Deriving simulations and comparing results

Be on the lookout for this fellow: The call-outs are ACTIONs for you to do!

When you see the check mark, compare your work to the marked element
Objectives

In this exercise you will learn how to

- Derive new ideas from other people’s work
  - Taking the role of a experimentalist/developer
  - How to clone an OCCAM object

- Run explorative experiments in OCCAM
  - Cache simulators

- Share your results and compare them
  - Comparing experiments
Outline

1 Deriving Experiments
   ◦ Cloning the XSim simulator
   ◦ Adding cache to the simulator

2 Explore cache configurations
   ◦ Running experiments with multiple configurations
   ◦ Testing different cache configurations

3 Sharing and comparing results
   ◦ Sharing experimental results with a partner
   ◦ Compare your results
① Deriving a simulator
Imagine you found XSim in a paper

- You ran some experiments (OCCAM ex.1)
- And thought:

What if their system included some cache?

Adding cache to the simulation

- Cache improves execution time
  - Providing faster access
  - Storing memory information likely to be used
With OCCAM and SST it’s easy to it

- OCCAM provides the tools to reuse simulators
Deriving experiments

With OCCAM and SST it’s easy to it

- **OCCAM provides the tools to reuse simulators**
  - Build on others work
  - Avoid re-implementation
  - Save time to produce new knowledge

- **SST provides a complete set of simulators**
  - With a small modification we can add cache
We need to make the following 3 steps:

1. **Clone the original simulator**
2. **Modify the configuration and output schema**
3. **Modify the SST simulation to include cache**
Deriving experiments

Step 1 — cloning the simulator

First lets go into the folder for this exercise

(a) From the home directory go into the exercises folder

```
user@tutorial ~ $ cd exercises
user@tutorial ~/exercises $ cd ex3
user@tutorial ~/exercises/ex3 $ 
```

(b) And go to the ex3 folder
① Deriving experiments

Step 1 — cloning the simulator

In order to clone we need the id of the original

(a) In the search bar search for the original simulator: XSim-demo

(b) Select it from the results list
Deriving experiments

Step 1 — cloning the simulator

In order to clone we need the id of the original

(a) In the Metadata tab

(b) Copy the object’s id
Deriving experiments

Step 1 – cloning the simulator

Create a clone of an OCCAM object

- **Syntax**: occam clone <id>
- OCCAM clone creates a copy of an object
  - This copy will have a new id (unique)
  - Creates a copy on the local directory
Create a clone of an OCCAM object

- Syntax: `occam clone <id>`
- OCCAM clone creates a copy of an object
  - This copy will have a new id (unique)
  - Creates a copy on the local directory

(a) Clone the XSim-demo simulator
1 Deriving experiments

Step 1 – cloning the simulator

The cloned object keeps the name

- To avoid confusion let’s change it

(a) Rename the folder with the cloned object to avoid confusion

```
user@tutorial ~/exercises/ex3 $ mv simulator-XSim-demo simulator-XSim-Cache
user@tutorial ~/exercises/ex3 $ cd simulator-XSim-Cache
```

(b) Go into that new folder
1 Deriving experiments

Step 1 – cloning the simulator

The cloned object keeps the name

- To avoid confusion let’s change it

(a) Open the object.json file

```
user@tutorial ~/exercises/ex3/simulator-XSim-Cache (master) $ vim object.json
```
Deriving experiments

Step 1 – cloning the simulator

The cloned object keeps the name

- To avoid confusion let’s change it

```
"name": "XSim-demo",

"name": "XSim-Cache",
```

(a) Find the name of the cloned simulator

(b) Change it to XSim-Cache
Deriving experiments

Step 1 – cloning the simulator

Still in the object.json file note that

- The cloned object’s link to the original
  - Preserving the provenance
- The object has a new id

```
"clonedFrom": {
  "id": "fdfebe32-875c-11e6-9e73-1c1b0d0a9044",
  "revision": "f711e035dd87da0b8c30dc544e8183d13f4b8b6c",
  "name": "Xsim-demo",
  "type": "simulator"
},
```

```
"id": "6e270712-56da-11e7-8885-080027de086d",
```

① Deriving experiments

Step 2 – Adding cache to schema

Adding cache configuration options:

- We need to modify the configuration schema
- And modify the output schema
Deriving experiments

Step 2 – Adding cache to schema

Adding cache configuration options:

- We need to modify the configuration schema
- And modify the output schema

Don’t worry we provide pre-prepared files in:
~/exercises/ex3/auxiliary_materials
Deriving experiments

Step 2 – Adding cache to schema

Modifying the configuration schema

(a) From the cloned simulator folder

user@tutorial ~$ cp ../auxiliary_materials/simulation_schema.json.
cp: overwrite './simulation_schema.json'? y
user@tutorial ~$ 

(b) Copy the pre-prepared file and overwrite the local file

Pre-prepared file location: ../auxiliary_materials/simulation_schema.json
1 Deriving experiments

Step 2 – Adding cache to schema

Inspect the new configuration options

(a) Open the configuration schema file

user@tutorial ~/exercises/ex3/simulator-XSim-Cache (master) $ vim simulation_schema.json
Deriving experiments

Step 2 – Adding cache to schema

Inspect the new configuration options

```
"l1cache":{
  "label":"L1 Cache options",
  "cache_frequency":{
    "label":"L1 cache frequency",
    "type":"string",
    "default":"2MHz"
  },
  "cache_size":{
    "label":"L1 cache size",
    "type":"string",
    "default":"8KiB"
  }
},
```

continues ...

The new configuration options are under “l1cache” and “l2cache”

This is a small subset of the SST cache options, you can check the complete set of options by running `sst-info memHierarchy.Cache`
Deriving experiments

Step 2 – Adding cache to schema

Inspect the new configuration options continued ...

```
"cache_line_size": {
   "label": "L1 block size",
   "type": "int",
   "default": 64
 },

"associativity": {
   "label": "L1 associativity",
   "type": "int",
   "default": 8
 },

"access_latency_cycles": {
   "label": "L1 access latency cycles",
   "type": "int",
   "default": 4
 }
```

Deriving experiments

Step 2 – Adding cache to schema

Inspect the new configuration options

```
"l2cache":{
  "label":"L2 Cache options",
  "cache_frequency":{
    "label":"L2 cache frequency",
    "type":"string",
    "default":"2MHz"
  },
  "cache_size":{
    "label":"L2 cache size",
    "type":"string",
    "default":"64KiB"
  }
}
```

continues ...
1. Deriving experiments

Step 2 – Adding cache to schema

Inspect the new configuration options

continued ...

"associativity":{
   "label":"L2 associativity",
   "type":"int",
   "default":16
},
"access_latency_cycles":{
   "label":"L2 access latency cycles",
   "type":"int",
   "default":6
}
**Deriving experiments**

*Step 2 – Adding cache to schema*

Modifying the output schema

(a) From the cloned simulator folder

(b) Copy the pre-prepared file and overwrite the local file

Pre-prepared file location:

`./auxiliary_materials/output_schema.json`
1 Deriving experiments

Step 2 – Adding cache to schema

Inspect the new outputs

(a) Open the output schema file

user@tutorial ~/exercises/ex3/simulator-XSim-Cache (master*) $ vim output_schema.json
Deriving experiments

Step 2 – Adding cache to schema

Inspect the new output schema

... l1_cache_configuration":{
  "cache_size":{"type":"string"},
  "access_latency_cycles":{"type":"string"},
  "cache_line_size":{"type":"string"},
  "cache_frequency":{"type":"string"},
  "associativity":{"type":"string"}
},
  "cache_misses":{"type":"string"},
...
① Deriving experiments

Step 3 – Modify the sst simulation

Modify the simulation to include cache
Deriving experiments

Step 3 – Modify the sst simulation

Modify the simulation to include cache

We just added new configurations, so we need to make sure the simulator parses them.
Deriving experiments

Step 3 – Modify the sst simulation

(a) In the cloned simulator folder

(b) Open the sim_parser.py file
Modify the simulation parser

```python
# Uncomment this
#
#def has_cache(self):
#  with open(self.args.sim_config[0], 'r') as f:
#    simulation_options = json.load(f)
#    l1_config = simulation_options.get("l1cache",{})
...
```

(a) Search for the line with “Uncomment this”

(b) Uncomment the subsequent lines (remove the ‘#’ character)

This code parses the new configuration options
Deriving experiments

Step 3 – Modify the sst simulation

Modify the simulation parser
  • Introduces 3 new functions

def has_cache(self):
  ...
  ...
def get_l1_params(self):
  ...
  ...
def get_l2_params(self):
  ...
  ...

Checks if there is cache in the configurations file

Parses the L1 cache parameters

Parses the L2 cache parameters
Now we can modify the simulation, and use those functions to configure the cache.
Deriving experiments

*Step 3 – Modify the sst simulation*

Modify the sst simulation to include cache

(a) In the cloned simulator folder

(b) Open the simulation.py file
Deriving experiments

*Step 3 – Modify the sst simulation*

Modify the sst simulation to include cache

```python
# Uncomment this
if(arguments.has_cache()):
    l1cache = sst.Component("l1cache","memHierarchy.Cache")
    l1cache.addParams(arguments.get_l1_params())
...
# Comment this
```

(a) Search for the line with “Uncomment this”

(b) Uncomment the subsequent lines (remove the ‘#’ character)

(c) Until you find this line
Deriving experiments

Step 3 – Modify the sst simulation

Modify the sst simulation to include cache

(a) When you reach the line with “Comment this”

```python
# Comment this
#cpu_memory_link = sst.Link("cpu_data_memory_link")
#cpu_memory_link.connect(
#    (cpu,"data_memory_link",arguments.get_link_latency()),
#    (memory,"direct_link",arguments.get_link_latency())
```

(b) Comment the subsequent lines (add a ‘#’ character)
Deriving experiments

Step 3 – Modify the sst simulation

Modify the sst simulation to include cache

import sim_parser
arguments = sim_parser.parse()
...
if(arguments.has_cache()):
    ...
else:
    ...

The file begins the same as the one you explored in OCCAM Ex. 1

However, if the configuration schema contains cache, then the cache is connected between the CPU and memory

Else, only memory is connected to the CPU
Deriving experiments

Step 3 – Modify the sst simulation

(a) Add files to the git repository:

```
git add object.json simulation_schema.json simulation.py sim_parser.py output_schema.json
```

(b) Commit the changes to git:

```
git commit -am "Added cache to the simulation"
```

(c) Commit the changes to OCCAM:

```
occam commit
```
1 Deriving experiments

Step 3 – Modify the sst simulation

(a) Build the simulator

user@tutorial ~/exercises/ex3/simulator-XSim-Cache (master) $ occam build
Deriving experiments

Step 3 – Modify the sst simulation

And we have a simulator with cache!
② Explore cache configurations
② Explore cache configurations

Create a new workset

Go to the web interface and create a workset

(a) Click on your username to go back to the Workset selection page

(b) Create a new Workset named Exercise 3
The Great Cache Challenge of 2017
② Explore cache configurations

The challenge

Find the cache configuration that makes a mystery program run the fastest

① Import a mystery program into OCCAM
   ◦ Check the link in your handouts

② Create an experiment with the new simulator

③ Test multiple cache configurations
② Explore cache configurations

The experiment

(a) Create the following experiment

Don’t include the plotting tool

Make sure you add the simulator with Cache

It should contain the mystery program you imported
2) **Explore cache configurations**

**Configuring the experiment**

(a) **Give** the experiment a name that differentiates your results!! (e.g. append your name in the end)

You will be sharing these results! So give them a distinct name.
2. Explore cache configurations

Configuring the experiment

Test different cache configurations

A simple trick would be making the cache huge! Play by the rules and keep the default value 😊

L1 Cache options

- L1 Cache frequency [MHz]: 2MHz
- L1 cache size [KiB]: 8KiB
- L1 block size []: 64
- L1 associativity [-]: 8
- L1 access latency [cycles]: 4

Change only these parameters

Update

Do not forget to click Update
② Explore cache configurations

The outputs

Data (application/json)

```json
{
  "cache_misses": cache_misses 64,
  "access_latency_cycles": access_latency_cycles: 4,
  "associativity": associativity: 8,
  "cache_frequency": cache_frequency: 2MHz,
  "cache_line_size": cache_line_size: 64,
  "I1_cache_configuration": cache_size cache_size: 8KIB
}
```

Check the number of cache misses (the lower the better)

Check your cache configurations

Check the total number of executed CPU cycles (the lower the better)
Hints for the challenge
Testing multiple configuration options

- Quickly test multiple hypothesis
  - Verify trends in data
  - Find problematic configurations
  - Optimize configurations

- OCCAM provides this feature
  - Run an experiment with multiple configurations
    - Only with numeric (integer and float) types
  - Permutates all possible combinations
Testing multiple configuration options

• Example

These options generate all possible permutations, thus creating 4 runs:

- block_size = 16, associativity=1
- block_size = 16, associativity=8
- block_size = 32, associativity=1
- block_size = 32, associativity=8

These must be integer or float types
③ Sharing and comparing results
③ Sharing and comparing results

Now we’ll compare your results with a partner

① You’ll share your results with a partner
   ◦ Choose another participant

② You’ll plot them side-by-side with yours
   ◦ Compare how much each improved the performance
In OCCAM’s web interface
- Open your best result

(a) Open the best result you got in the previous challenge

Note: The actual output names will be different
Sharing and comparing results

Sharing your results

Then get the address of your best result

- The address of your result should look like this

  https://<IP_ADDRESS>/objects/<id_of_your_result>

- It is a big number!
  - Lets make it easier to share
Sharing and comparing results

Sharing your results

Use tinyurl to make the URL shorter

(a) Go to https://tinyurl.com

(b) Paste the URL of your result in this box

(c) Give it an alias

(d) Make it smaller
Sharing and comparing results
Sharing your results

Share your link

Now please exchange your link with another participant
Sharing and comparing results

Sharing your results

(a) Click on your username to go back to the Workset selection page

(b) Go back to the Exercise 3 workset
③ Sharing and comparing results

Sharing your results

Get your partner’s results

(a) Get the link from another participant and type it into the import box

(b) Click Import
③ Sharing and comparing results

Sharing your results

Get your partner’s results

It will appear in the workset
③ Sharing and comparing results

Plot a comparison

Compare your results

(a) Create a new experiment to compare your results with your partner’s
③ Sharing and comparing results

Plot a comparison

Compare your results

(a) Put the plotter script that you used in exercise 1 in the workflow

(b) Click attach
③ Sharing and comparing results

Plot a comparison

Compare your results

(a) Attach your output
③ Sharing and comparing results

Plot a comparison

Compare your results

It should look like this

(a) Click the plus sign under your results
③ **Sharing and comparing results**

*Plot a comparison*

Compare your results

(a) Attach your partner’s output
③ Sharing and comparing results

*Plot a comparison*

Compare your results

It should look like this
③ **Sharing and comparing results**

*Plot a comparison*

Compare your results

(a) Configure your figure options and make a nice plot
Run the experiment to plot the comparison

(a) Click the run tab

(b) And click run

Plot a comparison
③ Sharing and comparing results

Plot a comparison

Check your results

(a) Once the run is ready, refresh the page

(b) And head to the Output tab
③ Sharing and comparing results

Plot a comparison

Check your results

You will have a plot like this one.

My Graph

- Xsim-results-user
- Xsim-results-user 2

200Hz 2KHz 20KHz 200KHz 2MHz

1M 0.5M
In this exercise

- You learned how to derive previous simulators
  - How simple it is with OCCAM and SST
- You learned how to explore variable spaces
  - Leveraging OCCAM ranged configurations
- You learned how to share your results
Acknowledgments

We gratefully acknowledge the support and collaboration of our research partners.

Portions of this material is based in part upon work supported by the National Science Foundation under Grant Numbers CCF-142331, CNS-1012070, ACI-1535232, CNS-1305220 and CCF-1148646. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.