

Week 8 Progress Report

Tim Russell

Department of Electrical and Computer Engineering

University of Maine

Orono, USA

tgrussell@eiu.edu

I. WRITING CODE

This week was spent writing the code which will emulate the *Ominous Ocelot* cache replacement algorithm in Jianhui's `Buffersim` program. The main program performs energy calculations and buffer management operations, but does so with the input of a specific cache replacement algorithm. So, in addition to the `Buffersim` program, Jianhui provided me with many files which implement various replacement schemes, such as LRU, LIRS, 2Q, etc. My task, then, was to code my algorithm and interface it with `Buffersim`.

Listing 1 shows the header file for *Ominous Ocelot*, while Listing 2 shows the implementation of the various functions. Additionally, several of the core files for `Buffersim` had to be altered to incorporate function calls from *Ominous Ocelot* into the program. The task of reading, understanding, and adapting Jianhui's code was more than a little arduous, and that occupied a great deal of time. One of the issues with this undertaking was understanding what different functions **did** when they were called. On more than one occasion I encountered function calls in one file, the function declaration in another, and the definition in a third, with no apparent hierarchical relationship between the three. The same was true of classes and structures, which were often redefined in multiple files. I found great friendship and cooperation with the `grep` command, which I exploited extensively to find where these things lived. Additionally, `Buffersim` uses a huge number of global objects which are, again, spread over several files. So, tracking down where these objects were initialized and modified was a cumbersome, time-consuming process.

II. TROUBLE ON THE BLOCK

After much work and plenty of input from Jianhui, I was able to finish the *Ominous Ocelot* code, which appears to interface properly with `Buffersim`. The only problem now is that apparently my overlap table `map` does not contain pertinent information for the replacement decisions. Specifically, I build the overlap table by first reading in the trace with the `analyzer` program developed last week. `analyzer` builds the frequency information and then outputs the polished block overlap information to a file. The `initOO()` function of the *Ominous Ocelot* code calls `build_map` which reads in the aforementioned file and builds the overlap table to be used in `Buffersim`.

The problem is that I just found out that the block numbers provided in the input file to `analyzer` are *Logical Block Numbers* (LBNs). The block numbers read into `buffersim` from the trace are relative block numbers, along with the associated inode. To get the LBN, `Buffersim` reconstructs the LBN from the block and inode, but the *Ominous Ocelot* code does not take this into account. The replacement decisions are thus based on erroneous data, and so this algorithm basically reverts to LRU. The next step will be to make the proper adjustments to get the LBNs correctly.

Listing 1. oo.h: Header file for *Omnivorous Ocelot* Algorithm

```

1  /*****
2  * Author: Tim Russell      *
3  * Created: July 21, 2008  *
4  * Last Mod'ed: July 22, 2008 *
5  *****/
6
7  #ifndef _oo_h
8  #define _oo_h_ 1
9
10 #include <fstream>
11 #include "arc.h"
12 #include <map>
13
14 /***** externs *****/
15 extern int bank_num;
16 extern int lru_bottom_len;
17 extern long TILength;
18 extern int ee_alg;
19
20 extern std::map<unsigned, int> inodeLastAccessBank;
21 extern int bank_accesses[20480];
22 extern unsigned hit;
23 extern unsigned miss;
24 int get_bank( unsigned inode , int fileD );
25 /*****
26
27 /***** From LRU *****/
28 #define LRU_REF_LIMIT 32
29 static int diff = 0;
30 static int diff1 = 0;
31 static int diff2 = 0;
32 static int diff3 = 0;
33 static int replacements = 0 ;
34
35 CDB *lruGetVictime_ee( unsigned inode);
36 /*****
37
38 /***** My typedefs and objects *****/
39 typedef std::map<unsigned int , unsigned long long> entryMap;
40 typedef std::map< unsigned int , entryMap > groupMap;
41 typedef std::pair<unsigned int , unsigned long long> entryType;
42 typedef std::pair<unsigned int , entryMap > rowType;
43
44 static groupMap groupTable;
45 /*****
46
47 /***** My functions *****/
48 void build_map(groupMap& groupTable , std::ifstream& in);
49 unsigned long long get_overlap_value( unsigned rowBlock , unsigned colBlock);
50 void initOO ();
51 void reportOO ();
52 int get_OO_victim_bank( unsigned block);
53 int OOfref( unsigned inode , unsigned block , int fileD , int op, int *pHit ,
54            int *pBankNum, int *pDirty);
55 CDB *chip_lru_remove( int bank);

```

```

56 void chip_mru_insert(CDB *bp, int bank);
57 void remove_from_chipList( CDB *bp, int bank );
58 /* *****/
59
60 /****** Included to avoid compilation problems *****/
61 #define ALG_LAST_BANK 0
62 #define ALG_LAST_INODE_BANK 1
63
64 class bank_rec2
65 {
66     public:
67     int freeBlk; // the number of free blks in the bank
68     long double lastAccess; //Actual Time access time
69     long double lastAccess1; //message arrives
70     long double lastActive; // time when the last active duration began
71     long double lastWaitDisk; // last time when the waitDisk event happens.
72     long double lastWait; //last time when the wait event happens.
73     long double lastArriveTime;
74     // when the bank is powering up, the nextActiveTime is the power up completion time
75     long double nextActiveTime;
76     long double energy, totalEnergy;
77     long double active_duration, activeEgy, active_duration1;
78     long double idle_duration, idleEgy;
79     long double nap_duration, napEgy;
80     long double tran_duration_a2d, tran_duration_d2a, tran_duration, tranEgy;
81     long double tran_duration_n2d, tran_duration_n2a, tran_duration_a2n;
82     long double lastJobStartTime, lastJobFinishTime;
83     long accesses;
84     int stat;
85     int busy;
86     int activeJobs;
87     double queueLenSum;
88     double samplingNum;
89     int devID; // where
90     int frq;
91     unsigned int finishedJobs;
92     int replacements;
93     //list< Job * > NICWaitList;
94     //list< Job * > DiskDMAChannelWaitList;
95     long double powerupTime; //future or last powerup time
96     int overlapJobs, overlapJobs1, overlapJobs2;
97     long double overlapTime, overlapTime1, overlapTime2;
98     int powerupNum; // powerup times
99     bank_rec2( ){};
100     // to record the chip access time and to determine whether its
101     // operation is overlapped with other chips if lastActiveBegin > lastActiveEnd,
102     // the chip is being active. Jianhui Oct 9 2007
103     long double lastActiveBegin, lastActiveEnd;
104
105     CDB chip_header;
106     CDB MRU_blocks;
107 };
108 /* *****/
109 #endif

```

Listing 2. oo.cc: Source file for *Omnivorous Ocelot* Algorithm

```

1  /*****
2  * Author: Tim Russell      *
3  * Created: July 21, 2008  *
4  * Last Mod'ed: July 22, 2008 *
5  *****/
6  using namespace std;
7
8  #include "oo.h"
9  #include "prefetch.h"
10 // #include "buffersim.h"
11
12 // #include <queue>
13 // #include <list>
14 // #include <iostream>
15 // #include <algorithm>
16
17
18 extern bank_rec2 *bank_info;
19
20 void build_map(groupMap& groupTable, std::ifstream& in)
21 {
22     unsigned int rowKey, colKey;
23     unsigned long long olap;
24     entryMap rowMap;
25     groupTable.clear();
26     rowMap.clear();
27
28     while (true)
29     {
30         in >> rowKey;
31         if (!in.good()) { break; }
32         while (in.good())
33         {
34             in >> colKey;
35             if (!colKey) { break; }
36             in >> olap;
37             if (olap)
38             {
39                 rowMap.insert(entryType(colKey, olap));
40             }
41         }
42         if (!rowMap.empty())
43         {
44             groupTable.insert(rowType(rowKey, rowMap));
45         }
46         rowMap.clear();
47     }
48 }
49
50 unsigned long long get_overlap_value(unsigned rowBlock, unsigned colBlock)
51 {
52     return groupTable[rowBlock][colBlock];
53 }
54
55

```

```

56 void initOO()
57 {
58     groupTable.clear(); // clear the map
59
60     std::ifstream fin("overlap_table.txt"); // open the frequencies file for reading
61     if (!fin.good())
62     {
63         fprintf(stderr, "Error while opening file in initOO(); aborting!\n");
64         exit(1);
65     }
66
67     build_map(groupTable, fin); // build the group map
68
69     int i;
70     // fprintf(stderr, "bank_num used in initOO() is %d; aborting!\n", bank_num);
71     for (i = 0; i < bank_num; i++)
72     {
73         bank_info[i].chip_header.lrunext = NULL;
74         bank_info[i].chip_header.lruprev = NULL;
75         // fprintf(stderr, "Accessing %d th memory chip MRU pointer in initOO()\n", i);
76         bank_info[i].MRU_blocks.chipList_next = NULL;
77         bank_info[i].MRU_blocks.chipList_prev = NULL;
78     }
79
80     fin.close();
81 }
82
83 int get_OO_victim_bank(unsigned block)
84 {
85     unsigned long long totalOverlap, maxOverlap = 0;
86     CDB *chipList, *sentinel;
87
88     int bank;
89
90     for (int i = 0; i < bank_num; i++) {
91         totalOverlap = 0;
92
93         chipList = &(bank_info[i].MRU_blocks);
94         sentinel = chipList;
95
96         // Sum the overlap values of the blocks on this chip
97         if (groupTable.count(chipList->block))
98         { // this is one of the elements of the group table
99             totalOverlap += get_overlap_value(block, chipList->block);
100         }
101         chipList = chipList->chipList_next;
102         while (chipList != sentinel)
103         {
104             if (groupTable.count(chipList->block))
105             {
106                 totalOverlap += get_overlap_value(block, chipList->block);
107             }
108
109             chipList = chipList->chipList_next;
110         }
111     }

```

```

112     if (totalOverlap > maxOverlap) {
113         totalOverlap = maxOverlap;
114         bank = i;
115     }
116 }
117
118 return bank;
119 }
120
121 void reportOO()
122 {
123     float diffRate = 0;
124
125     if (replacements != 0)
126     {
127         diffRate = (float)diff/(float)replacements;
128     }
129
130     printf("LRU: Accesses %d Hit %d Miss %d Ratio %d,diff %d,diff1 %d,diff2 %d",
131           accesses , hit , miss , (100*hit)/accesses ,diff ,diff1 ,diff2 );
132     printf(",diff3 %d,rep: %d, diff rate %f\n", diff3 ,replacements ,diffRate);
133 }
134
135
136 int OOfref(unsigned inode , unsigned block , int fileD , int op ,
137           int *pHit ,int *pBankNum ,int *pDirty)
138 {
139     // fprintf(stderr , "Got into OOfref\n");
140     struct CDB *temp , *bp1;
141     int bank , freeFlag;
142     *pHit = 0;
143     freeFlag = 1;
144     accesses++; // where is this initialized?
145
146     temp = locate(inode , block); // find the requested page
147     if (temp) // if (temp != NULL), i.e. this is a cache hit
148     {
149         hit++;
150         *pHit = 1;
151         remove_from_list(temp); // take off whichever list
152         mru_insert(temp , T1); // put it back at the MRU spot
153         remove_from_chipList(temp , temp->bank );
154         chip_mru_insert(temp , temp->bank);
155     }
156
157     else // this is a cache miss
158     {
159         miss++;
160         if ((unsigned)T1Length < cachesize)
161         { // there is free space in cache
162             temp = get_new_CDB();
163             temp->bank = get_bank(inode , fileD);
164             // insert_list(bank_info[temp->bank].chip_header , temp);
165             assert( temp->bank >= 0 && temp->bank < bank_num);
166         }
167         else // we have to make room by evicting some block

```

```

168 {
169     freeFlag =0;
170     if (groupTable.count(block))
171     { // this is a high-frequency block
172         // place this block in the chip with the most overlap
173         bank = get_OO_victim_bank(block);
174         temp = chip_lru_remove(bank);
175     }
176     else // this block is not in our groupMap
177     { // use Jianhui's approach
178         if ( !lru_bottom_len )
179         { // there is no LRU window, so just use straight LRU
180             temp = lru_remove(T1);
181         }
182
183         else
184         { // we are using an LRU window
185             bp1 = lruBottom();
186             bank = getLRU_victimBank();
187             temp = lruGetVictime_ee(inode);
188             if (temp)
189             {
190                 remove_from_list(temp);
191
192                 if (temp->bank != bank) { diff++; }
193                 if (bp1 != temp) { diff1++; }
194                 if (bp1->bank != temp->bank) { diff2++; }
195             }
196
197             else { temp = lru_remove(T1); }
198         } // end Jianhui's MRU/LRU approach
199     } // end of "not in groupMap" block
200
201     T1Length--;
202     replacements++;
203
204     remove_from_chipList(temp, temp->bank );
205
206
207 } // end of "no free space" block
208
209     mru_insert(temp, T1);
210     chip_mru_insert(temp, temp->bank);
211
212
213     T1Length++; // bookkeep:
214
215     hashout(temp); //remove from hash table if there
216     temp->inode = inode;
217     temp->block = block;
218     assert(temp->bank >= 0 && temp->bank < bank_num);
219
220     hashin(temp); // put it back in
221 } // end of cache miss block
222
223 *pDirty = temp->dirty;

```

```

224     if( !(*pHit) && op == READ)
225     {
226         temp->dirty = 0;
227     }
228
229     if(op == WRITE )
230     {
231         temp->dirty = 1;
232     }
233
234     *pBankNum = temp->bank;
235     assert(temp->bank >= 0 && temp->bank < bank_num);
236
237     // Got rid of "if"-statement, because cnt was always set to 0
238     memset(bank_accesses ,bank_num ,0);
239
240     bank_accesses [*pBankNum]++;
241     inodeLastAccessBank [inode]=*pBankNum;
242
243     return 1;
244 }
245
246 CDB *lruGetVictime_ee(unsigned inode)
247 {
248     CDB *bp, *bp1;
249     int bank, bank1;
250     std::map<unsigned, int >::iterator it;
251
252     if (ee_alg == ALG_LAST_INODE_BANK)
253     {
254         it = inodeLastAccessBank.find(inode);
255
256         if(it == inodeLastAccessBank.end())
257         {
258             bank = getLRU_victimBank(); // get LRU top block's bank
259         }
260         else
261         {
262             bank = inodeLastAccessBank[inode];
263         }
264     }
265
266     else if (ee_alg == ALG_LAST_BANK)
267     {
268         bank = getLRU_victimBank(); // get LRU top block's bank
269     }
270
271     //bank = getMax(bank_accesses ,bank_num);
272     if( bank < 0 || bank >= bank_num)
273     {
274         printf("bank is out of range @ lruGetVictime_ee( )\n");
275         return NULL;
276     }
277
278     bp = getLRU_victimBlock(bank);
279     if (bp == NULL)

```

```

280 {
281     return NULL;
282 }
283
284 bp1 = lruBottom();
285 bank1 = bp1->bank;
286 if (bank != bank1)
287 {
288     diff3++;
289 }
290
291 return bp;
292 }
293
294 /* Remove and return local LRU block from this memory chip */
295 CDB *chip_lru_remove(int bank)
296 {
297     CDB *bp;
298
299     bp = &(bank_info[bank].MRU_blocks);
300     bp = bp->chipList_prev;
301     (bp->chipList_prev)->chipList_next = bp->chipList_next;
302     (bp->chipList_next)->chipList_prev = bp->chipList_prev;
303     //bp->chipList_next = bp->chipList_prev = NULL;
304
305     return bp;
306 }
307
308
309 /*remove bp from bank's chip list*/
310 void remove_from_chipList( CDB *bp, int bank ){
311
312     CDB *header;
313
314     header = &(bank_info[bank].MRU_blocks);
315
316     /*only one element in this list*/
317     if( header->chipList_next == header->chipList_prev ){
318         header->chipList_next = header->chipList_prev = NULL;
319         bp->chipList_next =bp->chipList_prev = NULL;
320
321     }else{
322
323         bp->chipList_prev->chipList_next = bp->chipList_next;
324         bp->chipList_next->chipList_prev = bp->chipList_prev;
325         bp->chipList_next =bp->chipList_prev = NULL;
326     }
327
328     return;
329 }
330
331 /* Insert a block to the local MRU location on this memory chip
332 void chip_mru_insert(CDB *bp, int bank)
333 {
334     CDB *header;
335

```

```
336     header = &(bank_info[bank].MRU_blocks);
337
338
339     bp->chipList_next = header->chipList_next;
340     bp->chipList_prev = header;
341
342
343     if( header->chipList_next != NULL )
344         (header->chipList_next)->chipList_prev = bp;
345     else
346     {
347         bp->chipList_next = header;
348
349     }
350     header->chipList_next = bp;
351
352
353     if( header->chipList_prev == NULL )
354         header->chipList_prev = bp;
355
356
357 }
```