به نام خداوند بخشنده مهربان

معماری کامپیوتر

دانشکده مهندسی کامپیوتر

دکتر اسدی
بهار ۱۴۰۳

مهدی علی نژاد، ۴۰۱۱۰۶۲۶۶ و امیرحسین صوری، ۴۰۱۱۰۶۱۸۲

گزارش تمرین نهم بخش عملی

سوال ۱

۱. تمرین عملی اول

در این تمرین قصد داریم با انجام چند آزمون با شبیه‌ساز Gem5 و نحوه کار با این شبیه‌ساز آشنا شویم. برای کار با این شبیه‌ساز می‌توانید از ماشین مجازی که لینک آن در صفحه درس در CW قرار داده شده است و یا build کردن شبیه‌ساز Gem5 روی سیستم خودتان استفاده کنید. (در صورتی که روش دوم را انتخاب می‌کنید، توصیه می‌شود از یک سیستم‌عامل Unix-Based استفاده کنید.)

صورت تمرین

در این تمرین شما باید دو نمونه کد جمع و دو نمونه کد ضرب ماتریس را به وسیله شبیه‌ساز Gem5 اجرا کنید و نتایج آن‌ها را بررسی نمایید.

۱. **جمع دو ماتریس:** دو کد Adder1 و Adder2 را که در فایل add1.c و add2.c قرار دارند، در نظر بگیرید. این دو تابع که دو روش مختلف برای جمع دو ماتریس است را از لحاظ locality مقایسه کنید و قبل از انجام شبیه‌سازی، پیش‌بینی کنید که کدام عملکرد بهتری دارد. همچنین پس از اجرای شبیه‌سازی با بررسی فایل stats.txt عملکرد این دو کد را از لحاظ عملکرد L1DCache مقایسه کنید.

۲. **ضرب دو ماتریس:** کد ضرب ماتریس را در فایل mul.c در نظر بگیرید. روش کار این کد و عملکرد آن را با شبیه‌سازی در Gem5 بررسی کنید. همچنین با توجه به وجود حافظه L1DCache در شبیه‌سازی که انجام می‌دهید، یک پیاده‌سازی جدیدی برای ضرب دو ماتریس ارائه دهید. سپس کد جدید را با Gem5 شبیه‌سازی کنید. در نهایت فایل stats.txt را برای دو کد بررسی، تحلیل و مقایسه کنید.

راهنمایی

۱. درهنگام شبیه‌سازی در صورتی که کد کامپایل شده مناسب معماری X86 است، از X86TimingSimpleCpu استفاده کنید و اگر معماری سیستم شما ARM است، از ARMTimingSimpleCpu استفاده کنید.

۲. در شبیه‌سازی که انجام می‌دهید از دو لایه حافظه نهان استفاده کنید، به صورتی که حافظه نهان لایه اول برای داده و دستورالعمل‌ها به صورت جداگانه و حافظه نهان لایه دوم به صورت مشترک برای هر دو باشد.

۳. برای مشاهده ملموس تغییر نتایج در بین نمونه‌کدهای مختلف، لازم است که به اندازه ماتریس‌های و همچنین حافظه نهان توجه کنید.

۴. توصیه می‌شود در کدی که شبیه‌سازی می‌کنید خروجی در stdout نداشته باشید.

گزارش

گزارش شما باید شامل تمام کدهایی که به زبان C پیاده‌سازی کرده‌اید و همچنین شامل تحلیل‌های شما روی پارامترهای خروجی شبیه‌سازی انجام شده، باشد.

- build کردن Gem۵

طبق توضیحات گفته شده در این فیلم آموزشی YouTube از لینک زیر کلون کرده و با دنباله دستورات زیر آن را build می کنیم.

```
> git clone https://gem5.googlesource.com/public/gem5
> cd gem5
> scons build/X86/gem5.opt -j<number of threads>
```

- setup

در این تست سید رندوم به صفر ست شده و سایز ماتریس ها ۱۲۸ در ۱۲۸ است.

- کد های جمع با محلیت های متفاوت

کد اول دارای لوکالیتی مکانی کمتری است، این کد بر روی ستون های ماتریس حلقه می زند که برای حافظه، محلیت بدی حساب می شود زیرا خانه های ماتریس به صورت سطری کنار هم قرار گرفته اند.

```
matrix unoptimized_matrix_add(matrix a, matrix b, int m, int n)
{
    matrix c = matrix_rand_init(m, n);
    for (int j = 0; j < n; j++)
    {
        for (int i = 0; i < m; i++)
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    return c;
}
```

کد دوم ولی دارای لوکالیتی مکانی بیشتری است زیرا بر روی سطر های ماتریس حلقه می زند. پس پیش بینی می شود این کد بهتر عمل کند.

```
matrix optimized_matrix_add(matrix a, matrix b, int m, int n)
{
    matrix c = matrix_rand_init(m, n);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    return c;
}
```

پس از شبیه سازی کدها با شبیه ساز gem۵ به این نتایج می رسیم. حواسمان هست که سایز کش را به اندازه ای تعیین کنیم که تمام یک سطر ماتریس در یک بلاک جا نشود و حتما باعث miss شود.

```
system.cpu.dcache.overallMissRate::total       0.097139                       # miss rate for overall accesses (Ratio)
```

miss rate حاصل از کد جمع اول.

```
system.cpu.dcache.overallMissRate::total       0.085956                       # miss rate for overall accesses (Ratio)
```

miss rate حاصل از کد جمع دوم.

همانطور که مشاهده می شود، miss rate کد دوم کمتر از کد اول است پس لوکالیتی بهتری دارد.

- کد ضرب

کد ضرب داده شده، از یک ماتریس به صورت ستونی می خواند که همانطور که در جمع دیدیم، این حالت باعث لوکالیتی بد می شود، پس miss rate بیشتری خواهیم دید.

```
matrix unoptimized_matrix_mul(matrix a, matrix b, int l, int m, int n)
{
    matrix c = matrix_zero_init(l, n);
    for (int i = 0; i < l; i++){
        for (int j = 0; j < n; j++){
            for (int k = 0; k < m; k++){
                c[i][j] += a[i][k]*b[k][j];
            }
        }
    }
    return c;
}
```

کد ضرب دو ماتریس.

```
system.cpu.dcache.overallMissRate::total       0.100912                       # miss rate for overall accesses (Ratio)
```

miss rate حاصل از آن.

```
matrix optimized_matrix_mul(matrix a, matrix b, int l, int m, int n) {
    matrix c = matrix_zero_init(l, n);
    for (int i = 0; i < l; i++){
        for (int j = 0; j < m; j++){
            for (int k = 0; k < n; k++){
                c[i][k] += a[i][j]*b[j][k];
            }
        }
    }
    return c;
}
```

کد ضرب بهینه، این کد به این دلیل که درونی ترین حلقه حالا بر روی سطر های ماتریس است دچار miss کمتری شود.

```
system.cpu.dcache.overallMissRate::total       0.050950                       # miss rate for overall accesses (Ratio)
```