Heterogenous CPU-GPU ordinary differential equation solver

Dániel Nagy
Co-author: Dr. Ferenc Hegedűs

BME, Department of Hydrodynamic Systems

GPU Day 2024

MŰEGYETEM 1782

Department of Hydrodynamic Systems

Budapest University of Technology and Economics, Sonochemistry Research Group

Content

1 Introduction
- Motivation
- Per-thread approach

2 Methods
- CUDA Streams
- Overlapping
- Test equations

3 Results
- TEST1, Bernoulli
- TEST2, Duffing

4 Conclusion

Goal

Developing a heterogeneous CPU-GPU ordinary differential equation solver

- Using CUDA
- Utilizing both CPUs and GPUs for optimal performance
- Speeding-up the solution of parameter studies
- Possible application for high-complexity problems
  - Delay differential equations
  - Coupled differential equations

Parameter study

Systematic examination of the parameter dependent behavior of a dynamic system.

Per-thread approach for ODEs

Each thread calculates the same ODE[1]:

$$\dot{x} = f(x, t; p) \tag{1}$$

Can be solved numerically using the traditional 4th order Runge-Kutta method as

$$x_{n+1}^{\{i\}} = x_n^{\{i\}} + \frac{\Delta t}{6}\left(k_1^{\{i\}} + 2k_2^{\{i\}} + 2k_3^{\{i\}} + k_4^{\{i\}}\right) \tag{2}$$

$$k_1^{\{i\}} = f(x_n^{\{i\}}, t; p^{\{i\}}) \quad k_2^{\{i\}} = \dots \tag{3}$$

Notice

- i is the thread index, $i = 0 \dots N - 1$; N is the number of threads, n is the current time-step.
- The threads do not exchange data.

[1] Dániel Nagy, Lambert Plavecz, and Ferenc Hegedűs. "The art of solving a large number of non-stiff, low-dimensional ordinary differential equation systems on GPUs and CPUs". In: *Communications in Nonlinear Science and Numerical Simulation* 112 (2022), p. 106521.

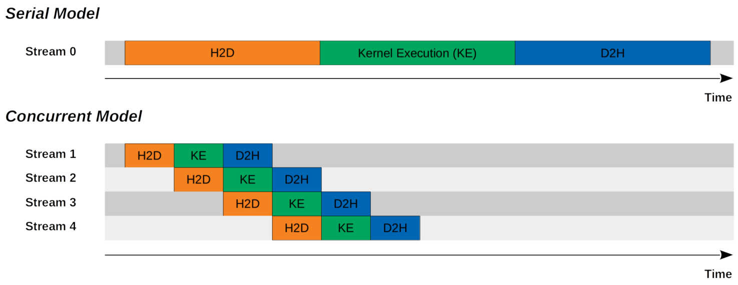
Limitations of the per-thread approach

- Thread divergence if a time-step is rejected (adaptive methods)
- Register-spills (e.g., solving the Keller-Miksis equation using RKCK45)
- Calculating delay terms (uncoalesced memory operations)
- Not enough register memory to store coupling matrices

Using CUDA streams

CUDA Stream

A sequence of operations that execute asynchronously on a CUDA device. It allows for concurrent execution of multiple tasks on the GPU, enabling parallel processing and overlapping of computation with data transfers.



Source: Lei Mao (`https://leimao.github.io/blog/CUDA-Stream/`)

Ideal overlap

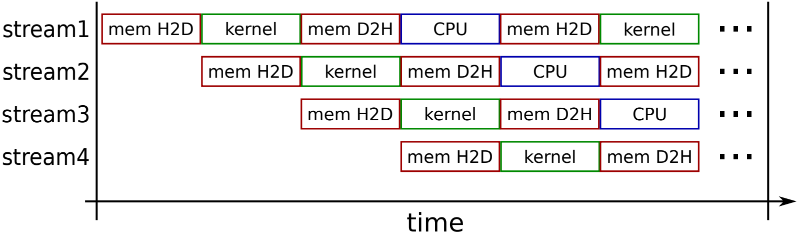
- each stream handles 1/4 of the workload

Ideal overlap

- each stream handles 1/4 of the workload
- each stream does the following algorithm
  1. copy x_n, t_n from CPU to GPU (mem H2D)
  2. kernel call: 1 Runge-Kutta step in parallel to determine x_{n+1}, t_{n+1}
  3. copy x_{n+1}, t_{n+1} from GPU to CPU (mem D2H)
  4. execute serial operations on the CPU on the data

Ideal overlap

- each stream handles 1/4 of the workload
- each stream does the following algorithm
  1. copy x_n, t_n from CPU to GPU (mem H2D)
  2. kernel call: 1 Runge-Kutta step in parallel to determine x_{n+1}, t_{n+1}
  3. copy x_{n+1}, t_{n+1} from GPU to CPU (mem D2H)
  4. execute serial operations on the CPU on the data
- each substep requires the same amount of time



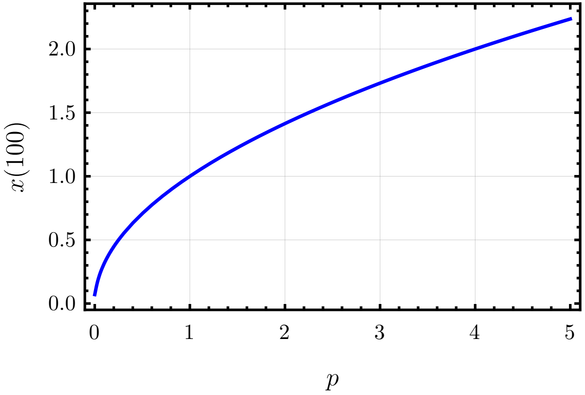
TEST1: Bernoulli type ODE

$$\dot{x}(t) = px - x^3; \quad x(0) = 1 \quad (4)$$

- Analytical solution exists, used for validating the results

$$x(t) = \frac{\sqrt{p}e^{pt}}{\sqrt{-1 + e^{2pt} + p}} \quad (5)$$

- Solution in $t \in [0,100]$ and $p \in [0,5]$
- $\Delta t = 0.1$, RK4 is used



End values at $t = 100$

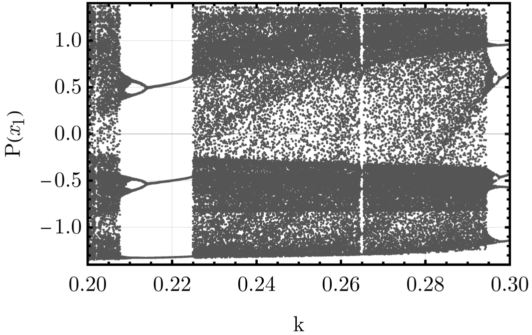
TEST2: Bernoulli type ODE

$$\dot{x}_1 = x_2 \tag{6}$$

$$\dot{x}_2 = x_1 - x_1^3 - kx_2 + 0.3\cos(t) \tag{7}$$

$$x_1(0) = -0.5 \quad x_2(0) = -0.1 \tag{8}$$

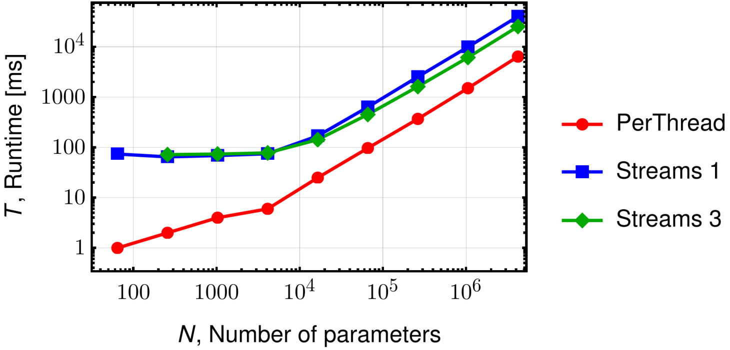
- $k \in [0.2, 0.3]$
- Transient simulation for $t \in [0, 1024 \cdot 2\pi]$
- 32 Poincare sections at $t = (1024 + i) \cdot 2\pi$, where $i = 1 \ldots 32$
- Adaptive 5th order Runge-Kutta Cash-Karp method using abs. and rel. tolerance 10^{-6}



Bifurcation diagram as a result

Results of TEST1 (Bernoulli)

- Pure GPU (PerThread) is the fastest
- Some overlapping is possible, using 3 streams is 1.6× faster



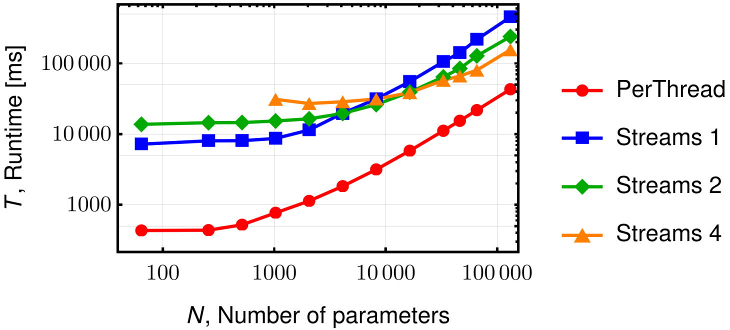
TEST1 Profiling

Problem:

- A single RK4 step requires much less time than the memory operations
- Streams are idle

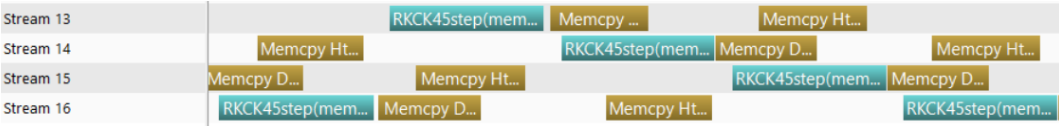
Results of TEST2 (Duffing)

- Pure GPU (PerThread) is the fastest
- Using 2 streams is 1.9× faster than 1 stream
- Using 4 streams is 2.9× faster than 1 stream



TEST2 Profiling

- CPU calculations are short (only checks if the end time is reached)
- Overlapping is possible and makes things faster



Conclusion

- Overlapping memory operations and CPU/GPU calculations is possible
- For the test problems slower than the per-thread approach
- Problem: **global memory operations are slow**

Future plans:

- Can be good if a lot of serial calculations are required.
- Usage for an **adaptive delay differential equation solver** where the delayed terms are found at arbitrary memory locations.

Thank you for the attention!

1 Introduction
- Motivation
- Per-thread approach

2 Methods
- CUDA Streams
- Overlapping
- Test equations

3 Results
- TEST1, Bernoulli
- TEST2, Duffing

4 Conclusion

References

Nagy, Dániel, Lambert Plavecz, and Ferenc Hegedűs. "The art of solving a large number of non-stiff, low-dimensional ordinary differential equation systems on GPUs and CPUs". In: *Communications in Nonlinear Science and Numerical Simulation* 112 (2022), p. 106521.