Parallel Architecture, Assignment 2

s1140740

The purpose of this report is to outline implementation details of a cache coherence simulator, argue its validity and discuss results of experiments that were performed by the simulator.

1 Implementation Details

All parts are completed fully. Both MSI and MESI protocols are supported. The cache coherence simulator is in a Python script `cache.py`.

1.1 Command Line Options

There are several command line options available to the Python script:

```
$ python3 cache.py [-h] [--lines LINES] [--words WORDS] [--mesi MESI] tracefile
```

1.2 Data Structures

Instruction Describes an operation that is executed (read or write), an identifier of the CPU executing the instruction, and an address of the word that is being used in the operation.

Line Describes a single line of cache. Contains the tag and index of the line.

Event Describes an event (hit or miss) that occurs after a certain CPU requests a line from a cache.

States are represented by a string with one capital letter and can be one of `M`, `E`, `S`, `I`, or possibly `None` if the line is not present in the cache. An operation is also a string with one capital letter can be either `R` (read) or `W` (write). Finally, an event can be either a `miss` or a `hit`.

1.3 Transitions

Transitions are represented as a map of triplets (current state, operation and event) to a new state. For example, the following entry `('M', 'R', 'miss'): 'S'` describes the following transition rule:

> If the line is in state `M(odified)` and a CPU performs a `R(ead)` and gets a `miss` from the cache, transition the line state to `S(hared)`.

Some transitions, could result in multiple states depending on states of other caches. For example, the following entry `('I', 'R', 'miss'): lambda *args: 'E' if exclusive(*args) else 'S'` describes the following transition rule:

If the line is in state `I(nvalid)` and a CPU performs a `R(ead)` and gets a `miss` from the cache, transition to `E(xclusive)` if no other cache has access to the line or to `S(hared)` if the cache line is present in caches of other CPUs.

There are four such maps altogether: local and remote transitions for both protocols MSI and MESI.

1.4 Process

1. After a line is read from the trace file it is parsed and converted into an `Instruction` object.
2. A `Line` object is created from the address. This gives us access to the tag and index associated with an address.
3. Next, a cache lookup is performed and results in an `Event` object.
4. Metrics are calculated and the cache states are logged into standard output if logging is enabled.
5. Finally state transitions are performed for the line in the local cache as well as other caches that contain the line. The final state is extracted from transition rules described in chapter 1.2.

2 Validity of the Simulator

There are unit tests in file `test.py`.