Schrodinger Equation Accelerator on PYNQ via HLS

Haifei Wang
Wuhan University, Hubei

OpenHW2022

AMD XILINX

On board test by PYNQ-Z2

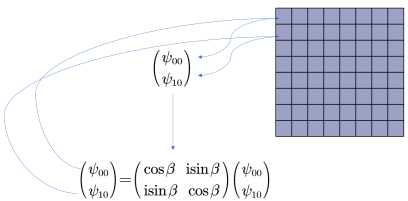
INTRODUCTION

Traditional numerical approaches for solving a PDE, such as the Runge-Kutta methods, are not suitable for quantum settings because they do not **numerically preserve the probability** (sum of the squared modulus of the wave function):

$$\int |\varphi|^2 dr$$

quantum mechanics should always be one.

To preserve the probability, we should turn to other algorithms, such as the Chebyshev algorithm and the Trotter-Suzuki algorithm. The accelerator is based on the **Trotter-Suzuki algorithm** [1], in which the transformation matrix from t=0 to t=Δt has a determinant of one, meaning that it is a proper rotation matrix and will not alter the squared modulus of the vector it is acting on.



Principle of Trotter-Suzuki algorithm

[1] De Raedt, Hans. "Computer simulation of quantum phenomena in nanoscale devices." Annual Reviews of Computational Physics IV (1996): 107-146.

This hardware design has employed several hardware acceleration techniques to reduce the **latency and the initiation interval** (II).

(1) Distributed ROM

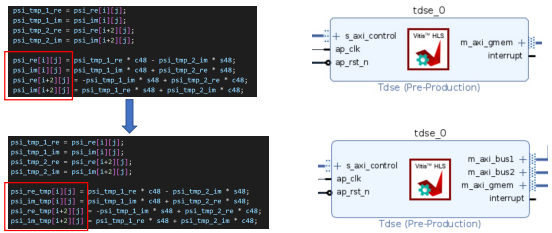
ROM is relatively easy to replicate the storage because it only reads. The Vitis HLS tool will perform this optimization for an array that is initialized at the beginning and never modified after. Wherever the array is needed, a ROM will be synthesized, reducing the path length.

(2) Removing self-dependency

Many algorithms have to accumulate the data, however, on the hardware level, the next data can get into accumulation only after the last accumulation is finished. It is possible **to avoid such self-dependency** and increase the throughput by **alternating the logic of the C++ code** (e.g., loop interchange).

(3) Adding read ports for BRAMs

It is possible to split one large BRAM into several smaller BRAMs, increasing the number of read ports to get more data in a single clock cycle.



Removing self-dependency (left) and adding read ports (right)

Also, it is useful to **bundle** different array arguments (with AXI4 Master interface) to different AXI4 ports if they are both accessed in one loop. This is because, by default, array arguments are mapped to a single interface bundle that only allows one read and one write in a clock cycle [2].

[2] ug1399 > HLS Programmers Guide > Interface of the HLS design > Defining Interfaces > AXI Adatper Interfaces Protocol > AXI4 Master Interface > M_AXI Bundles

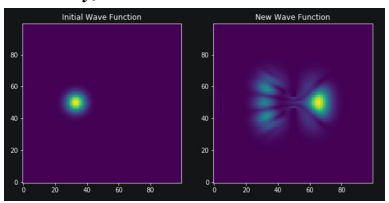
CREATIVE DESIGN

RESULT

The hardware implementation (1.84s) of a 2-D quantum wave packet passing through an aperture is **200+ times faster** than the software (524.4s) implementation. The hardware acceleration techniques have helped reduce the initiation interval to one clock cycle, meaning a **fully pipelined** dataflow. As shown below, it only takes about 100M clock cycles to perform 2000 evolutions.

Modules && Loops	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT
tdse	99154012	9.920E8	-	99154013	-	no	134	54	13184	16865
sin_or_cos_float_s	19	190.000	-	1	-	yes	0	6	2497	3002
copy_o_copy_i	10002	1.000E5	4	1	10000	yes	-	-	-	-
top	99144000	9.910E8	49572	-	2000	no	-	-	-	-
R11_o_R11_i	5014	5.014E4	16	1	5000	yes	-	-	-	-
R12_o_R12_i	4814	4.814E4	16	1	4800	yes	-	-	-	-
R21_o_R21_i	5013	5.013E4	15	1	5000	yes	-	-	-	-
R22_o_R22_i	4913	4.913E4	15	1	4900	yes	-	-	-	-
R31_o_R31_i	5012	5.012E4	14	1	5000	yes	-	-	-	-
R32_o_R32_i	4812	4.812E4	14	1	4800	yes	-	-	-	-
R41_o_R41_i	5011	5.011E4	13	1	5000	yes	-	-	-	-
R42_o_R42_i	4911	4.911E4	13	1	4900	yes	-	-	-	-
R5_o_R5_i	10053	1.010E5	55	1	10000	yes	-	-	-	-

Latency, interval and resource utilization



A 2-D quantum wave packet passing through an aperture