CS4223 Assignment 3

Laurence Putra Franslay (U096833E)

14th Nov 2012

Appendix starts on page 3 of the report.

1 Tools to run the experiment

The code was written in Python, and will run using Python 2.x. However, for this experiment, as the data set is enormous, pypy was used as it is 40% more efficient than Python 2.x. 2 different environments were used to run this software, CentOS6 and MacOSX 10.8.2.

To run the software you have to install either pypy(recommended) or Python 2.x on the machine (Python 3.x is not supported due to its many performance issues). On CentOS6 it is *sudo yum install pypy* and on MacOSX 10.8.2 it is *brew install pypy*. Note that for MacOSX 10.8.2 you need to have Homebrew (http://mxcl.github.com/homebrew/) to install.

It also should be noted that the tracefiles should be stored in the format

```
dump
    \weather1
        WEATHER1.PRG
    \weather2
        WEATHER1.PRG    WEATHER2.PRG
    \weather4
        WEATHER1.PRG    WEATHER2.PRG    WEATHER3.PRG    WEATHER4.PRG
    \weather8
        WEATHER1.PRG    WEATHER2.PRG    WEATHER3.PRG    WEATHER4.PRG
        WEATHER5.PRG    WEATHER6.PRG    WEATHER7.PRG    WEATHER8.PRG
```

where dump is in the same directory as the code file. To run the code, simply run

```
pypy mesi_sim.py WEATHER 1 1024 64
```

where the first argument is the script name, second argument is the filename, third argument is the cache size, and fourth argument is the cache block size.

1.1 Implementation Details

Due to the lack of proper threading support in Python, the processors were assumed to run in order, i.e. Processor 1 followed by Processor 2 and so on.

The cache is represented in a 3D list, the first dimension pointing to the 2D cache of each individual processor. Each record held 2 values, the state as well as the tag. The state of each cache record is initially initialised to 'I'.

The bus for each processor was implemented using a *deque*, which is essentially a queue. When a processor starts its cycle, it will run through all the messages in the queue and update the cache depending on the contents of each message. The message is a tuple containing the operation (read or write), as weill as the tag and cache index.

When there is a cache miss, the processer will stall for 10 cycles, but the bus queue is still being fed with updates.
The full code can be found in *Section A of the Appendix*.

2 Conclusion

The cache miss ratio is calculated via the following equation.

$(CacheMissRatio) = (CacheMissCount)/(CacheAccessCount)$

2.1 Increasing Cache Size

From the experiment results, which can be found in *Section B of the Appendix*, it is clear that increasing the cache size results in a smaller cache miss ratio. For the multiprocessor traces, when you compare the cache miss ratio of the same processors for different cache sizes, it also shows that the larger the cache size, the smaller the cache miss ratio.

This observation is probably due to the fact that with a larger cache size, one can store more data within the cache, and hence, have less cache misses.

2.2 Increasing number of processors

There is no visible relation between the number of processors and the cache miss ratio. When the processor count is increased, the cache miss ratio sometimes increase and sometimes decrease.

This observation is probably due to the fact that the processor cannot choose which data it gets to read, and hence the cache miss ratio as a result of increasing the processor count is unpredictable.

A Code

```
import sys
import math
from collections import deque

if len(sys.argv) == 5:
    filename = sys.argv[1]
    no_processors = int(sys.argv[2])
    cache_size = int(sys.argv[3])
    cache_block_size = int(sys.argv[4])
else:
    filename = "WEATHER"
    no_processors = 4
    cache_size = 4096
    cache_block_size = 64

#initialisation of variables
cache_blocks = cache_size / cache_block_size
cache_block_offset = int(math.log(cache_blocks, 2))
word_size_in_bits = 16
word_size = word_size_in_bits / 8
memory_address_size_in_bits = 32
memory_address_size = memory_address_size_in_bits / 8
memory_block_offset = int(math.log(cache_block_size, 2))
tag_size = memory_address_size_in_bits - memory_block_offset -
    cache_block_offset



#analytics variables
cycle_count = 0
cache_access = 0
cache_miss = 0
cache_access_indie = []
cache_miss_indie = []
for x in range(no_processors):
    cache_miss_indie.append(0)
    cache_access_indie.append(0)
#opening files
files = []
for x in range(no_processors):
    files.append(open('dump/weather' + str(no_processors) + '/' +
        filename + str(x + 1) + '.PRG', 'r'))

#initialisation of cache
```

```
cache = []
for x in range(no_processors):
    cache.append([])
    i = 0
    while i < cache_blocks:
        cache[x].append(['I',0])
        i += 1

#initialisation of buses
bus = []
for x in range(no_processors):
    bus.append(deque([]))

#initialisation of stall variables
stall = []
for x in range(no_processors):
    stall.append(0)

#load mem access line into buffer
line = []
for x in range(no_processors):
    line.append(files[x].readline())
#start main processing
completed = False
count = 0
while not completed:
    for x in range(no_processors):
        inst_read = False
        data_read = False
        #clear bus
        while bus[x]:
            op, index, tag = bus[x].popleft()
            if cache[x][index][1] == tag:
                if op == 'r':
                    cache[x][index][0] = 'S'
                elif op == 'w':
                    cache[x][index][0] = 'I'
        #process line/s else, stall and minus 1 from CPU cycle
        if stall[x] == 0:
            for y in range(2):
                if line[x]:
                    inst_type = line[x][0]
                    inst_memory = line[x][2:]
                    if inst_type == '0':
                        if not inst_read:
                            inst_read = True
                        else:
                            break
                    elif inst_type == '2' or inst_type == '3':
                        if data_read:
                            break
                        else:
                            data_read = True
                            cache_access += 1
                            cache_access_indie[x] += 1
                            memory_block_number = bin(int(inst_memory, 16))[2:].zfill(32)
                            tag_index = memory_block_number[:-memory_block_offset]
                            cache_index = tag_index[-cache_block_offset:]
                            tag = tag_index[:-cache_block_offset]
                            #start of cache lookup
                            int_cache_index = int(cache_index,2)
                            int_tag = int(tag, 2)
                            if cache[x][int_cache_index][1] != int_tag or cache[x][int_cache_index][0] == 'I':
                                cache_miss += 1
                                cache_miss_indie[x] += 1
                                stall[x] = 10
                                for z in range(no_processors):
                                    if z != x and cache[y][int_cache_index][1] == int_tag:
                                        if cache[z][int_cache_index][0] == 'M':
                                            cache[z][int_cache_index][0] = 'E'
                                break
                            elif inst_type == '2':
                                pass
                            elif inst_type == '3':
                                cache[x][int_cache_index][0] = 'M'
                                for z in range(no_processors):
                                    tmp = 'w', int_cache_index, int_tag
                                    if x != z:
                                        bus[y].append(tmp)
                    line[x] = files[x].readline()
        else:
            stall[x] -= 1
            while bus[x]:
                op, index, tag = bus[x].popleft()
                if cache[x][index][1] == tag:
                    if op == 'r':
                        cache[x][index][0] = 'S'
                    elif op == 'w':
                        cache[x][index][0] = 'I'
            if stall[x] == 0:
                inst_type = line[x][0]
                inst_memory = line[x][2:]
                memory_block_number = bin(int(inst_memory, 16))[2:].zfill(32)
                tag_index = memory_block_number[:-memory_block_offset]
                cache_index = tag_index[-cache_block_offset:]
                tag = tag_index[:-cache_block_offset]
                #start of cache lookup
                int_cache_index = int(cache_index,2)
                int_tag = int(tag, 2)
                if inst_type == '3':
                    cache[x][int_cache_index][0] = 'M'
                    cache[x][int_cache_index][1] = int_tag
                    for y in range(no_processors):
                        tmp = 'w', int_cache_index, int_tag
                        if x != y:
                            bus[y].append(tmp)
                elif inst_type == '2':
                    state = 'E'
                    for y in range(no_processors):
                        if x != y and cache[y][int_cache_index][1] == int_tag:
                            state = 'S'
                            break
                    cache[x][int_cache_index][0] = state
                    cache[x][int_cache_index][1] = int_tag
                line[x] = files[x].readline()
    #stops when all processor files are done checking
    completed = True
    for x in range(no_processors):
        if line[x]:
            completed = False


for x in range(no_processors):
    print "Processor␣" + str(x + 1)
    print "Cache␣Miss␣␣␣␣␣␣␣=␣" + str(cache_miss_indie[x])
    print "Cache␣Access␣␣␣␣␣=␣" + str(cache_access_indie[x])
    print "Cache␣Miss␣Ratio␣=␣" +  str(cache_miss_indie[x]/float(cache_access_indie[x]))
    print ""
```

B Experiment results

B.1 1 Processor

B.1.1 Cache Size: 1KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	14677614	18125829	0.809762356249
Overall	14677614	18125829	0.809762356249

B.1.2 Cache Size: 2KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	14590128	18125829	0.804935763214
Overall	14590128	18125829	0.804935763214

B.1.3 Cache Size: 4KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	14470247	18125829	0.798321941579
Overall	14470247	18125829	0.798321941579

B.1.4 Cache Size: 8KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	14444872	18125829	0.796922005609
Overall	14444872	18125829	0.796922005609

B.1.5 Cache Size: 16KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	14381035	18125829	0.793400125313
Overall	14381035	18125829	0.793400125313

B.1.6 Cache Size: 32KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	14379670	18125829	0.793324818412
Overall	14379670	18125829	0.793324818412

B.2 2 Processor

B.2.1 Cache Size: 1KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	7565265	9062884	0.834752491591
2	7569500	9062945	0.835214160518
Overall	15134765	18125829	0.834983326832

B.2.2 Cache Size: 2KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	7509633	9062884	0.828614048243
2	7513657	9062945	0.829052476871
Overall	15023290	18125829	0.828833263295

B.2.3 Cache Size: 4KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	7422706	9062884	0.819022509832
2	7402491	9062945	0.81678648607
Overall	14825197	18125829	0.817904494189

B.2.4 Cache Size: 8KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	7398607	9062884	0.816363422504
2	7377055	9062945	0.813979892849
Overall	14775662	18125829	0.815171653666

B.2.5 Cache Size: 16KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	7350018	9062884	0.811002104849
2	7316131	9062945	0.807257574663
Overall	14666149	18125829	0.809129833455

B.2.6 Cache Size: 32KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	7341061	9062884	0.810013788105
2	7307170	9062945	0.806268823214
Overall	14648231	18125829	0.808141299358

B.3 4 Processors

B.3.1 Cache Size: 1KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	3730387	4531371	0.823235837454
2	3738672	4531558	0.825030155192
3	3728923	4531513	0.82288696954
4	3717382	4531387	0.820362948475
Overall	14915364	18125829	0.8228789977

B.3.2 Cache Size: 2KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	3685957	4531371	0.813430857902
2	3687713	4531558	0.813784795428
3	3681779	4531513	0.812483380275
4	3669681	4531387	0.809836149506
Overall	14725130	18125829	0.812383808763

B.3.3 Cache Size: 4KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	3638737	4531371	0.803010170653
2	3636115	4531558	0.80239842456
3	3631885	4531513	0.801472929682
4	3620475	4531387	0.798977222647
Overall	14527212	18125829	0.801464694387

B.3.4 Cache Size: 8KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	3627905	4531371	0.800619724141
2	3623026	4531558	0.799510013995
3	3619226	4531513	0.798679381478
4	3610225	4531387	0.796715222072
Overall	14480382	18125829	0.798881088418

B.4 4 Processors

B.4.1 Cache Size: 16KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	3595380	4531371	0.793441984777
2	3581420	4531558	0.790328624283
3	3577234	4531513	0.789412719328
4	3568609	4531387	0.787531279054
Overall	14322643	18125829	0.790178645071

B.4.2 Cache Size: 32KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	3592148	4531371	0.792728734858
2	3578329	4531558	0.789646518924
3	3574161	4531513	0.788734579378
4	3565379	4531387	0.786818473019
Overall	14310017	18125829	0.789482070034

B.5 8 Processors

B.5.1 Cache Size: 1KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	1930214	2269692	0.850429926175
2	1946099	2269847	0.857370122303
3	1940240	2269754	0.854823914838
4	1853819	2261703	0.81965625018
5	1853822	2261679	0.81966627448
6	1855236	2261711	0.820279867764
7	1855193	2261759	0.820243447688
8	1927886	2269684	0.849407230258
Overall	15162509	18125829	0.836513960272

B.5.2 Cache Size: 2KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	1901344	2269692	0.837710138644
2	1915059	2269847	0.84369519179
3	1910585	2269754	0.8417586223
4	1821863	2261703	0.805527074068
5	1821875	2261679	0.80554092778
6	1824877	2261711	0.806856844221
7	1824806	2261759	0.806808329269
8	1898190	2269684	0.83632347058
Overall	14918599	18125829	0.823057472295

B.5.3 Cache Size: 4KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	1871102	2269692	0.824385863809
2	1881520	2269847	0.828919306015
3	1876649	2269754	0.82680722228
4	1791328	2261703	0.792026185578
5	1791306	2261679	0.792024862945
6	1792825	2261711	0.792685272345
7	1793187	2261759	0.792828502064
8	1865131	2269684	0.821758006841
Overall	14663048	18125829	0.808958751625

B.5.4 Cache Size: 8KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	1865844	2269692	0.822069249925
2	1875488	2269847	0.826261858178
3	1871516	2269754	0.824545743724
4	1786931	2261703	0.790082075321
5	1786935	2261679	0.790092227942
6	1787403	2261711	0.790287972248
7	1787838	2261759	0.790463528608
8	1859556	2269684	0.819301717772
Overall	14621511	18125829	0.806667159885

B.5.5 Cache Size: 16KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	1844818	2269692	0.812805437918
2	1849769	2269847	0.81493113853
3	1844850	2269754	0.812797333984
4	1764478	2261703	0.780154600317
5	1764482	2261679	0.780164647591
6	1764941	2261711	0.78035655307
7	1764900	2261759	0.780321864531
8	1833791	2269684	0.807949917257
Overall	14432029	18125829	0.796213458706

B.5.6 Cache Size: 32KB

Processor No.	Cache Miss	Cache Access	Cache Miss Ratio
1	1843866	2269692	0.812385997748
2	1848326	2269847	0.814295412863
3	1843234	2269754	0.812085362555
4	1763003	2261703	0.779502436881
5	1763031	2261679	0.77952308882
6	1763585	2261711	0.779757006974
7	1763526	2261759	0.779714372751
8	1831888	2269684	0.807111474549
Overall	14420459	18125829	0.795575143073