Calculation of dissipative phase space corrections

Denes Molnar, Purdue University

- GPU Day 2018 -

The Future of Computing, Graphics and Data Analysis

June 21-22, Wigner RCP, Budapest, Hungary

" $\sim 70\%$ of problems: Linear Algebra

$\sim 30\%$ of problems: Monte Carlo "

remaining $\mathcal{O}(1\%)$: something else

Outline

I. Physics question

II. Math/computing problem - single-threaded solution

III. Creating a multi-threaded solution for GPUs

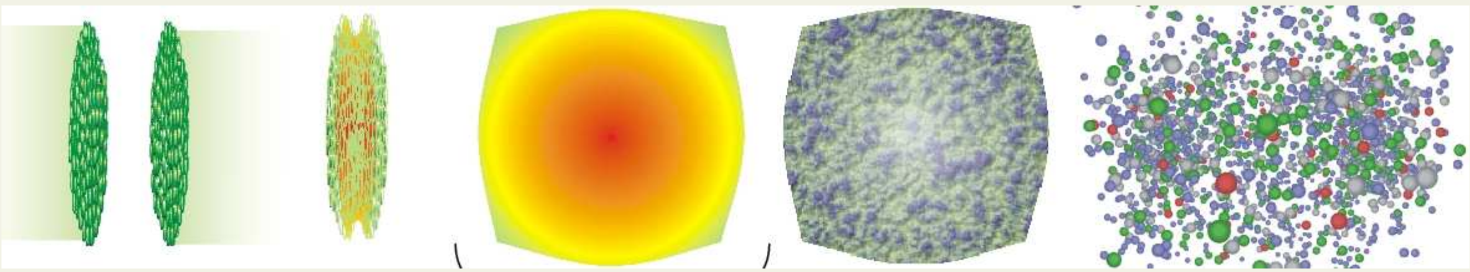
IV. Results, future steps

I. 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 δf problem (hydro $\longrightarrow$ particles)

hydro gives N^μ & $T^{\mu\nu}$, 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**

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

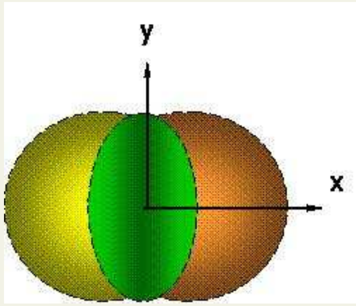
- **near local equilibrium (viscous hydro) - "few to many"**

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

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

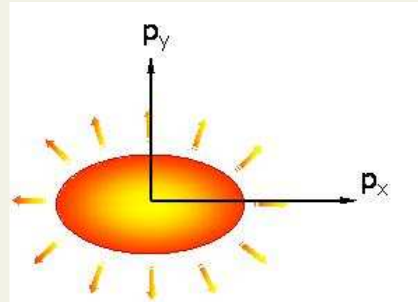
Elliptic flow (v_2) and viscosity

initial spatial anisotropy converts to final momentum space anisotropy



$$\longrightarrow$$

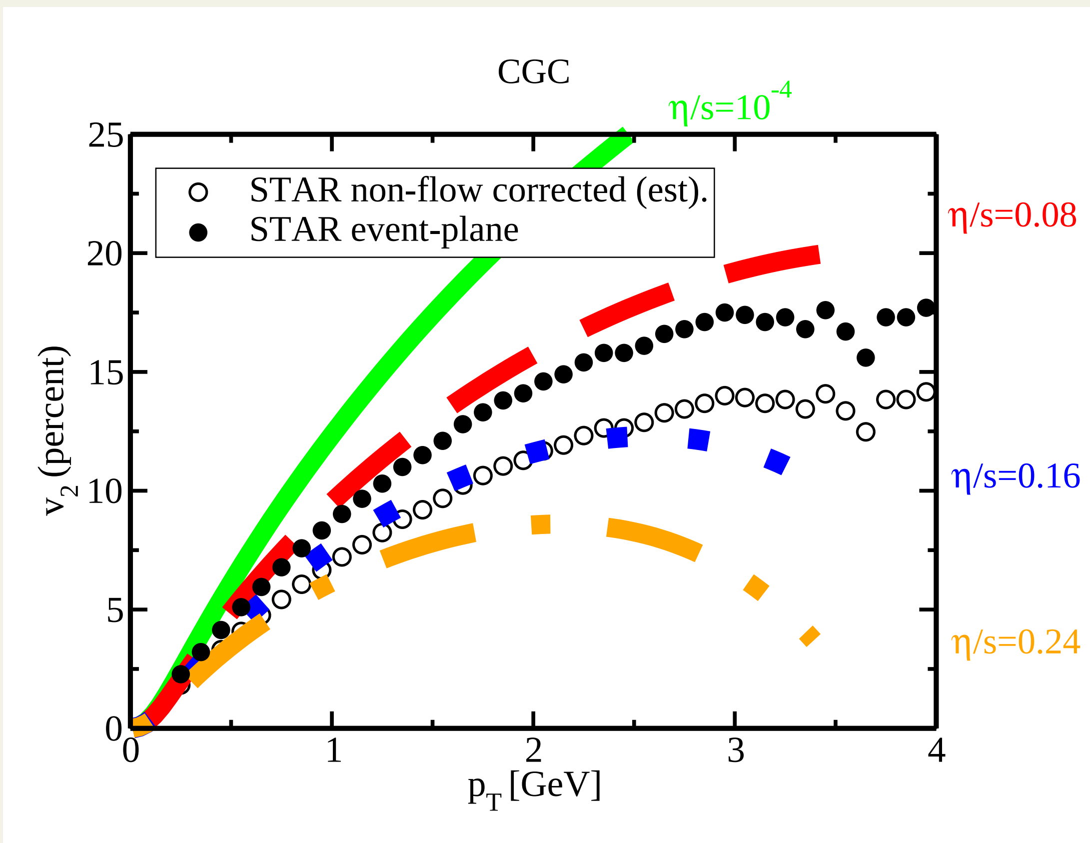
$$\varepsilon \equiv \frac{\langle x^2 - y^2 \rangle}{\langle x^2 + y^2 \rangle}$$



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

can be used to measure viscosity

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



however, result sensitive to δf

DM & Wolff, PRC95 ('17);
Wolff & DM, PRC96 ('17)
Dusling et al, PRC81 ('09)
...

II. What δf to take?

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

$$\delta f = (A(x) + B_\mu(x)p^\mu + C_{\mu\nu}(x)p^\mu p^\nu) f_{eq}$$

$\rightarrow$ **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**

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

$\rightarrow$ **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

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

$\rightarrow$ **in practice, 4D numerical integration for A_{kl} (for isotropic cross sections)**

Single-threaded computation

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

61-point Gauss-Kronrod:

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

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

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

$\rightarrow$ **always bisect interval that has <u>largest error</u> next**

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

$$\downarrow$$

$f_x(x_i)$

$\quad\hookrightarrow f_y(y_j)$

$\qquad\hookrightarrow f_t(t_k)$

$\qquad\quad\hookrightarrow f_z(z_l)$

$\rightarrow$ **dependency tree**

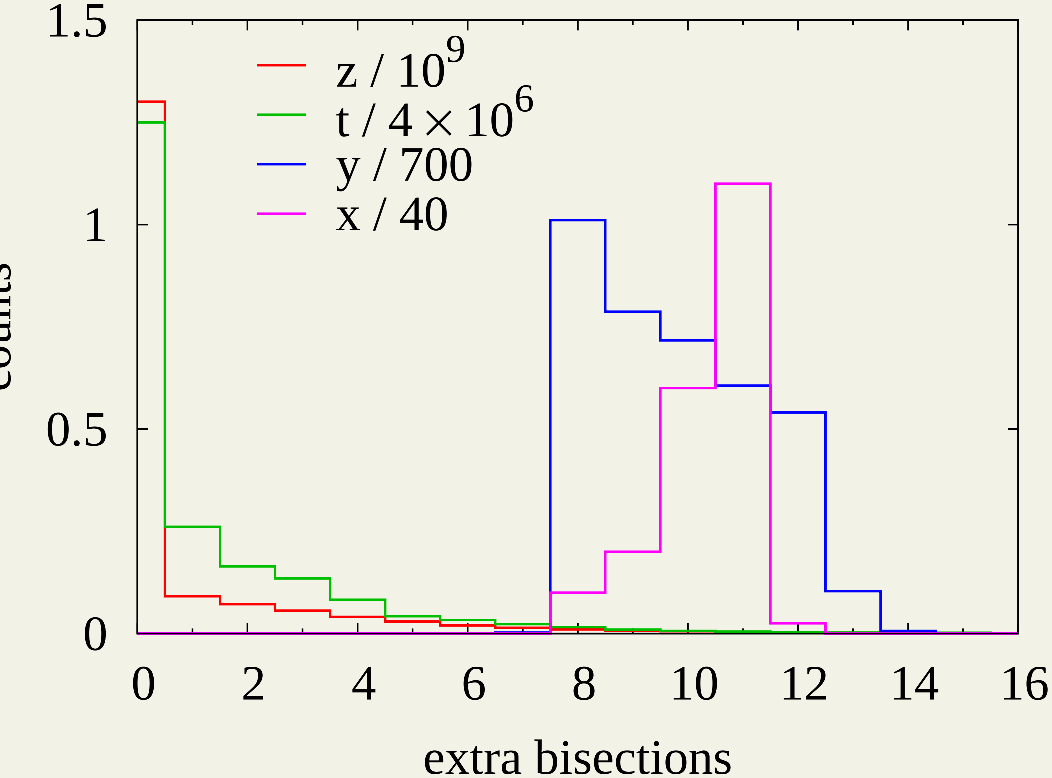
III. Let's do it on GPU...

Progression of ideas

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

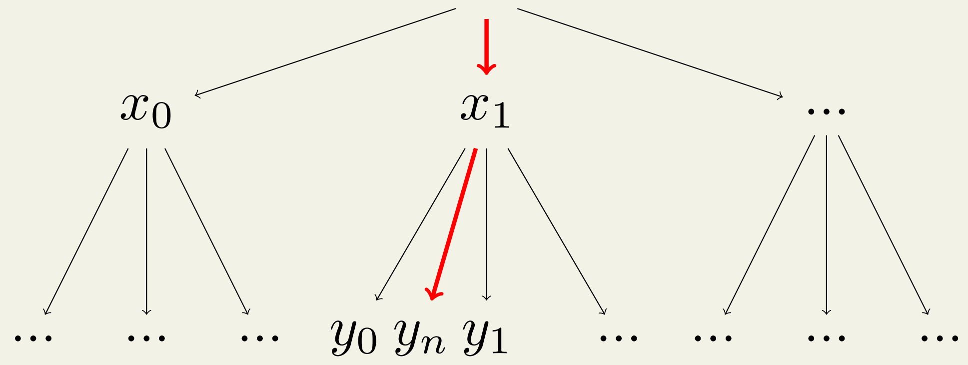
Idea 1: nested, adaptive G-K for innermost 2 integrals on GPU

- **each thread computes full $\int dz\, dt$, for given (different) x, y**
- **minimal $\mathcal{O}(10)$ storage needed, also not that many conditionals**



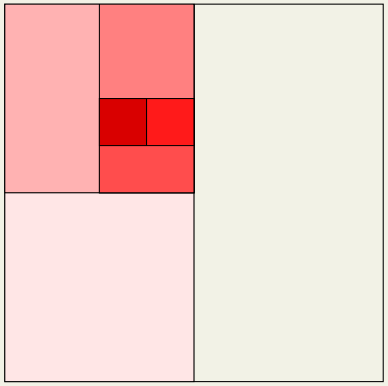

```
 iv1 = [-1,1];
integrate(iv1);
N = 0;
while (error_big && N < SPACE) {
  sections[N] = iv1;
  i = worst_section(sections);
  iv1 = left_half(sections[i]);
  iv2 = right_half(sections[i]);
  integrate(iv1);
  integrate(iv2);
  update_sum_and_error(iv1, iv2);
  N ++;
  sections[i] = iv2;
}
```

outer two integrals: **initial 61× 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)**

Key new idea (Idea 2): **"bisections" in 2D** (x, y)



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

parallel 2D Gauss-Kronrod $\approx$ two successive reductions ($\times 2$)

$$
\begin{tikzcd}
x_0, y_0 \arrow[r] \arrow[d] & x_0, y_1 \arrow[r] \arrow[d] & x_0, y_2 \arrow[r, dashed] \arrow[d] & \cdots \arrow[r] & x_0 \arrow[d] \\
x_1, y_0 \arrow[r] \arrow[d] & x_1, y_1 \arrow[r] \arrow[d] & x_1, y_2 \arrow[r, dashed] \arrow[d] & \cdots \arrow[r] & x_1 \arrow[d] \\
x_2, y_0 \arrow[r] \arrow[d, dashed] & x_2, y_1 \arrow[r] \arrow[d, dashed] & x_2, y_2 \arrow[r, dashed] \arrow[d, dashed] & \cdots \arrow[r] & x_2 \arrow[d, dashed] \\
\cdots \arrow[d] & \cdots \arrow[d] & \cdots \arrow[d] & \cdots & \cdots \arrow[d] \\
y_0 \arrow[r] & y_1 \arrow[r] & y_2 \arrow[r, dashed] & \cdots \arrow[r] & TOT
\end{tikzcd}
$$

- do all $f(x_i, y_j)$ evals (1 eval/thread)
- G-K sum along y first, then x $\rightarrow$ error is ϵ_x
- G-K sum along x first, then y $\rightarrow$ error is ϵ_y

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

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

Performance of Idea 2: bulk viscous δf, with basis: $\phi_n(x) = x^{n/2}$

$$n = 0 \ldots 16 \quad \Rightarrow \quad 17^2 = 289 \;\; A_{kl} \text{ elements}$$

OpenCL: ~ 10 **minutes** (Opteron 6376 @ Wigner GPU Lab, 32 cores)

single-threaded: ~ 2 **hours** (Intel Core i7 @ Purdue)

...but register analysis predicted bottleneck on GPUs (kernel too complex)

Idea 3a: nest two adaptive 2D G-K integrators $\int dxdy \int dtdz(...)$

- put only inner $f(t,z)$ evals on GPU $\to 61 \times 61 \times 2$ evals, 1/thread
- keep all other logic single-threaded on CPU

Idea 3b: also do first reduction in 2D G-K on GPU (weighted sum of 61 #s)

- cuts GPU$\to$CPU data transfer at end by factor 61
- second (final) reduction for $\int dtdz$ still done on CPU

Final idea (Idea 4): pool GPU part of all A_{kl} calculations $(N \times N)$

- **CPU$\to$ GPU:** x, y, $t-z$ **ranges, ...** ~ 10 **doubles** $\sim$ **100 bytes** $\times N^2$
- **GPU:** $\sim$ **7000 integrand evals** $\times N^2$, $\sim 61^2 \times 2$ **doubles** $\sim$ **0.6 MB** $\times N^2$

  $\sim$ **120 reductions** $\times N^2$
- **GPU$\to$ CPU:** $\sim 61 \times 2 \times 2$ **doubles** $\sim$ **2 kB** $\times N^2$

instead of:

```
loop over k,l {
    evaluate_A(k,l); // <-- uses GPU inside
}
```

reorganize:

```
while (! Akl_evaluators.converged()) {
    tz_region_list = empty;
    for (eval in Akl_evaluators) {
        tz_regions = eval->next_tz_regions();
        tz_region_list.append(tz_regions);
    }
    results = evaluate_f_on_GPU(tz_region_list);
    Akl_evaluators.update(results);
}
```

Benchmarks

wall clock time in <u>minutes</u> for calculation of full $N \times N$ matrix $(m/T = 3)$

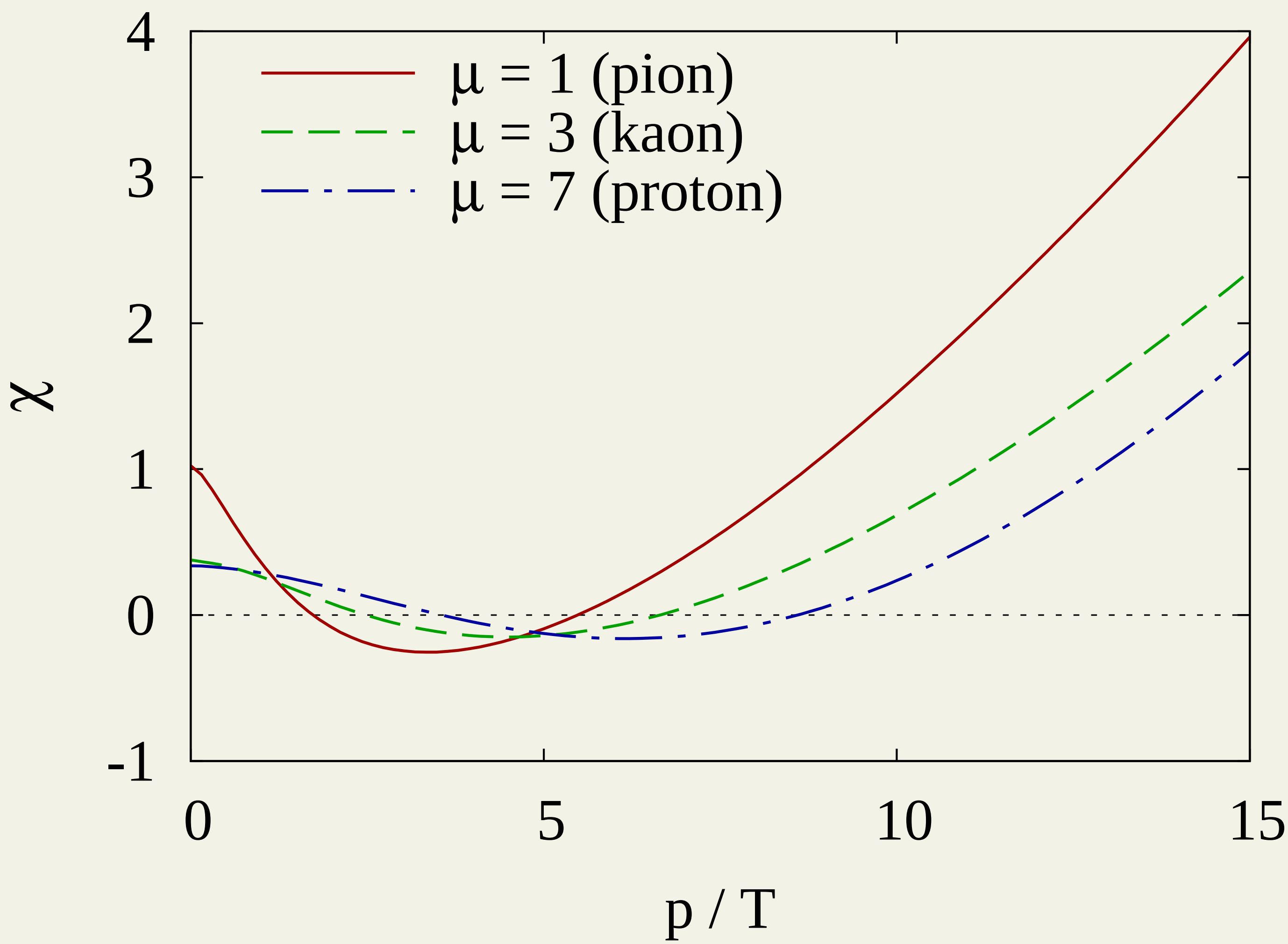
Problem Size	single-threaded (Core i7)	Idea 2 (Opteron)	Idea 3 (GTX 1080)	Idea 4 (GTX 1080)
6×6	20		17	1.6
11×11	67		42	3.0
16×16	134	12	80	5.2
21×21			131	7.8
26×26				11

$\Rightarrow$ all parallel versions run faster than single-threaded

10-25x speedup using nested 2D G-K with pooled evals on GPU

IV. Some results

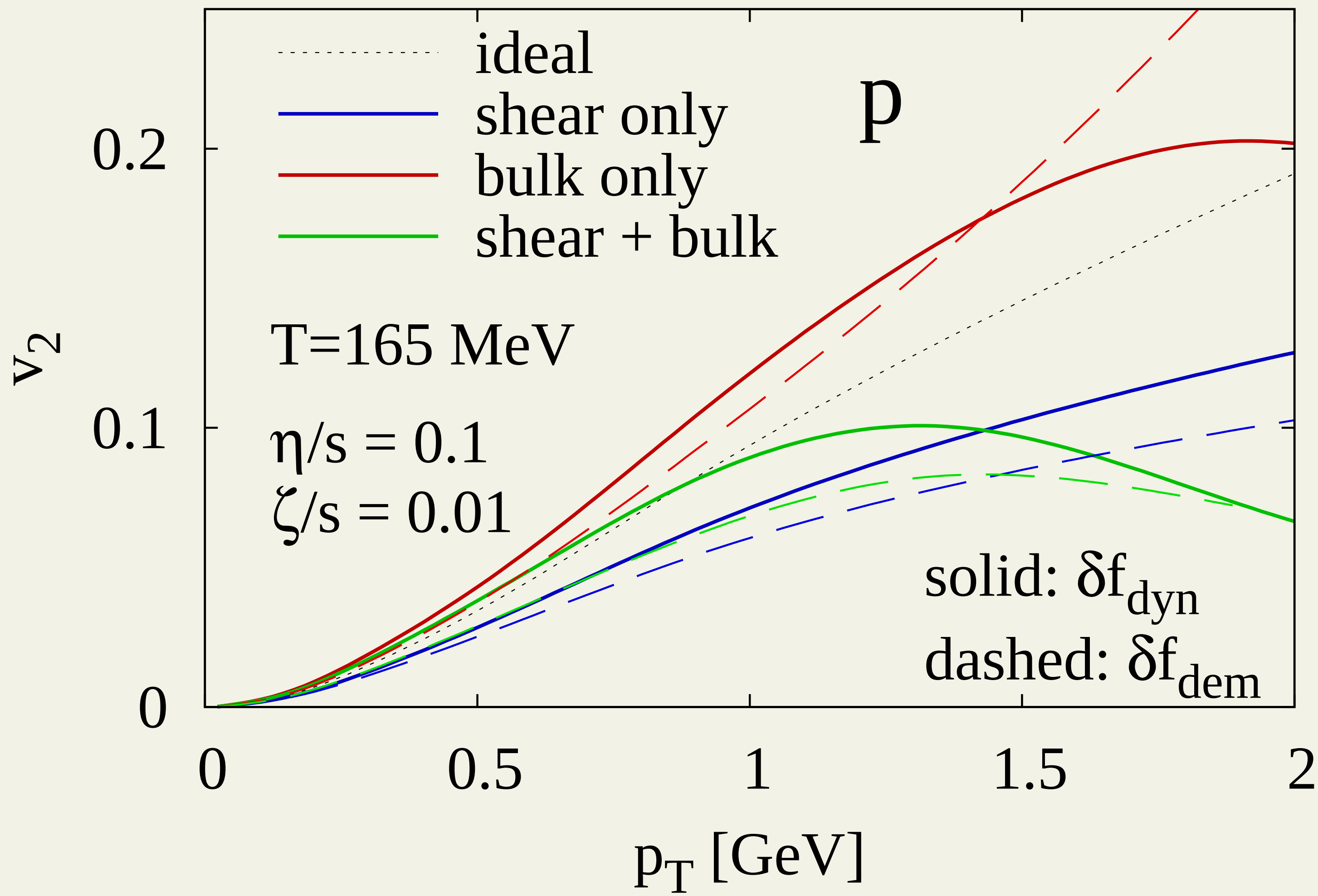
$\chi_{bulk} \propto \delta f_{bulk}/f_{eq}$ **vs particle type and momentum** $(\mu \equiv m/T)$



bulk viscous δf **non-mononotonic vs momentum**

significant viscous correction to proton anisotropy v_2 in Au+Au at RHIC

DM @ QM2018



black: $\delta f = 0$

dashed: ad-hoc δf

solid: kin. theory δf

Summary

Comparison 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 (δf) obtained from kinetic theory. This requires the numerical evaluation of certain 4D integrals.

Our multi-threaded integrator nests adaptive two-dimensional generalizations of Gauss-Kronrod quadrature and pools the evaluations of the integrand on GPU while calculating all needed matrix elements in one go. This is 10-25x faster than single-threaded adaptive Gauss-Kronrod (4 nested 1D integrals).

Next steps:

- more tricks / optimizations: problem size has extra $N_{species}^2 \sim 3000$ factor
  interleave evaluation and reduction kernels

- attack harder problems
  shear viscous δf on GPU (slower kernel, more math to do)
  angle-dependent cross sections (5D integrals)

Summary

Comparison 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 (δf) obtained from kinetic theory. This requires the numerical evaluation of certain 4D integrals.

Our multi-threaded integrator nests adaptive two-dimensional generalizations of Gauss-Kronrod quadrature and pools the evaluations of the integrand on GPU while calculating all needed matrix elements in one go. This is 10-25x faster than single-threaded adaptive Gauss-Kronrod (4 nested 1D integrals).

Next steps:

- **more tricks / optimizations: problem size has extra $N^2_{species} \sim 3000$ factor**
  interleave evaluation and reduction kernels

- **attack harder problems**
  shear viscous δf on GPU (slower kernel, more math to do)
  angle-dependent cross sections (5D integrals)

!! Many thanks to Máté and Dani at the GPU Lab !!

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

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

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.

Bulk viscous $Q[\chi]$

The bulk viscous corrections $\chi_i(p)$ for species i are given by maximizing

$$
\begin{aligned}
Q[\chi] \quad = \quad & \frac{1}{2T^8} \sum_{ijk\ell} \iiiint_{1\,2\,3\,4} f_{1i,eq} f_{2j,eq}\, \chi_{1i} \left(\chi_{3k} + \chi_{4\ell} - \chi_{1i} - \chi_{2j}\right)\, W_{12\to 34}^{ij\to k\ell} \delta^4(12-34) \\
& - \frac{1}{3T^4} \sum_i \int_1 f_{1i,eq} \chi_{1i} p_1^2 + \alpha_E \sum_i \int_1 f_{1i,eq} \chi_{1i} E_1^2 + \sum_c \alpha_c \sum_i q_{c,i} \int_1 f_{1i,eq} \chi_{1i} E_1
\end{aligned}
$$

with the shorthands $\delta^4(12-34) \equiv \delta^4(p_1 + p_2 - p_3 - p_4)$, and

$$
\int_a \equiv \int \frac{d^3 p_a}{2E_a} \,, \quad f_{ai,eq} \equiv f_{i,eq}(p_a) \,, \quad \chi_{ai} \equiv \chi_i(p_a) \,, \quad W_{12\to 34}^{ij\to k\ell} = \frac{4}{\pi} s p_{cm}^2 \frac{d\sigma_{12\to 34}^{ij\to k\ell}}{dt} \,.
$$

Here, i, j, k, and l each run throuh all particle species in the system, while $\alpha_{E,c}$ **are Lagrange multipliers**, ensuring that δf^{bulk} does not contribute to scalars conserved by the collision operator.

The maximum of Q directly gives the bulk viscosity: $\zeta = 4Q_{max} T^3$. We do not only need ζ but the full $\chi_i(p)$ that maximizes Q.