

CSE220 Project Part 3

Rax - a radix tree

implementation in ANSI C

Adil Rahman

Link to Github Repo

- <https://github.com/antirez/rax>

The screenshot shows the GitHub repository page for `antirez / rax`. The page includes a navigation bar with links to Pull requests, Issues, Marketplace, and Explore. Below the navigation bar, there are buttons for Watch (31), Star (547), Fork (71), and Insights. The repository summary shows 192 commits, 1 branch, 0 packages, 0 releases, 6 contributors, and BSD-2-Clause license. A pull request button and a clone/download button are also present. The commit history lists several recent changes, including updates to `.gitignore`, `COPYING`, `Makefile`, `README.md`, `TODO.md`, `crc16.c`, and `rax-oom-test.c`.

File	Description	Time Ago
<code>.gitignore</code>	Git ignore updated.	3 years ago
<code>COPYING</code>	Add email address in copyright notice.	3 years ago
<code>Makefile</code>	Cluster fuzz mode: emulate Redis issue 5693 more closely.	11 months ago
<code>README.md</code>	README: add info about debugging Rax.	last year
<code>TODO.md</code>	Do not allow the compiler to optimized failed lookup call.	3 years ago
<code>crc16.c</code>	Cluster fuzz mode: emulate Redis issue 5693 more closely.	11 months ago
<code>rax-oom-test.c</code>	Copyright year updated.	last year

Brief description of what the workload is

- **Rax** is a radix tree implementation in ANSI C that was originally created to fix a performance issue with Redis.
 - Redis: an open source in-memory data structure store that can be used as a database, cache, or message broker [1].
 - Radix tree: a space-optimized trie in which each node that is the only child is merged with its parent [2].
 - A trie is a specific type of search tree or ordered tree data structure that is used to store dynamic sets or associative sets where the keys are strings [3].
- The intention of Rax as an application was to create a balance between performance and memory usage through the use of radix trees [1].
 - The project uses fuzz testing techniques to test the implementation on a large amount of potential states [1].

Brief description of what the workload does

- The file rax-test.c is present to test the actual implementation.
- Functions:
 - hash table look-up, addition and removal of an element, locating an element within the table, and freeing the entire hash table.
 - uint32 to uint32 conversion, uint32 to key, int to key, and returning UNIX time in microseconds.
- Benchmark Tests:
 - do_benchmark: creates radix tree with 5,000,000 keys and elements, perform linear and random lookups by key and then deletes all 5,000,000 elements from tree.
 - do_units: perform walks through radix tree while testing proper functionality of iterator, and insertion of elements.
 - do_fuzz_cluster: fuzz test for generating random keys and inserting and removing elements
 - do_fuzz: fuzz test for generating random keys, inserting and removing elements, iterator functionality, and reporting of all elements.
 - do_regression: series of regression tests testing functionality of functions for radix tree
 - do_hugekey: testing creation of key bigger than $(2^{29}-1)$ characters

Steps to compile and run the workload

- Setting up Internet for VirtualBox
- Running CSE220_Arch on VirtualBox
- Git clone repo into VirtualBox
- Adding Static Flags to Makefile
- Editing Input File
- Compiling the Files
- Making the Script to Run the Workload

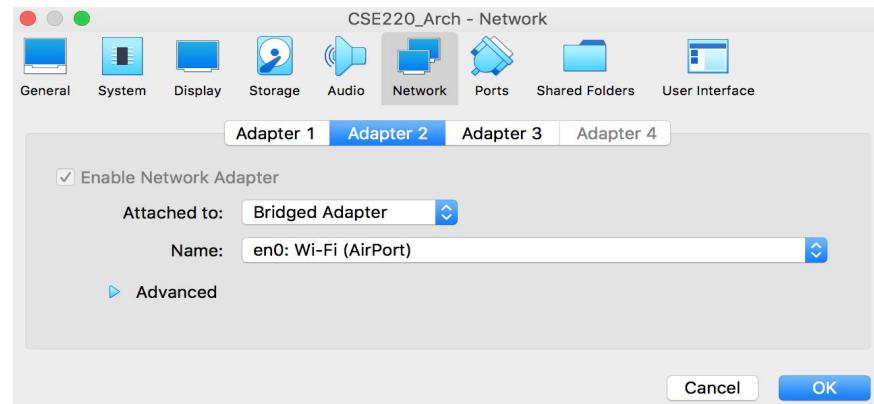
Setting up Internet for VirtualBox

- Ensure that your VirtualBox is connected to the internet. This will be necessary when we use git clone to copy the repo contents into the VirtualBox. The following steps were performed on the Mac High Sierra OS.
- Open VirtualBox and click the settings button:



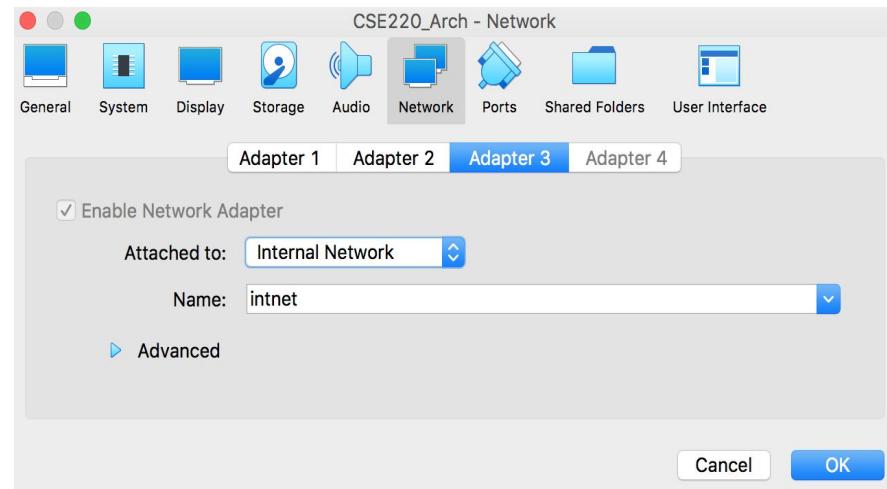
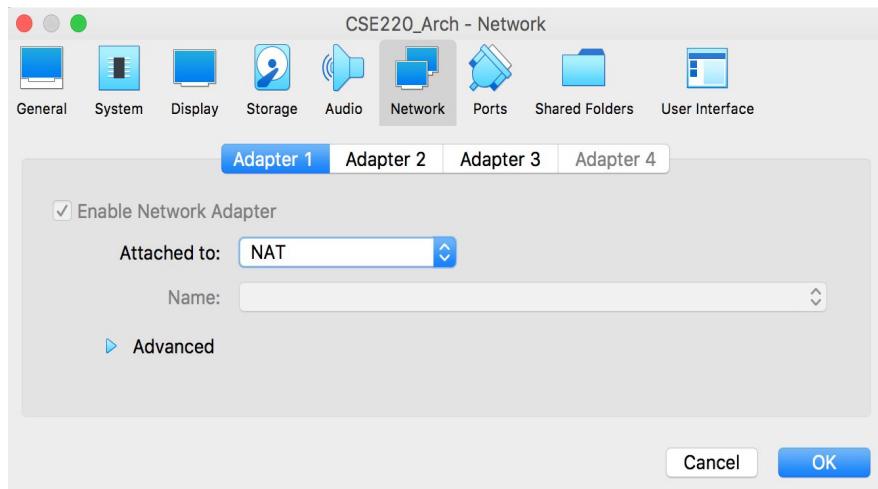
Setting up Internet for VirtualBox (2)

- Click on the ‘Network’ button
- Ensure that Adapter 1, Adapter 2, and Adapter 3 are set to the following settings:
 - Adapter 1:
 - Attached to: NAT
 - Adapter 2:
 - Attached to: Bridged Adapter
 - Name: en0: Wi-Fi (AirPort)
 - Adapter 3:
 - Attached to: Internal Network
 - Name: intnet
- Click the ‘OK’ button.



Setting up Internet for VirtualBox (3)

- Screenshots of Adapter 1 and Adapter 3 settings



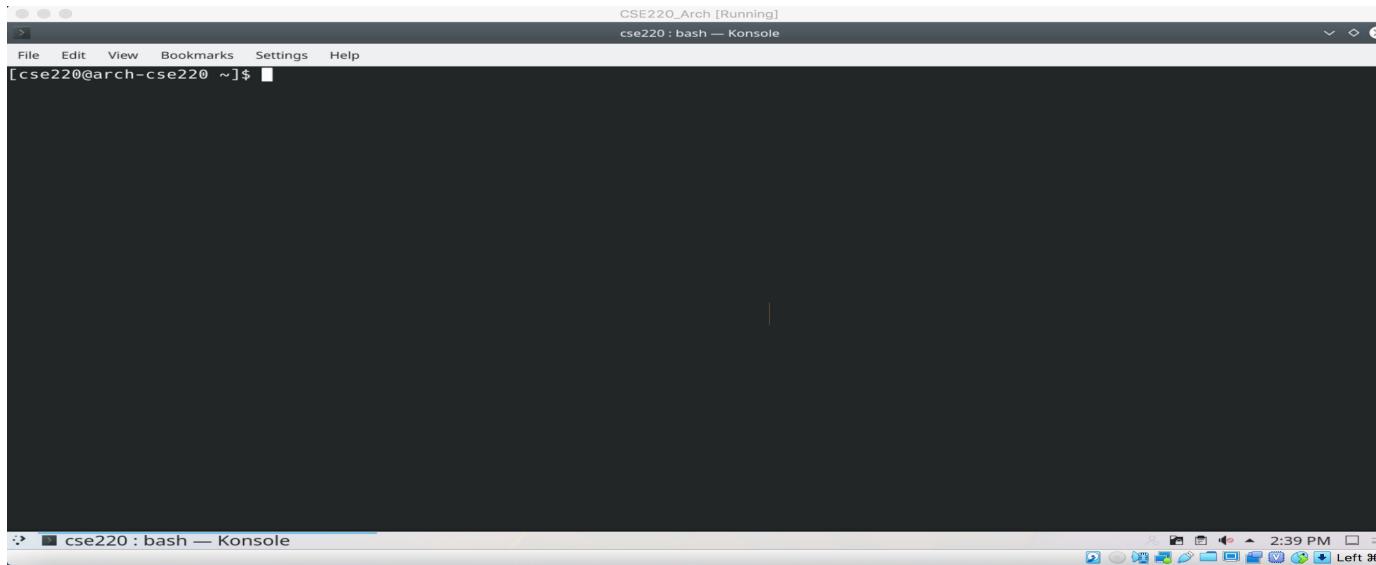
Running CSE220_Arch on VirtualBox

- Open the VirtualBox application and run ‘CSE220_Arch’ as seen below:



Running CSE220_Arch on VirtualBox (2)

- Following the steps of Project Part 1, you should successfully see the Linux OS.
- Open up a new Konsole at the bottom left of the screen as seen below:



Git clone repo into VirtualBox

- Click on the following link to find the repo for Rax:
 - <https://github.com/antirez/rax>
- Locate the ‘Clone or download’ button on the repo. It should look like this:
 - <https://github.com/antirez/rax.git>
- Copy this previous link for a future step. If your VirtualBox does not have a shared clipboard capability with your actual operating system, you will have to manually type this command in the git clone step.

The screenshot shows a GitHub repository page for 'rax' by 'antirez'. The page title is 'A radix tree implementation in ANSI C'. Key statistics shown are 192 commits, 1 branch, 0 packages, 0 releases, 6 contributors, and BSD-2-Clause license. A 'Clone or download' button is highlighted in green. Below the stats, there's a list of files: .gitignore, COPYING, Makefile, README.md, TODO.md, crc16.c, rax-oom-test.c, rax-test.c, rax.c, rax.h, and rax_malloc.h. The 'Clone with HTTPS' field contains the URL <https://github.com/antirez/rax.git>.

File	Description	Last Commit
.gitignore	Git ignore updated.	3 years ago
COPYING	Add email address in copyright notice.	last year
Makefile	Cluster fuzz mode: emulate Redis issue 5693 more closely.	11 months ago
README.md	README: add info about debugging Rax.	last year
TODO.md	Do not allow the compiler to optimized failed lookup call.	3 years ago
crc16.c	Cluster fuzz mode: emulate Redis issue 5693 more closely.	11 months ago
rax-oom-test.c	Copyright year updated.	last year
rax-test.c	Cluster fuzz test: report final number of keys.	11 months ago
rax.c	Ability to dynamically enable/disable debug messages.	last year
rax.h	raxSetDebugMsg() prototype added.	last year
rax_malloc.h	Ability to switch allocator without touching rax.c.	3 years ago

Git clone repo into VirtualBox (2)

- First change to the run folder by inputting the following command from the home directory:
 - cd build/release/run
- Ensure simu.conf.apache is inside of your run folder.
- From the previous slide, input the following command to obtain the files from the repo into the run folder.
 - git clone <https://github.com/antirez/rax.git>

Git clone repo into VirtualBox (3)

CSE220_Arch [Running]
run : bash — Konsole

File Edit View Bookmarks Settings Help

```
[cse220@arch-cse220 ~]$ cd build/release/run
[cse220@arch-cse220 run]$ ls
bins          IL1_core_assoc    nArchRegs      scripts
blockSize     instQueueSize    netBench.conf  simu.conf
'C Code'      json.sh         otree.sh       simu.conf.a72like
cJSON         L1CacheAssoc    peq.conf      simu.conf.acc
cmake-modules lodepng        percore_dtlb_assoc simu.conf.apache
data          maxbranches     pete.sh       simu.conf.aries
esesc.conf    maxLoads        pfstride     simu.conf.boom2
fetchports   maxprefetch    PrivL2_size   simu.conf.exynos4
fetchwidth   maxSaves        ProjectPartIScripts simu.conf.n1
flp.conf     memLatency     pwth.conf    simu.conf.projConf
gcc_out.s    memLatency     pwth.conf.exynos  simu.conf.samurai
hello.rv      memory-arch.dot router.conf  spec00_crafty.sh
[cse220@arch-cse220 run]$ git clone https://github.com/antirez/rax.git
Cloning into 'rax'...
remote: Enumerating objects: 643, done.
remote: Total 643 (delta 0), reused 0 (delta 0), pack-reused 643
Receiving objects: 100% (643/643), 195.69 KiB | 1.42 MiB/s, done.
Resolving deltas: 100% (402/402), done.
[cse220@arch-cse220 run]$ ls
bins          instQueueSize    otree.sh       simu.conf.a72like
blockSize     json.sh         peq.conf      simu.conf.acc
'C Code'      L1CacheAssoc    percore_dtlb_assoc simu.conf.apache
cJSON         lodepng        pete.sh       simu.conf.aries
cmake-modules maxbranches     pfstride     simu.conf.boom2
data          maxLoads        PrivL2_size   simu.conf.exynos4
esesc.conf    maxprefetch    ProjectPartIScripts simu.conf.n1
fetchports   maxSaves        pwth.conf    simu.conf.projConf
fetchwidth   memLatency     pwth.conf.exynos  simu.conf.samurai
flp.conf     memLatency     rax          spec00_crafty.sh
gcc_out.s    memory-arch.dot router.conf  spec00_gap.sh
[cse220@arch-cse220 run]$
```

rax folder has been created

Home — Dolphin run : bash — Konsole 2:52 PM Left 96

Adding Static Flags to Makefile

- Now go into the rax folder by inputting the following command:
 - cd rax
- Static flags will need to be added to the Makefile. Open up the file by inputting the following command:
 - vim Makefile

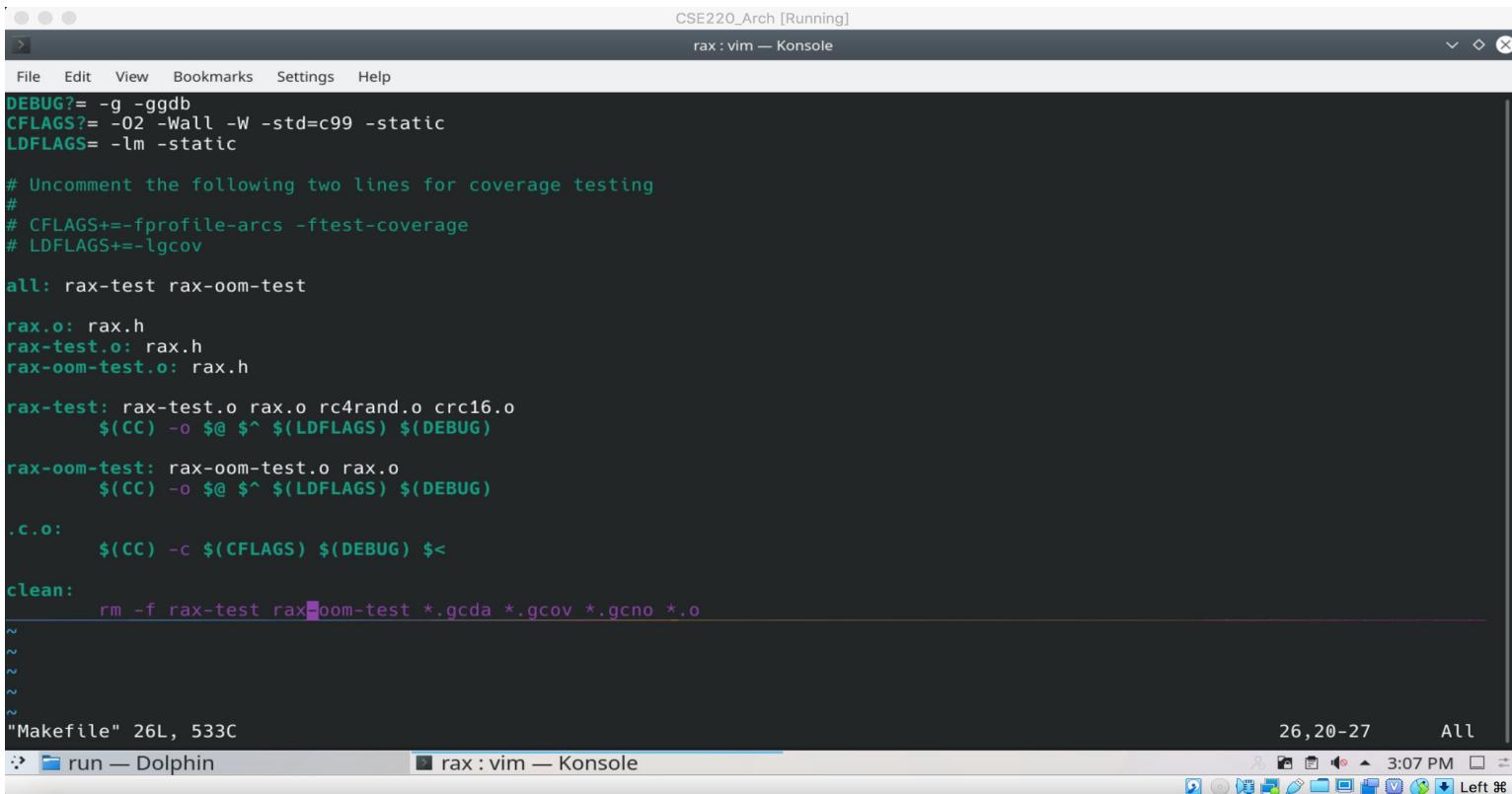
```
CSE220_Arch [Running]
rax : bash — Konsole
File Edit View Bookmarks Settings Help
[csse220@arch-cse220 ~]$ cd build.release.run
bash: cd: build.release.run: No such file or directory
[csse220@arch-cse220 ~]$ cd build/release/run
[csse220@arch-cse220 run]$ ls
bins          hello.rv      memLatency      pwth.conf    simu.conf.boom2      spec00_vortex_assoc.sh
blockSize     l1core_assoc  memory-arch.dot  pwth.conf.exynos  simu.conf.exynos4  spec00_vortex_memlatency.sh
Code          instanceSize   nArchRegs      rax.conf     simu.conf.rax        spec00_vortex_ArchRegnorm.sh
cJSON         json.sh       mtBench.conf  rax.sh       simu.conf.projConf  spec00_vortex_ArchRegs.sh
cmkE-modules LiCacheAssoc  otree.sh       router.conf  simu.conf.samurai  spec00_vortex.sh
data          lodepng       peq.conf      scripts     simu.conf.a72like  vortex1.out
esesc.conf    maxbranches   percore_dtlb_assoc  simu.conf      simu.conf.acc      vortex1.msg
fetchports   maxloads      pete.sh       simu.conf.frist  spec00_crafty.sh
fetchwidth   maxprefetch   pfstride     simu.conf.gap    spec00_gap.sh
flp.conf     maxSaves      PrivL2_size  simu.conf.apache  spec00_gcc.sh
gcc_out.s    memlatency   ProjectPartIScripts  simu.conf.aries  spec00_parser.sh
[csse220@arch-cse220 run]$ cd rax
[csse220@arch-cse220 rax]$ ls
COPYING  Makefile  rax.h      rax_oom_malloc.h  rax-test.c  rc4rand.h  TODO.md
crc16.c  rax.c      rax_malloc.h  rax-oom-test.c  rc4rand.c  README.md
[csse220@arch-cse220 rax]$ vim Makefile
```

The screenshot shows a terminal window titled "rax : bash — Konsole". The terminal displays a series of Linux shell commands and their outputs. It starts with navigating to a non-existent directory "build.release.run", then to "build/release/run". It lists files in this directory, which include various configuration files like "pwth.conf", "simu.conf", and "scripts", along with executables like "hello.rv" and "rax.sh". The user then navigates to the "rax" directory and lists its contents, which include "COPYING", "Makefile", "rax.h", "crc16.c", "rax.c", and "README.md". Finally, the user runs the command "vim Makefile" to edit the Makefile.

Adding Static Flags to Makefile (2)

- Once you are in the editor, ensure that ‘-static’ has been added to CFLAGS? and LDFLAGS.
- While in vim, press ‘i’ to be in the INSERT mode. Add ‘-static’ to the sections mentioned in the previous bullet. Then follow these steps:
 - Press the ‘esc’ button to exit the INSERT mode
 - Press the ‘:’ button, followed by ‘wq’ to make the writes to the file
- It should look like the following after the steps are done:
 - CFLAGS?= -O2 -Wall -W -std=c99 -static
 - LDFLAGS= -lm -static

Adding Static Flags to Makefile (3)



The screenshot shows a Kubuntu desktop environment with a terminal window titled "rax : vim — Konsole". The terminal displays a Makefile with static flags added. The Makefile includes definitions for DEBUG, CFLAGS, and LDFLAGS, and targets for rax-test, rax-oom-test, .c.o, and clean.

```
DEBUG?= -g -ggdb
CFLAGS?= -O2 -Wall -W -std=c99 -static
LDFLAGS= -lm -static

# Uncomment the following two lines for coverage testing
#
# CFLAGS+=--fprofile-arcs -ftest-coverage
# LDFLAGS+=--lgcov

all: rax-test rax-oom-test

rax.o: rax.h
rax-test.o: rax.h
rax-oom-test.o: rax.h

rax-test: rax-test.o rax.o rc4rand.o crc16.o
          $(CC) -o $@ $^ $(LDFLAGS) $(DEBUG)

rax-oom-test: rax-oom-test.o rax.o
              $(CC) -o $@ $^ $(LDFLAGS) $(DEBUG)

.c.o:
          $(CC) -c $(CFLAGS) $(DEBUG) $<

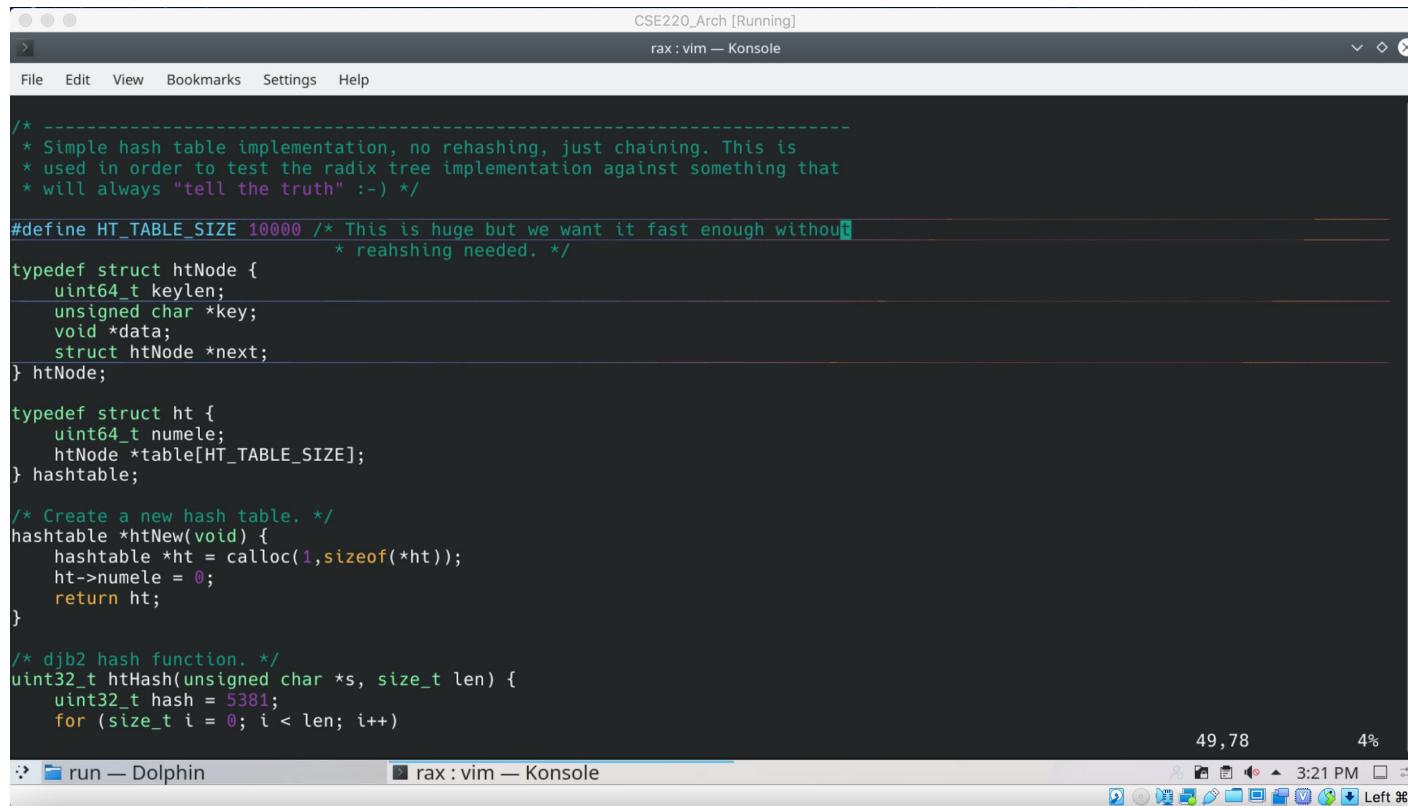
clean:
          rm -f rax-test rax-oom-test *.gcda *.gcov *.gcno *.o
```

The terminal window has a title bar "CSE220_Arch [Running]" and a status bar at the bottom showing "26,20-27 All" and the current time "3:07 PM". The bottom of the screen shows the Kubuntu desktop interface with icons for various applications like dolphin, file manager, and system settings.

Editing Input File

- You will now need to make an edit to the following input file:
 - rax-test.c
- Input the following command to edit the file:
 - vim rax-test.c
- Line 49 of the code should have the following:
 - #define HT_TABLE_SIZE 100000
 - Edit this so that the table size is now 10000 instead:
 - Press ‘i’ for INSERT mode, make the change, then press ‘esc’, then ‘:’, then ‘wq’.
 - The line should now look like this: #define HT_TABLE_SIZE 10000

Editing Input File (2)



The screenshot shows a terminal window titled "CSE220_Arch [Running]" with the command "rax : vim — Konsole". The terminal is displaying a C program. The code defines a hash table structure using chaining. It includes a comment about the radix tree implementation, a large table size of 10000, and a djb2 hash function. The terminal window has a dark background and light-colored text. The status bar at the bottom shows "49,78" and "4%" on the right, and "rax : vim — Konsole" on the left. The bottom of the window features a toolbar with various icons.

```
/* -----
 * Simple hash table implementation, no rehashing, just chaining. This is
 * used in order to test the radix tree implementation against something that
 * will always "tell the truth" :-) */
#define HT_TABLE_SIZE 10000 /* This is huge but we want it fast enough without
                           * reahsing needed. */
typedef struct htNode {
    uint64_t keylen;
    unsigned char *key;
    void *data;
    struct htNode *next;
} htNode;
typedef struct ht {
    uint64_t numele;
    htNode *table[HT_TABLE_SIZE];
} hashtable;
/* Create a new hash table. */
hashtable *htNew(void) {
    hashtable *ht = calloc(1,sizeof(*ht));
    ht->numele = 0;
    return ht;
}
/* djb2 hash function. */
uint32_t htHash(unsigned char *s, size_t len) {
    uint32_t hash = 5381;
    for (size_t i = 0; i < len; i++)

```

Compiling the Files

- Ensure that you are still in the rax folder.
- Input the following command to commence the compiling:
 - make CC=riscv64-linux-gnu-gcc
- You should now see the riscv file in the rax folder for rax-test as follows:

```
[cse220@arch-cse220 run]$ cd rax
[cse220@arch-cse220 rax]$ ls
COPYING  esesc_rax.xAGLQP  rax.h      rax_oom_malloc.h  rax-oom-test.o  rax-test.o  rc4rand.o
crc16.c  Makefile        rax_malloc.h  rax-oom-test    rax-test      rc4rand.c  README.md
crc16.o  rax.c          rax.o       rax-oom-test.c   rax-test.c    rc4rand.h  TODO.md
```

Compiling the Files (2)

The screenshot shows a terminal window titled "CSE220_Arch [Running]" with the command "rax : bash — Konsole". The terminal output is as follows:

```
bash: cd: build.release.run: No such file or directory
[cse220@arch-cse220 ~]$ cd build/release/run
[cse220@arch-cse220 run]$ ls
bins          hello.rv      memLatency      ppth.conf      simu.conf.boom2    spec00_vortex_assoc.sh
blockSize     IL1_core_assoc memory-arch.dot  ppth.conf.exynos  simu.conf.exynos4
'C Code'      instQueueSize  nArchRegs      rax           simu.conf.n1
cJSON         json.sh       netBench.conf   rax.sh        simu.conf.projConf
cmake-modules L1CacheAssoc  otree.sh       router.conf   simu.conf.samurai
data          lodepng       peq.conf       scripts       spec00_crafty.sh
esesc.conf    maxbranches   percore_dtlb_assoc simu.conf       spec00_gap.sh
fetchports   maxloads      pete.sh        simu.conf.a72like spec00_gcc.sh
fetchwidth   maxprefetch   pfstride      simu.conf.acc   spec00_parser.sh
flp.conf     maxSaves      Prvl2_size    simu.conf.apache spec00_twolf.sh
gcc_out.s    memlatency   ProjectPartIScripts simu.conf.aries spec00_vortex_...
[cse220@arch-cse220 run]$ cd rax
[cse220@arch-cse220 rax]$ ls
COPYING  Makefile  rax.h      rax_oom_malloc.h  rax-test.c  rc4rand.h  TODO.md
crc16.c  rax.c     rax_malloc.h  rax-oom-test.c  rc4rand.c  README.md
[cse220@arch-cse220 rax]$ vim Makefile
[cse220@arch-cse220 rax]$ ls
COPYING  Makefile  rax.h      rax_oom_malloc.h  rax-test.c  rc4rand.h  TODO.md
crc16.c  rax.c     rax_malloc.h  rax-oom-test.c  rc4rand.c  README.md
[cse220@arch-cse220 rax]$ vim rax-test.c
[cse220@arch-cse220 rax]$ make CC=riscv64-linux-gnu-gcc
riscv64-linux-gnu-gcc -c -O2 -Wall -W -std=c99 -static -g -ggdb rax-test.c
riscv64-linux-gnu-gcc -c -O2 -Wall -W -std=c99 -static -g -ggdb rax.c
riscv64-linux-gnu-gcc -c -O2 -Wall -W -std=c99 -static -g -ggdb rc4rand.c
riscv64-linux-gnu-gcc -c -O2 -Wall -W -std=c99 -static -g -ggdb crc16.c
riscv64-linux-gnu-gcc -o rax-test rax-test.o rax.o rc4rand.o crc16.o -lm -static -g -ggdb
riscv64-linux-gnu-gcc -c -O2 -Wall -W -std=c99 -static -g -ggdb rax-oom-test.c
riscv64-linux-gnu-gcc -o rax-oom-test rax-oom-test.o rax.o -lm -static -g -ggdb
[cse220@arch-cse220 rax]$
```

The terminal is running in a KDE desktop environment, indicated by the "Dolphin" file manager icon in the taskbar.

Making the Script to Run the Workload

- Ensure that you are now in the run folder.
- Create a script called ‘rax.sh’ to perform the run in ESESC
- This can be done through the GUI. In the run folder, perform the following steps:
 - Right click and select ‘Create New’ and then ‘Text File ...’
 - Name the file ‘rax.sh’
 - In the Konsole, ensure you are still in the run folder and input ‘vim rax.sh’
 - Fill in the file with the text seen in the following slide
- Once this is done, you will need to give execute permission for the file. While in the run folder where rax.sh is present, input the following command:
 - chmod +x rax.sh

Making the Script to Run the Workload (2)

- Type the following into the rax.sh file and save it

Making the Script to Run the Workload (3)

- Now run the script file rax.sh
 - In the run folder, input the following command to run the script file:
 - `./rax.sh`
 - The code will now run

Initial results showing low IPC and reasonable instruction count

GitHub Stars	Instruction Count	IPC
547	182998019	0.54

The screenshot shows the Piazza platform interface. At the top, there's a navigation bar with links for Apps, MS Project_2018..., 2019-20calendar..., Spring 2013 -- Co..., Login to MyUCSC, UC Santa Cruz - S..., Dashboard, tpc-c_v5.11.0.pdf, py-tpcc/sqlitedrv..., and a user profile for Adil Rahman.

The main area displays a discussion thread under the heading "CSE 220". The thread has several posts:

- A post from Ross Edward Mawhorter with the title "Instr Use this thread to claim rep..." and a link to <https://github.com/23Prover/z3>.
- A post from Adil Rahman with the title "Help compiling for RISC-V" and a link to <https://github.com/AdilRahman/json-parser>.
- A post from Adil Rahman with the title "How long should the brief description ..." and a link to <https://github.com/antirez/rax>.
- A post from Alexander Rinaldi with the title "Instr Fix for EESVC VM" and a link to <https://github.com/vnraude/libesvc/>.
- A post from Alexander Rinaldi with the title "Script for controlling the vm" and a link to <https://github.com/woltapp/blurhash>.

At the bottom of the page, there are statistics: Average Response Time: 50 min, Online Now: 7, This Week: 41, and a footer note: Copyright ©2019 Piazza Technologies, Inc. All Rights Reserved. Privacy Policy, Conduit Policy, Terms of Use, Blog, Report Bug.

Raw Report from running it on ESESC

```
CSE220_Arch [Running]
run : bash — Konsole
File Edit View Bookmarks Settings Help

*****
# File : esesc_rax.xAGLQP : Mon Nov 11 15:26:40 2019
*****
Sampler 0 (Procs 0)
    Rabbit Warmup Detail Timing Total KIPS
    KIPS 76010 N/A 461070 964 18345
    Time 10.9% 0.0% 2.0% 87.1% : Sim Time (s) 244.239 Exe 198.574 ms Sim (1700MHz)
    Inst 45.3% 0.0% 50.2% 4.6% : Approx Total Time 4342.530 ms Sim (1700MHz)
*****
Proc : Delay : Avg.Time : BPType : Total : RAS : BPred : BTB : iBTB :
BTAC : WasteRatio
    0 : 2 : 48.947 : 2level : 75.20% : 100.00% of 8.84% : 93.74% of 73.12% : 61.48% of 52.50% : 0.00% :
    0.00% : 14.25%
    0 : 4 : 48.947 : 2level : 90.52% : 0.00% of 0.00% : 88.96% of 85.58% : 93.52% of 0.59% : 49.40% : (
    0.00% fixed) :
-----
Proc : nCommit : nInst : AALU : BALU : CALU : LALU : SALU : LD Fwd : Replay : Worst Unit (clk)
    0 : 182997978 : 182998019 : 45.21% : 19.50% : 0.58% : 20.81% : 13.89% : 4.64% : N/A : 0.00
-----
Proc IPC uIPC Active Cycles Busy LDQ STQ IWin ROB Regs IO maxBr MisBr Br4Clk brDelay
    0 0.54 0.54 1.00 337576226 27.1 0.0 0.1 25.0 0.9 2.1 0.0 0.0 0.0 0.0 4.1
*****
Cache Occ AvgMemLat MemAccesses MissRate ( RD , WR , BUS )
IL1(0) 0.0 2.0 88870495 0.0% 0.0% ( 100.0% , 0.0% , 0.0% ) 28.0 0.0 GB/s
DL1(0) 0.0 12.6 64784869 5.1% 8.0% ( 91.6% , 98.7% , 0.0% ) 20.4 0.0 GB/s
-----
ITLB(0) 0.0 2.0 88870492 0.0% 0.0% ( 100.0% , 0.0% , 0.0% ) 28.0 0.0 GB/s
L2(0) 0.0 94.0 3290387 94.0% 94.0% ( 5.2% , 0.0% , 0.0% ) 1.0 0.0 GB/s
Memory(0) 0.0 60.0 3100193 0.0% 0.0% ( 100.0% , 100.0% , 0.0% ) 1.0 0.0 GB/s
PTLB(0) 0.0 12.7 63513781 3.6% 3.6% ( 96.4% , 0.0% , 0.0% ) 20.0 0.0 GB/s
*****
```

Explanation of bottlenecks and how you found it

- The following bottlenecks are occurring as seen in the initial raw report:
 - **Very low branch prediction success**, with the first 2 level branch predictor only having a 93.74% accuracy and the second 2 level branch predictor only having a 88.96% accuracy.
 - Was found by looking at the report percentage accuracies.
 - **Very low branch target buffer (BTB) success**, with the first 2 level branch predictor only having a 61.48% accuracy and the second 2 level only having a 93.52% accuracy.
 - Was found by looking at the report percentage accuracies.
 - **Very high primary and secondary miss rates at 94.8% in the L2 cache level of the memory hierarchy**. There are 3,290,387 memory accesses actually going into the L2 cache that miss in the L1 cache.
 - Was found by looking at the report percentage accuracies.
 - **Low instruction level parallelism (ILP)**. Since the insertions and removals in the radix tree are not dependent on each other, multiple instructions should be able to be performed simultaneously, however they are limited currently by the architecture.
 - Was found by increasing issueWidth and retireWidth parameters.

Adjusting the branch predictor parameters for bpred

- We tackle the first branch predictor that was using a 2 level as the bpred parameter.
- The code has multiple forward loops and if statements causing branches and jump instructions in the underlying assembly.
- The benchmark loops many times to do several insertions and removals in the radix tree. Inside of these forward loops are if statements to track whether errors have occurred or not.
- Since there is incredibly low BTB success, it was hypothesized that the BTB simply was not big enough to hold enough target PCs for the number of times the code jumped in the benchmark.
- In addition, there may have been some associativity issues that was causing misses in the BTB.

What architectural parameters you changed to improve performance for bpred

- We have decided to increase the size and also decrease the associativity.
 - btbSize went from 64 to 4096.
 - assoc went from 32 to 4.

Parameters for hybrid predictor for bpred

CSE220_Arch [Running]
simu.conf.apache — Kate

File Edit View Projects Bookmarks Sessions Tools Settings Help

simu.conf.apache (2) simu.conf.apache

160
161 [CUNIT_CALU]
162 Num = 2
163 Occ = 1
164
165 [BPredIssueX]
166 type = "2level"
167 #type = "2bit"
168 addrShift = 1
169 #useDolc = true
170 bpred4Cycle = 1 # bpred for cycle
171 BTACDelay = 4 # no BTAC
172 l1size = 1
173 l2size = 2*1024
174 l2bits = 2
175 size = 2*1024
176 Bits = 2
177 historySize = 9
178 Metasize = 2*1024
179 MetaBits = 2
180 localSize = 2*1024
181 localBits = 2
182 #btbSize = 64
183 btbSize = 4096
184 btbBsize = 1
185 #btbAssoc = 32
186 btbAssoc = 4
187 btbReplPolicy = 'RANDOM'
188 btbHistorySize = 9
189 rasSize = 8
190 rasPrefetch = 0
191
192 [BPredIssueX2]
193 type = "2level"
194 addrShift = 1

Filesystem Browser Projects Documents

Line 381, Column 22

INSERT en_US Soft Tabs: 4 UTF-8 Normal

Search and Replace Current Project Terminal

run : bash — Konsole run — Dolphin simu.conf.apache — Kate

3:22 PM Left %

Explanation of how your changes fix the bottleneck

- Increasing the size allows more target PCs to be held, and changing the associativity to make it lower makes it so there are more blocks within an index and increases the chance of hitting in the BTB.
- Making this change results in a BTB success of 98.51%.

Adjusting the branch predictor parameters for bpred2

- We tackle the second branch predictor that was originally using a 2 level as the bpred2 parameter.
- This branch predictor had an incredibly low success rate at 88.96% in branch prediction.
- It was decided that a more sophisticated branch predictor could be used to improve the performance.
- The hybrid predictor was chosen because it could retain more knowledge of how the forward loops and if statements occur in the code. Having bits to keep track of the history would help this along with an increased BTB size and adjusted associativity.

What architectural parameters you changed to improve performance for bpred2

- Since the original simu.config.apache file does not have a hybrid predictor, we established one.
 - The initial hybrid branch predictor was taken from the simu.config file which was located in the home folder with the directory change build/debug/run
- We have decided to increase the history size, increase the BTB size and also decrease the BTB associativity.
 - historySize went from 11 to 17.
 - btbSize went from 64 to 4096.
 - assoc went from 32 to 4.

Parameters for hybrid predictor for bpred2

CSE220_Arch [Running]
simu.conf.apache — Kate

File Edit View Projects Bookmarks Sessions Tools Settings Help

simu.conf.apache (2) simu.conf.apache

```
238 rasSize      = 0
239 numBanks     = 1
240
241
242 [BPredIssueX4]
243 type          = "hybrid"
244 bpred4Cycle   = 4 # bpred for cycle
245 #BTACDelay   = 0 # no BTAC
246 BTACDelay    = 3
247 lisize        = 1
248 l2size        = 16*1024
249 l2Bits        = 1
250 #historySize = 11
251 historySize   = 17 # gets more bits more history to predict at a 0.8 IPC
252 Metasize      = 16*1024
253 MetaBits      = 2
254 localSize     = 16*1024
255 localBits     = 2
256 #btbSize      = 512
257 btbSize       = 4096
258 btbBSIZE      = 1
259 #btbAssoc     = 2
260 btbAssoc      = 4
261 btbRepIPolicy = 'LRU'
262 rasSize       = 0
263 rasPrefetch   = 0
264 # Power Parameters
265 tbits         = 2 # Bits for each table entry
266 tsize         = 16*1024 # Size of each table
267 numBanks     = 1
268
269
270 [BPredIssueX_alt]
271 type          = "ogehl"
272 #type         = "oracle"
273 BTACDelay    = 3
```

Line 381, Column 22

INSERT en_US Soft Tabs: 4 UTF-8 Normal

Search and Replace Current Project Terminal

run : rax.sh — Konsole run — Dolphin simu.conf.apache — Kate

3:26 PM Left %

Explanation of how your changes fix the bottleneck

- Increasing the number of bits to keep track of the history allowed for more success of the branch predictor to track how the code progressed through if statements and forward loops. More information could be retained since there were more bits available to store these patterns.
- Increasing the size allows more target PCs to be held, and adjusting the associativity makes it so there are more blocks within an index and increases the chance of hitting in the BTB.
- Making this change results in a branch prediction accuracy of 97.18% and a success of 99.31% in the BTB.

Why the architectural parameter changes are reasonable for bpred and bpred2

- Increasing the size of the BTB to 4096 and the associativity to 4 is a reasonable change because these parameter settings are seen within the **Intel Ivy Bridge CPU** [4].
- Since branch predictors take up a large portion of the space on CPUs, having this size increase is not unreasonable, especially considering the performance gain from it.
- In addition, having 6 additional bits for the historySize is a very small addition and does not take up much additional space.
- 6 bits should also be very cheap to add in terms of cost.

Adjusting the L1 cache parameters

- We now tackle the L1 cache.
- There are many accesses that are missing the L1 cache and going into the L2 cache causing a high miss rate in the L2 cache. We attempt to minimize the amount of memory accesses that reach the L2 cache by creating more hits in the L1 cache.
- The benchmark itself first tests various insertions of all int keys, then string keys, and then alphanumeric keys.
- There will be many challenges dealing with storing the alphanumeric keys due to the varying byte length between an int (4 bytes) and a char (1 byte).

Adjusting the L1 cache parameters (2)

- However many of these generated keys were first placed into an array. We attempted to take advantage of this since the keys would be still next to each other since these arrays are stored contiguously in memory.
- It is hypothesized that there will be many hits when the keys are consisting of only ints or only chars. The majority of the misses will then potentially occur when alphanumeric keys are generated.
- Varying tests were performed where the associativity was adjusted to values of 1, 2, 4, 8, and 16.
 - bsize had the original value of 32
 - size had the original value of $64 * 1024$

Changes in associativity with capacity as 64 KB and block size as 32 B

bsize	assoc	size	IPC after
32	1	64*1024	0.52
32	2	64*1024	0.6
32	4	64*1024	0.62
32	8	64*1024	0.62
32	16	64*1024	0.62

Changes in associativity with set capacity and block size

- It was seen that increasing associativity did help the IPC. This gave evidence that the type of cache misses that were occurring could be due to conflict misses.
- In order to reduce the possibility of these misses, we decided to increase the associativity so that more of the cache could be properly utilized.
- In addition to this, we decided that increasing the block size to 64 would also help with this. By increasing the block size, more data could fit inside a single block.
- This also would make it more likely that with a given index you would get a hit based off the index and tag bits.

Changes in associativity with capacity as 64 KB and block size as 64 B

bsize	assoc	size	IPC after
64	1	64*1024	0.52
64	2	64*1024	0.61
64	4	64*1024	0.63
64	8	64*1024	0.64
64	16	64*1024	0.64

What architectural parameters you changed to improve performance for DL1_core

- We have decided to increase the block size and also increase the associativity.
 - bsize went from 32 to 64
 - assoc went from 4 to 8.

Parameters for L1 cache for DL1_core

CSE220_Arch [Running]
simu.conf.apache — Kate

File Edit View Projects Bookmarks Sessions Tools Settings Help

simu.conf.apache (2) simu.conf.apache

```
334 replPolicy      = 'LRU'
335 lowerLevel     = "IL1_core IL1"
336 lowerTLB        = "Shared_TLB STLB shared"
337 lowerTLB_delay  = 20
338
339 [DL1_core]
340 deviceType     = 'cache'
341 coreCoupledFreq = true
342 inclusive      = true
343 directory      = false
344 blockName      = "dcache"
345 numBanks       = 1
346 sendFillPort0ccp = 0
347 sendFillNumPorts = 1
348 maxRequests   = 8
349 size          = 64*1024
350 #assoc        = 4
351 assoc          = 8 # Intel Ivy Bridge
352 skew           = false
353 #bsize         = 32
354 bszie          = 64 # Intel Ivy Bridge
355 replPolicy    = 'LRU'
356 bkNumPorts    = 1
357 bkPort0ccp   = 1
358 hitDelay      = 4
359 missDelay     = 4
360 lowerLevel    = "PrivL2 L2 sharedby 2" # I and D cache
361 #lowerLevel   = "${memLevel}"
362 bankShift     = 4
363 fillBuffSize  = 4
364 pfetchBuffSize = 16
365 nlprefetch    = 0
366 wbBuffSize    = 16
367
368 [PrivL2]
369 deviceType     = 'cache'
```

Line 381, Column 22

INSERT en_US Soft Tabs: 4 UTF-8 Normal

Search and Replace Current Project Terminal

run : bash — Konsole run — Dolphin simu.conf.apache — Kate

3:30 PM Left 86

Explanation of how your changes fix the bottleneck

- Increasing the block size helped to take advantage of the spatial locality between the varying keys that were stored contiguously in memory. It also allowed the cache to be arranged in such a way that more hits could occur for a given index.
- Increasing the associativity helped arrange the cache in such a way that more blocks of the cache were being utilized than before, which reduces conflict misses.
- This resulted in the memory accesses going to L2 reducing from 3,290,387 memory accesses to 2,357,703 memory accesses, reducing a big amount of latency cost.

Adjusting the L2 cache parameters

- We now tackle the L2 cache.
- A large majority of the latency is due to misses in the L2 cache, as the miss rate was originally 94.8%.
- It is hypothesized that the alphanumeric keys that are being generated are the majority of the causes for misses due to the varying byte length of the keys depending on how many ints versus chars there were in the key.
- After all of the changes above, only a little more would be required to obtain the 1.5x speedup necessary.
- Varying tests were performed where the associativity was adjusted to values of 2, 4, and 8.
 - bsize had the original value of 32 and was changed to 8
 - size had the original value of $2*1024*1024$ and was changed to $256*1024$

Changes in associativity with capacity as 256 KB and block size as 8 B

bsize	assoc	size	IPC after
8	2	256*1024	0.63
8	4	256*1024	0.75
8	8	256*1024	0.75

What architectural parameters you changed to improve performance for PrivL2

- We have decided to decrease the capacity and the block size and also decrease the associativity.
 - size went from $2*1024*1024$ to $256*1024$
 - bsize went from 32 to 8
 - assoc went from 16 to 4
- In addition to this, since the capacity was decreased, the hit delay and miss delays were reduced.
 - hitDelay went from 30-4 to 10
 - missDelay went from 30-4 to 5

Parameters for L2 cache for PrivL2

CSE220_Arch [Running]
simu.conf.apache — Kate

File Edit View Projects Bookmarks Sessions Tools Settings Help

simu.conf.apache (2) simu.conf.apache

367
368 [PrivL2]
369 deviceType = 'cache'
370 coreCoupledFreq = false
371 inclusive = true
372 directory = true
373 blockName = "L2"
374 numBanks = 1
375 sendFillNumPorts = 1
376 sendFillPort0ccp = 2
377 maxRequests = 32
378 #size = 2*1024*1024
379 size = 256*1024 # Intel Ivy_Bridge
380 #assoc = 16
381 assoc = 4
382 #bsize = 32
383 bsize = 8
384 replPolicy = 'LRU'
385 bkNumPorts = 1
386 bkPort0ccp = 1
387 #hitDelay = 30-4 # 4 for L1 miss delay
388 hitDelay = 10 # Intel Westmere
389 #hitDelay = 12 # Intel Ivy_Bridge
390 #missDelay = 30-4
391 missDelay = 5
392 lowerLevel = "\${memLevel}"
393 #lowerLevel = "L3Cache L3 shared"
394 fillBuffSize = 4
395 pfetchBuffSize = 16
396 nlprefetch = 0
397 wbBuffSize = 16
398 forceLkg = 0.0061637333/2 #0.0184912 #Cacti6.5
399
400 [Shared_TLB]
401 deviceType = 'tlb'
402 blockName = "STLB"

Line 381, Column 22

INSERT en_US Soft Tabs: 4 UTF-8 Normal

Search and Replace Current Project Terminal

run : bash — Konsole run — Dolphin simu.conf.apache — Kate

3:32 PM Left 96

Explanation of how your changes fix the bottleneck

- For the L2 cache, the size was actually decreased quite a bit from the original value of 2 MB to 256 KB. By decreasing the cache size, we were able to deal with much smaller hit and miss delays in the second level cache.
- Thus although the L2 cache miss rate was relatively unchanged, having a smaller cache to search through to allow us to change the delays to be much lower allowed the IPC to increase. Exploring the 2 MB space previously was useless and created a lot of wasted time in just searching the cache that was not going to succeed in obtaining hits.
- This bottleneck was dealt with by just placing a much smaller L2 cache so that the exploration space was smaller. It will still get some advantage of having that 5% hit rate, but take way less time to do so due to the decreased capacity.

Why the architectural parameter changes are reasonable for L1 cache and L2 cache

- For the L1 cache, the size was not touched upon at all. The associativity and the bsize were changed. These values in associativity and bsize are also seen in the **Intel Ivy Bridge CPU** [4]. The latencies for 4 as the hit and miss delay are accurate, as many CPUs with varying sizes of 32 KB or 64 KB take around this much time.
- For the L2 cache, the size decrease allowed these hit and miss delays, which are seen in the **Intel Ivy Bridge as well as the Intel Westmere CPU** [5, 6].
- The associativity and block size are reasonable, as they do not require much hardware complexity to implement.

Improving upon Instruction Level Parallelism

- Lastly, it was recognized that the code itself has very few read after write (RAW) hazards to deal with with every insertion.
- In the insertion process, there is a check to see how to optimize the placing of keys as child nodes in the structure. Even if multiple insertions occur at the same time, future insertions will rearrange the tree such that the tree is correct. The last insertion will also ensure that the entirety of the tree is built and organized correctly.
- Although the instruction fetch is originally set to 4, the issue width and retirement width are only set to 2. This gives suspicion that a bottleneck appears because they are not all equivalent.

What architectural parameters you changed to improve performance ILP

- We have increased the issue width and retirement width to match the instruction fetch.
 - issueWidth went from 2 to 4
 - retireWidth went from 2 to 4

Parameters for tradCORE

CSE220_Arch [Running]
simu.conf.apache — Kate

File Edit View Projects Bookmarks Sessions Tools Settings Help

simu.conf.apache (2) simu.conf.apache

```
10 # MIPS64R4 apache-like configuration
11 [tradCORE]
12 type      = "ooo"
13 areaFactor = 2
14 fetchWidth = 4
15 alignedFetch = true
16 fetchPorts = 1
17 instQueueSize = 24
18 throttlingRatio = 1.0
19 #issueWidth = 2
20 issueWidth = 4
21 #retireWidth = 2
22 retireWidth = 4
23 decodeDelay = 4
24 renameDelay = 4
25 retireDelay = 3 # cycles between execute and retire
26 maxBranches = 32
27 drainOnMiss = true
28 bb4Cycle = 1
29 bpredDelay = 2 #
30 maxIRequests = 3 # +1 icache hit delay -> 1 outs miss
31 interClusterLat = 0 # P4 intra +?
32 clusterScheduler = "RoundRobin"
33 #clusterScheduler = "Use"
34 cluster[0] = 'AUNIT'
35 cluster[1] = 'MUNIT'
36 cluster[2] = 'CUNIT'
37 bpred = 'BPredIssueX'
38 #bpred2 = 'BPredIssueX2'
39 bpred2 = 'BPredIssueX4'
40 robSize = 64
41 stForwardDelay = 4 # +1 clk from the instruction latency
42 maxMemory = 24
43 maxLoads = 24
44 maxStores = 24
45 prefetcher = "PrefetchEngine"
```

Line 381, Column 22

INSERT en_US Soft Tabs: 4 UTF-8 Normal

Search and Replace Current Project Terminal

run : bash — Konsole run — Dolphin simu.conf.apache — Kate

3:33 PM Left

Parameters for tradCORE (2)

CSE220_Arch [Running]
simu.conf.apache — Kate

File Edit View Projects Bookmarks Sessions Tools Settings Help

simu.conf.apache (2) simu.conf.apache

25 retireDelay = 3 # cycles between execute and retire
26 maxBranches = 32
27 drainOnMiss = true
28 bb4Cycle = 1
29 bpredDelay = 2 #
30 maxIRRequests = 3 # +1 icache hit delay -> 1 outs miss
31 interClusterLat = 0 # P4 intra +1?
32 clusterScheduler = "RoundRobin"
33 #clusterScheduler = "Use"
34 cluster[0] = 'AUNIT'
35 cluster[1] = 'MUNIT'
36 cluster[2] = 'CUNIT'
37 bpred = 'BPredIssueX'
38 #bpred2 = 'BPredIssueX2'
39 bpred2 = 'BPredIssueX4'
40 robSize = 64
41 stForwardDelay = 4 # +1 clk from the instruction latency
42 maxMemory = 24
43 maxLoads = 24
44 maxStores = 24
45 prefetcher = "PrefetchEngine"
46 #DL1 = "DL1_core DL1"
47 #IL1 = "IL1_core IL1"
48 DL1 = "PerCore_DTLB PTLB"
49 IL1 = "PerCore_ITLB ITLB"
50 MemoryReplay = false
51 enableICache = true
52 enableDCache = true
53 noMemSpec = false
54 StoreSetSize = 8192
55 instWidth = 32
56 opcodeWidth = 11
57 nArchRegs = 32
58 nTotalRegs = 40 # Some delay penalty for recycling the registers
59 scbSize = 8
60

Line 32, Column 33

INSERT en_US Soft Tabs: 4 UTF-8 Normal

Search and Replace Current Project Terminal

run : bash — Konsole run — Dolphin simu.conf.apache — Kate 3:34 PM Left %

Why the architectural parameter changes are reasonable for issueWidth and retire Width

- It is seen in the **AMD Zen and Zen 2** that have issueWidths of 4 and 8 respectively [7]. Since these exist, it is reasonable to implement these changes.
- retireWidths have also been seen to have been increased from 2 to values of 4 to 8 [7].

What architectural parameters you changed to improve performance

- The following github link contains the simu.conf.apache file with all of the above proposed changes:
 - <https://github.com/rahmanan4/CompArch/blob/master/simu.conf.apache>
- The screenshots in the previous slides also show the proposed changes made.
- Final IPC: 0.82
- $0.82 / 0.54 = 1.52x$ speedup

ESESC report showing final results of improved IPC

```
CSE220_Arch [Running]
run : bash — Konsole
File Edit View Bookmarks Settings Help

*****
# File : esesc_rax.xhw50      : Sun Dec 8 15:29:22 2019
*****
Sampler 0 (Procs 0)
    Rabbit Warmup Detail Timing Total KIPS
    KIPS 62536 N/A 368614 767 14650
    Time 10.6% 0.0% 2.0% 87.4% : Sim Time (s) 311.059 Exe 130.914 ms Sim (1700MHz)
    Inst 45.3% 0.0% 50.2% 4.6% : Approx Total Time 2862.927 ms Sim (1700MHz)
*****
Proc : Delay : Avg.Time : BPType      : Total : RAS      : BPred      : BTB       : iBTB      : BTAC      : WasteR
atio
    0 : 2 : 62.419 : 2level      : 94.63% : 100.00% of 8.84% : 94.61% of 85.19% : 98.51% of 52.56% : 0.00% : 0.00% : 14.36%
    0 : 4 : 62.419 : hybrid      : 97.22% : 0.00% of 0.00% : 97.18% of 85.40% : 99.31% of 53.79% : 0.00% : ( 0.00% fixed) :
-----
Proc : nCommit : nInst : AALU : BALU : CALU : LALU : SALU : LD Fwd : Replay : Worst Unit (clk)
    0 : 182997978 : 182997999 : 45.21% : 19.50% : 0.58% : 20.81% : 13.89% : 7.10% : N/A : 0.00
-----
Proc IPC uIPC Active Cycles Busy LDQ STQ IWin ROB Regs IO maxBr MisBr Br4Clk brDelay
    0 0.82 0.82 1.00 222554220 20.6 0.3 0.1 38.2 1.7 4.0 0.0 0.0 0.0 0.0 2.6
*****
Cache Occ AvgMemLat MemAccesses MissRate ( RD , WR, BUS )
IL1(0) 0.0 2.0 88194635 0.0% 0.0% (100.0%, 0.0%, 0.0%) 42.1 0.0 GB/s
-----
DL1(0) 0.0 10.4 64869168 3.6% 6.6% ( 93.9%, 99.6%, 0.0%) 31.0 0.0 GB/s
-----
ITLB(0) 0.0 2.0 88194632 0.0% 0.0% (100.0%, 0.0%, 0.0%) 42.1 0.0 GB/s
L2(0) 0.0 72.3 2357703 94.9% 94.9% ( 5.1%, 0.0%, 0.0%) 1.1 0.0 GB/s
Memory(0) 0.0 60.0 2240609 0.0% 0.0% (100.0%, 100.0%, 0.0%) 1.1 0.0 GB/s
PTLB(0) 0.0 10.5 63513696 3.7% 3.7% ( 96.3%, 0.0%, 0.0%) 30.3 0.0 GB/s
*****[cse220@arch-cse220 run]$ █
run : bash — Konsole      run — Dolphin      simu.conf.apache — Kate
3:34 PM Left 86
```

References

- [1] “Introduction to Redis”, Redis, 14 November 2019,
<https://redis.io/topics/introduction>
- [2] “Radix tree”, Wikipedia, 18 September 2019,
https://en.wikipedia.org/wiki/Radix_tree
- [3] “Trie”, Wikipedia, 5 November 2019, <https://en.wikipedia.org/wiki/Trie>
- [4] “Inside the Ivy Bridge and Haswell BTB”, Matt Godbolt’s blog, 23 February 2016,
<https://xania.org/201602/haswell-and-ivy-btb>
- [5] “Intel Ivy Bridge”, 7-cpu, 8 December 2019,
<https://www.7-cpu.com/cpu/IvyBridge.html>
- [6] “Intel Westmere”, 7-cpu, 8 December 2019,
<https://www.7-cpu.com/cpu/Westmere.html>
- [7] “Zen - Microarchitectures - AMD”, Wikichip, 8 December 2019,
<https://en.wikichip.org/wiki/amd/microarchitectures/zen>