Проект 2: Откривање на асоцијативност на Л1-кеш на ГПУ со архитектура Ферми

Димитриј Мијоски

Универзитет Св.Кирил и Методиј

Факултет за Информатички науки и Компјутерско Инженерство

111132, [mijoski.dimitrij@students.finki.ukim.mk](mailto:mijoski.dimitrij@students.finki.ukim.mk)

Abstract

In the second project we use a method that consists of one micro-benchmark and analysis of the measured data to discover characteristic of memory cache like cache size, cache line size, number of sets and associativity (lines inside set). The public data from the manufacturer states only the first two parameters (total size and line size).

Keywords

Memory, cache, cache associativity, GPU.

Резиме

Во вториов проект користиме метод кој се состои од еден микробенчмарк и од анализа на измерените податоци, за да откриеме карактеристики на мемориски кеш како големина на кеш, големина на кеш-линија, број на множества (комплети) и асоцијативност (број на линии во множество). Јавните податоци од производителот ги опишуваат само првите два параметри (вкупна големина и големина на линија).

Клучни зборови

Меморија, кеш, асоцијативност на кеш, ГПУ.

# Вовед

Кај компјутерските системи од поново време наменети за брза обработка на податоци меморијата е тесно грло – процесорот може да обработува податоци многу побрзо отколку што меморијата го опслужува. Истиот проблем се јавува и кај ГПУ. Јасно, проблемот се решава со намалување на трансферите на податоци од меморијата до процесорот. Ако имаме податоци кои се користат повеќе пати, тогаш тие би ги користеле додека се наоѓаат во процесорот. Сепак, меморијата врз која прецесорот работи директно, односно регистрите, е многу ограничена и се мери во бајти. Заради тоа изградена е мемориска хиерархија каде блиску до процесорот ставаме мали и брзи мемории, а подалеку ставаме поголеми но и побавни мемории. Конкретно кај архитектурата *Fermi* (Ферма) помеѓу графичкиот процесор имаме две нивоа на кеш. Првото ниво, Л1 се наоѓа на самиот мултипроцесор, додека второто ниво е заедничко за повеќе мултипроцесори. Кешовите можат да бидат од неколку пати па се до стотици пати побрзи од главната меморија. Заради ваквите добивки на перформанси тие се честа тема на истражување од областа компјутерски архитектури.

# Архитектура

Прво на кратко ќе ја опишеме архитектурата на модерните графички од Енвидија.

Од логичка гледна точна, една паралелна рутина се вика и јадро (*kernel*). Едно јадро содржи инстукции и при повик се назначува големина на мрежа од блокови и големина на блок со нишки. Различни блокови може да се извршуваат на различни мултипроцесори (може и на ист) по неопределен редослед. Еден блок се извршува на ист мултипроцесор.

Сега, еден блок може да има најмногу 1024 нишки, а еден мултипроцесор има релативно мал број на т.н. КУДА-јадра. Има 8 на архитектура Тесла (верзии 1.0 до 1.3), 32 на Ферми итн. Тоа значи тие 1024 нишки не можат одеднаш сите да се извршат паралелно, туку има некакво распоредување. Има уште една поделба на блокот по ворпови каде еден ворп содржи 32 последователни нишки. Еден ворп е најмала извршувачка единица и нишките во еден ворп **во еден момент** извршуваат **иста инструкција.** Кога една инструкција од ворп ќе дојде на ред, распоредувачот го назначува тој ворп над функциските единици (КУДА-јадра) во онолку циклуси колку што е потребно. Пример ако имаме 8 јадра, ворпот ќе биде целосно **издаден** за 4 такт циклуси.

Кога ќе дојде инструкција за мемориски трансфер до или од глобалната меморија, ворпот се не назначува на КУДА-јадрата, туку на слична низа од јадра (функциски единици) за мемориски трансфери. Барањата до секоја единица се збор што нишката го бара и тој може да биде 1, 2, 4, 8 или 16 **бајтен** збор. Потоа сите тие барања, ако е можно (ако се последователни и соодветно порамнети), се спојуваат во една **голема 128-бајтна мемориска трансакција** кај Ферми. Таа трансакција се проследува до кешот, а ако не се најде линијата се проследува до меморискиот контролер кој треба да ги собере тие 128 бајти во онолку такт циклуси колку што е потребно (тоа зависи од фреквензијата на меморијата и од ширината на нејзината магистрала).

Дека мемориската хиерархија е многу осетлива работа ни кажува тоа што алгоритмите за пристапување на меморијата кај неколкуте современи архитектури, од Тесла па наваму, се разликуваат кај сите архитектури помалку или повеќе. Кај првата архитектура, Тесла со верзија 1.0 и 1.1, нема кеш, има спојување на барање во поголеми трансакции но не е воопшто паметно. Трансакциите се со големина од 32 и 64 бајти. Кај верзија 1.2 и 1.3 има попаметно спојување во трансакции и постојат 128 бајтни, 64 бајтни и 32 бајтни трансакции. Кај Ферми (тоа е верзија 2.0 и 2.1) пак веќе имаме кеш и тоа две нивоа. Трансакциите што носат меморија во Л1 се 128 бајти, а кај Л2 се 32 бајтни. Кај поновите има уште некои помали разлики. За подетален опис најдобро е да се види официјалното упатство (*CUDA Programming Guide*, глава 4, глава 5 и додаток *C*).

Л1-кешот може да се постави да биде 16 килобајти или 48 килобајти. Ние меревме со 16 килобајти.

# Методологија за мерење

Искористена е методологија опишана во труд (Henry Wong, Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos, *Demystifying GPU Microarchitecture through Microbenchmarking*).

Работиме само со една нишка. Изминуваме една низа од почеток до крај **барем два пати**. При првото изминување, сите кеш-линии ќе направат по едно промашување. При второто изминување, ако целата низа ја збира во кешот, тоа ќе биде брзо. Ако низата е малку поголема и опфаќа **една кеш-линија повеќе**, тогаш уште при првото изминување таа последна линија ќе се преслика во првото множество на местото од првата линија и ќе ја поништи првата линија. Веднаш потоа одиме на првата линија, но таа е веќе поништена од кешот па ќе направи промашување и ќе ја замени следната линија во првото множество. Кога таа заменетата ќе дојди на ред за читање и таа ќе направи промашување.

Значи ако низата е само малку поголема од кешот, сите линии што се пресликуваат во првото множество ќе бидат промашувања при второто изминување. Линиите што се пресликуваат во останатите множества ќе бидат погодоци. Како и да е, во вкупното време ќе имаме значителен скок.

Ако низата е опфатена во целиот кеш плус две други линии, при второто изминување ќе има промашување и во првото и во второто множество.

Ние оваа цела постапка на две изминувања ја мериме со прецизни хардверски бројачи во такт циклуси. Таа бројка ја делиме со бројот на пристапи и добиваме просечно време во циклуси по пристап т.е. добиваме циклуси по инструкција (CPI). За големини на низата за кои таа целата влегува во кешот ЦПИ ќе биде исто и ќе биде најмало. Веднаш кога низата ќе порасне една линија плус, за тие големини за кои зафаќа само една линија плус, одеднаш ЦПИ ќе рипне. Кога почне да зафаќа две линии плус ЦПИ пак ќе рипне. Сега важи:

1. Крајната точка на големина без скок ја означува големината на кешот.
2. Разликата на големини на низа од еден скок до друг скок е големина на кеш-линија.
3. Бројот на скокови на ЦПИ е бројот на множества.
4. Асоцијативноста се добива ако големината на кешот ја поделаме со големината на линија и бројот на множества.

# Резултати

Затоа што знаме дека кешот е голем 16 килобајти (16384 бајти), тргнавме со низа од 4096 елементи (секој елемент е 4 бајти). Таа низа целосно ја збира. Правевме експерименти со големини до 8192 елемента. Ова се резултатите за ЦПИ во однос на големина на низата.

Слика 1. Измерени ЦПИ за големини од 4096 до 8192.

Слика 2. Попрецизен график со измерени ЦПИ за големини од 4096 до 5244.

За Л1 кешот нас не интересира вториот график. Од еден до друг скок имаме разлика на големини од 32 елемента односно 32 \* 4 бајти = 128 бајти со што се потврдува јавно објавениот податокот дека кеш линија во Л1 е 128 бајти. Имаме 32 скока што значи дека кешот има 32 множества. 4096 елементи / 32 множества = 128 елементи во едно множество. 128 елементи во множество / 32 елементи во линија = 4 линии во множество. Тоа е асоцијативноста што требаше да ја откриеме.

На слика 1 можеме да видиме две скалести криви. Втората скала очигледно е заради некој друг кеш (можеби Л2) што не ни е тема на истражување на проектов.

# Заклучок

Како што видовме во глава 1 и 2, мемориската хиерархија е осетлива и може многу да влијае врз перформансите на еден алгоритам. Нејзино добро познавање секако е од полза при пишување алгоритам кој треба брзо да се извршува. Со откривањето на асоцијативноста овде појаснивме еден аспект повеќе од таа хиерархија.