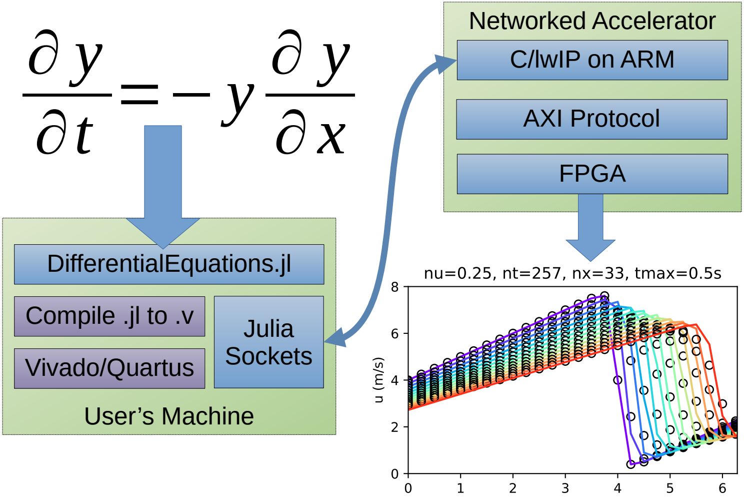
Progress on DDAs for Differential Equation Simulation

Jonathan García-Mallén, Shuohao Ping, Alex Miralles-Cordal, Ian Martin, Mukund Ramakrishnan, Yipeng Huang

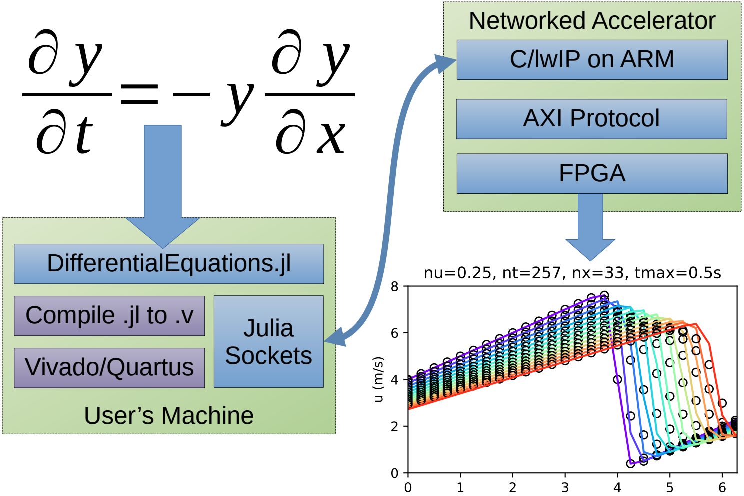
Rutgers University

2024 April 3

$$\frac{\partial y}{\partial t} = -y \frac{\partial y}{\partial x}$$



$$\frac{\partial y}{\partial t} = -y\frac{\partial y}{\partial x}$$



- Moore's law is dead

- Moore's law is dead
- Hardware acceleration permits better performance with fewer transistors, but are difficult to program.

- Moore's law is dead
- Hardware acceleration permits better performance with fewer transistors, but are difficult to program.
- Differential Equations are everywhere.

- Moore's law is dead
- Hardware acceleration permits better performance with fewer transistors, but are difficult to program.
- Differential Equations are everywhere.
- What is the best front-end for a researcher to interface?

- Moore's law is dead
- Hardware acceleration permits better performance with fewer transistors, but are difficult to program.
- Differential Equations are everywhere.
- What is the best front-end for a researcher to interface?
- Consider MATLAB, Python/SciPy, C++, and Julia

- Moore's law is dead
- Hardware acceleration permits better performance with fewer transistors, but are difficult to program.
- Differential Equations are everywhere.
- What is the best front-end for a researcher to interface?
- Consider MATLAB, Python/SciPy, C++, and Julia
- How naturally are DiffEq's expressed in the language?

- Moore's law is dead
- Hardware acceleration permits better performance with fewer transistors, but are difficult to program.
- Differential Equations are everywhere.
- What is the best front-end for a researcher to interface?
- Consider MATLAB, Python/SciPy, C++, and Julia
- How naturally are DiffEq's expressed in the language?
- How mature and accessible is the accelerator/gpu ecosystem? This is a proxy for ease of accelerator development for the language.

Expressing an ODE in Python

```
import numpy as np
from scipy.integrate import odeint
def sincos(y, t):

    u1, u2 = y
    du1 = u2
    du2 = -u1
    return [du1, du2]

init_cond = [0.0, 1.0]
tspan = np.linspace(0, 10, 101)
sol = odeint(sincos, init_cond, tspan)
```

Expressing an ODE in Julia

```
using DifferentialEquations, Plots
function sincos!(du, u, p, t)

    du[1] = u[2]
    du[2] = -u[1]

end
init_cond = [0.0; 1.0]
tspan = (0.0, 100.0)
prob = ODEProblem(sincos!, init_cond, tspan)
sol = solve(prob)
plot(sol, idxs = (0, 1))
```

Accelerating an ODE in Julia with a GPU

```
using DifferentialEquations, DiffEqGPU, CUDA
function sincos!(du, u, p, t)

    du[1] = u[2]
    du[2] = -u[1]

end
init_cond = [0.0; 1.0]
tspan = (0.0, 100.0)
prob = ODEProblem(sincos!, init_cond, tspan)
sol = solve(prob, GPUTsit5())
```

Accelerating an ODE in Julia with a GPU

```
using DifferentialEquations, DiffEqGPU, CUDA
function sincos!(du, u, p, t)

    du[1] = u[2]
    du[2] = -u[1]

end
init_cond = [0.0; 1.0]
tspan = (0.0, 100.0)
prob = ODEProblem(sincos!, init_cond, tspan)
sol = solve(prob, GPUTsit5())
```

- Julia is most natural for expressing math and also best for accelerator programming.

I'll take a detour away from Differential Equations now. I want to elucidate the significance of the convergence rate of a numerical method.

Round-off Error

1. Round-off error becomes problematic if the pivots are close to zero.

Round-off Error

1. Round-off error becomes problematic if the pivots are close to zero.
2. Round-off error can arise due to finite-precision arithmetic. `Float64(1/3) != Rational(1)/Rational(3)`. All of us here have probably heard of this before.

Round-off Error

1. Round-off error becomes problematic if the pivots are close to zero.
2. Round-off error can arise due to finite-precision arithmetic. `Float64(1/3) != Rational(1)/Rational(3)`. All of us here have probably heard of this before.
3. Truncation error is different.

Truncation Error: e^x

Consider e^x.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \dots$$

Truncation Error: e^x

Consider e^x.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}...$$

Unable to compute an infinite sum directly, we approximate by truncating everything after the third term:

$$e^x \approx 1 + x + \frac{x^2}{2!}$$

The truncation error is then

$$\frac{x^3}{3!} + \frac{x^4}{4!}...$$

Euler's Method

Given f and an initial y_0 and x_0,
find y where

$$\frac{dy}{dx} = f(x, y); y(x_0) = y_0$$

Euler's Method

Given f and an initial y_0 and x_0, find y where

$$\frac{dy}{dx} = f(x, y); y(x_0) = y_0$$

Pick a step size h:

$$y_{k+1} = y_k + h \cdot f(x_k, y_k) : k \geq 0$$

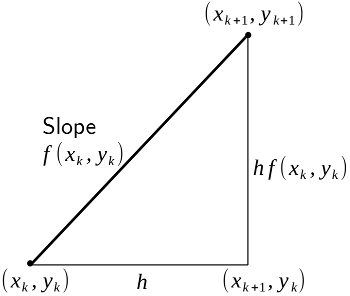
Euler's Method

Given f and an initial y_0 and x_0, find y where

$$\frac{dy}{dx} = f(x, y); y(x_0) = y_0$$

Pick a step size h:

$$y_{k+1} = y_k + h \cdot f(x_k, y_k) : k \geq 0$$



Euler's Method's Errors

- First order method for solving ODEs: $err \propto h$

Euler's Method's Errors

- First order method for solving ODEs: $err \propto h$
- More accurate with smaller h

Euler's Method's Errors

- First order method for solving ODEs: $err \propto h$
- More accurate with smaller h
- Worse round-off error with smaller h

Euler's Method's Errors

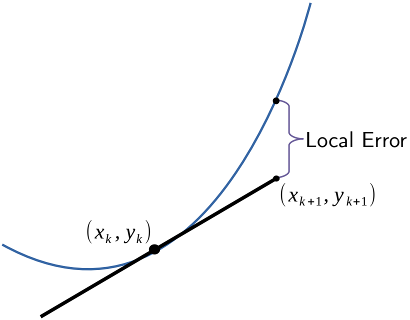
- First order method for solving ODEs: $err \propto h$
- More accurate with smaller h
- Worse round-off error with smaller h
- Local truncation error

Euler's Method's Errors

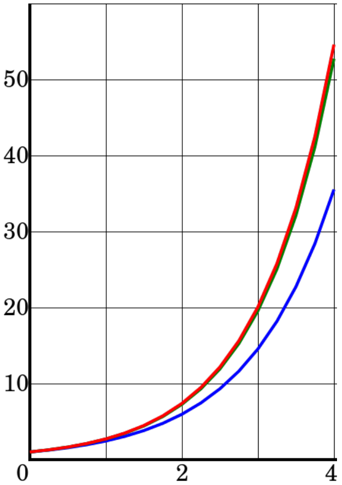
- First order method for solving ODEs: $err \propto h$
- More accurate with smaller h
- Worse round-off error with smaller h
- Local truncation error
- Gobal truncation error

Euler's Method's Local Truncation Error

$$
\begin{aligned}
y(x_0 + h) &\approx y(x_0) - hf(x_0, y(x_0)) \\
err &= |y(x_0 + h) - y(x_0) - hf(x_0, y(x_0))|
\end{aligned}
$$



Euler's Method's Global Truncation Error



- Red: $y = e^t$
- Blue: y approximated by Euler Method with $h = 0.25$

Euler's Method on an FPGA

$y_{k+1} = y_k + h \cdot f(x_k, y_k) : k \geq 0$

```
phase 1: [dz,r] <= r + dt*y;
phase 2: y <= y + dy;
```

Euler's Method on an FPGA

$y_{k+1} = y_k + h \cdot f(x_k, y_k) : k \geq 0$

```
phase 1: [dz,r] <= r + dt*y;
phase 2: y <= y + dy;
```

- y: signed register

Euler's Method on an FPGA

$y_{k+1} = y_k + h \cdot f(x_k, y_k) : k \geq 0$

```
phase 1: [dz,r] <= r + dt*y;
phase 2: y <= y + dy;
```

- `y`: signed register
- $h \rightarrow$`dt`

Euler's Method on an FPGA

$y_{k+1} = y_k + h \cdot f(x_k, y_k) : k \geq 0$

```
phase 1: [dz,r] <= r + dt*y;
phase 2: y <= y + dy;
```

- `y`: signed register
- $h \rightarrow$`dt` is a power of 2; multiplication is costly

Euler's Method on an FPGA

$y_{k+1} = y_k + h \cdot f(x_k, y_k) : k \geq 0$

```
phase 1: [dz,r] <= r + dt*y;
phase 2: y <= y + dy;
```

- `y`: signed register
- $h \rightarrow$`dt` is a power of 2; multiplication is costly
- This is a Digital Differential Analyzer

Euler's Method on an FPGA

$y_{k+1} = y_k + h \cdot f(x_k, y_k) : k \geq 0$

```
phase 1: [dz,r] <= r + dt*y;
phase 2: y <= y + dy;
```

- `y`: signed register
- $h \rightarrow$`dt` is a power of 2; multiplication is costly
- This is a Digital Differential Analyzer
- Don't accelerate first-order methods

.

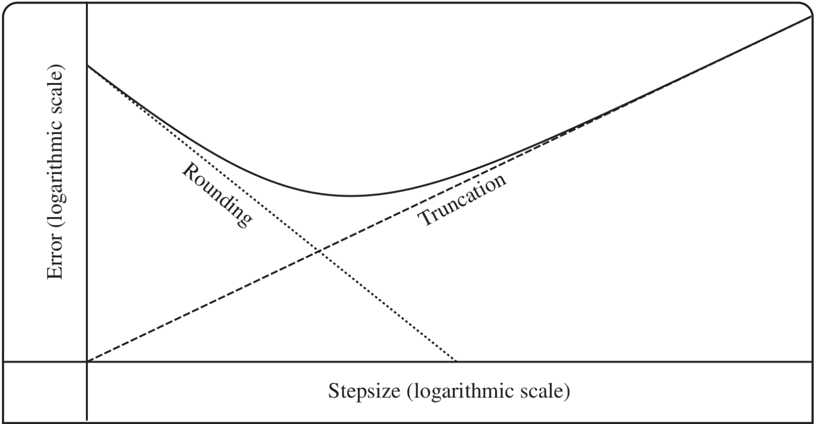
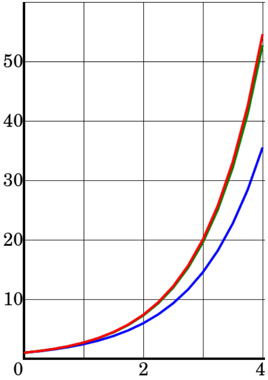
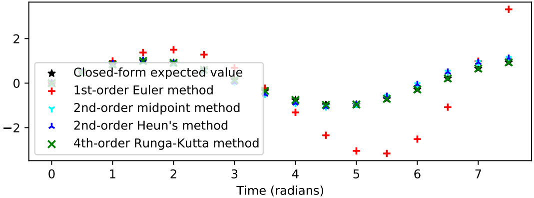
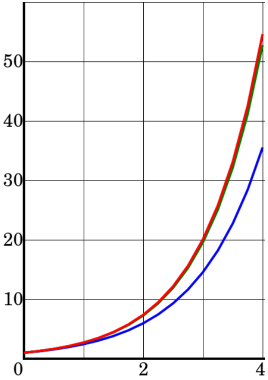
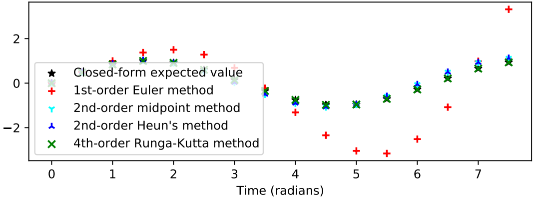


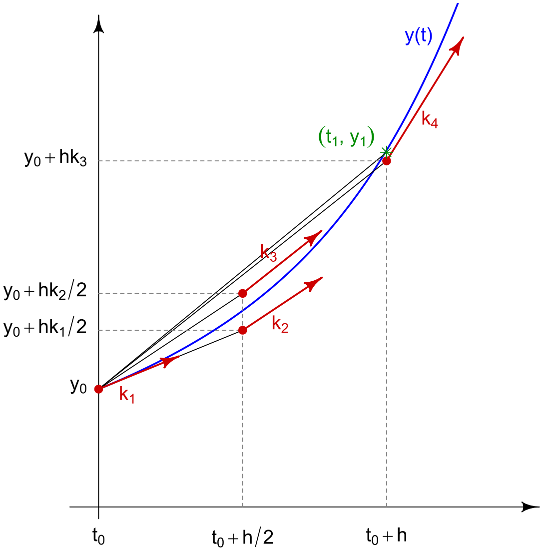
Figure 218(i) Schema showing effects of rounding error


50
40
30
20
10
0
2
4


Closed-form expected value
1st-order Euler method
2nd-order midpoint method
2nd-order Heun's method
4th-order Runga-Kutta method
Time (radians)


50
40
30
20
10
0
2
4


Closed-form expected value
1st-order Euler method
2nd-order midpoint method
2nd-order Heun's method
4th-order Runga-Kutta method
Time (radians)



Slopes used by 4th-order Runge-Kutta

```
phase 1: [dz,r] <= r + dt*y;
phase 2: y <= y + dy;
```

phase 1: [dz,r] <= r + dt*y;

phase 2: y <= y + dy;

phase i: $\mathtt{dy}_{i-1}$ <= dy;

$$\mathtt{[dz,r]{<=}r{+}dt{*}(y{+}}\sum_{j=1}^{i-2} a_{ij}\mathtt{{*}dy}_j\mathtt{+}a_{i,i-1}\mathtt{{*}dy);}$$

$$\texttt{phase 1: [dz,r] <= r + dt*y;}$$

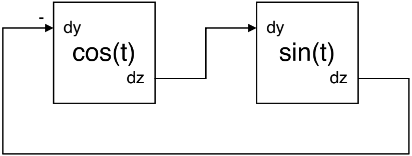
$$\texttt{phase 2: y <= y + dy;}$$

$$\texttt{phase i: dy}_{i-1} \texttt{ <= dy;}$$

$$\texttt{[dz,r]<=r+dt*(y+}\sum_{j=1}^{i-2} a_{ij}\texttt{*dy}_j\texttt{+}a_{i,i-1}\texttt{*dy);}$$

$$\texttt{phase s+1: y <= y + } \sum_{j=1}^{s-1} b_j\texttt{*dy}_j \texttt{ + } b_s\texttt{*dy;}$$

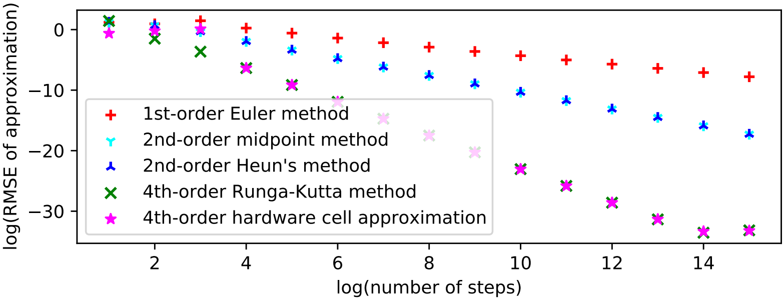
```
1: [dz,r] <= r + dt*y;
2: y <= y + dy;
```
i: $\mathtt{dy}_{i-1}$ <= dy;



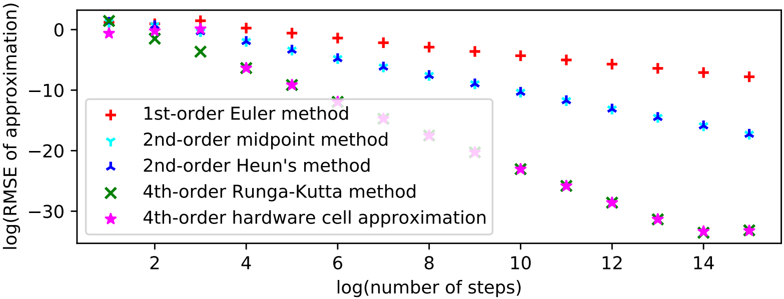
$$\mathtt{[dz,r]<=r+dt*(y+}\sum_{j=1}^{i-2} a_{ij}\mathtt{*dy}_j\mathtt{+}a_{i,i-1}\mathtt{*dy);}$$

$$\mathtt{s+1:\ y\ <=\ y\ +}\ \sum_{j=1}^{s-1} b_j\mathtt{*dy}_j\ \mathtt{+}\ b_s\mathtt{*dy;}$$

Accuracy of sine from Hardware



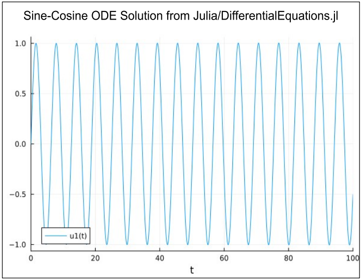
Accuracy of sine from Hardware

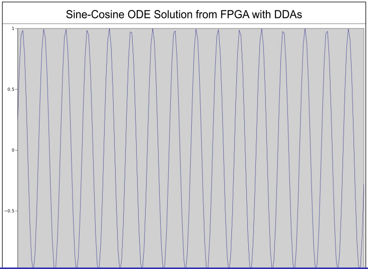


Timing Sine

Time to produce 16 periods

Hardware	Seconds
Intel i7-6600U	0.000165
AMD EPYC 7352	0.000367
FPGA (projected)	0.000016





Timing: What about Latency?

- Future metric.

Timing: What about Latency?

- Future metric.
- Amazon EC2 F1 Instances - "Enable faster FPGA accelerator development and deployment in the cloud"

Timing: What about Latency?

- Future metric.
- Amazon EC2 F1 Instances - “Enable faster FPGA accelerator development and deployment in the cloud”
- Intel Infrastructure Processing Unit
  - “FPGA-based and ASIC-based IPU platforms”
  - Already shipped to Google & other cloud service providers.

Power: Rough Measurements using Zybo 7Z (Zynq-7000)

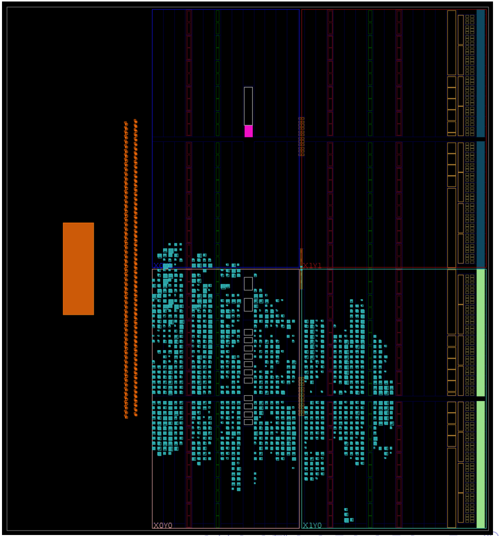
- FPGA was roughly 30$\times$ more efficient than the i7 CPU
  - 4.5 Watt max Zybo Z7 power draw
  - 15 Watt TDP as proxy for i7-6600U power draw
  - FPGA used $7.2e-5$ joules while the CPU used $2.55e-3$ joules to calculate sine

Power: Rough Measurements using Zybo 7Z (Zynq-7000)

- FPGA was roughly 30$\times$ more efficient than the i7 CPU
  - 4.5 Watt max Zybo Z7 power draw
  - 15 Watt TDP as proxy for i7-6600U power draw
  - FPGA used $7.2e-5$ joules while the CPU used $2.55e-3$ joules to calculate sine
- Rough, preliminary measurements

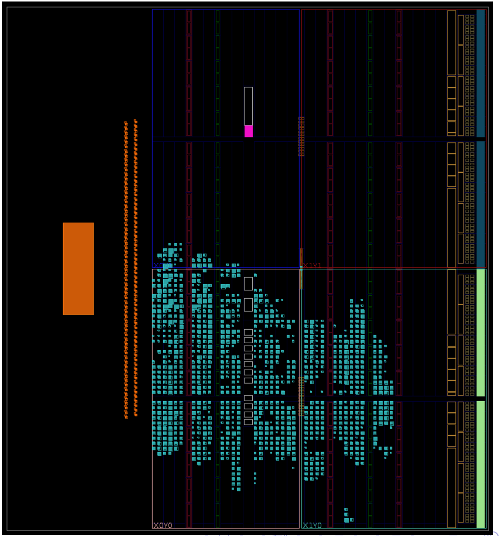
Area: Does it Scale?

- One thousand cells can be instantiated on a commodity FPGA.
- We have not yet tried multiplexing the cells
- Can still slice/batch the timespan



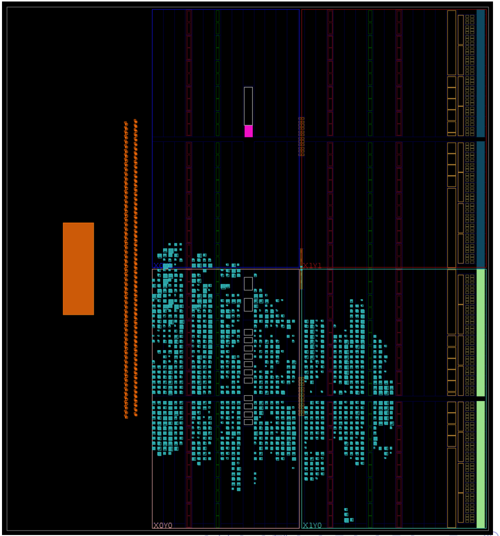
Area: Does it Scale?

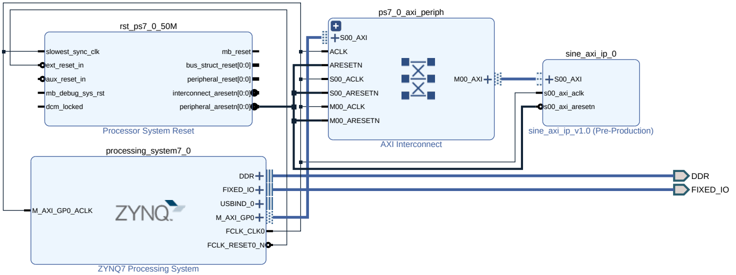
- One thousand cells can be instantiated on a commodity FPGA.
- We have not yet tried multiplexing the cells
- Can still slice/batch the timespan



Area: Does it Scale?

- One thousand cells can be instantiated on a commodity FPGA.
- We have not yet tried multiplexing the cells
- Can still slice/batch the timespan




rst_ps7_0_50M
slowest_sync_clk
ext_reset_in
aux_reset_in
mb_debug_sys_rst
dcm_locked
mb_reset
bus_struct_reset[0:0]
peripheral_reset[0:0]
interconnect_aresetn[0:0]
peripheral_aresetn[0:0]
Processor System Reset
ps7_0_axi_periph
S00_AXI
ACLK
ARESETN
S00_ACLK
S00_ARESETN
M00_ACLK
M00_ARESETN
M00_AXI
AXI Interconnect
sine_axi_ip_0
S00_AXI
s00_axi_aclk
s00_axi_aresetn
sine_axi_ip_v1.0 (Pre-Production)
processing_system7_0
M_AXI_GP0_ACLK
ZYNQ
DDR
FIXED_IO
USBIND_0
M_AXI_GP0
FCLK_CLK0
FCLK_RESET0_N
ZYNQ7 Processing System
DDR
FIXED_IO

Progress on DDAs for Differential Equation Simulation

Jonathan García-Mallén, Shuohao Ping, Alex Miralles-Cordal, Ian Martin, Mukund Ramakrishnan, Yipeng Huang

Rutgers University

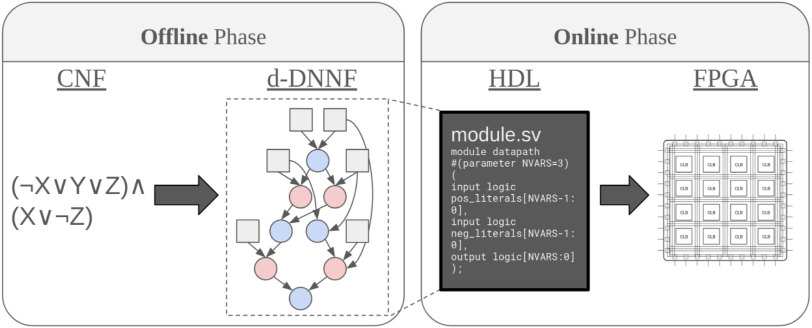
Questions?

FPGA-Accelerated Weighted Model Counting for Quantum Circuit Simulation

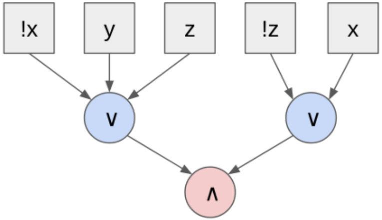
Mayank Barad, Neel Shejwalkar, Maria Xu, J. García-Mallén, Y. Huang

Rutgers University

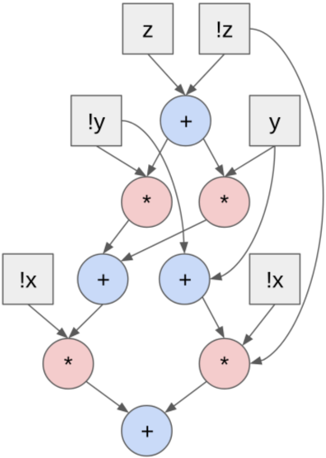
2024 April 3

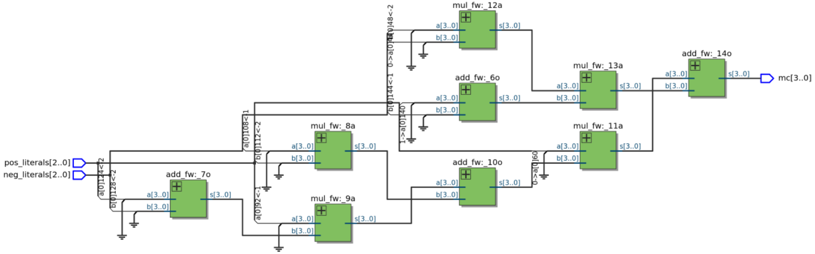

Offline Phase
CNF
d-DNNF
(¬X∨Y∨Z)∧
(X∨¬Z)
Online Phase
HDL
FPGA
module.sv
module datapath
#(parameter NVARS=3)
(
input logic
pos_literals[NVARS-1:
0],
input logic
neg_literals[NVARS-1:
0],
output logic[NVARS:0]
);

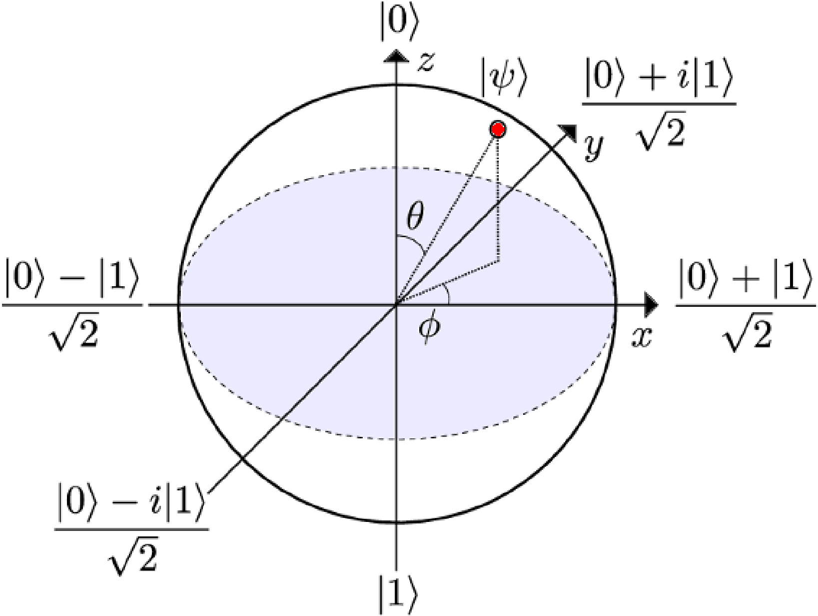
Flat



Nested




mul_fw:_12a
add_fw:_14o
mul_fw:_13a
add_fw:_6o
mul_fw:_8a
mul_fw:_11a
add_fw:_10o
add_fw:_7o
mul_fw:_9a
pos_literals[2..0]
neg_literals[2..0]
mc[3..0]
a[3..0]
b[3..0]
s[3..0]


$|0\rangle$
z
$|\psi\rangle$
$\frac{|0\rangle + i|1\rangle}{\sqrt{2}}$
y
θ
$\frac{|0\rangle - |1\rangle}{\sqrt{2}}$
$\frac{|0\rangle + |1\rangle}{\sqrt{2}}$
ϕ
x
$\frac{|0\rangle - i|1\rangle}{\sqrt{2}}$
$|1\rangle$

FPGA-Accelerated Weighted Model Counting for Quantum Circuit Simulation

Mayank Barad, Neel Shejwalkar, Maria Xu, J. García-Mallén, Y. Huang

Rutgers University

Thank You!