دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

عنوان:

تمرین شبیه‌سازی سری ۲

نگارش

رضا آدینه پور - ۴۰۲۱۳۱۰۵۵

استاد راهنما

جناب آقای دکتر فربه

۱۰ آبان ۱۴۰۲

• سوال اول

دو شبه کد زیر را برای مسئله ضرب ماتریس‌ها در نظر بگیرید و گام‌های زیر را با توجه به آن کامل کنید.

• الگوریتم اول:

Algorithm 1 Matrix multiplication

Require: Input Matrix A and B
Require: Let C be a new matrix of the appropriate size
  for i from 1 to n: **do**
    for j from 1 to p: **do**
      $sum \leftarrow 0$
      for k from 1 to m: **do**
        $sum \leftarrow A_{ik} \times B_{kj}$
      end for
    end for
  end for
  return C

• الگوریتم دوم:

Algorithm 2 Matrix multiplication

Require: Input Matrix A and B
Require: Let C be a new matrix of the appropriate size
Require: Pick a tile size $T = \Theta(\sqrt{M})$
  for I from 1 to n: **do**
    for J from 1 to p: **do**
      for K from 1 to m: **do**
        $A_{I:I+T,K:K+T} \times B_{K:K+T,J:J+T} \times C_{I:I+T,J:J+T}$
        for i from I to min(I+T, n) **do**
          for j from J to min(J+T, p) **do**
            $sum \leftarrow 0$
          end for
          for k from K to min(K+T, m) **do**
            $sum \leftarrow sum + A_{ik} \times B_{kj}$
          end for
          $C_{ij} \leftarrow C_{ij} + sum$
        end for
      end for
    end for
  end for
  return C

• گام اول:

شبه کد های آورده شده را به کد Python یا C++ تبدیل کرده و آن‌ها را کامپایل و فایل کامپایل شده را بر روی یک سیستم ساده فاقد حافظه نهان در شبیه‌ساز Gem5 اجرا کنید. برای درک بهتر تفاوت ها و هدف این تمرین، نیاز است ماتریس‌هایی با بیش از ۱۰۰ سطر و ستون تعریف کنید. مقادیر مربط به درایه‌های ماتریس‌ها را با تابع Rand تولید کنید. برای این گام باید

کدهای مربوطه و تصاویر اجرای کدها بر روی Gem5 و نیز فایل stats تولید شده را درقالب یک فایل زیپ با نام step1 در فایل نهایی پاسخ خود قرار دهید.

● **راهنمای گام اول:**

برای اجرای کد بر روی شبیه‌ساز از دستور زیر استفاده می‌شود:

```
build/X86/gem5.opt[path of the config file][path of your compiled code]
```

برای پیکربندی شبیه‌ساز به صورت یک سیستم فاقد حافظه نهان، باید فایل config مربوطه را اجرا کنید. آموزش نحوه کدنویسی و ایجاد این فایل در آدرس زیر، اینجا موجود است.

شما می‌توانید درصورت تمایل، توضیحات را مطالعه و کدهای این فایل را بررسی کنید. در غیر این صورت می‌توانید از فایل آماده آن که در مسیر configs/learning_gem5/part1/simple.py وجود دارد استفاده کنید.

● **گام دوم:**

فایل stats مربوط به کدها را بررسی کرده و پارامترهای زیر را برای دو کد پیدا کرده و در گزارش خود بیاورید. فایل stats را می‌توانید در مسیر gem5/m5out پیدا کنید.

- IPC
- Number of busy cycles

● **گام دوم:**

در این گام باید کدهای مربوط به دو شبه‌کد را بر روی یک سیستم دارای دو سطح حافظه نهان L۱ و L۲ اجرا کنید. برای این گام مد پردازنده را Timing Simple CPU درنظر بگیرید. Associativity حافظه نهان لایه اول داده و دستورالعمل را ۴ و حافظه نهان لایه دوم را ۸ و نیز اندازه هر بلاک را ۶۴ درنظر بگیرید. سایز حافظه نهان لایه اول و دوم اختیاری است. اما باید در گزارش ذکر شود. باید تصاویر اجرای کدها بر روی gem5 و نیز فایل stats تولید شده را در قالب یک فایل Zip با نام step2 در فایل نهایی پاسخ خود قرار دهید.

- IPC
- Number of busy cycles
- L1i cache hit and miss rate
- L1d cache hit and miss rate
- L2 cache hit and miss rate

● **راهنمای گام اول:**

برای اینکه gem5 را به‌صورت یک سیستم دارای حافظه نهان پیکربندی کنید، از فایل config موجود در مسیر configs/deprecated/example/se.py استفاده کنید. این فایل به شما اجازه می‌دهد که حافظه نهان L۱ و L۲ را تعریف کنید و سایز هرکدام و نیز Associativity آن‌ها و دیگر پارامترهای مرتبط را تعریف و مقداردهی کنید. می‌توانید از دستورات زیر برای این منظور استفاده کنید:

- `--cpu-type=TimingSimpleCPU||DerivO3CPU||MinorCPU`
- `--cache` → `enable L1 cache.`
- `--L2cache` → `enable L2 cache.`
- `--L1d_size = Size`
- `--L1i_size = Size`

- `--L2_size = Size`
- `--L1d_assoc = Associativity`
- `--L1i_assoc = Associativity`
- `--L2_assoc = Associativity`
- `--cacheline_size = cache line (block) size`

• گام چهارم:
فایل `stats` مربوط به کدها را بررسی کرده و پارامترهای زیر را برای دو کد پیدا کرده و در گزارش ارسالی خود بیاورید.

- IPC
- Number of busy cycles
- L1i cache hit and miss rate
- L1d cache hit and miss rate
- L2 cache hit and miss rate

• گام پنجم:
در این گام ابتدا اختلاف مقادیر بدست آمده در گام دوم و چهارم را مقایسه کنید و در ادامه علت اختلاف مقادیر پارامترهای گام چهارم برای دو شبه‌کد را تحلیل کنید. (درواقع باید تفاوت عملکرد دو کد را بررسی و تاثیر آن بر رفتار سیستم را بیان کنید)

• سوال دوم
در پوشه تمرین یک برنامه با نام `sample.c` قرار دارد که یک لیست پیوندی با اندازه ۱۰۰۰۰ ایجاد کرده و سپس با استفاده از یک تابع بازگشتی، لیست ایجاد شده را معکوس می‌کند. این برنامه را در `gem5` و با حالت `se` اجرا کنید و گام‌های زیر را کامل کنید:

• گام اول:
در دو نوع پردازنده Atomic Simple Cpu و DerivO۳CPU پارامترهای IPC و Sim Second را گزارش کنید و دلیل تفاوت در مقدار این پارامترها برای دو نوع پردازنده را ذکر کنید.

• گام دوم:
سعی کنید تابع مورد استفاده به منظور معکوس کردن لیست پیوندی را به شکلی تغییر دهید که عملکرد سیستم را از نظر پارامتر IPC بهبود دهد. با ذکر دلیل مشخص کنید که در کدام نوع پردازنده بهبود بیشتری می‌توان ایجاد کرد؟

• سوال سوم
بررسی کنید که سه Mode مربوط به سیستم حافظه یعنی Timing mode و Atomic mode و Functional mode چه تفاوتی دارند و هرکدام برای چه منظور در شبیه‌سازی استفاده می‌شوند؟ همچنین فایل `config` استفاده شده در گام اول سوال اول را بررسی کرده و مقادیر پارامترهای زیر را در گزارش خود بیاورید.

- فرکانس کلاک
- مد شبیه‌ساز حافظه
- سایز حافظه

۱ پاسخ سوال اول

در این قسمت به بررسی نتایج بدست آمده در سوال اول می‌پردازیم. همچنین شما می‌توانید کدهای نوشته شده در این بخش را در اینجا یا در انتهای گزارش در «» مشاهده کنید. در هر دو کد، ماتریس A را 150×150 و ماتریس B را 200×150 تعریف کردیم.

۱.۱ گام اول

۱.۱.۱ شبه‌کد اول (پایتون)

در مرحله اول ابتدا کد پایتون را تست می‌کنیم. برای انجام این مرحله ابتدا باید فایل با پسوند py. را به فایل execute تبدیل کنیم. برای انجام این کار ابتدا با دستور زیر کتابخانه مورد نیاز برای انجام این کار را نصب میکنیم:

```
pip install pyinstaller
```

پس از نصب کتابخانه با دستور زیر می‌توان فایل execute را تولید کرد:

```
pyinstaller --onefile [your file name]
```

حالا فایل config مورد نیاز با شبیه‌سازی خودمان را می‌سازیم. از دایرکتوری زیر

```
gem5/configs/learning_gem5/part1
```

فایل sample.py را باز کرده و در قسمت

```
binary = os.path.join()
```

مسیر فایل execute خود را وارد می‌کنیم.

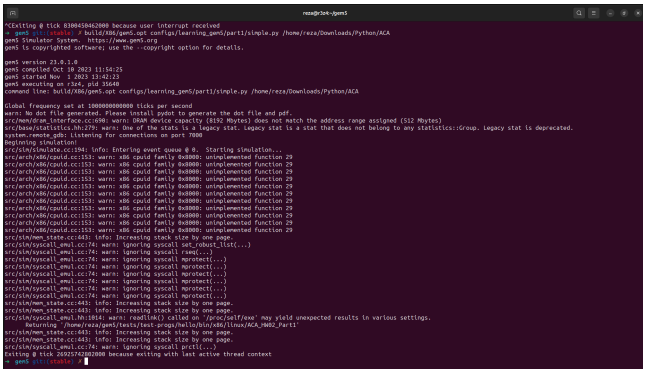
```
# Here we set the X86 "hello world" binary. With other ISAs you must specify
# workloads compiled to those ISAs. Other "hello world" binaries for other ISAs
# can be found in "tests/test-progs/hello".
thispath = os.path.dirname(os.path.realpath(__file__))
binary = os.path.join(
    thispath,
    "../../../",
    "tests/test-progs/hello/bin/x86/linux/ACA_HW02_Part1",
)

system.workload = SEWorkload.init_compatible(binary)
```

شکل ۱: تغییر مسیر فایل کامپایل شده Python

سپس با دستور زیر شبیه‌سازی را اجرا می‌کنیم:

```
build/X86/gem5.opt configs/
learning_gem5/part1/simple.py
```

پس ار تکمیل فرایند شبی‌سازی که قدری زمان‌بر بود خروجی به صورت زیر است:



شکل ۲: خروجی شبیه‌سازی اول با Python

متاسفانه شبیه‌سازی با فایل python به‌درستی کار نکرد و نتوانستم مشکل آن‌را پیدا کنم. پس ادامه شبیه‌سازی ها را با کد C++ انجام می‌دهیم.

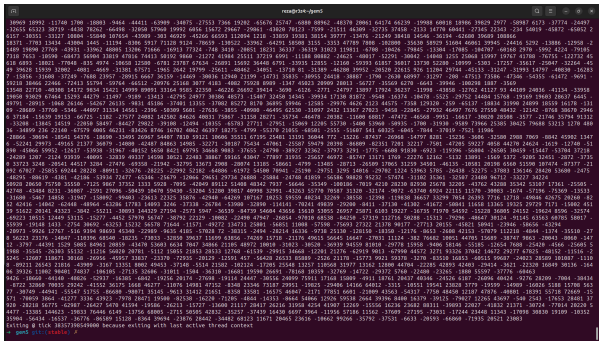
۲.۱.۱ شبه‌کد اول (C++)

مشابه با کد پایتون، در فایل config مسیر فایل کامپایل شده C++ را می دهیم.

```
# Here we set the X86 "hello world" binary. With other ISAs you must specify
# workloads compiled to those ISAs. Other "hello world" binaries for other ISAs
# can be found in "tests/test-progs/hello".
thispath = os.path.dirname(os.path.realpath(__file__))
binary = os.path.join(
    thispath,
    "../../../../",
    "Downloads/Python/ACA_HW02_Part1/ACA_HW02_Part1",
)

system.workload = SEWorkload.init_compatible(binary)
```

شکل ۳: تغییر مسیر فایل کامپایل شده C++

خروجی شبیه‌ساز برای شبه‌کد اول C++ به صورت زیر است:



شکل ۴: خروجی شبیه‌سازی اول با C++

۲.۱ گام دوم

۱.۲.۱ شبه‌کد اول (Python)

در گام بعدی به سراغ فایل stats می‌رویم و پارامترهای خواسته شده را گزارش می‌کنیم.

- IPC
- Number of busy cycles

۲.۲.۱ شبه‌کد اول (C++)

پارامترهای خواسته شده برای کد C++ به صورت زیر است:

- IPC = 0.011162 $\frac{count}{cycle}$
- Number of busy cycles: 38357398548.999001

۳.۲.۱ شبه‌کد دوم (C++)

پارامترهای خواسته شده برای کد C++ به صورت زیر است:

- IPC = 0.009821 $\frac{count}{cycle}$
- Number of busy cycles: 59261119214.999001

۳.۱ گام سوم

فایل کانفیگ two_level.py را از اینجا دانلود کردیم و از این فایل به عنوان فایل کانفیگ برای این بخش استفاده می‌کنیم.
مسیر فایل کامپایل شده را به صورت زیر در فایل two_level.py تغییر می‌دهیم:

```
# Default to running 'hello', use the compiled ISA to find the binary
# grab the specific path to the binary
thispath = os.path.dirname(os.path.realpath(__file__))
default_binary = os.path.join(
    thispath,
    "../../../../",
    "Downloads/Python/ACA_HW02_cpp/ACA_HW02_Part1",
)
```

شکل ۷: تغییر مسیر فایل کد C++ کامپایل شده

۱.۱.۱ شبه‌کد دوم (C++)

برای شبه‌کد دوم، صرفا کد C++ آن را تست می‌کنیم. مسیر فایل را در فایل config تنظیم می‌کنیم.

```
# Here we set the X86 "hello world" binary. With other ISAs you must specify
# workloads compiled to those ISAs. Other "hello world" binaries for other ISAs
# can be found in "tests/test-progs/hello".
thispath = os.path.dirname(os.path.realpath(__file__))
binary = os.path.join(
    thispath,
    "../../../../",
    "Downloads/Python/ACA_HW02_cpp/ACA_HW02_Part2"
)

system.workload = SEWorkload.init_compatible(binary)
```

شکل ۵: تغییر مسیر فایل کامپایل شده C++

سپس شبیه‌سازی را اجرا می‌کنیم:

شکل ۶: خروجی شبیه‌سازی دوم با C++

۱.۳.۱ شبیه‌سازی شبه‌کد اول

با دستور زیر، شبه‌کد اول را شبیه‌سازی می‌کنیم:

```
build/X86/gem5.opt configs/
learning_gem5/part1/two_level.py
--l1d_size=128kB --l1i_size=64kB
--l2_size=1MB
```

در این بخش سایز L۱ Cache را ۶۴ کیلوبایت و سایز L۲ Cache را ۱ مگابایت درنظر گرفتیم.
خروجی شبیه‌سازی به صورت زیر است:

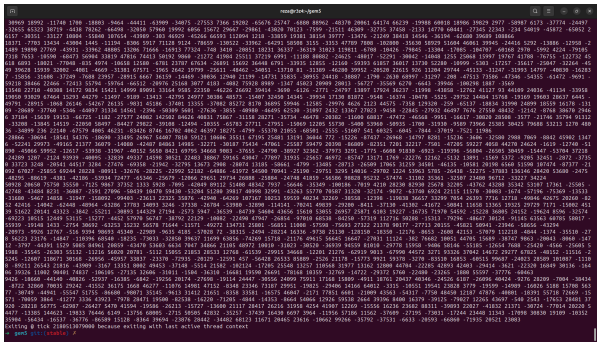
شکل ۸: خروجی شبیه‌سازی شبه‌کد اول

۲.۳.۱ شبیه‌سازی شبه‌کد دوم

مسیر فایل کامپایل شده را در فایل کانفیگ تغییر داده و با دستور زیر کد را شبیه‌سازی می‌کنیم: سایز Cache ها همان مقدار قسمت قبل هستند.

```
build/X86/gem5.opt configs/
learning_gem5/part1/two_level.py
--l1d_size=128kB --l1i_size=64kB
--l2_size=1MB
```

خروجی شبیه‌سازی به صورت زیر است:



شکل ۹: خروجی شبیه‌سازی شبه‌کد دوم

۴.۱ گام چهارم

۱.۴.۱ شبه‌کد اول

- IPC = 0.263273 $\frac{count}{cycle}$
- Number of busy cycles: 1626196851.999000
- L1i cache hit rate: not found
- L1i cache miss rate: not found
- l1d cache hit rate: not found
- l1d cache miss rate: not found
- l2i cache hit rate: 169
- l2d cache hit rate: 26420
- l2i cache miss rate: 2129
- l2d cache miss rate: 14245

۲.۴.۱ شبه‌کد دوم

- IPC = 0.266907 $\frac{count}{cycle}$
- Number of busy cycles: 2180513078.999000
- L1i cache hit rate: not found
- L1i cache miss rate: not found
- l1d cache hit rate: not found
- l1d cache miss rate: not found
- l2i cache hit rate: 154
- l2d cache hit rate: 20844
- l2i cache miss rate: 0.932692
- l2d cache miss rate: 0.405934

۵.۱ گام پنجم

در هر دو شبه‌کد از یک تابع تولید اعداد تصادفی استفاده شده است. و ماتریس‌های بکار رفته در هر دو کد یمسان است.
در شبه‌کد اول صرفا هر سطر را در ستون ماتریس متناظرش ظرب کرده و حاصل ضرب هر درایه را با مقدار قبلی خودش جمع میکنیم. اما در شبه‌کد دوم، الگوریتم نوشته شده برای ضرب دو ماتریس بسیار پیچیده‌تر از الگوریتم اول است.
می‌دانیم که بار محاسباتی حلقه های تو در تو برای سخت افزار بسیار سنگین است و تاجایی که ممکن است باید از نوشتن حلقه‌های تو در تو غیر ضروری پرهیز کنیم. پس می‌توان یکی از معیارهای مقایسه این دو شبه‌کد را حلقه‌های تو در تو قرار داد.
در شبه‌کد اول از ۳ حلقه `for` تو در تو استفاده شده است اما در شبه‌کد دوم از ۶ حلقه استفاده شده است. بنابر این می‌توان گفت که شبه‌کد دوم از نظر Performance کند تر و بار محاسباتی بیشتری را بر سیستم متحمل می‌شود و در بهترین حالت (با فرض برابر بودن باقی قسمت‌های کد) انتظار داریم شبه‌کد دوم حداقل ۲ برابر کندتر از شبه‌کد اول باشد.
یکی از پارامترهای مقایسه `performance` دو پردازنده، `Instruction Per Cycle` یا با اختصار `IPC` است. این پارامتر، نشان دهنده میانگین `instruction` های انجام شده توسط `cpu` در هر کلاک است.
پارامتر دیگری که برای مقایسه `performance` مورد استفاده قرار می‌گیرد، `Clock Per Instruction` یا همان `CPI` است. این پارامتر تعداد کلاک های زده شده برای انجام هر `instruction` را نشان می‌دهد.
`CPI` و `IPC` دقیقا عکس هم هستند. برای مقایسه `performance` این دو شبه کد از پارامتر `CPI` استفاده می‌کنیم.
در فایل `stats` شبه‌کد اول، **CPI=۸۹.۵۹** گزارش شده است. همچنین در شبه‌کد دوم، **CPI=۱۰۱.۸۲** است. همانطور که مشاهده می‌شود در شبه‌کد دوم، بار محاسباتی بیشتر بوده است و برای هر `instruction` به‌طور میانگین ۱۰۱ لبه کلاک مصرف شده است. که این تحلیل با نکته ای که در مورد حلقه‌های تو در تو در ابتدای این بخش گفتیم مطابقت دارد.
پارامتر بعدی ای که می‌توان برای مقایسه `performance` دو کد استفاده کرد، `Number of Busy Clock` است. که نشان دهنده تعداد کلاک‌هایی است که `cpu` مشغول اجرای برنامه بوده است. شاید نتوان از این پارامتر برای مقایسه هر نوع برنامه ای استفاده کرد چون ممکن است معیار مناسبی برای مقیسه دو برنامه متفاوت نباشد، اما در اینجا چون ورودی‌ها و خروجی‌های هر دو برنامه یکسان است و فقط الگوریتم‌ها تفاوت دارد، معیار خوبی برای مقایسه `performance` است.

این پارامتر برای شبه کد اول، ۳۸۳۵۷۳۹۸۵۴۹ و برای شبه‌کد دوم، ۵۹۲۶۱۱۹۲۱۵ بدست‌امده است. همانطور که مشاهده می‌شود، در این معیار هم بازنده `performance` شبه‌کد دوم است و اجرای شبه‌کد اول در تعداد کلاک های کمتری طول کشیده است. این خروجی هم مطابق با پیش‌بینی ای است که در ابتدا کرده بودیم.

۲ پاسخ سوال دوم

مطابق با دستور زیر، فایل را در دو حالت AtomicSim-pleCPU و DerivO۳CPU شبیه‌سازی می‌کنیم:

```
build/X86/gem5.opt configs/example/
se.py ../../../Downloads/sample
```

۱.۲ گام اول

پارامترهای IPC و Sim_second برای حالت Atomic-SimpleCPU به‌صورت زیر بدست آمده است:

1. IPC = 0.262431 $\frac{count}{cycle}$
2. Sim_second: 20.413799

پارامترهای IPC و Sim_second برای حالت De-rivO۳CPU به‌صورت زیر بدست آمده است:

1. IPC: 0.278193 $\frac{count}{cycle}$
2. Sim_second: 38.17364

تفاوت اصلی بین حالت های AtomicSimpleCPU و DerivO۳CPU در سطح جزئیات و پیچیدگی آنها در مدل سازی CPU نهفته است. برای مثال این یک نمایش اولیه از پایپ‌لاین CPU را بدون ویژگی های ریزمعماری دقیق ارائه می دهد. همچنین Instruction ها به صورت اتمی اجرا می شوند، به این معنی که در یک کلاک بدون در نظر گرفتن مراحل پایپ‌لاین یا تأخیرها کامل می شوند. Atom-icSimpleCPU از نظر سرعت شبیه سازی سریعتر است اما دقت کمتری نسبت به DerivO۳CPU دارد. به دلیل سادگی، AtomicSimpleCPU به طور کلی IPC و مقدار sim_second کمتری نسبت به DerivO۳CPU دارد. که همین موضوع از نتایج بدست آمده از شبیه‌سازی هم مشهود است.

۲.۲ گام دوم

برای افزایش سرعت دسترسی به حافظه می‌توان به جای تخصیص حافظه برای هر گره به صورت مجزا، با استفاده از `malloc` یک بلوک پیوسته از حافظه را برای همه گره‌ها به صورت همزمان اختصاص داد.
انجام عملیات بازگشتی بازگشتی بار محاسباتی سنگینی را به CPU تحمیل می‌کند. می‌توان این اجرای بازگشتی را با یک تابع `iterative` جایگزین کرد.
کد اصلاح شده را می‌توانید در «پیوست آ» مشاهده کنید

۳ پاسخ سوال سوم

برای شبیه‌سازی حافظه‌ها در gem5 سه `mode` کاری وجود دارد. `Timing mode` و `Atomic mode` و `mode` `Functional` که در ادامه به بررسی تفاوت‌های آنها می‌پردازیم.

۱. **Functional mode:**

(آ) این حالت، ساده ترین و سریع‌ترین حالت است.

(ب) به دلیل اینکه این حالت ساده‌ترین حالت است و بسیاری از آپشن‌های حالات دیگر را ندارد، سرعت بیشتری هم دارد.

(ج) این حالت برای شبیه‌سازی اولیه مناسب است.

(د) در این حالت، همه دسترسی‌های حافظه به‌صورت دقیق مدل نمی‌شود.

(ه) بین دقت و سرعت، trade-off وجود دارد. بنابر این نمی‌توان از این حالت انتظار دقت بسیار بالایی داشت.

۲. **Atomic mode:**

(آ) این حالت سطح بالاتری ازجزئیات را در مقایسه با حالت functional ارائه می‌دهد.

(ب) در این حالت تمام دسترسی‌های حافظه بدون درنظر گرفتن ملاحظات زمانی مدل می‌شوند.

(ج) این حالت، سریع‌تر از حالت timing است و دقت بیشتری نسبت به حالت functional دارد.

۳. **Timing mode:**

(آ) دقیق‌ترین و جزءی ترین شبیه‌سازی را این حالت انجام می‌دهد.

(ب) در این حالت بر خلاف حالت قبل، دسترسی‌ها به حافظه با ملاحظات و در نظر گرفتن زمان‌بندی ها و تاخیرها مدل‌سازی می‌شود.

(ج) این حالت از شبیه‌سازی رفتاری واقع گرایانه از سلسله‌مراتب کش، کنترل‌کننده های حافظه و تاخیرهای اتصال را شبیه‌سازی می‌کند.

(د) این حالت بیشتر در ارزیابی و تاثیر بهینه‌سازی های مرتبط با حافظه استفاده می‌شود.

(ه) این حالت از شبیه‌سازی دقیق‌ترین نتایج را ارائه می‌دهد اما در مقایسه با دو حالت قبل، به دلیل حجم محاسبات و شبیه‌سازی بیشتر، زمان شبیه‌سازی بیشتر استو

انتخاب mode حافظه به نیازهای خاص شبیه سازی ما بستگی دارد. اگر به نتایج سریع و تقریبی نیاز داشته باشیم، می‌توان از حالت functional استفاده کرد. اما اگر به ویژگی های مرتب سازی حافظه اولیه نیاز داشته باشیم، می توان از حالت atomic استفاده کرد. برای تجزیه و تحلیل دقیق عملکرد و مطالعات ریزمعماری، حالت timing بهترین انتخاب برای شبیه‌سازی است.
با بررسی فایل `config` گام اول سوال اول پارامتر های خواسته شده را به صورت زیر گزارش می‌کنیم:
فرکانس کلاک: ۱ گیگاهرتز است.

```
# Set the clock frequency of the system (and all of its children)
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = "1GHz"
system.clk_domain.voltage_domain = VoltageDomain()
```

شکل ۱۰: فرکانس کلاک شبیه‌سازی

مد شبیه‌سازی: مد شبیه‌سازی روی Timing mode تنظیم شده است.

```
# Set up the system
system.mem_mode = "timing"  # Use timing accesses
system.mem_ranges = [AddrRange("512MB")]  # Create an address range

# Create a simple CPU
# You can use ISA-specific CPU models for different workloads:
# `RiscvTimingSimpleCPU`, `ArmTimingSimpleCPU`.
system.cpu = X86TimingSimpleCPU()
```

شکل ۱۱: مد شبیه‌سازی

سایز حافظه: سایز حافظه ۵۱۲ مگابایت تعریف شده است.

```
# Set up the system
system.mem_mode = "timing"  # Use timing accesses
system.mem_ranges = [AddrRange("512MB")]  # Create an address range
```

شکل ۱۲: سایز حافظه

پیوست آ

- کد پایتون نوشته شده برای شبه‌کد اول:

Listing 1: Python code for part1

```
#           ********************************************************
#          **    course          : Advanced Computer Architecture  **
#         ***    HomeWork        : 03                               ***
#        ****    Topic           : Simulation on gem5               ****
#        ****    AUTHOR          : Reza Adinepour                   ****
#         ***    Student ID:     : 402131055                        ***
#          **    Github          : github.com/rezaAdinepour/        **
#           ********************************************************


import numpy as np
import os
import random


# clear terminal, if you use windows, replace 'clear' with cls
clear = lambda: os.system('clear')
clear()

# function definition
def matrix_mult(A, B):
    try:
        n = np.shape(A)[0]
        m = np.shape(B)[0]
        p = np.shape(B)[1]

        # checking the multiplicity condition
        if(m != np.shape(A)[1]):
            raise ValueError("The number of columns in matrix A must be equal to the number of rows in matrix B")

        # create result matrix
        C = np.zeros((n, p))

        for i in range(n):
            for j in range(p):
                sum = 0
                for k in range(m):
                    sum += A[i][k] * B[k][j]
                C[i][j] = sum

        return C
    except:
        raise ValueError("The matrices have incompatible dimensions for multiplication")


```

```
46
47 # Example usage
48 A_row = A_col = 150
49 B_row = 150
50 B_col = 200
51 lower_limit = -100
52 upper_limit = 100
53
54
55 random.seed(42)
56
57 # Create A_row*A_row random matrix, each of element between lower_limit to
       upper_limit
58 A = np.random.randint(lower_limit, upper_limit, size=(A_row, A_col))
59 B = np.random.randint(lower_limit, upper_limit, size=(B_row, B_col))
60
61
62 print('A is {} matrix'.format(np.shape(A)))
63 print('B is {} matrix\n'.format(np.shape(B)))
64
65 result = matrix_mult(A, B)
66 print('A * B = {}' .format(result))
```

- کد C++ نوشته شده برای شبه‌کد اول:

Listing 2: C++ code for part1

```
1  //            **************************************************
2  //          **   course          : Advanced Computer Architecture  **
3  //         ***   HomeWork        : 03                              ***
4  //        ****   Topic           : Simulation on gem5              ****
5  //        ****   AUTHOR          : Reza Adinepour                  ****
6  //         ***   Student ID:     : 402131055                       ***
7  //          **   Github          : github.com/rezaAdinepour/       **
8  //            **************************************************
9
10
11 #include <iostream>
12 #include <vector>
13 #include <random>
14
15 using namespace std;
16
17
18
19 // Function to impleament random matrix
20 vector<vector<int>> random_number_gen(int row, int col, int lower_limit, int
       upper_limit)
21 {
22     srand(42);
23
24     random_device rd;
25     mt19937 gen(rd());
26     uniform_int_distribution<int> dist(lower_limit, upper_limit);
27
28     vector<vector<int>> matrix(row, vector<int>(col));
29
```

```
    for (auto& row : matrix)
    {
        for (int& num : row)
            num = dist(gen);
    }

    return matrix;
}


// Function to perform matrix multiplication
vector<vector<int>> matrix_mult(const vector<vector<int>>& A, const vector<vector<int>>& B)
{
    try
    {
        int n = A.size();
        int m = B.size();
        int p = B[0].size();

        // Checking the multiplicity condition
        if (m != A[0].size())
        {
            throw invalid_argument("The number of columns in matrix A must be equal to the number of rows in matrix B");
        }

        // Create result matrix C
        vector<vector<int>> C(n, vector<int>(p, 0));

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < p; j++)
            {
                int sum = 0;
                for (int k = 0; k < m; k++)
                    sum += A[i][k] * B[k][j];
                C[i][j] = sum;
            }
        }

        return C;
    }
    catch (const exception& e)
    {
        throw invalid_argument("The matrices have incompatible dimensions for multiplication");
    }
}



int main()
{
    // Example usage
```

```cpp
    int A_row = 150;
    int A_col = 150;

    int B_row = 150;
    int B_col = 200;

    int lowerLimit = -100;
    int upperLimit = 100;

    vector<vector<int>> A = random_number_gen(A_row, A_col, lowerLimit, upperLimit);
    vector<vector<int>> B = random_number_gen(B_row, B_col, lowerLimit, upperLimit);

    cout << "A is (" << A.size() << " x " << A[0].size() << ')' << " matrix" << endl;
    cout << "B is (" << B.size() << " x " << B[0].size() << ')' << " matrix" << endl << endl;

    try
    {
        vector<vector<int>> result = matrix_mult(A, B);
        cout << "A * B = " << endl;
        for (const auto& row : result)
        {
            for (int elem : row)
            {
                cout << elem << " ";
            }
            cout << endl;
        }
    }
    catch (const exception& e)
    {
        cout << e.what() << endl;
    }

    return 0;
}
```

- کد پایتون نوشته شده برای شبه‌کد دوم:

Listing 3: Python code for part2

```python
#           **************************************************
#         **   course          : Advanced Computer Architecture  **
#        ***   HomeWork        : 03                              ***
#       ****   Topic           : Simulation on gem5              ****
#       ****   AUTHOR          : Reza Adinepour                  ****
#        ***   Student ID:     : 402131055                       ***
#         **   Github          : github.com/rezaAdinepour/       **
#           **************************************************


import numpy as np
import os
import random
```

```
14
15
16 # clear terminal, if you use windows, replace 'clear' with cls
17 clear = lambda: os.system('clear')
18 clear()
19
20
21
22 def matrix_mult(A, B):
23     try:
24         n = np.shape(A)[0]
25         m = np.shape(B)[0]
26         p = np.shape(B)[1]
27
28         # checking the multiplicity condition
29         if(m != np.shape(A)[1]):
30             raise ValueError("The number of columns in matrix A must be equal to
    the number of rows in matrix B")
31
32         # set tile size
33         T = int(pow(m, 0.5))
34
35         # create result matrix
36         C = np.zeros((n, p))
37
38         for I in range(0, n, T):
39             for J in range(0, p, T):
40                 for K in range(0, m, T):
41                     for i in range(I, min(I + T, n)):
42                         for j in range(J, min(J + T, p)):
43                             sum_val = 0
44                             for k in range(K, min(K + T, m)):
45                                 sum_val += A[i][k] * B[k][j]
46                             C[i][j] += sum_val
47
48         return C
49     except ValueError as e:
50         return str(e)
51     except Exception as e:
52         return "An error occurred: " + str(e)
53
54
55 # Example usage
56 A_row = A_col = 150
57 B_row = 150
58 B_col = 200
59 lower_limit = -100
60 upper_limit = 100
61
62 random.seed(42)
63
64 # Create A_row*A_row random matrix, each of element between lower_limit to
    upper_limit
65 A = np.random.randint(lower_limit, upper_limit, size=(A_row, A_col))
66 B = np.random.randint(lower_limit, upper_limit, size=(B_row, B_col))
67
```

```
print('A is {} matrix'.format(np.shape(A)))
print('B is {} matrix\n'.format(np.shape(B)))

result = matrix_mult(A, B)
print('A * B = {}' .format(result))
```

- کد C++ نوشته شده برای شبه‌کد دوم:

Listing 4: C++ code for part2

```
//            ******************************************************
//          **    course          : Advanced Computer Architecture  **
//         ***    HomeWork        : 03                              ***
//        ****    Topic           : Simulation on gem5              ****
//        ****    AUTHOR          : Reza Adinepour                  ****
//         ***    Student ID:     : 402131055                       ***
//          **    Github          : github.com/rezaAdinepour/       **
//            ******************************************************



#include <iostream>
#include <vector>
#include <cmath>
#include <random>

using namespace std;


// Function to impleament random matrix
vector<vector<int>> random_number_gen(int row, int col, int lower_limit, int
    upper_limit)
{
    srand(42);

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> dist(lower_limit, upper_limit);

    vector<vector<int>> matrix(row, vector<int>(col));

    for (auto& row : matrix)
    {
        for (int& num : row)
            num = dist(gen);
    }

    return matrix;
}


// Function to perform matrix multiplication
vector<vector<int>> matrix_mult(const vector<vector<int>>& A, const vector<vector<
    int>>& B)
{
    try
    {
```

```
        int n = A.size();
        int m = B.size();
        int p = B[0].size();

        if (m != A[0].size())
            throw invalid_argument("The number of columns in matrix A must be equal to the number of rows in matrix B");

        // Set tile size
        int T = static_cast<int>(sqrt(m));

        // Create result matrix C
        vector<vector<int>> C(n, vector<int>(p, 0));

        for (int I = 0; I < n; I += T)
        {
            for (int J = 0; J < p; J += T)
            {
                for (int K = 0; K < m; K += T)
                {
                    for (int i = I; i < min(I + T, n); i++)
                    {
                        for (int j = J; j < min(J + T, p); j++)
                        {
                            int sum_val = 0;
                            for (int k = K; k < min(K + T, m); k++)
                                sum_val += A[i][k] * B[k][j];
                            C[i][j] += sum_val;
                        }
                    }
                }
            }
        }

        return C;
    }
    catch (const exception& e)
    {
        return {{-1}};
    }
}

int main()
{
    // Example usage
    int A_row = 150;
    int A_col = 150;

    int B_row = 150;
    int B_col = 200;

    int lowerLimit = -100;
    int upperLimit = 100;

    vector<vector<int>> A = random_number_gen(A_row, A_col, lowerLimit, upperLimit);
```

```
    vector<vector<int>> B = random_number_gen(B_row, B_col, lowerLimit, upperLimit);

    cout << "A is (" << A.size() << " x " << A[0].size() << ')' << " matrix" << endl;
    cout << "B is (" << B.size() << " x " << B[0].size() << ')' << " matrix" << endl;

    vector<vector<int>> result = matrix_mult(A, B);
    if (result[0][0] == -1)
        cout << "An error occurred." << endl;
    else
    {
        cout << "A * B = " << endl;
        for (const auto& row : result)
        {
            for (int elem : row)
                cout << elem << " ";
            cout << endl;
        }
    }

    return 0;
}
```

- کد موجود در فایل Sample.c:

Listing 5: Sample C code

```
//           ******************************************************
//          **   course          : Advanced Computer Architecture  **
//         ***   HomeWork        : 03                              ***
//        ****   Topic           : Simulation on gem5              ****
//        ****   AUTHOR          : Reza Adinepour                  ****
//         ***   Student ID:     : 402131055                       ***
//          **   Github          : github.com/rezaAdinepour/       **
//           ******************************************************


#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int label;
    struct Node* next;
};


struct Node* createNode(int label)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->label = label;
    newNode->next = NULL;
    return newNode;
};

```

```c

struct Node* reverse(struct Node* head)
{
    if (head == NULL || head->next == NULL)
    {
        return head;
    } else
    {
        struct Node* new_head = reverse(head->next);
        head->next->next = head;
        head->next = NULL;
        return new_head;
    }
}


int main()
{

    struct Node* head = NULL;
    struct Node* current = NULL;

    for (int i = 1; i <= 10000; i++)
    {
        if (head == NULL)
        {
            head = createNode(i);
            current = head;
        } else
        {
            current->next = createNode(i);
            current = current->next;
        }
    }

    reverse(head);

    return 0;
}
```

- کد Sample.c بهبود داده شده:

Listing 6: Improved C code

```c
//           **************************************************
//          **   course          : Advanced Computer Architecture  **
//         ***   HomeWork        : 03                              ***
//        ****   Topic           : Simulation on gem5              ****
//        ****   AUTHOR          : Reza Adinepour                  ****
//         ***   Student ID:     : 402131055                       ***
//          **   Github          : github.com/rezaAdinepour/       **
//           **************************************************


#include <stdio.h>
#include <stdlib.h>

```

```
struct Node
{
    int label;
    int nextIndex;
};

struct Node* createNode(int label, int nextIndex)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->label = label;
    newNode->nextIndex = nextIndex;
    return newNode;
}

int main()
{
    const int SIZE = 10000;

    struct Node* nodes = (struct Node*)malloc(SIZE * sizeof(struct Node));

    for (int i = 0; i < SIZE; i++)
    {
        nodes[i] = *createNode(i + 1, i + 1);
    }

    nodes[SIZE - 1].nextIndex = -1;

    int currentIndex = 0;
    int nextIndex = nodes[currentIndex].nextIndex;

    while (nextIndex != -1)
    {
        int temp = nodes[nextIndex].nextIndex;
        nodes[nextIndex].nextIndex = currentIndex;
        currentIndex = nextIndex;
        nextIndex = temp;
    }

    return 0;
}
```