testing.py

```python
import unittest

from bounded_queue import BoundedQueue


class BoundedQueueTest(unittest.TestCase):

    def test_import(self):
        # init
        queue = BoundedQueue(name="test",
                             maxsize=5)
        # init_queue
        collection = [1.0, 2.0, 3.0, 4.0, 5.0]
        queue.import_data(collection)
        # check size
        self.assertEqual(queue.size(), len(collection))
        # check if data added in the right order
        self.assertEqual(queue.try_peek_last(), collection[len(collection) - 1])
        # check exception for overfilling queue
        self.assertRaises(RuntimeError, queue.import_data, 6 * [1.0])

    def test_enq_deq(self):
        # init
        queue = BoundedQueue(name="test",
                             maxsize=1,
                             collection=[1.0])
        # check size
        self.assertEqual(queue.size(), 1)
        # empty queue, check element value
        self.assertEqual(queue.dequeue(), 1.0)
        # check size
        self.assertEqual(queue.size(), 0)
        # check size
        self.assertTrue(queue.is_empty())
        # check exception on underflow
        self.assertRaises(RuntimeError, queue.dequeue)
        # enqueue element
        queue.enqueue(1.0)
        # check size
        self.assertTrue(queue.is_full())
        # check exception on overflow
        self.assertRaises(RuntimeError, queue.enqueue, 2.0)

    def test_try_enq_deq(self):
        # init
        queue = BoundedQueue(name="test",
                             maxsize=1,
                             collection=[1.0])
        # check size
        self.assertEqual(queue.size(), 1)
        # empty queue, check element value
        self.assertEqual(queue.try_dequeue(), 1.0)
        # check size
        self.assertEqual(queue.size(), 0)
        # check size
        self.assertTrue(queue.is_empty())
        # dequeue from emtpy queue, check return value
        self.assertFalse(queue.try_dequeue())
        # enqueue, into non-full list, check return value
        self.assertTrue(queue.try_enqueue(1.0))
        # check size
        self.assertTrue(queue.is_full())
        # enqueue into full queue, check return value
        self.assertFalse(queue.try_enqueue(1.0), 2.0)

    def test_peek(self):
        # init
        queue = BoundedQueue(name="test",
                             maxsize=2,
                             collection=[1.0, 2.0])
        # check value at index 0
        self.assertEqual(queue.peek(0), 1.0)
        # check value at index 1
        self.assertEqual(queue.peek(1), 2.0)
        # check value at last location
        self.assertEqual(queue.try_peek_last(), 2.0)
        # empty queue
        queue.dequeue()
        queue.dequeue()
        # peek on empty queue, check return value
        self.assertFalse(queue.try_peek_last())


from calculator import Calculator
from numpy import cos


class CalculatorTest(unittest.TestCase):

    def test_calc(self):
        # init vars
        variables = dict()
        variables["a"] = 7.0
        variables["b"] = 2.0
        # init calc
        computation = "cos(a + b) if (a > b) else (a + 5) * b"
        calculator = Calculator()
        # do manual calculation and compare result
        result = cos(variables["a"] + variables["b"]) if (variables["a"] > variables["b"]) else (variables["a"] + 5) * \
                                                                                                    variables["b"]
        self.assertEqual(calculator.eval_expr(variables, computation), result)


class RunProgramTest(unittest.TestCase):

    def test(self):
        pass  # not a general test case, since dace and intel fgpa opencl sdk has to be installed and configured


import helper
import numpy as np


class HelperTest(unittest.TestCase):

    def test(self):
        # check max_dict_entry_key
        self.assertEqual(helper.max_dict_entry_key({"a": [1, 0, 0], "b": [0, 1, 0], "c": [0, 0, 1]}), "a")
        # check list_add_cwise
        self.assertEqual(helper.list_add_cwise([1, 2, 3], [3, 2, 1]), [4, 4, 4])
        # check list_subtract_cwise
        self.assertEqual(helper.list_subtract_cwise([1, 2, 3], [1, 2, 3]), [0, 0, 0])
        # check dim_to_abs_val
        self.assertEqual(helper.dim_to_abs_val([3, 2, 1], [10, 10, 10]), 321)
        # check convert_3d_to_1d
        self.assertEqual(helper.convert_3d_to_1d([10, 10, 10], [3, 2, 1]), 321)
        # check load_array
        self.assertListEqual(list(helper.load_array({"data": "testing/helper_test.csv", "data_type": "float64"})),
                             [7.0, 7.0])
        self.assertListEqual(list(helper.load_array({"data": "testing/helper_test.dat", "data_type": "float64"})),
                             [7.0, 7.0])
        # check save_array / load_array
        out_data = np.array([1.0, 2.0, 3.0])
        file = {"data": "test.dat", "data_type": "float64"}
        helper.save_array(out_data, file["data"])
        in_data = helper.load_array(file)
        self.assertTrue(helper.arrays_are_equal(out_data, in_data))
        os.remove(file["data"])
        # check unique
        not_unique = [1.0, 2.0, 1.0]
        self.assertListEqual(sorted(helper.unique(not_unique)), [1.0, 2.0])


from compute_graph import ComputeGraph
import json
import os
```

testing.py

```
class ComputeGraphTest(unittest.TestCase):

    def test(self):
        # define example computation
        computation = "out = cos(3.14);res = A[i,j,k] if (A[i,j,k]+1 > A[i,j,k]-B[i,j,k]) else out"
        # instantiate ComputeGraph and generate internal state
        graph = ComputeGraph()
        graph.generate_graph(computation)
        graph.calculate_latency()
        # load operation latency manually to compare result
        with open('compute_graph.config') as json_file:
            op_latency = json.load(json_file)
        # check if latencies match
        self.assertEqual(op_latency["op_latency"]["cos"] + op_latency["op_latency"]["add"] + 1, graph.max_latency)
        # save plot
        filename = "compute_graph_unittest.png"
        graph.plot_graph(filename)  # write graph to file
        # delete plot
        os.remove(filename)


from kernel import Kernel
import dace.types


class KernelTest(unittest.TestCase):

    def test(self):
        # define global problem size
        dimensions = [100, 100, 100]
        # instantiate example kernel
        kernel = Kernel(name="dummy",
                        kernel_string="SUBST = a[i,j,k] + a[i,j,k-1] + a[i,j-1,k] + a[i-1,j,k]; res = SUBST + a[i,j,k]",
                        dimensions=dimensions,
                        data_type=dace.types.float64,
                        boundary_conditions={"a": {"type": "constant", "value": 1.0}})
        # check if the string matches
        self.assertEqual(kernel.generate_relative_access_kernel_string(),
                         "SUBST = (((a[0] + a[-1]) + a[-100]) + a[-10000]); res = (SUBST + a[0])")


from kernel_chain_graph import KernelChainGraph


class KernelChainGraphTest(unittest.TestCase):

    def test(self):
        chain = KernelChainGraph(path='stencils/simple_input_delay_buf.json', plot_graph=False)
        # Note: Since e.g. the delay buffer sizes get tested using different cases (e.g. through the simulator), we only
        # add a basic (no exception) case in here for the moment.


from optimizer import Optimizer


class OptimizerTest(unittest.TestCase):

    def test(self):
        # instantiate example KernelChainGraph
        chain = KernelChainGraph(path='stencils/simple_input_delay_buf.json', plot_graph=False)
        # instantiate the Optimizer
        opt = Optimizer(chain.kernel_nodes, chain.dimensions)
        # define bounds
        com_bound = 10000
        fast_mem_bound = 1000
        slow_mem_bound = 100000
        ratio = 0.5
        # run all optimization strategies
        opt.minimize_fast_mem(communication_volume_bound=com_bound)
        opt.minimize_comm_vol(fast_memory_bound=fast_mem_bound, slow_memory_bound=slow_mem_bound)
        opt.optimize_to_ratio(ratio=ratio)


from simulator import Simulator


class SimulatorTest(unittest.TestCase):

    def test(self):
        # set up all sample configs with their (paper) result
        samples = {
            "sample1": {
                "file": "stencils/simulator.json",
                "res": [5.14, 4.14, 5.14, 11.14, 7.14, 8.14]
            },
            "sample2": {
                "file": "stencils/simulator2.json",
                "res": [3., 4., 3., 4., 5., 4., 3., 4., 3.]
            },
            "sample3": {
                "file": "stencils/simulator3.json",
                "res": [4., 7., 8., 13., 20., 19., 16., 25., 20.]
            },
            "sample4": {
                "file": "stencils/simulator4.json",
                "res": [3., 3., 3., 3., 3., 3., 3., 3., 3.]
            },
        "sample5": {
            "file": "stencils/simulator5.json",
            "res": [7., 9., 7., 9., 11., 9., 7., 9., 7.]
        },
        "sample6": {
            "file": "stencils/simulator6.json",
            "res": [14., 18., 14., 18., 22., 18., 14., 18., 14.]
        },
        "sample7": {
            "file": "stencils/simulator7.json",
            "res": [20.25, 20.25, 19.25, 20.25, 20.25, 19.25, 16.25, 16.25, 16.25]
        },
        "sample8": {
            "file": "stencils/simulator8.json",
            "res": [4., 8., 12., 16., 20., 24., 28., 32., 36.]
        },
        "sample9": {
            "file": "stencils/simulator9.json",
            "res": [3., 5., 7., 9., 11., 13.]
        },
        "sample10": {
            "file": "stencils/simulator10.json",
            "res": [1., 6., 11., 16., 21., 26.]
        },
        "sample11": {
            "file": "stencils/simulator11.json",
            "res": [4., 2., 3., 10., 5., 6., 16., 8., 9.]
        },
        "sample12": {
            "file": "stencils/simulator12.json",
            "res": [20.25, 20.25, 19.25, 20.25, 20.25, 19.25, 16.25, 16.25, 16.25, 20.25, 20.25, 19.25, 20.25,
                    20.25,
                    19.25, 16.25, 16.25, 16.25, 20.25, 20.25, 19.25, 20.25, 20.25, 19.25, 16.25, 16.25, 16.25]
        }
    }
    # run all samples
    for sample in samples:
        chain = KernelChainGraph(path=samples[sample]['file'], plot_graph=False)
        sim = Simulator(input_nodes=chain.input_nodes,
                        input_config=chain.inputs,
                        kernel_nodes=chain.kernel_nodes,
                        output_nodes=chain.output_nodes,
                        dimensions=chain.dimensions,
                        input_config_name="test",
                        write_output=False,
                        verbose=False)
        sim.simulate()
        # check if result matches
        self.assertTrue(helper.arrays_are_equal(np.array(samples[sample]['res']), sim.get_result()['res'], 0.01))
```

```
if __name__ == '__main__':
    """
        Run all unit tests.
    """
    unittest.main()
```