gpu.wigner.mta.hu

CODE GENERATION
FOR PARALLEL DIFFERENTIAL EQUATION SOLVERS

Dániel Berényi

Wigner RCP, Budapest, Hungary

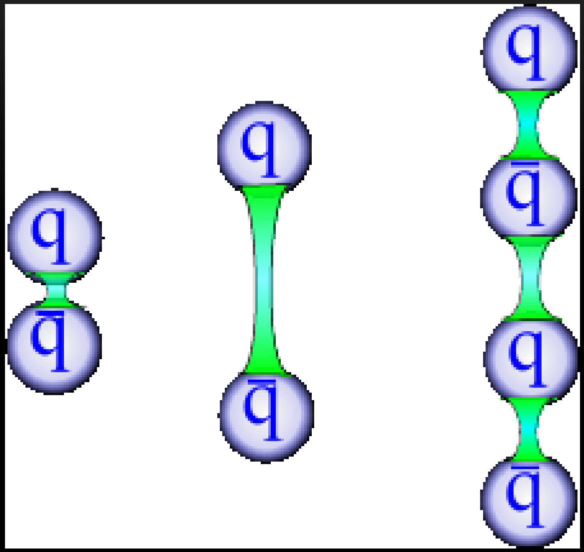
Colleagues: Péter Lévai, Sándor Varró, Vladimir V. Skokov

GPU Day 2015 – The Future of Many-Core Computing in Science
21. May 2015

TABLE OF CONTENTS

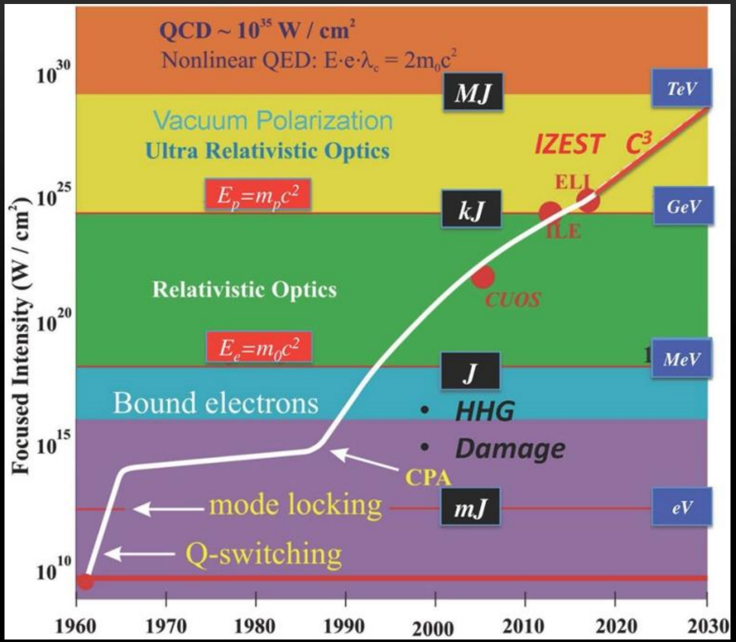
MOTIVATION

- Pair production from vacuum:
  - $q\bar{q}$ in heavy ion collisions:
    - Description of the early stages of collisions,
    - Formation and breaking of color strings,
    - Successful family of models,
    - Understanding of LHC data, particle spectras, etc.



MOTIVATION

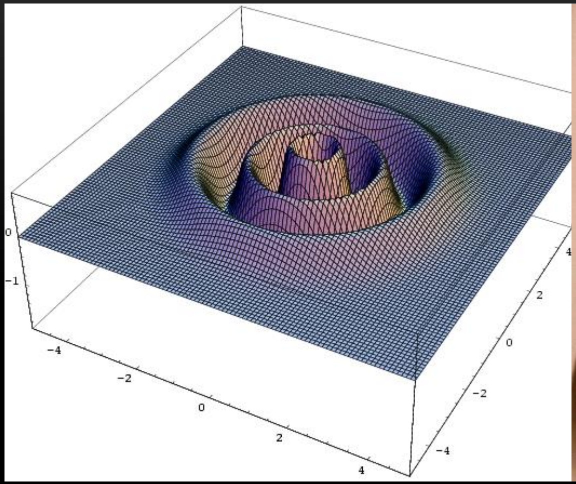
- Pair production from vacuum:
  - e^+e^- in extreme strong laser fields:
    - Holy grail of QED
    - interesting non-linear phenomena in this regime
  - Also possible near compact astrophysical objects



T. Tajima, G. Mourou

THEORETICAL MODEL

- Wigner-functions



Eugene Wigner

- Quantum analogue of the classical one particle distribution function

- Definition:

$$\hat{C}\left(\vec{x},\vec{s},t\right) = \exp\left(-iq\int_{-1/2}^{1/2}\vec{\boldsymbol{A}}(\vec{x}+\lambda\vec{s},t)\vec{s}\,\mathrm{d}\lambda\right)\left[\Psi\left(\vec{x}+\frac{\vec{s}}{2}\right),\bar{\Psi}\left(\vec{x}-\frac{\vec{s}}{2}\right)\right]$$

$$\boldsymbol{W}(\vec{\boldsymbol{x}},\vec{\boldsymbol{p}},\boldsymbol{t}) = -\frac{1}{2}\int e^{-i\vec{p}\vec{s}}\langle 0|\hat{C}\left(\vec{x},\vec{s},t\right)|0\rangle \mathrm{d}\vec{s}$$

I. Bialynicki-Birula et al, Phys. Rev. **D44**, 1825-1835. (1991)

THEORETICAL MODEL

- Time evolution equations:

  A Boltzmann like equations for the quantum one particle distribution function: (QED: 3+3+1 dimension, 16 components, F. Hebenstreit et al, Phys. Rev. **D82** (2010) 105026.)

$$
\begin{aligned}
D_t\mathbb{s} & & &- 2\vec{P}\cdot\vec{\mathbb{t}}_{1} &&= 0\\
D_t\mathbb{p} & & &+ 2\vec{P}\cdot\vec{\mathbb{t}}_{2} &&= 2m\mathbb{a}_0\\
D_t\mathbb{v}_0 &+ \vec{D}_{\vec{x}}\cdot\vec{\mathbb{v}} & & &&= 0\\
D_t\mathbb{a}_0 &+ \vec{D}_{\vec{x}}\cdot\vec{\mathbb{a}} & & &&= 2m\mathbb{p}\\
D_t\vec{\mathbb{v}} &+ \vec{D}_{\vec{x}}\mathbb{v}_0 & &+ 2\vec{P}\times\vec{\mathbb{a}} &&= -2m\vec{\mathbb{t}}_1\\
D_t\vec{\mathbb{a}} &+ \vec{D}_{\vec{x}}\mathbb{a}_0 & &+ 2\vec{P}\times\vec{\mathbb{v}} &&= 0\\
D_t\vec{\mathbb{t}}_{1} &+ \vec{D}_{\vec{x}}\times\vec{\mathbb{t}}_{2} & &+ 2\vec{P}\mathbb{s} &&= 2m\mathbb{v}\\
D_t\vec{\mathbb{t}}_{2} &- \vec{D}_{\vec{x}}\times\vec{\mathbb{t}}_{1} & &- 2\vec{P}\mathbb{p} &&= 0
\end{aligned}
$$

THEORETICAL MODEL

- The Wigner function based description can be extended to higher symmetries (non-Abelian case) too.

  (Quark Wigner function evolution: A.V. Prozorkevich, S.A. Smolyansky, S.V. Ilyin)

- There will be more and more components

- The intermixing of components will be more complex

- If the SU(N) color matrices replaced by unity, the QED equations are recovered.

COMPUTATIONAL ASPECTS

Challenges of the evolution equations:

- Too many dimensions →Simplified configurations are obligatory
  Magnetic field usually neglected
- Handling of the non-local differential operators is not trivial
- Extreme separation of time scales for realistic field parameters

- But most importantly:
  extremely intense computational problem!

COMPUTATIONAL ASPECTS

Challenges of the evolution equations:

- Too many dimensions →Simplified configurations are obligatory
  Magnetic field usually neglected
- Handling of the non-local differential operators is not trivial
- Extreme separation of time scales for realistic field parameters

- But most importantly:
  extremely intense computational problem!

Fits well to GPUs!

WORKFLOW

- The expected workflow is:

1. Formulation
   (mathematical equations, symbolic manipulation)
2. Spectral expansion into a dense matrix problem
   (maybe inside a finite difference time integrator)
3. Efficient parallel implementation/execution

- Possible tools:

1. Maxima, Mathematica, …
2. Maxima, Mathematica, …
   with or exported to
   Host high level language: C++, Fortran…
3. Parallel APIs & libraries:
   OpenCL, CUDA, …
   BLAS implementations…

PROGRAMMING ASPECTS

- We'd like to investigate many **electromagnetic vector field** configurations.
- But the equations may be drasticly simplified by some choices, via symbolic algebraic manipulations.
- **One may not want to rewrite completely the equations each time, when the field is changed!**
- Less tools → less headache…
- Development time constrained!
- More automatization and generic solvers needed!
- We aim to generate the final specialized solver code

  from the mathematical equations!

PROPOSED ALTERNATIVE SOLUTION

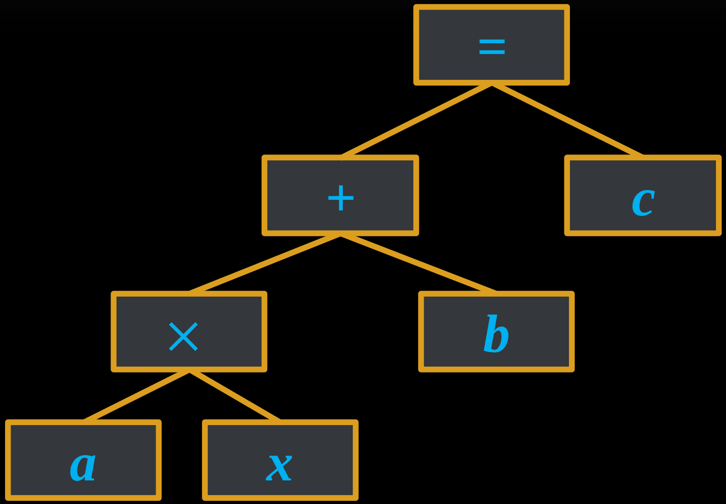
- Run-time construction and manipulation of the abstract syntax tree (AST) from the symbolic equations and dynamic code-generation!

- Main advantages:
  - We can work with only one tool!
  - But, we can support many language back-ends and APIs
  - infer as much as possible from the symbolic equations!
  - eliminate many error sources and inconsistencies!

ABSTRACT SYNTAX TREES

- Mathematical formulas and equations can be represented by trees:

$$ax + b = c$$



AST MANIPULATIONS

Since all of the needed constructs are trees the workflow can be seen as a series of transformations on the ASTs!

- Symbolic (math) stage:

  simplifications ($0 \cdot a \rightarrow a$, $3a + 4a \rightarrow 7a$)

  symbolic differentiation ($\frac{d\sin(x)}{dx} \rightarrow \cos(x)$)

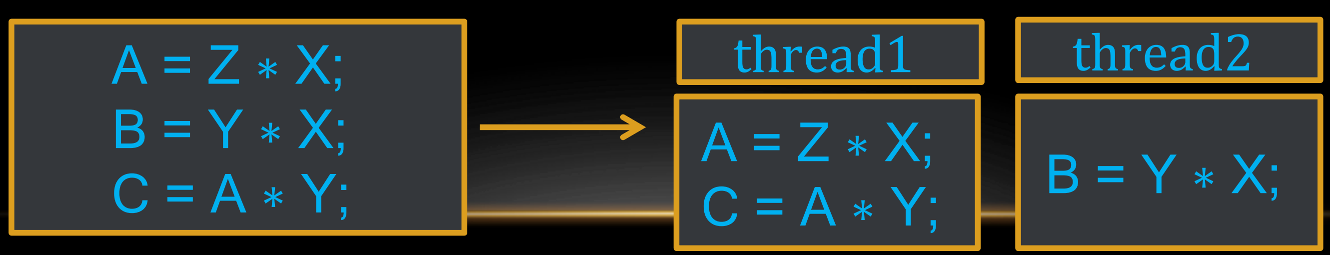
  series expansions ($f(x) \rightarrow \sum_{i=0}^{n} f_i \Phi_i\,(x)$)
  check for argument sanity and rank mismatch

- Programming stage:

  Infer types, further sanity checks

a = dot(vector<2, double>(2., 9.) , vector<2, double>(1., 0.)) →a is scalar double

Parallelization from data dependency (consider matrix operations):



AST MANIPULATIONS

At the symbolic stage a general model is specialized according to user defined constants and parameters and simplified symbolically.
Numerical solvers are just higher-order functions operating on the equations.

Example: Spectral Expansion (like Fourier, Chebyshev series)

1. Equation:

$$\frac{df(x)}{dx} = -af(x)$$

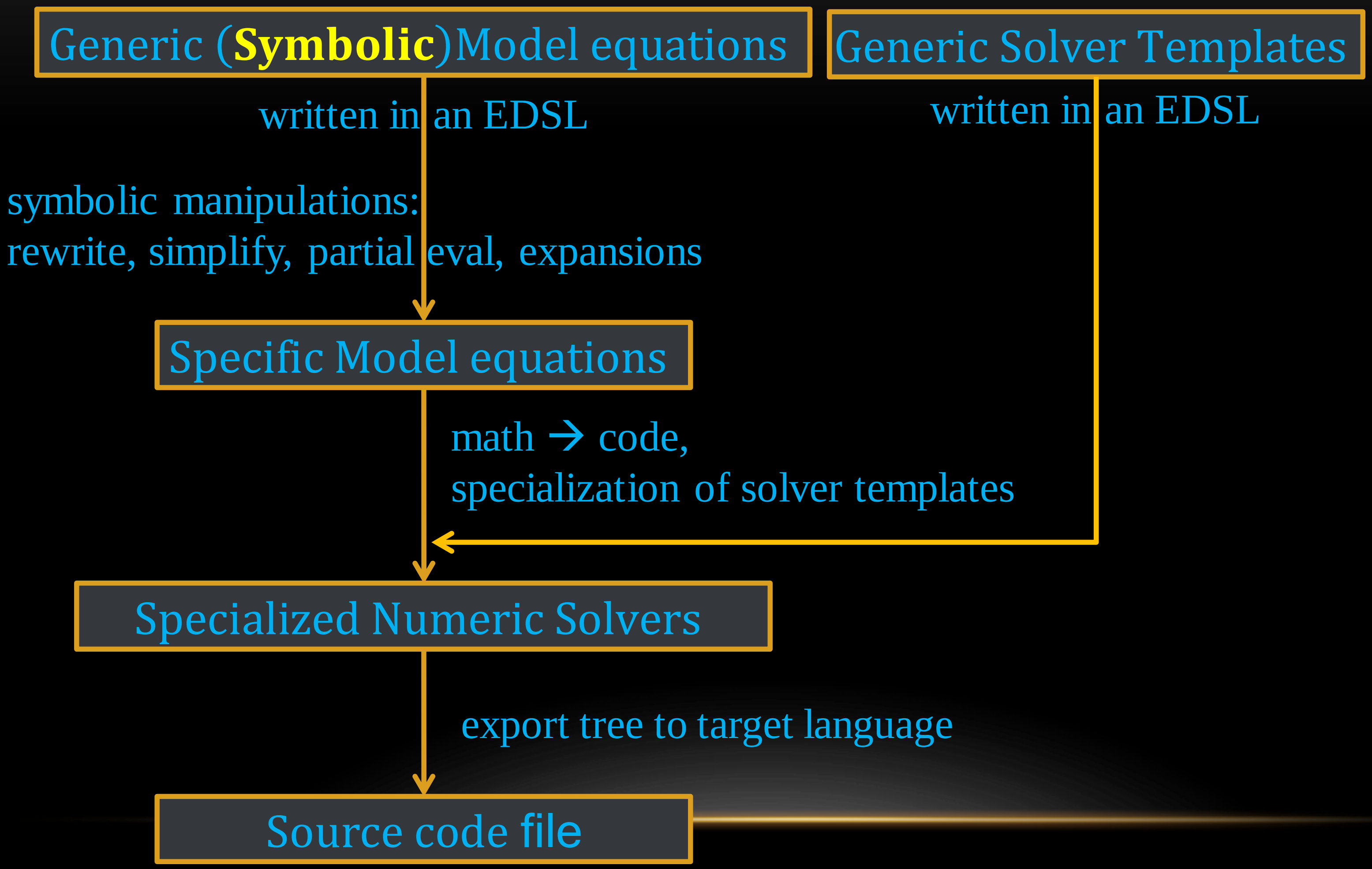
2. Expansion: $f(x) = \sum_{i=0}^{N} f_i \Phi_i(x)$

$$\sum_{i=0}^{N} f_i \frac{d\Phi_i(x)}{dx} = -a \sum_{i=0}^{N} f_i \Phi_i(x)$$

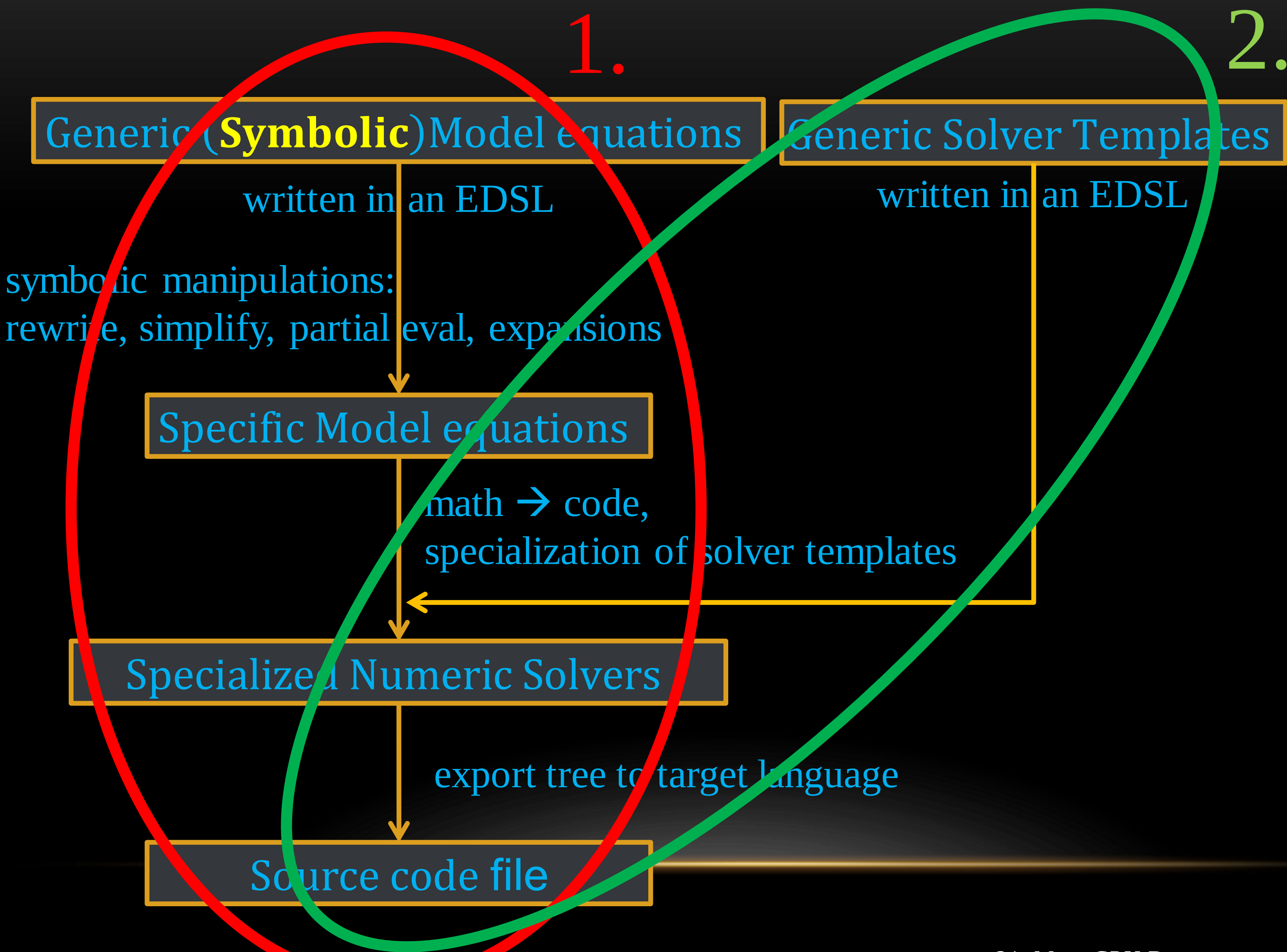
3. Differentiation: $\Phi_i(x) = \cos(iNx)$

$$-\sum_{i=0}^{N} f_i iN \sin(iNx) = -a \sum_{i=0}^{N} f_i \cos(iNx)$$

CODE-GENERATION FLOW



CODE-GENERATION FLOW



CURRENT STATUS

1. Math EDSL Project:

- Symbolic formulation and manipulation of equations, Spectral expansion
- Solution on the GPU with hand written solver
- Successfully tested on many PDEs:
  Wave eq., Fokker-Planck eq., Vlasov eq., sQED Wigner

```
SymbolicDE DE;
DE.DimensionsSymbols() << t << x;
DE.UnknownSymbols() << f;
DE.Equations() << diff(t, f) - D*diff(x, diff(x, f)) + v*diff(x, f);

DE.Constants() << equate(t0, 0.0) << equate(D, 0.75) << equate(v, 1.2);

DE.BoundaryConditions() << f(t0, x) - exp(-sq(x+2.)/(4*D*t0))/sqrt(4*pi*D*t0);

DE.SpectralBases() << SpectralExpansion(L"Chebyshev", 42, 1.0, 5.0)
                   << SpectralExpansion(L"RationalChebyshev", 42, 0.0., 1.5);

DE.ProcessAsFullSpectral();
```

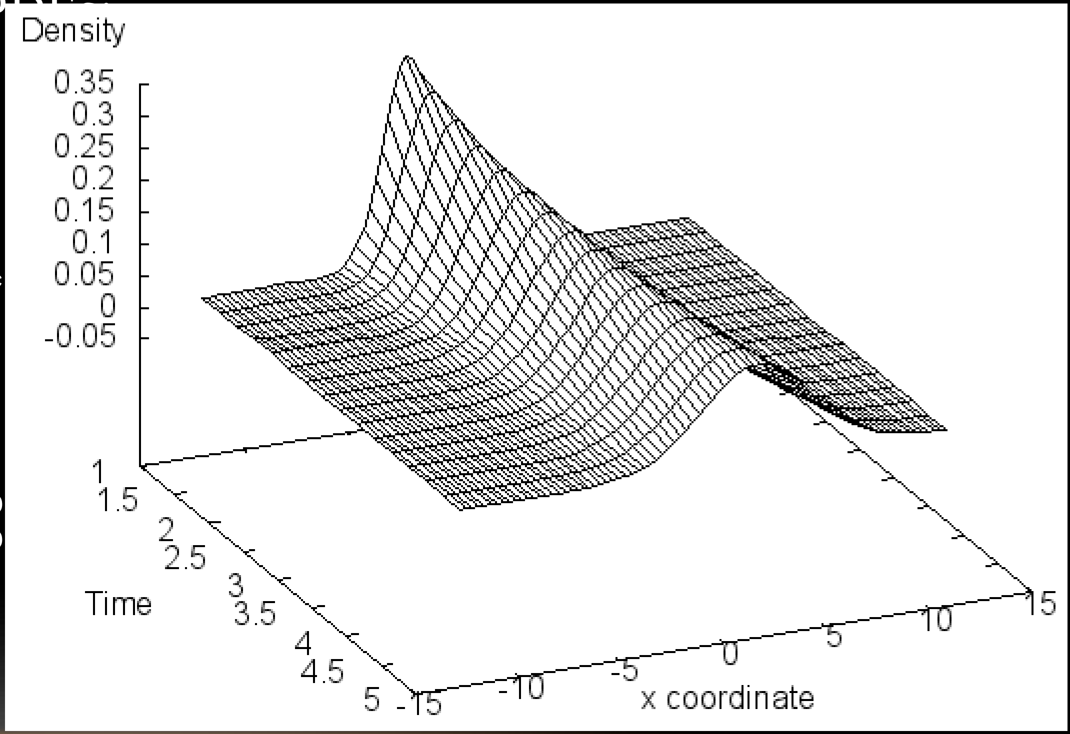
Syntax is subject to further optimization

CURRENT STATUS

1. Math EDSL Project:

- Symbolic formulation and manipulation of equations, Spectral expansion
- Solution on the GPU with hand written solver
- Successfully tested on many PDEs:
  Wave eq., Fokker-Planck eq.,

```
SymbolicDE DE;
DE.DimensionsSymbols() << t << x;
DE.UnknownSymbols() << f;
DE.Equations() << diff(t, f) - D*

DE.Constants() << equate(t0, 0.0)

DE.BoundaryConditions() << f(t0,

DE.SpectralBases() << SpectralExp
                   << SpectralExp

DE.ProcessAsFullSpectral();
```



CURRENT STATUS

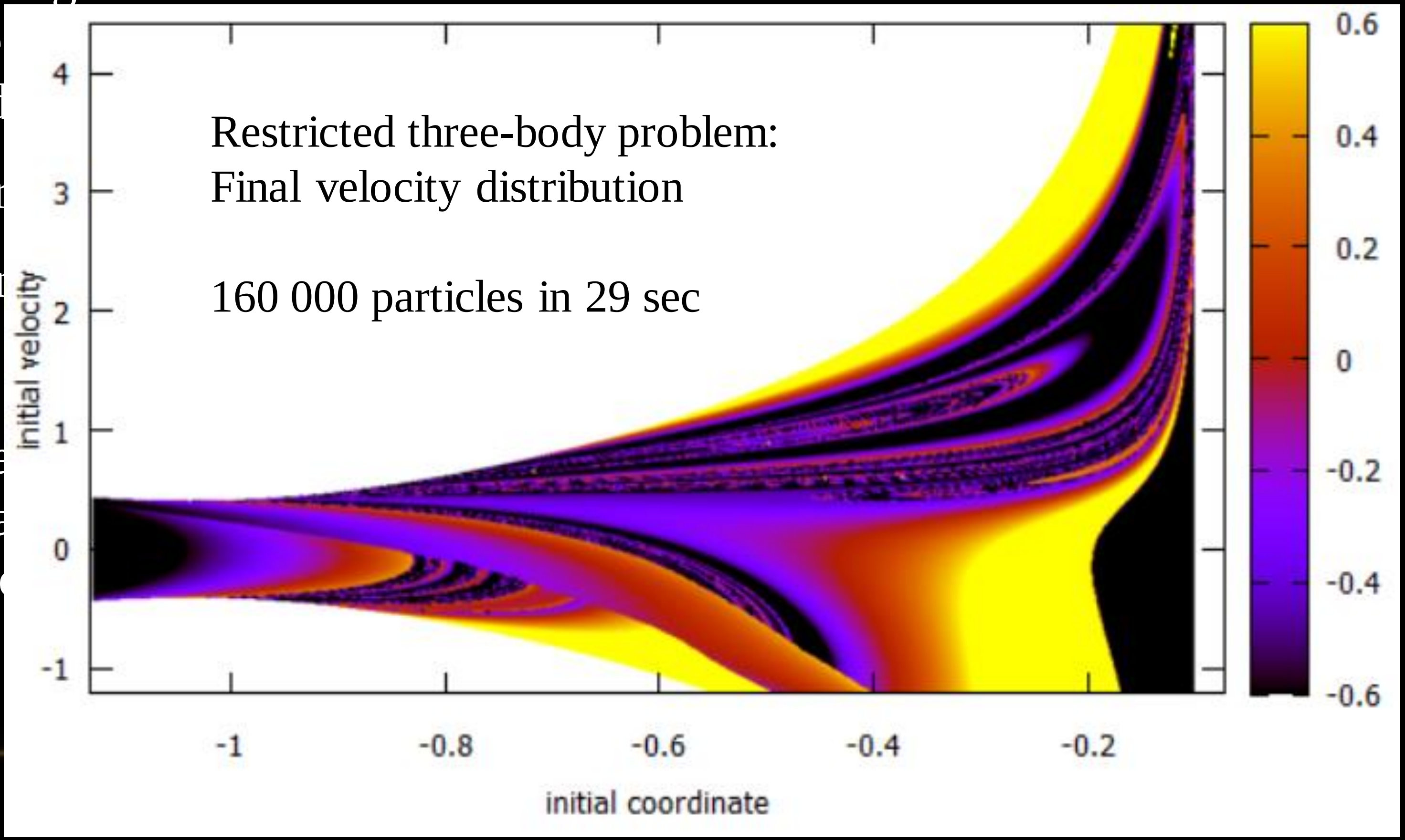
2. Solver Template EDSL Project:

- Expression of parallel programs with higher-order functions
- Automatic generation of host side C++ code and client side OpenCL code
- 8th order adaptive Runge-Kutta stepper implemented
- Tested on ODE systems, ≈30x faster then single thread CPU code.
- Current work is to cleanup and improve syntax of the EDSL in C++
- Current application:
  - collaboration with Gábor Drótos (Eötvös University, Budapest) to study chaotic scattering in the restricted three-body problem.
  - Collaboration with Alexandru Nicolin (Horia Hulubei National Institute, IFIN-HH) to model Bose-Einstein condensates.

CURRENT STATUS

2. Solver Template EDSL Project:

- Expression of parallel programs with higher-order functions
- Automatic generation of host side client sid
- 8th order
- Tested or
- Current a collabora to study



SUMMARY

- Efficient modeling of complicated equations in pair production demand some novel tools to handle special cases automatically while delivering high-performance computation e.g. on GPUs.
- Abstract mathematical models are easier to understand and automatically manipulate symbolically
- Same for abstract program code
- Dynamic code generation can combine these into efficient GPU simulations in a generic way

This work was supported by the Hungarian OTKA Grants No. 104260, No. 106119.

THANK YOU

- Questions?

BACKUP SLIDES

THEORETICAL MODEL

- Hartee-type approximation: $\langle F^{\mu\nu} C \rangle \rightarrow \langle F^{\mu\nu} \rangle \langle C \rangle$
- No back reaction, no radiation corrections, etc.
- Compact form of evolution equation:

$$\boldsymbol{D_t W} = -\frac{1}{2}\overrightarrow{\boldsymbol{D}}_{\vec{x}}[\gamma^0\vec{\gamma},\boldsymbol{W}] - im[\gamma^0,\boldsymbol{W}] - i\overrightarrow{\boldsymbol{P}}\{\gamma^0\vec{\gamma},\boldsymbol{W}\}$$

- The differential operators are non trivial operator series:

$$\boldsymbol{D_t} = \partial_t + e\overrightarrow{\boldsymbol{E}}\vec{\nabla}_p + \ldots$$

- Expansion on 4x4 Dirac basis:

$$\boldsymbol{W}(x,p,t) = \frac{1}{4}\left[1\mathbb{s} + i\gamma_5\mathbb{p} + \gamma^\mu\mathbb{v}_\mu + \gamma^\mu\gamma_5\mathbb{a}_\mu + \sigma^{\mu\nu}\mathbb{t}_{\mu\nu}\right]$$

THEORETICAL MODEL

- In case of no magnetic field (B=0) and homogeneous electric field (E(t)):

$$\frac{\mathrm{d}f}{\mathrm{dt}} = \frac{\mathrm{eE}\varepsilon_\perp}{\omega^2}\mathrm{v}$$

$$\frac{\mathrm{d}v}{\mathrm{dt}} = \frac{1}{2}\frac{eE\varepsilon_\perp}{\omega^2}(1-2f) - 2\omega u$$

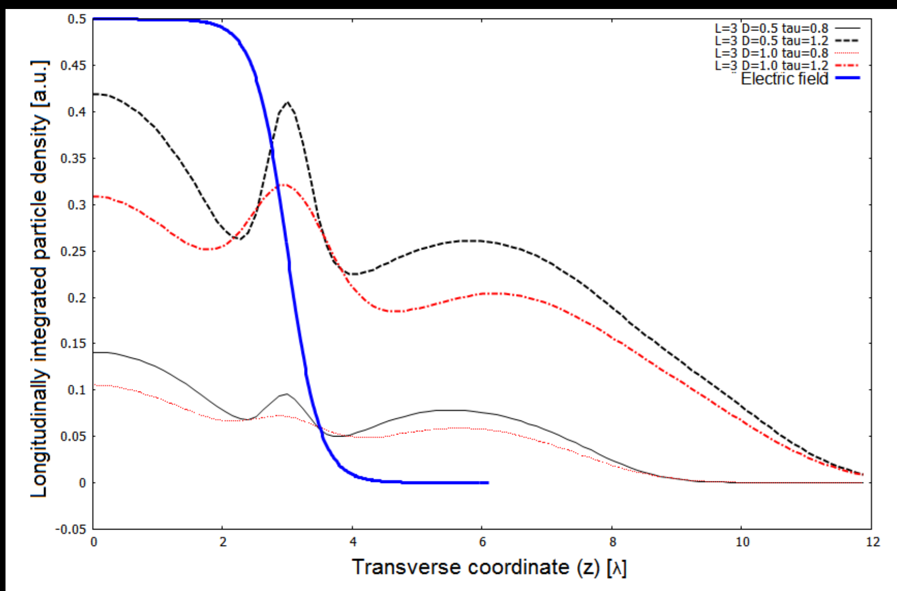
$$\frac{\mathrm{d}u}{\mathrm{d}t} = 2\omega u$$

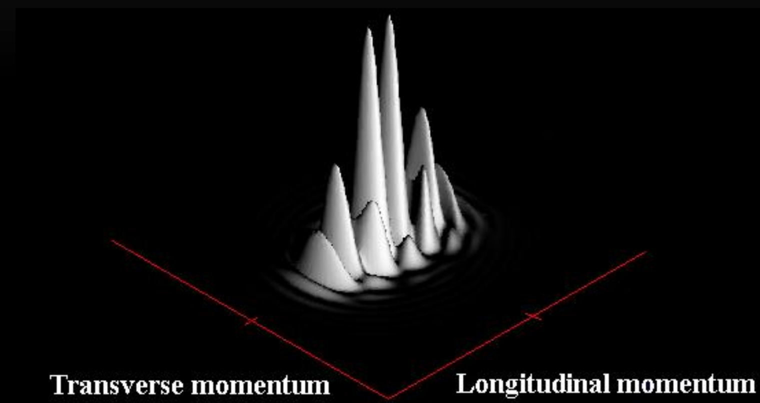
where:

$$\vec{p} = \left(\vec{q}_\perp, q_\parallel - eA(t)\right)$$

PHYSICS

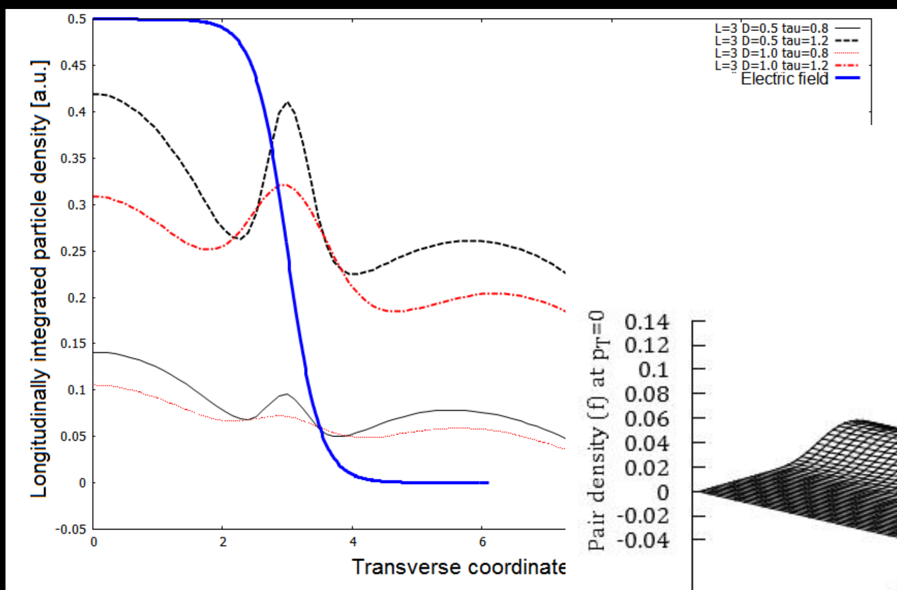
- We can calculate asymptotic ($t \to \infty$) Wigner functions, so we have:
- Energy-, Momentum-, Mass-, Charge- and Spindensity.
- We can calculate particle spectras:

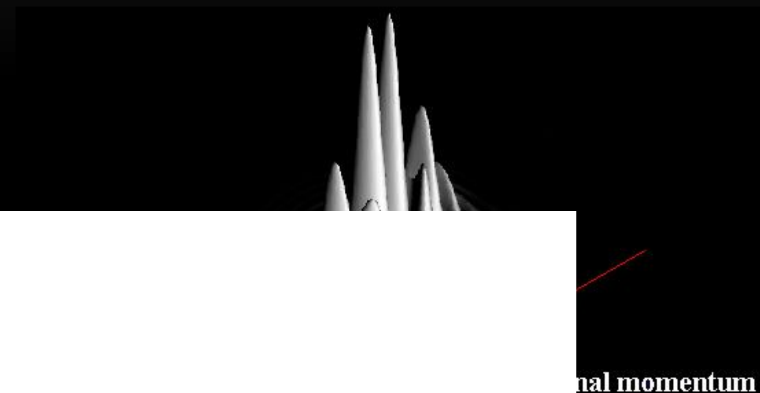


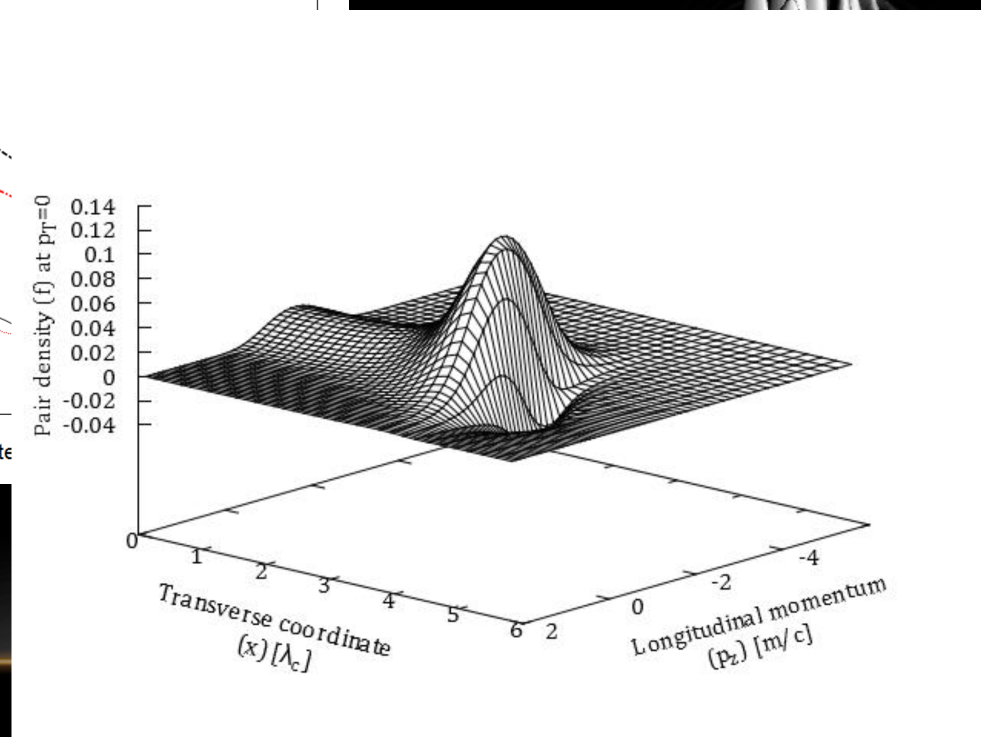


PHYSICS

- We can calculate asymptotic ($t \rightarrow \infty$) Wigner functions, so we have:
- Energy-, Momentum-, Mass-, Charge- and Spindensity.
- We can calculate particle spectras:



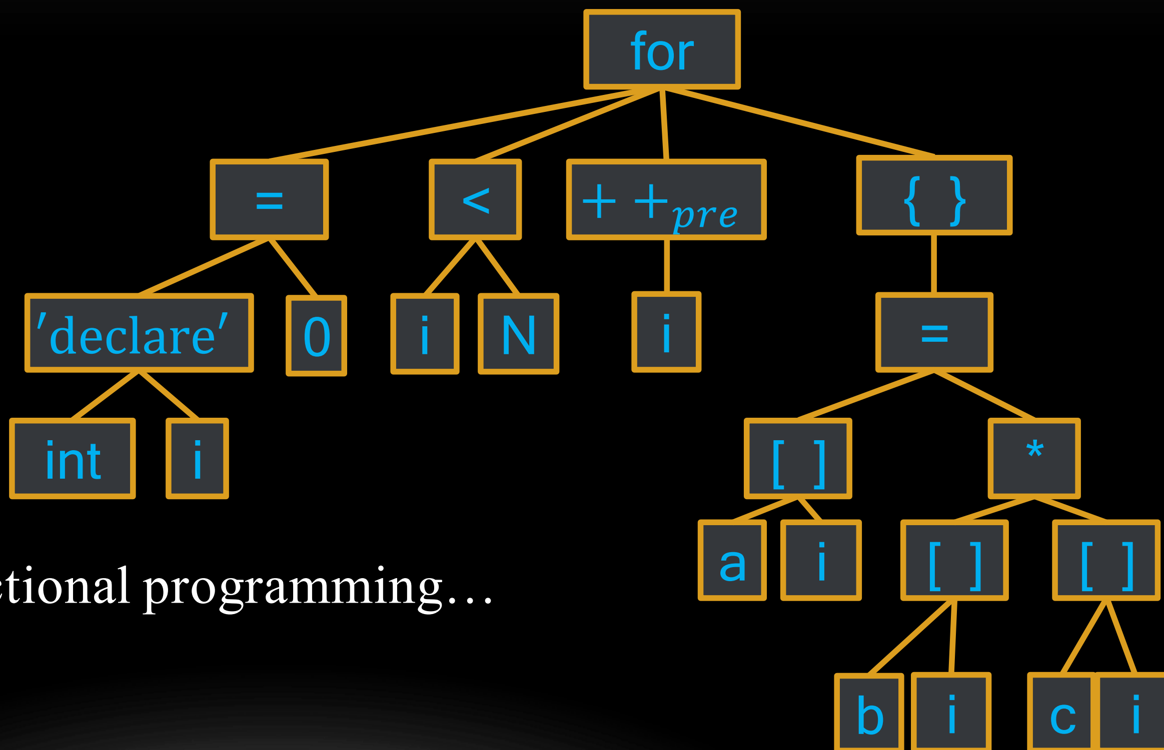




ABSTRACT SYNTAX TREES

- Imperative programming constructs can also be represented by trees:

```
for(int i=0; i<N; ++i)
{
    a[i] = b[i] * c[i];
}
```



- Not to mention functional programming…

SYMBOLIC→PROGRAMMING CONVERSION

All Math objects are given a type deduced from the leaves and propagated upwards.

Function definitions, signatures constructed, defunctionalization applied.

One important construct: ParallelFunction created from vector, matrix operations!

$$f1(a, \vec{x}, \vec{y}) := a\vec{x} + \vec{y}$$

$$\vec{z} = f1(a, \vec{x}, \vec{y})$$

```
void f1(range1 i, double a, double* z,
             double* x, double* y)
{
             z[i] = a * x[i] + y[i];
}
```

Calls are generated as:

```
ParallelCall( f1, RangeOf(z), a, z, x, y);
```

FINAL CODE-GENERATION

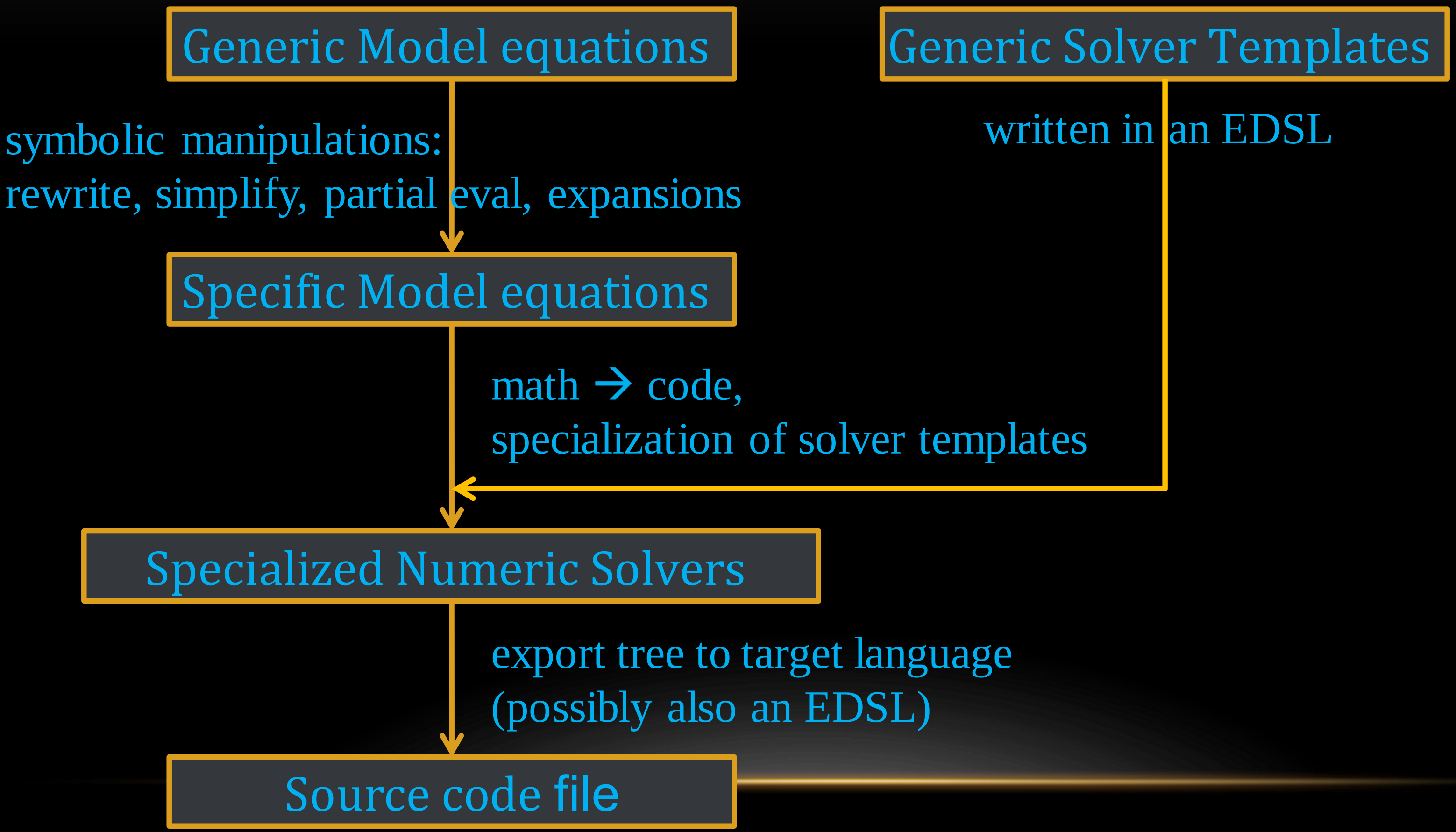
- When all the conversions are ready the program tree is traversed and all the branch operators are converted to their textual equivalents in the selected languages.

- Currently C++ / C / OpenCL export is considered.

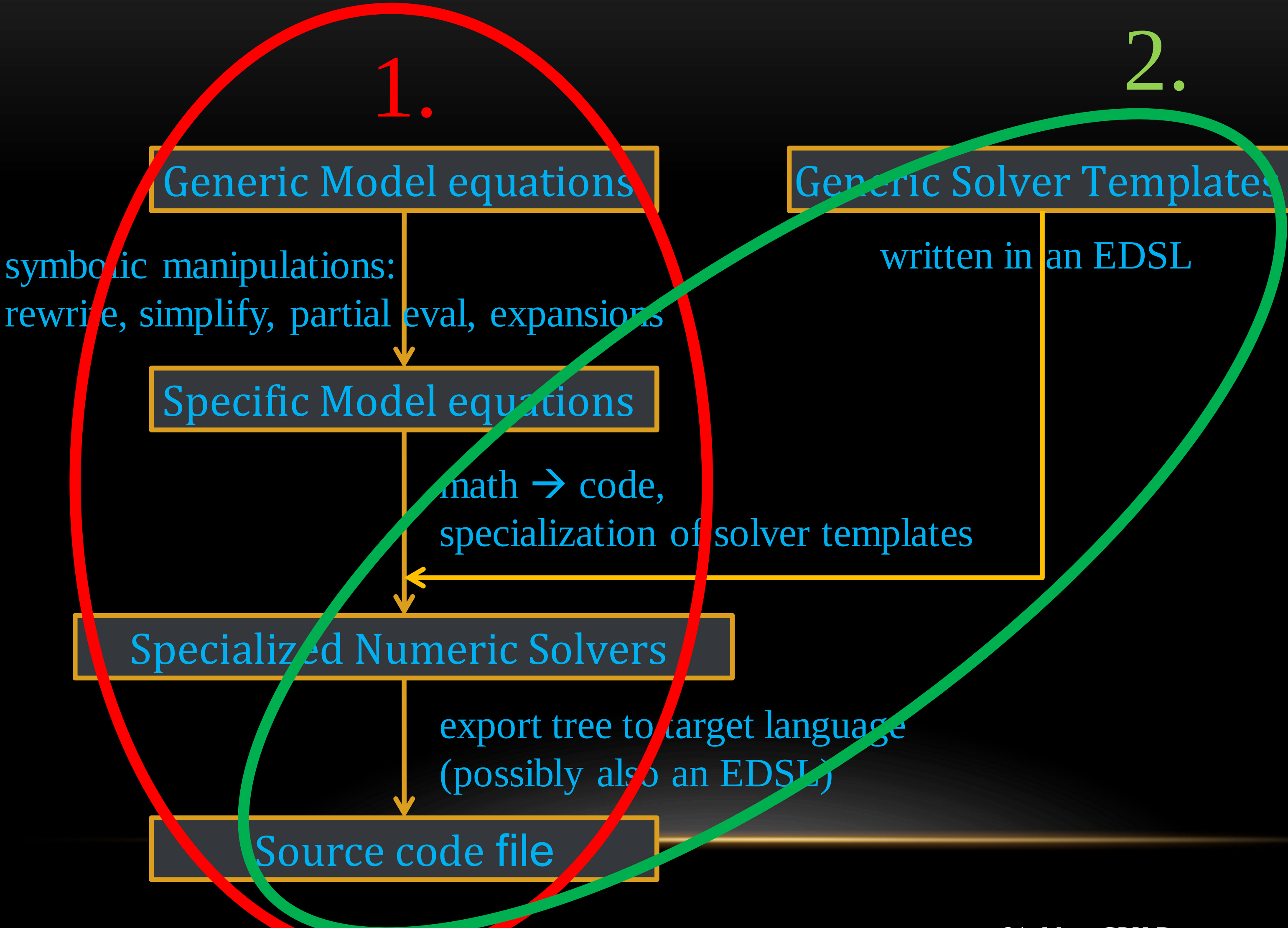
  Why the C family?
  Largest common set of features supported by the compute and rendering APIs:

  OpenCL kernel C, OpenGL GLSL, DirectCompute, DirectX HLSL

CODE-GENERATION FLOW

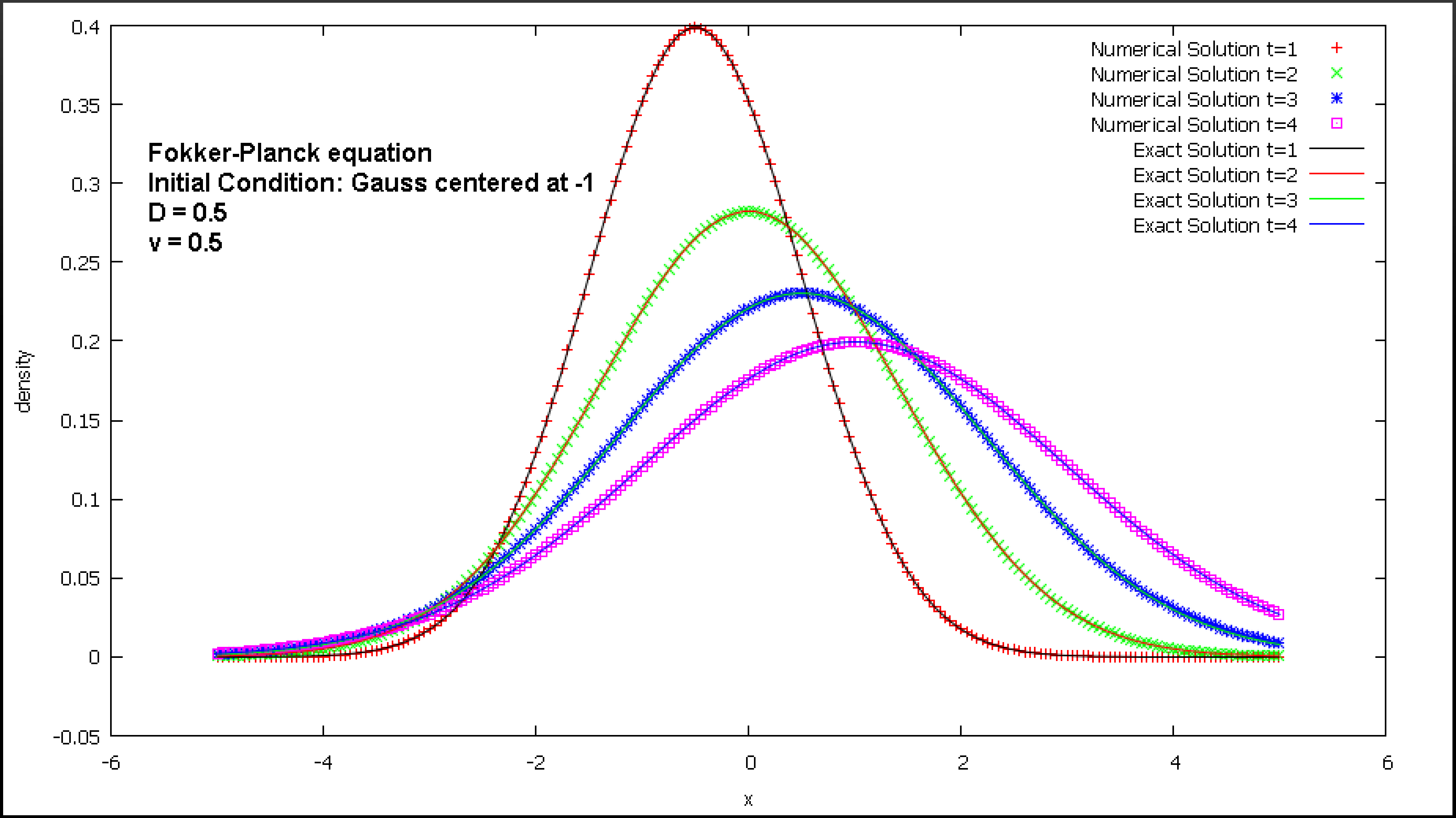


CODE-GENERATION FLOW



DEMONSTRATION

```
#include <Phys/DifferentialEquations.h>

void FokkerPlanckEquation()
{
    SymbolicDE DE;

    MathExpr t(L"t", 1, 1);
    MathExpr x(L"x", 1, 1);
    MathExpr f(L"f", 1, 1);
    MathExpr D(L"D", 1, 1);
    MathExpr v(L"v", 1, 1);
    MathExpr t0(L"t0", 1, 1);
    MathExpr pi(L"PI");

    DE.DimensionSymbols() << t << x;
    DE.UnknownSymbols() << f;
    DE.Equations() << diff(t, f) - diff(x, diff(x, D(x)*f)) + diff(x, v*f);
    DE.Constants() << equate(t0, 0.0);
    DE.Functions() << equate(D, 0.5) << equate(v, 0.5);

    DE.BoundaryConditions()
                << f(t0, x) - exp(-sq(x+v*t0)/(D*4*t0))/sqrt(pi*4.0*t0*D);


    DE.SpectralBases()  << SpectralExpansion(L"RationalChebyshev", 48, 0.0, 1.0)
                        << SpectralExpansion(L"RationalChebyshev", 48, 0.0, 1.5);

    DE.ProcessAsFullSPectral();

    arr<double> ev; ev << 1.0 << 0.0;
    DE.SampleSolutionToFile1(L"out.txt", -5.0, 5.0, 0.05, 1, ev );
    arr<double> ev2; ev2 << 2.0 << 0.0;
    DE.SampleSolutionToFile1(L"out2.txt", -5.0, 5.0, 0.05, 1, ev2 );
    arr<double> ev3; ev3 << 3.0 << 0.0;
    DE.SampleSolutionToFile1(L"out3.txt", -5.0, 5.0, 0.05, 1, ev3 );
    arr<double> ev4; ev4 << 4.0 << 0.0;
    DE.SampleSolutionToFile1(L"out4.txt", -5.0, 5.0, 0.05, 1, ev4 );

    DE.SampleSolutionToFile2(L"fp.txt", -20.0, 20.0, 0.5, 0, -10.0, 10.0, 0.25, 1, ev4 );
}
```


Fokker-Planck equation
Initial Condition: Gauss centered at -1
D = 0.5
v = 0.5
Numerical Solution t=1
Numerical Solution t=2
Numerical Solution t=3
Numerical Solution t=4
Exact Solution t=1
Exact Solution t=2
Exact Solution t=3
Exact Solution t=4
density
x
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-6
-4
-2
0
2
4
6

```cpp
#include "Computation.h"
#include "odes.h"
#include "Ex2.h"

struct RKState{ double x, v, t; double& operator[]( int i ){ return ((double*)&x)[i]; } };

void RK8Test()
{
    using namespace Metaprogramming;

    MetaProgram p;
    ID(a); //identifier for user input
    {
        ID(x); ID(v); ID(t); ID(s); ID(i); ID(ss); //identifiers

        Type Num("double"), State("State"), Int("int"); //type identifiers

        //State struct
        p |= decllist( Num|x, Num|v, Num|t ) | State;

        //RHS of DE
        p |= signature(State, State) | Id("rhs") = $(s, { !(State|ss), ss[~x] = s[~v], ss[~v] = -2.0*s[~x], rt(ss) });

        //Indexer function for State
        p |= signature(Num, State, Int) | Id("indexer") = $( ids(s,i), { rt( Select( i==0, s[~x], s[~v])) });

        //Higher order solver function definition imported:
        p |= getRK8(Id("indexer"));

        //Main entry point and solver invoke (translates to kernel call)
        p |= signature(Type::Void(), vec(Num) )|Id("main") = $(a, async_block( !Id("rk8")(domof(a), a, Id("rhs"), 2, Id("indexer")) ) );
    }

    Namespace ns;
    au state = CreateBuffer<RKState>(200); //user buffer in CPU RAM
    for( int i=0; i<state.ext[0]; i++ ){ state[i].x = 0.0; state[i].v = 2.0*i; state[i].t = 0.0; }

    ns.CreateBuffer(a, state ); //Bind to identifier
    ns.AddCode(p);              //Compile metaprogram
    ns.exec( a );               //Compile and Launch with identifier as parameter
    ns.ReadBuffer("a");         //Read back to user buffer
}
```

DEMONSTRATION