АРХИТЕКТУРА И ОРГАНИЗАЦИЈА РАЧУНАРА 2
ШКОЛСКА 2020/2021
ПРОЈЕКАТ

Пројекат из предмета Архитектура и организација рачунара 2 се школске 2020/2021 ради самостално. Пројекат носи до 45 поена. У првом и другом делу пројекта постоји могућност за бонус поенима. Максималан бонус износи 5 поена и оцењује се по принципу тачно-нетачно.

Први део пројекта: **Кеш меморија**

Полазећи од имплементације симулатора кеш меморије која је разматрана током курса потребно је имплементирати одређен број алгоритама замене сходно одговарајућем задатку. Постојеће и развијене алгоритме замене је потребно тестирати користећи два независна скупа трагова извршавања. Први траг извршавања је потребно формирати користећи алат *Valgrind* примењен на неку од стандардних апликација, док се други траг извршавања може преузети са локације: *https://crc2.ece.tamu.edu/?page_id=41*. Тестирање је потребно реализовати варирањем величине кеш меморије, броја блокова кеш меморије, сет-асоцијативношћу.

Задаци:

1. Полазећи од постојећег кода симулатора потребно је имплементирати алгоритам под редним бројем: 1 + (*gggg*+bbbb) % 5, где *gggg* представља годину уписа, а *bbbb* број индекса студента.

2. Креирати први траг извршавања користећи *Valgrind* и неки од стандардних апликација. Траг извршавања треба да садржи барем 1.000.000 захтева за приступ меморији.

3. Одабрати два трага извршавања из другог скупа тест трагова. Први одабрати према једначини 1 + (*gggg+bbbb*) % *broj_tragova*, док други треба да буде први наредни траг.

4. Тестирање обавити варирањем величине кеш меморије. Кеш меморија може бити величине 2KB, 8KB, 32KB, 128KB, 512KB, 2MB. Величина блока може бити 16B, 32B, 64B и 128B. Сет асоцијативност 4, 8, 16, 32. Упоредно приказати и коментарисати добијене резултате. За приказ резултата користити *Excel*.

5. **(бонус 5 поена)** Уместо алгоритма одређеног у тачки 1. имплементирати један од алгоритама датих у делу са предлозима. Тестирати га и упоредити са преосталим алгоритмима за који постоји готов код. Направити детаљну документацију за реализован алгоритам, као и конфигурације које су тестиране над алгоритмом.

Алгоритми:

1. *Random* псеудо *LRU* алгоритам (*Ordered selection*)
2. *2Q* алгоритам
3. *LRU-K*
4. *SLRU* алгоритам (*Segmented LRU*)
5. *ARC* (*Adaptive Replacement Cache*)

Предлог 1. (машинско учење)

Испитати могућност да се као алгоритам замене користи неки од алгоритама машинског учења. Варирати параметре датог алгоритма како би се утврдиле оптималне перформансе решења.

Предлог 2. (неуралнa мрежа)

Варирати број слојева и конфигурације неуралне мреже, као и кернела који се користи за рачунање активационе функције.

Предлог 3. (генетски алгоритам)

Предлог како доћи до новог алгоритма замене: Прекидачка функција која описује машину стања по којој се ажурирају бити стања се може реализовати користећи матрицу. Ова матрица садржи информацију на основу којих се користећи информацију да је било сагласности (један бит), кодиране вредности улаза где је пронађена сагласности (n бита), и текућег стања (m бита) чита вредност која представља следеће стање (m бита). Ако није пронађена сагласност онда се на основу те информације (1 бит) као и текућег стања (m бита) одређује улаз за замену (n бита). Код сет асоцијативности 4 и *FIFO* алгоритма замене n је једнако 2, и m је једнако 2. Ова табела за случај да је откривена сагласност ажурира следеће стање тако да буде једнако текућем стању. У случају да нема сагласност враћа за један већу вредност у односу на текућу.

Један од начина за тражење нових алгоритама би био коришћењем неке од хеуристика. Коју би требало понављати велики број пута. На почетку се k (~100) оваквих табела попунити случајним вредностима и убаци се једна табела која одговара *FIFO* алгоритму и једна која одговара псеудо *LRU* алгоритму. Пропусти се кроз алгоритам замене и сортирају се према оствареним резултатима. Од ових се одабере 9% из првих 10% резултата и 1% из преосталих 90% резултата. Онда се од свака од тих одабраних табела начини 9 нових табела помоћу малих модификација (случајна промена неких од улаза). Након овога се алгоритам замене поново примени и тако велики број пута.

Други део пројекта: **Предвиђање скока**

Полазећи од имплементације симулатора система за предвиђање скока која је разматрана током курса потребно је имплементирати одређен број алгоритама предвиђања скока сходно одговарајућем задатку. Постојеће и развијене алгоритме је потребно тестирати користећи два независна скупа трагова извршавања. Први траг извршавања је потребно формирати користећи алат *Valgrind* (потребно је прилагодити подалат *Lackey* – датотеку: *lk_main.c*; или искористи неки други подалат) примењен на неку од стандардних апликација, док се други траг извршавања може преузети са локације:

- http://cis.upenn.edu/~cis501/previous/fall2017/homework/trace-format.html, односно
- http://cis.upenn.edu/~cis501/previous/fall2015/homework/trace-format.html.

Тестирање је потребно реализовати варирањем параметара алгоритма за предвиђање скока.

Задаци:

1. Полазећи од постојећег кода симулатора потребно је имплементирати алгоритме под редном бројем: 1 + (*gggg*+*bbbb* + *i*) % 5, где *gggg* представља годину уписа, *bbbb* број индекса студента, а број *i* може имати вредност 0 или 1.

2. Креирати први траг извршавања користећи *Valgrind* и неки од стандардних апликација. Траг извршавања треба да садржи барем 100.000 условних скокова.

3. Одабрати два трага извршавања из другог скупа тест трагова.

4. Тестирање обавити варирањем параметара датог алгоритма, као и избором различите шема за предвиђање са 2 бита. Упоредно приказати и коментарисати добијене резултате. За приказ резултата користити *Excel*.

5. **(бонус 5 поена)** Предложити нови алгоритам за предвиђање скока. Тестирати га и упоредити са преосталим алгоритмима за који постоји готов код. Предложено решење се сматра успешним ако буде имало већи проценат успешно погођених скокова у односу на реализоване алгоритме из задатка 1.

Алгоритми:

1. *TAGE* предиктор
2. *YAGS* предиктор
3. *gskew* предиктор
4. *e-gskew* предиктор
5. *2bc-gskew* предиктор

Трећи део пројекта: **Оптимизација програма**

Потребно је предложити програм сегмент који ће се оптимизовати на основу познавања архитектуре и организација рачунара. Програмски сегмент треба да има следеће карактеристике:

1. Програмски сегмент је део неког од стандардних програма или библиотека (није наменски писан за овај тест).
2. Програма обавља велики број приступа меморији и скокова.
3. Подаци које користи не могу да стану у кеш меморију.
4. Програм обавља рад са матрицама или другим више димензионим низовима.
5. На програм се може применити техника поделе на блокове.
6. На програм се могу применити барем три векторске инструкције.
7. Векторске инструкције се могу применити унутар петљи.
8. Програмски сегмент има барем 1000 линија кода.
9. Ни један други студент се не бави оптимизацијом датог програмског сегмента.
10. Програмски сегмент је писан у *C/C++* програмском језику.
11. Програмски сегмент није претходно разматран на овом курсу.

Препоручује се коришћење изворног кода неке од библиотека која садржи алгоритме машинског учења. Примери алгоритама су доступни на адресама:

- https://analyticsindiamag.com/top-10-libraries-in-c-c-for-machine-learning/,
- https://hackernoon.com/top-cc-machine-learning-libraries-for-data-science-nl183wo1,
- https://github.com/josephmisiti/awesome-machine-learning.

Задаци:

1. Превести дати оригинални програм са програмским сегментом користећи *GCC* преводилац са подразумеваним подешавањима, као и са подешавањима за брзину рада и утрошак меморије.

2. За неколико, барем две, различите архитектуре и организације рачунара обавити што већи број оптимизација датог програмског сегмента. Коментарисати сваку од оптимизација.

3. За сваку од примењених оптимизација посебно превести програм.

4. Измерити време извршавања програмског сегмента за сваки од ових случајева. Упоредно приказати и коментарисати добијене резултате. За приказ резултата користити *Excel*.