Quantum Computing/FPGA Activities in Perugia

Mirko Mariotti [1,2] Giulio Bianchini [1] Loriano Storchi [3,2] Giacomo Surace [2]
Daniele Spiga [2] Diego Ciangottini [2] Giuseppe Prudente [2]

[1]Dipartimento di Fisica e Geologia, Universitá degli Studi di Perugia

[2]INFN sezione di Perugia

[3]Dipartimento di Farmacia, Universitá degli Studi G. D'Annunzio

Quantum Computing Activities in Perugia

We started experimenting with quantum computing. Our main interested is using FPGA to simulate quantum computers.

The goal is to experiment with classical/quantum hybrid computing backed by the CPU/FPGA hardware.

The work plan goes on three main directions:

- Learning and experimenting with reference quantum tools. Activity 1
- Create a HLS based quantum simulator. Activity 2
- Create a BondMachine based quantum simulator. Activity 3

Quantum Computing

Activity 3

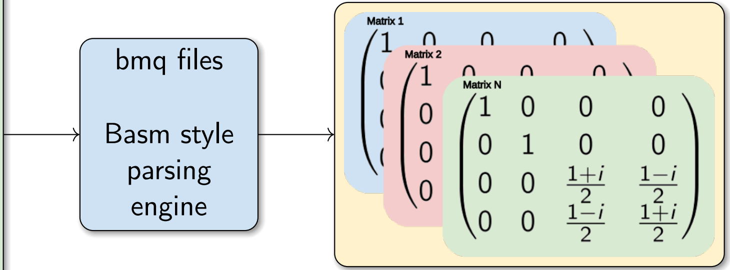
With all the capabilities of the BondMachine in terms of parallelism and speed, of customizability of the instruction set and the numerical precision, it is a natural question to ask whether the BondMachine could be used to simulate quantum computers.

A quantum computer simulator called bmqsim has been developed and is available within the BondMachine project.

Quantum Circuit

The first ingredient for bmqsim is a quantum circuit. The quantum circuit is a sequence of quantum gates represented by a sequence of matrices. the “program” is a .bmq file that contains code similar to the Qasm code.

```
%block code1 .sequential
       qbits    q0,q1
       zero     q0,q1
       h        q0
       cx       q0,q1
%endblock

%meta bmdef  main:code1
```



$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}$$

Indepedently of the backend, bmqsim translates the .bmq file into N matrices.

Backends

bmqsim may use different backends to operate. different backends create different hardware to simulate the same quantum circuit. Moreover, each backend may have different flavors to further fine-tune the HDL.

Sofware Simulation

Hardcoded matrices sequence

Loadable matrices sequence

Partially implemented

Full hardware deploy

Partially implemented

A command line option allows to choose the backend to use.

Backend: Software Simulation

In here, the quantum gates are simulated by the CPU. This is the slowest backend, but it useful for circuit design, debugging and testing. An example:

```
%%bash
cat program.bmq
```

✓ 0.0s

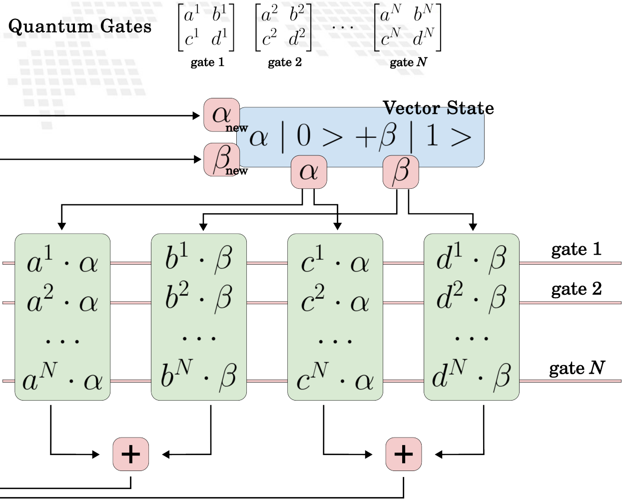
```
%block code1 .sequential
        qbits   q0,q1
        zero    q0,q1
        x       q0
        cx      q0,q1
%endblock

%meta bmdef global main:code1
```

```
%%bash
bmqsim -software-simulation -software-simulation-input inputs.json -software-simulation-output outputs.json program.bmq
```

Backend: Hardcoded matrices sequence

This backend creates a hardware that for each state of the quantum register, it applies the sequence of matrices.

For each matrix operation a dedicated processor is used. Within the processor, the matrix elements of all the gates are hardcoded.



Backend: Hardcoded matrices sequence

Pros and Cons

Pros:

- The matrices elements of the gates are already inside each processor. There no movement of big matrices.
- Fast

Cons:

- The circuit is fixed. to use a different circuit hardware has to be re-synthesized.
- Matrices are fully expanded. This may lead to a big hardware.
- Sparse matrices uses hardware anyway.

Backend: Loadable matrices sequence

Similar to the previous backend, but the matrices are loaded from the final application command line. This allows to change the matrices without recompiling the hardware.

To do so a small boot loader is needed on every processor. And a protocol to load the matrices elements from the final application.

Backend: Loadable matrices sequence

Pros and Cons

Pros:

- The matrices elements of the gates are already inside each processor. There no movement of big matrices.
- Fast
- The circuit is fixed, but a new circuit can be injected by the final application.

Cons:

- Matrices are fully expanded. This may lead to a big hardware.
- Sparse matrices uses hardware anyway.

Backend: Full hardware deploy

In this backend, the quantum circuit is synthesized in full hardware. Instead of having a state that is updated by each gate, only the relevant parts of the state are updated. Keeping track of the entalgment of the qubits and the sparce nature of the matrices.

Backend: Full hardware deploy

Pros and Cons

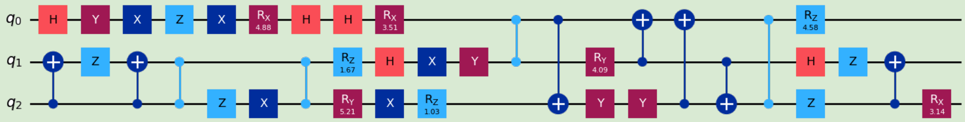
Pros:

- Fast
- Less resources used with respect to the previous backends

Cons:

- The circuit is fixed and cannot be changed.

Validation




```
Overall: Passed

Detailed results:
        pennylane: Passed
        qiskitrot: Passed
        quest: Passed
        bmqsim_hw: Passed
        bmqsim: Passed
        bmqsim_alveo: Passed
```

The validation is done by comparing the results of the simulation with the results of the same quantum circuit simulated by a well-known quantum simulator. Randomizing both the quantum circuit and the input state.

Validation

Activity 1

The validation tests are available in the bmqsimtests repository.
at the repository: `https://github.com/BondMachineHQ/bmqsimtests`

The README.md file contains the instructions to run the tests and describe the two layer directory structure of the tests.

The first layer is the quantum circuit to test. The second layer is the specific simulator to use.

Applications

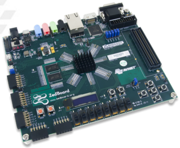
Alongside the FPGA hardware, bmqsim can create the end application that can be used to simulate quantum circuits.

Three types of applications are available:

- Jupiter Notebook using the PYNQ framework
- Standalone C application using pynq-api
- C++/OpenCL application

Boards

The boards that are supported by bmqsim are:



Xilinx Zynq-7000 SoC (ZedBoard)

Alveo U55C

Future Work

- Implement the full hardware deploy backend
- Implement the loadable matrices sequence backend
- Addressing routing problems over 4 qubits
- Power consumption analysis
- Numerical precision change analysis

HLS Simulation

Activity 2

With the things learned from the Activity 1 and the Activity 3, we are going to implement a HLS based quantum simulator.