Parallel Algorithms for Semiclassical Quantum Dynamics

Manfred Liebmann
Technische Universität München
Chair of Optimal Control
Center for Mathematical Sciences, M17
manfred.liebmann@tum.de

February 2, 2016

Quantum Molecular Dynamics

- **Time-dependent Schrödinger Equation**: Let $\varepsilon > 0$ be a small parameter. We want to find solutions to

$$\mathrm{i}\,\varepsilon\,\partial_t\psi(t,x) = -\tfrac{1}{2}\,\varepsilon^2\,\triangle\,\psi(t,x) + V(x)\,\psi(t,x)$$

for initial datum $\psi(0,\cdot) = \psi_0 \in L^2(\mathbb{R}^d)$ with norm $\|\psi_0\| = 1$.

- Assume $\mathcal{H}^\varepsilon = -\frac{\varepsilon^2}{2}\triangle + V$ to be a self-adjoint linear operator. Then there exists a unique solution

$$\psi(t,\cdot) = \mathcal{U}_t^\varepsilon\psi_0 := \mathrm{e}^{-\frac{\mathrm{i}}{\varepsilon}\mathcal{H}^\varepsilon t}\psi_0.$$

for all times $t \in \mathbb{R}$ by means of the unitary group $\mathcal{U}_t^\varepsilon$ generated by the Hamiltonian $\mathcal{H}^\varepsilon$.

Wave Packets

- A Gaussian wave packet $g_z^\varepsilon \in \mathcal{S}(\mathbb{R}^d)$ centered at $z = (q,p) \in \mathbb{R}^{2d}$ shall be defined by

$$g_z^\varepsilon(x) = (\pi\varepsilon)^{-\frac{d}{4}} \exp\left(-\frac{1}{2\varepsilon}\left|x-q\right|^2 + \frac{\mathrm{i}}{\varepsilon}\, p\cdot(x-q)\right).$$

- Any $\psi \in L^2(\mathbb{R}^d)$ may be represented by wave packets in the way

$$\psi = (2\pi\varepsilon)^{-d} \int_{\mathbb{R}^{2d}} g_z^\varepsilon \, \langle g_z^\varepsilon, \psi\rangle \, dz.$$

- From this we get the formal identity

$$\mathcal{U}_t^\varepsilon \psi = (2\pi\varepsilon)^{-d} \int_{\mathbb{R}^{2d}} (\mathcal{U}_t^\varepsilon g_z^\varepsilon) \, \langle g_z^\varepsilon, \psi\rangle \, dz.$$

Herman–Kluk Propagator

Definition 1 (Herman & Kluk, 1984). *The Herman–Kluk propagator is given by*

$$\mathcal{I}_t^\varepsilon \psi := (2\pi\varepsilon)^{-d} \int_{\mathbb{R}^{2d}} \left(u(t,z)\, \mathrm{e}^{\frac{\mathrm{i}}{\varepsilon} S(t,z)} g^\varepsilon_{\Phi^t(z)} \right) \langle g^\varepsilon_z, \psi \rangle \, dz.$$

The Herman–Kluk propagator incorporates the classical Hamiltonian flow $\Phi^t = (X^t, \Xi^t) : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ and classical action S, as well as the Herman–Kluk prefactor

$$u(t,z) = \sqrt{2^{-d} \det(\partial_q X^t(z) + \partial_p \Xi^t(z) + \mathrm{i}(\partial_q \Xi^t(z) - \partial_p X^t(z)))}.$$

to give an approximation to the time evolution defined by the unitary group

$$\mathcal{U}_t^\varepsilon \psi = (2\pi\varepsilon)^{-d} \int_{\mathbb{R}^{2d}} (\mathcal{U}_t^\varepsilon g^\varepsilon_z) \, \langle g^\varepsilon_z, \psi \rangle \, dz.$$

.

Approximation Property of the Herman–Kluk Propagator

Theorem 1 (Swart & Rousse, 2009). *The Herman–Kluk propagator is a bounded operator on $L^2(\mathbb{R}^d)$ which approximates the unitary propagator in the following sense. For every $T > 0$, there exists $C > 0$ such that for all $\varepsilon > 0$*

$$\sup_{t\in[0,T]} \|\mathcal{I}_t^\varepsilon - \mathcal{U}_t^\varepsilon\| \leq C\,\varepsilon.$$

Discretization of Phase Space

- We may rewrite the function $z \mapsto \langle g_z^\varepsilon, \psi \rangle$ as

$$(2\pi\varepsilon)^{-d} \langle g_z^\varepsilon, \psi \rangle =: \mu_\psi(z) r_\psi(z),$$

such that $\mu_\psi(z) \geq 0$ and $\int_{\mathbb{R}^{2d}} \mu_\psi(z) dz = 1$.

- The Herman–Kluk propagator is then given by

$$\mathcal{I}_t^\varepsilon \psi = \int_{\mathbb{R}^{2d}} \left(r_\psi(z) u(t,z) \mathrm{e}^{\frac{\mathrm{i}}{\varepsilon} S(t,z)} g^\varepsilon_{\Phi^t(z)} \right) d\mu_\psi(z)$$

and can be interpreted as a weighted integration over $\mathbb{R}^{2d}$.

Discretization of phase space

Theorem 2 (Lasser & Sattlegger, 2014). *Let $M \in \mathbb{N}$ and $\{z_1, \ldots, z_M\} \subset \mathbb{R}^{2d}$ be a μ_ψ distributed low discrepancy set. For all $t \in \mathbb{R}$ and $x \in \mathbb{R}^d$ let us define*

$$\psi_M(t,x) = \frac{1}{M} \sum_{m=1}^{M} r_\psi(z_m)\, u(t,z_m)\, \mathrm{e}^{\frac{\mathrm{i}}{\varepsilon} S(t,z_m)}\, g^{\varepsilon}_{\Phi^t(z_m)}.$$

Then

$$\lim_{M\to\infty} \|\psi_M(t,\cdot) - \mathcal{I}^{\varepsilon}_t \psi\| = 0.$$

Discretization of time

Theorem 3 (Lasser & Sattlegger, 2013). *Let $\tau > 0$. Compute approximate solutions $\tilde{u}$, $\tilde{S}$, and $\tilde{\Phi}^\tau$ of the classical mechanical system using a symplectic integrator of order γ and define*

$$\tilde{\mathcal{I}}_\tau^\varepsilon := (2\pi\varepsilon)^{-d} \int_{\mathbb{R}^{2d}} \tilde{u}(\tau, z)\, \mathrm{e}^{\frac{\mathrm{i}}{\varepsilon}\tilde{S}(\tau,z)}\, g^\varepsilon_{\tilde{\Phi}^\tau(z)}\, \langle g^\varepsilon_z, \psi\rangle\, dz.$$

Then there exist constants $C_1, C_2 > 0$ such that for all $\varepsilon > 0$ and $\tau > 0$

$$\left\| \tilde{\mathcal{I}}_\tau^\varepsilon - \mathcal{I}_\tau^\varepsilon \right\| \leq C_1 \tau^\gamma \varepsilon^{-1} + C_2 \tau^\gamma.$$

Egorov method

Theorem 4 (Egorov, 1969). *For the solution ψ of the semiclassical Schrödinger equation and a general Weyl-quantized operator $\mathrm{op}^\varepsilon(a)$ we have for the expectation value*

$$\langle \mathrm{op}^\varepsilon(a)\psi(t,\cdot),\psi(t,\cdot)\rangle = \langle \mathrm{op}^\varepsilon(a\circ\Phi^t)\psi_0,\psi_0\rangle + \mathcal{O}(\varepsilon^2)$$

The key idea is to rephrase the last expression in terms of the Wigner function $W(\psi_0) : \mathbb{R}^d \to \mathbb{R}$ corresponding to ψ_0 which is defined by

$$W(\psi_0)(z) = (2\pi\varepsilon)^{-d} \int_{\mathbb{R}^d} \mathrm{e}^{\frac{\mathrm{i}}{\varepsilon}p\cdot y}\, \overline{\psi_0\left(q+y/2\right)}\, \psi_0\left(q-y/2\right)\, dy$$

for every $z=(q,p)\in\mathbb{R}^{2d}$. The approximation may then be restated as

$$\langle \mathrm{op}^\varepsilon(a)\psi(t,\cdot),\psi(t,\cdot)\rangle = \int_{\mathbb{R}^{2d}} (a\circ\Phi^t)(z)\, W(\psi_0)(z)\, dz + \mathcal{O}(\varepsilon^2).$$

Discretization of time

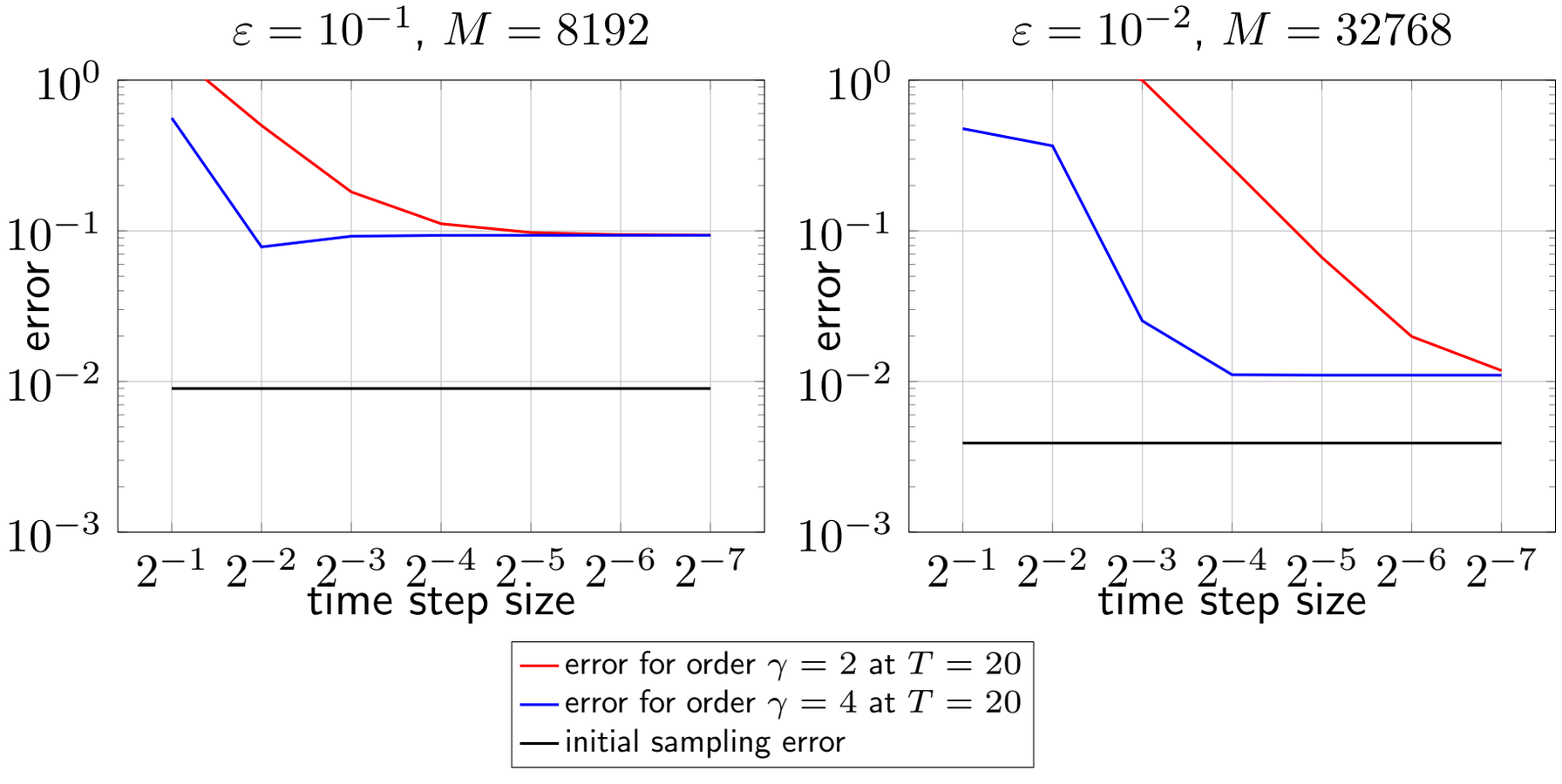
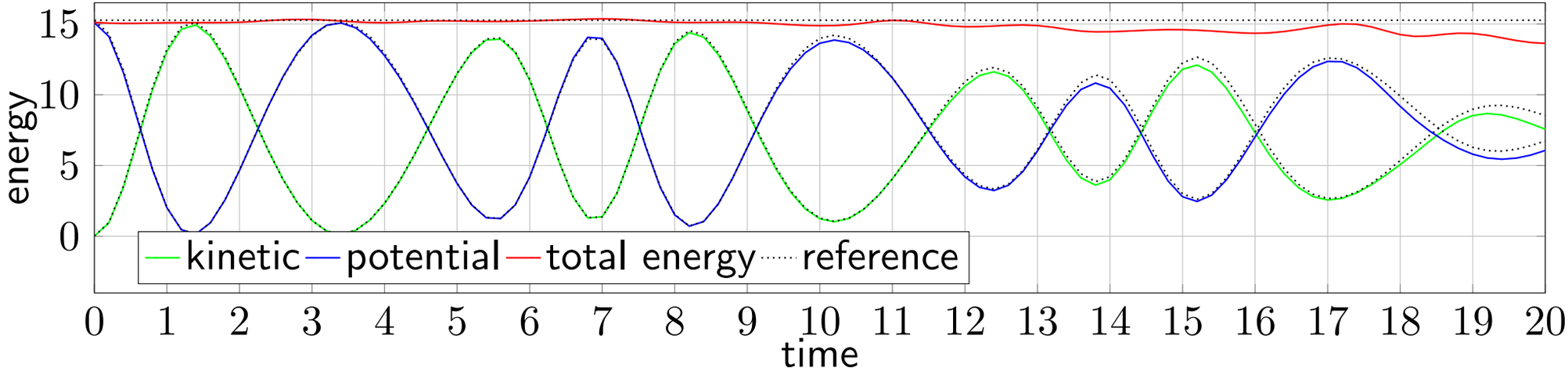


Figure 1: Dependence of the error $\|\psi(T) - \psi_{\mathrm{ref}}(T)\|$ on time step size.

A six-dimensional quantum system

Calculation of energy expectation values for the **6d** Henon–Heiles potential

$$V(x) = \sum_{k=1}^{6} \frac{1}{2} x_k^2 + \sigma \sum_{k=1}^{5} \left(x_k x_{k+1}^2 - \frac{1}{3} x_k^3 \right) + \frac{\sigma^2}{16} \sum_{k=1}^{5} \left(x_k^2 + x_{k+1}^2 \right)^2 .$$



Parameters: $\varepsilon = 10^{-2}$, $\sigma = \frac{1}{\sqrt{80}}$ with a Gaussian at $(2, \ldots, 2, 0 \ldots, 0)^T$ as initial datum.

High Performance Computing

- Automated C++ code generation for Störmer–Verlet method using Mathematica
  - Produced ≈ 22500 lines of code
- Parallelization "on all levels"
  - Hybrid MPI + OpenMP + SSE/AVX vectorization
  - **Vectorization class library** for SSE/AVX is integrated with code generator

Störmer–Verlet Time Stepping Scheme

Consider the Hamilton function $H(q,p) = \frac{1}{2}p^T M^{-1} p + V(q)$ with $q, p \in \mathbb{R}^n$ then the equations of motion are

$$\dot{q}_i = \frac{\partial}{\partial p_i} H(q,p), \quad \dot{p}_i = -\frac{\partial}{\partial q_i} H(q,p), \quad i = 1, \ldots, n$$

and the Størmer–Verlet time stepping scheme is given by

$$\begin{aligned} p^{n+1/2} &= p^n - \frac{\Delta t}{2} \nabla_q V(q^n) \\ q^{n+1} &= q^n - \Delta t M^{-1} p^{n+1/2} \\ p^{n+1} &= p^{n+1/2} - \frac{\Delta t}{2} \nabla_q V(q^{n+1}) \end{aligned}$$

Herman–Kluk Prefactor

The calculation of the Herman–Kluk prefactor requires the evaluation of the continuous complex square root of the determinant of a complex matrix.

$$u(t,z) = \sqrt{2^{-d} \det(\partial_q X^t(z) + \partial_p \Xi^t(z) + \mathrm{i}(\partial_q \Xi^t(z) - \partial_p X^t(z)))}.$$

With the Hamiltonian flow given by

$$\Phi^t : \mathbb{R}^{2d} \to \mathbb{R}^{2d}, \quad z_0 \mapsto \begin{pmatrix} X^t(z_0) \\ \Xi^t(z_0) \end{pmatrix}$$

such that $z(t) = \Phi^t(z_0)$ satisfies the Hamilton equations of motion with the initial datum $z(0) = z_0$.

Mathematica Code Generator

First, the $2d$ components of the vector $z = (q, p) \in \mathbb{R}^{2d}$ are initialized by

```
q = Array[Symbol[StringJoin["q",ToString[#]]]&,d];
p = Array[Symbol[StringJoin["p",ToString[#]]]&,d];
```

and then the $4d^2$ entries of the matrices $\partial_q X^t, \partial_p X^t, \partial_q \Xi, \partial_p \Xi^t \in \mathbb{R}^{d \times d}$ by

```
Xq =Array[Symbol[StringJoin["Xq",ToString[#1],"x",ToString[#2]]]&,{d,d}];
Xp =Array[Symbol[StringJoin["Xp",ToString[#1],"x",ToString[#2]]]&,{d,d}];
Xiq =Array[Symbol[StringJoin["Xiq",ToString[#1],"x",ToString[#2]]]&,{d,d}];
Xip =Array[Symbol[StringJoin["Xip",ToString[#1],"x",ToString[#2]]]&,{d,d}];
```

We will illustrate the further course of action using the example of a Henon–Heiles type potential which shall be given by

$$V(x) = \frac{1}{2}\sum_{k=1}^{d} x_k^2 + \sigma \sum_{k=1}^{d-1}\left(x_k x_{k+1}^2 - \frac{1}{3}x_k^3\right) + \beta \sum_{k=1}^{d-1}\left(x_k^2 + x_{k+1}^2\right)^2 \tag{1}$$

where $\sigma > 0$ and $\beta > 0$ are two positive constants. We enter the formula of the potential and Mathematica algebraically computes the gradient and the Hessian and simplifies the resulting expressions.

```
V = 1/2 Sum[q[[k]]^2, {k, 1, d}] +
    sigma Sum[q[[k]] q[[k + 1]]^2 - q[[k]]^3/3, {k, 1, d - 1}] +
    beta Sum[(q[[k]]^2 + q[[k + 1]]^2)^2, {k, 1, d - 1}] // FullSimplify;
GradV = D[V, {q}] // FullSimplify;
HessV = D[V, {q, 2}] // FullSimplify;
```

Furthermore we compute the matrix-matrix multiplication of the Hessian and W symbolically and again simplify the results.

```
HessVXq = HessV.Xq //FullSimplify;
HessVXp = HessV.Xp //FullSimplify;
```

With a series of elaborate string operations the source code of the time stepping scheme is produced automatically. Let us just give an example of the string manipulation needed for one step in p, $\partial_q \Xi$, and $\partial_p \Xi$ of the Störmer time stepping scheme.

```
ToString[ Array[ StringJoin[
    ToString[p[[#1]]], "+=", ToString[CForm[dt N[GradV[[#1]]]
    //. {Power[x_,2] -> HoldForm[x*x], Power[x_,3] -> HoldForm[x*x*x]}]],
    ";"] &, {d, 1}]] <>

 ToString[ Array[ StringJoin[
     ToString[Xiq[[#1,#2]]], "+=", ToString[CForm[dt N[HessVXq[[#1,#2]]]]
       //. {Power[x_, 2] -> HoldForm[x*x]}], ";"] &, {d, d}]] <>

 ToString[ Array[ StringJoin[
     ToString[Xip[[#1,#2]]], "+=", ToString[CForm[dt N[HessVXp[[#1,#2]]]]
       //. {Power[x_, 2] -> HoldForm[x*x]}], ";"] &, {d, d}]];
```

C++ Library for Explicit Vectorization

The vectorization library defines classes for the vector data types `double2`, `double4`, and `double8` and provides all elementary arithmetic operators $+, -, *, /$ by overloading the standard built-in operators. Likewise all common functions, e.g. `exp`, `sqrt`, `sin`, `cos`, are provided for the vector data types by the library.

```
class double2
{
    __m128d v;
    double2& operator*=(const double2& rhs)
    {
        v = _mm_mul_pd(v, rhs.v);
        return *this;
    }
    double2& sqrt()
    {
        v = _mm_sqrt_pd(v);
        return *this;
    }
};
```

The arithmetic operators and functions are not only overloaded to act directly on the

instances of the vector data types as member functions, for example `v.sqrt()` or `v *= 2.0`, but are also provided as globally defined operators to allow a more natural use of the operators similar to the built in types.

For example the expression `v = sqrt(x) * y + 2.0 * z` is valid code for all vector data types.

```
inline double2 operator*(double2 lhs, const double2& rhs)
{
    lhs *= rhs;
    return lhs;
}

inline double2 sqrt(double2 lhs)
{
    lhs.sqrt();
    return lhs;
}
```

Benchmark for Herman–Kluk Method and Henon–Heiles Model

- Benchmark: 6d Henon–Heiles potential
  - 2^{22} sampling points resulting in 331,350,016 ODEs
  - 100 time steps $+$ 10 evaluations of energy expectation values

MPI + AVX	CPU time	speedup
1 core	350.41 s	
2 cores	175.86 s	1.99
4 cores	88.20 s	3.97
8 cores	43.58 s	8.04
16 cores	22.03 s	15.91

Table 1: Mephisto cluster: 1 compute node with 2×Intel Xeon CPU E5-2650 @ 2.00GHz

Vectorization and Parallelization Efficiency

vector size	64bit (FPU)	128bit (SSE)	256bit (AVX)
CPU time	53.32 s	30.76 s	22.03 s
speedup		1.73	2.42

Table 2: Vectorization efficiency: 16 cores on 2×Intel Xeon CPU E5-2650 @ 2.00GHz

Hybrid MPI + OpenMP + SSE	CPU time	speedup
6 cores (1 MPI × 6 OpenMP)	102.23 s	
12 cores (2 MPI × 6 OpenMP)	52.86 s	1.93
24 cores (4 MPI × 6 OpenMP)	27.37 s	3.74
48 cores (8 MPI × 6 OpenMP)	13.53 s	7.56

Table 3: Mephisto cluster: 4 compute nodes with 8×Intel Xeon CPU X5650 @ 2.67GHz

Benchmark for Egorov Method and Henon–Heiles Model

dim	1x Nvidia Tesla K20	2x Intel Xeon E5-2650
16	2.746	28.054
32	5.544	57.723
64	24.713	126.527
128	198.988	544.650
256	482.869	1317.154
512	1036.739	2625.686

Table 4: Egorov method for Henon–Heiles potential

Compilation Time Scaling

The automatically generated code gets very complex with increasing dimension. The table compares GPU and CPU code generation.

dim	nvcc	g++
16	9.008s	2.130s
32	9.843s	3.279s
64	11.520s	5.681s
128	20.077s	20.677s
256	30.850s	71.105s
512	115.483s	351.046s

Table 5: Compilation times for Nvidia nvcc 7.0 and GNU g++ $4.8.2$