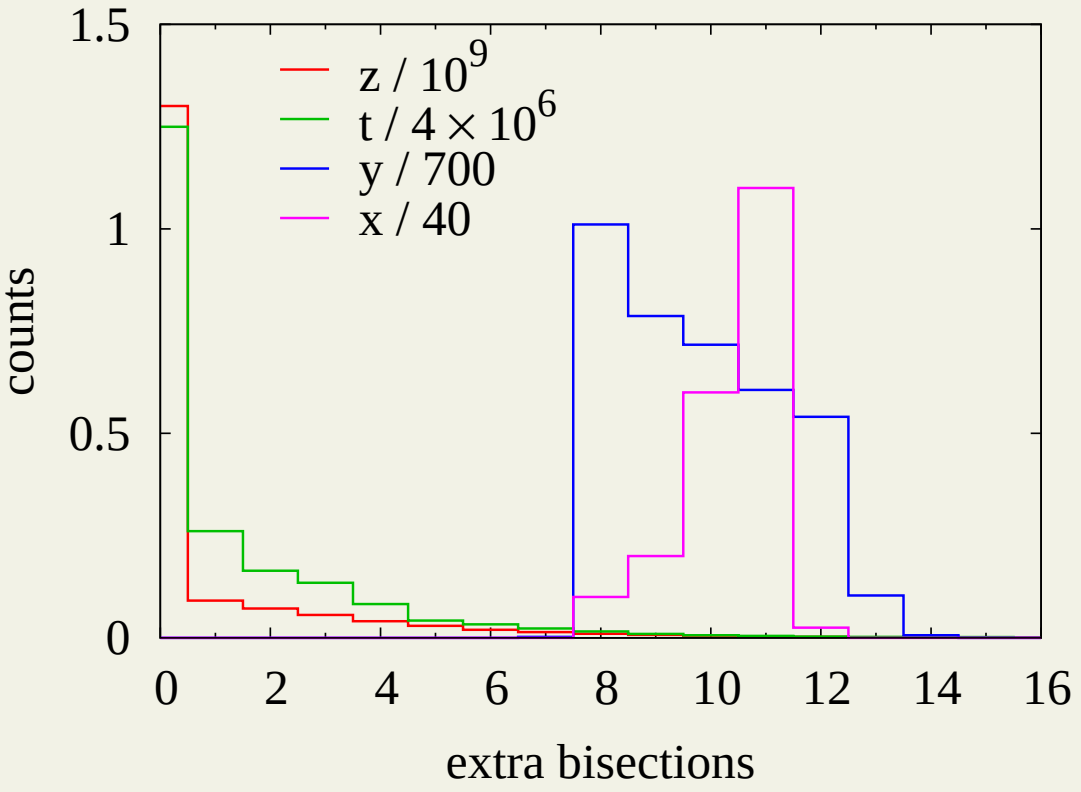
**works, but slow convergence with $N$ to desired $\mathcal{O}(10^{-11})$ relative accuracy**

**Idea 2: bite bullet, put nested adaptive G-K for innermost 2 integrals**

- **each thread computes full $\int dz\, dt$ , for given (different) $x$, $y$**

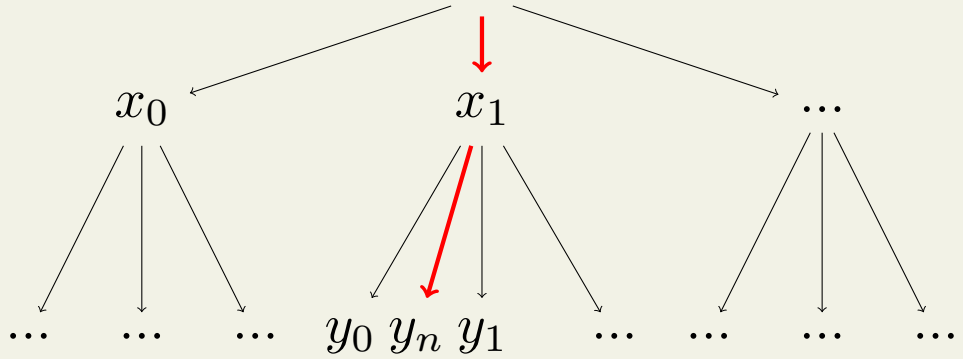
  **good: minimal $\mathcal{O}(10)$ storage needed, also not that many conditionals**

  **bad: duplicate G-K codes for $\int dz$ and $\int dt$ (can't pass ptr to function)**




```
integrate([-1,1]);
N = 0;
iv1 = [-1,1];
while (error_big && N < SPACE) {
  sections[N] = iv1;
  i = worst_section(sections);
  iv1 = left_half(sections[i]);
  iv2 = right_half(sections[i]);
  integrate(iv1);
  integrate(iv2);
  update_sum_and_error();
  N ++;
  sections[i] = iv2;
}
```

**outer two integrals:** initial 61$\times$ 61 evals can be done in parallel but dependency problem afterwards

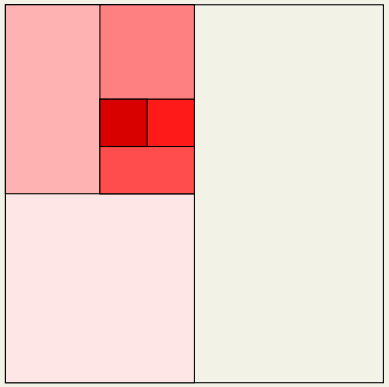



$\rightarrow$ any bisection at bottom level ($y$) holds back decision at top level ($x$)

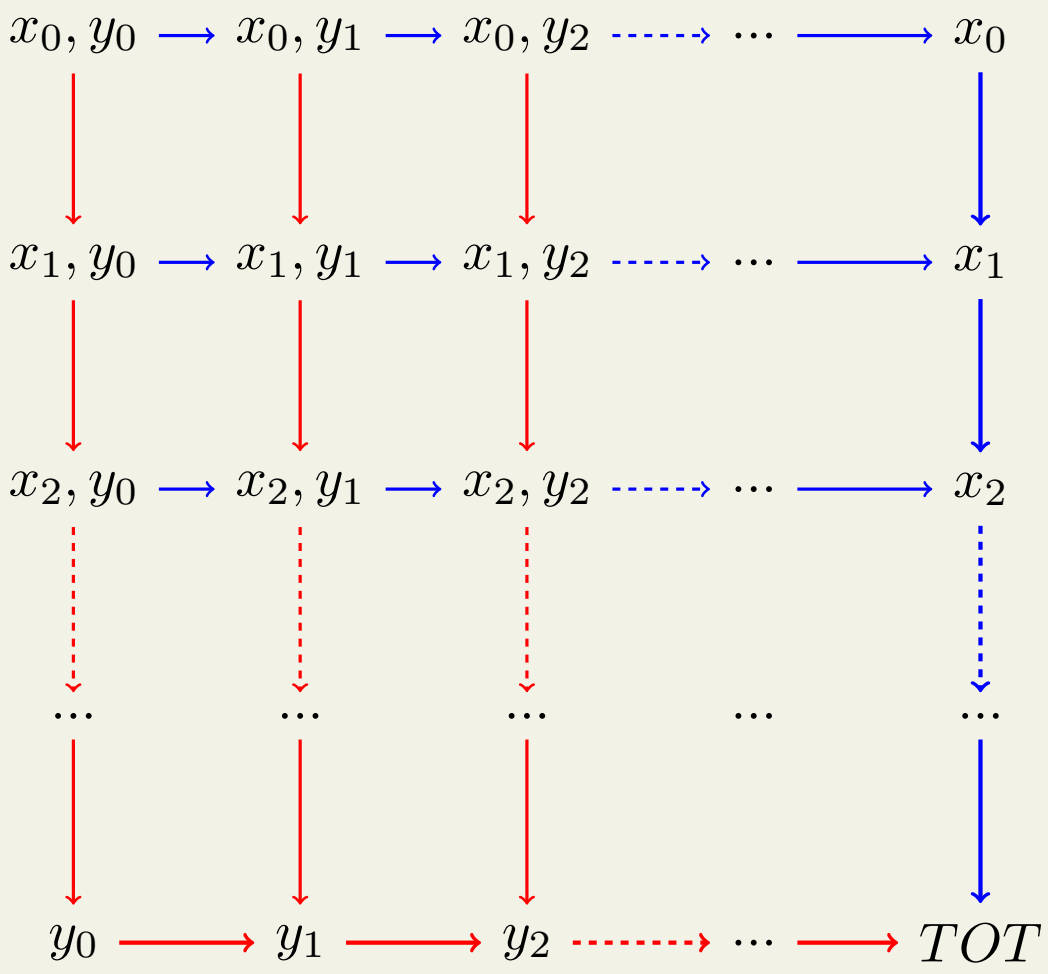
Idea 3: guess ahead where to bisect on top level $\rightarrow$ kinda hard...

Idea 4: "bisections" in 2D

- parallel G-K evals over each 2D "section"
- always halve 2D region with largest error



## parallel 2D Gauss-Kronrod




- **do all $f(x_i, y_j)$ evals (1 eval/thread)**

- **G-K sum along $y$ first, then $x$ $\rightarrow$ error is $\epsilon_x$**

- **G-K sum along $x$ first, then $y$ $\rightarrow$ error is $\epsilon_y$**

**est. error over 2D region:**
$\epsilon_{2D} = \max(\epsilon_x, \epsilon_y)$

**when halving ($\epsilon_{2D}$ too large):** **if $\epsilon_x > \epsilon_y$, cut horizontally (along $x$ axis)**
**if $\epsilon_y > \epsilon_x$, cut vertically (along $y$ axis)**

# Some results

**bulk viscous $\delta f$, basis: $\phi_n(x) = \sqrt{x^n}$, $n = 0 \ldots 16$** **($17^2 = 289$ elements)**

- **inner two integrals in OpenCL, outer two single-threaded in C++**
- **Opteron 6376 @ Wigner GPU Lab (32 cores)** **$\sim 30$ minutes**

$$A^{(31)}_{kl} = \begin{pmatrix}
0.49 & 0.844 & 1.57 & 3.1 & 6.5 & 14.4 & 33.3 & 80.7 & 203 \\
0.844 & 1.51 & 2.9 & 5.92 & 12.8 & 29.1 & 69.4 & 173 & 446 \\
1.57 & 2.9 & 5.74 & 12.1 & 27 & 63.3 & 155 & 395 & 1.05e3 \\
3.1 & 5.92 & 12.1 & 26.4 & 60.5 & 146 & 366 & 958 & 2.6e3 \\
6.5 & 12.8 & 27 & 60.5 & 143 & 353 & 911 & 2.44e3 & 6.79e3 \\
14.4 & 29.1 & 63.3 & 146 & 353 & 896 & 2.37e3 & 6.52e3 & 1.86e4 \\
33.3 & 69.4 & 155 & 366 & 911 & 2.37e3 & 6.43e3 & 1.81e4 & 5.27e4 \\
80.7 & 173 & 395 & 958 & 2.44e3 & 6.52e3 & 1.81e4 & 5.21e4 & 1.55e5 \\
203 & 446 & 1.05e3 & 2.6e3 & 6.79e3 & 1.86e4 & 5.27e4 & 1.55e5 & 4.72e5 \\
532 & 1.19e3 & 2.87e3 & 7.3e3 & 1.95e4 & 5.45e4 & 1.58e5 & 4.76e5 & 1.48e6 \\
1.44e3 & 3.31e3 & 8.13e3 & 2.12e4 & 5.79e4 & 1.65e5 & 4.9e5 & 1.5e6 & 4.76e6 \\
4.03e3 & 9.46e3 & 2.38e4 & 6.32e4 & 1.77e5 & 5.15e5 & 1.56e6 & 4.88e6 & 1.58e7 \\
1.16e4 & 2.79e4 & 7.14e4 & 1.94e5 & 5.54e5 & 1.65e6 & 5.09e6 & 1.62e7 & 5.34e7 \\
3.46e4 & 8.44e4 & 2.21e5 & 6.12e5 & 1.78e6 & 5.41e6 & 1.7e7 & 5.53e7 & 1.85e8 \\
1.05e5 & 2.62e5 & 7e5 & 1.98e6 & 5.87e6 & 1.82e7 & 5.82e7 & 1.93e8 & 6.57e8 \\
3.3e5 & 8.37e5 & 2.28e6 & 6.56e6 & 1.98e7 & 6.24e7 & 2.04e8 & 6.86e8 & 2.38e9 \\
1.06e6 & 2.74e6 & 7.58e6 & 2.22e7 & 6.84e7 & 2.19e8 & 7.28e8 & 2.49e9 & 8.78e9 & 3.
\end{pmatrix}$$

**compare to Intel Core i7 @ Purdue** $\sim 10$ **hours**

- **single-threaded solution in C++ (4 nested adaptive G-K routines)**

- **relative difference in results** $\times 10^{-13}$ $\rightarrow$ **better agreement than** $1.8 \times 10^{-11}$

$$\delta A_{kl}^{(31)} = \begin{pmatrix}
-180 & -140 & -72 & -60 & -51 & -49 & -38 & -61 & -38 & 5.6 & 1.8 \\
-147 & -110 & -55 & -55 & -48 & -50 & -37 & -62 & -39 & 4.2 & 23 \\
-120 & -64 & -44 & -51 & -46 & -49 & -38 & -63 & -38 & 29 & 20 \\
-100 & -16 & -36 & -47 & -43 & -47 & -36 & -60 & -5.4 & 26 & 16 \\
-74 & -61 & -32 & -44 & -40 & -45 & -34 & -0.26 & -4 & 22 & 13 \\
-70 & -69 & -25 & -40 & -37 & -41 & 2.1 & -20 & -2.9 & 18 & 9.8 \\
-64 & -50 & 27 & -36 & -33 & -4 & 2.1 & -17 & -1.9 & 14 & 7.5 \\
-58 & 4.1 & -18 & 2.1 & 2.5 & -3.4 & 1.9 & -14 & -1.2 & 10 & 5.7 \\
-22 & -33 & 15 & 1.9 & 2.2 & -2.8 & 1.7 & -11 & -0.69 & 7.7 & 4.3 \\
-44 & 77 & 21 & 1.8 & 1.9 & -2.3 & 1.4 & -8 & -0.36 & 5.5 & 3.5 \\
-14 & 96 & 20 & 1.3 & 1.5 & -1.8 & 1.2 & -5.8 & -0.14 & 3.8 & 2.9 \\
-10 & 110 & 19 & 0.85 & 1.2 & -1.3 & 0.93 & -4.2 & -0.037 & 2.5 & 2.5 \\
-7.2 & 120 & 18 & 0.48 & 0.89 & -0.99 & 0.75 & -2.9 & 0.016 & 1.7 & 2.3 \\
-5 & 130 & 16 & 0.15 & 0.65 & -0.7 & 0.59 & -1.9 & 0.027 & 1 & 2.1 \\
-4.7 & 140 & 15 & -0.11 & 0.46 & -0.47 & 0.45 & -1.3 & 0.018 & 0.63 & 0.46 \\
-3.4 & 140 & 13 & -0.31 & 0.32 & -0.3 & 0.35 & -0.81 & 0.056 & 0.5 & 0.37 \\
-2.4 & 140 & 11 & -0.46 & 0.21 & -0.18 & 0.27 & -0.47 & 0.029 & 0.33 & 1.9
\end{pmatrix}$$

# Next steps / Issues

- **real GPU?**

  - **so far OpenCL compilation fails ...** CL_BUILD_PROGRAM_FAILURE

- **understand occasional NaNs...**

  - **even initial G-K evals fail sometimes $\rightarrow$ number representation?**

- **parallel computation for shear viscous $\delta f$**

  $\rightarrow$ **involves more complicated kernels**

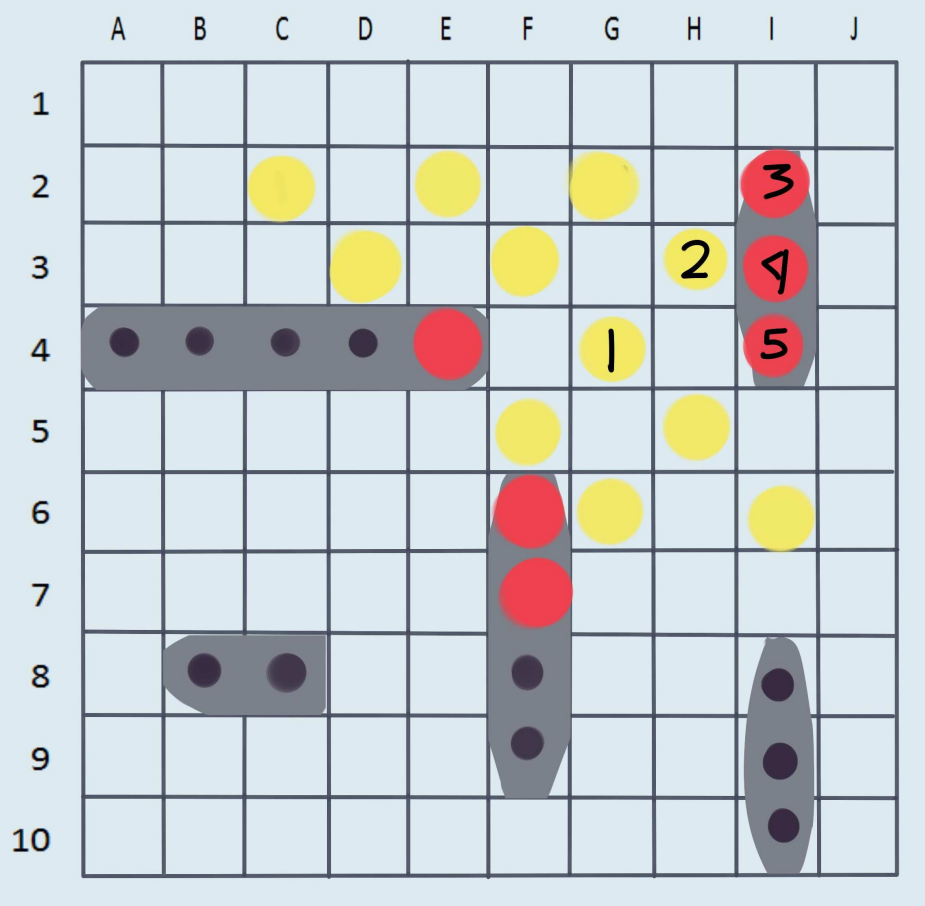
- **test/benchmark new 2D bisection algorithm in single-threaded mode**

  $\rightarrow$ **typically only $\mathcal{O}(10)$ tries needed**

- **scaling of algorithm: $\rightarrow$ degree of parallelizaton limited to $61^2 \sim 3700$**

**other potential strategies:** - **bisect multiple intervals per iteration**

- **cut one interval into several pieces per step**




**finding ship:** $\mathcal{O}(N^2)$ **task, sinking it:** $\mathcal{O}(1)$

**can we sink substantially faster if we can fire many times per round?**

# Summary

Comparision of hydrodynamics to heavy-ion data requires a model to convert the fluid fields to particles. Instead of $ad\ hoc$ parameterizations, one should use self-consistent viscous phase space corrections ($\delta f$) obtained from kinetic theory. This requires the numerical evaluation of certain 4D integrals.

For single-threaded computation, a straightforward "fire-and-forget" method is adaptive Gauss-Kronrod integration (4 nested 1D integrals).

We developed a multi-threaded algorithm that nests adaptive Gauss-Kronrod in OpenCL for the innermost 2 integrals (one 2D integral for each thread). The outermost 2 integrals are done in C++ via adaptive 2D generalization of the Gauss-Kronrod method (bisects 2D regions in each iteration).

Initial tests performed at the Wigner GPU Laboratory look promising. The parallel routine performs with excellent accuracy. The primary challenge right now is getting the OpenCL code to compile on real GPUs.

—

Many thanks to Máté Nagy-Egri and Dániel Berényi for help and advice, and to Gergely Barnaföldi and the Wigner RCP for supporting this work.

# Integrals over $[0, \infty)$

**As in GSL, map to $(0, 1]$ via switching to variables $x \to (1 - t)/t$**

$$\int_0^\infty dx\, f(x) = \int_0^1 \frac{dt}{t^2}\, f\left(\frac{1-t}{t}\right)$$

**One cannot evaluate Jacobian at $t = 0$, so in practice we do G-K with**

$$\int_\epsilon^1 \frac{dt}{t^2}\,(...)$$

**In our case, integrand cuts off exponentially,**

$$f(x) \propto e^{-\sqrt{x^2 + a^2}}$$

**so $\epsilon = 10^{-5} - 10^{-3}$ is sufficiently accurate.**