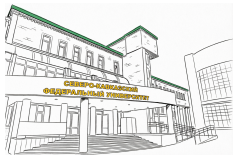

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

Департамент цифровых, робототехнических систем и электроники

АРХИТЕКТУРА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СУПЕРЭВМ

Методические указания
к выполнению практических работ

Направление подготовки: 09.04.02 «Информационные системы и технологии»
Профиль: «Искусственный интеллект, математическое моделирование и суперкомпьютерные технологии в разработке информационных систе»

Квалификация (степень) выпускника: магистр
Форма обучения: очная

Ставрополь – 2022

УДК 004.31:004.93
ББК 32.973

Аннотация: Методические указания содержат комплект из 10 практических работ по дисциплине «АРХИТЕКТУРА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СУПЕРЭВМ» с фокусом на моделирование современных вычислительных систем с использованием открытого эмулятора gem5. Каждая работа включает цель, задачи, теоретический минимум, пошаговый разбор примера, фрагменты конфигурационных скриптов на Python, индивидуальное задание для самостоятельного выполнения, а также вопросы для самоконтроля. Материал охватывает полный цикл исследования архитектуры: от базовых моделей CPU и иерархии кешей до механизмов внеочередного исполнения, протоколов когерентности, NUMA-топологий, аппаратной предвыборки, DVFS-управления и гетерогенных акселераторов. Рекомендуемый объем – 36 часов лабораторных занятий. Работы выполняются в среде Linux с использованием gem5 v23+, наборов бенчмарков SPEC/MiBench и утилит анализа статистики.
Составитель: к.ф.-м.н. Д.Л. Винокурский
Рецензент: к.ф.м.н. Е.В. Крахоткина

Содержание

1 Введение в gem5: базовые модели CPU

Цель: Освоить установку, сборку и базовое конфигурирование эмулятора gem5, сравнить производительность и точность базовых моделей процессоров.
Задачи:

- Настроить окружение: компиляция gem5 в режимах `X86` или `RISCV`.
- Изучить структуру конфигурационных скриптов на Python и режимы симуляции (SE/FS).
- Написать скрипт запуска простой программы в режимах `SimpleCPU`, `AtomicSimpleCPU` и `TimingSimpleCPU`.
- Проанализировать выходные файлы `stats.txt` и сравнить IPC, total cycles и время симуляции.

Теоретическое описание: gem5 – модульный симулятор компьютерных систем, позволяющий моделировать архитектуру на уровне транзакций. Базовые CPU-модели различаются детализацией: `SimpleCPU` (функциональное выполнение, без временной модели), `AtomicSimpleCPU` (атомарное выполнение инструкций с тактовой задержкой), `TimingSimpleCPU` (пошаговое выполнение конвейера). Режим SE (System Call Emulation) запускает пользовательские приложения без полной ОС, что ускоряет отладку конфигураций. **Разобранный пример:** 1. Создание системы: `system = System()`, подключение шины `SystemXBar()`.
2. Выбор CPU: `cpu = AtomicSimpleCPU()` или `cpu = TimingSimpleCPU()`.
3. Подключение процесса: `process = Process(cmd=["./hello"]); cpu.workload = process`.
4. Запуск: `m5.instantiate()`, `m5.simulate()`. **Программный код:**

```
from m5.objects import *
system = System()
system.cpu = TimingSimpleCPU()
system.membus = SystemXBar()
system.cpu.icache_port = system.membus.cpu_side_ports
system.cpu.dcache_port = system.membus.cpu_side_ports
process = Process()
process.cmd = ["./benchmarks/mibench/basicmath"]
system.cpu.workload = process
system.cpu.createThreads()
root = Root(full_system=False, system=system)
m5.instantiate()
m5.simulate()
```

Индивидуальное задание: Запустите 3 CPU-модели на одном бенчмарке. Постройте таблицу сравнения: IPC, simulated seconds, simulation speed (MHz). Объясните расхождения. **Вопросы для самопроверки:**

1. В чём принципиальное отличие режимов SE и FS в gem5?
2. Почему `AtomicSimpleCPU` выполняется быстрее, но даёт менее точные результаты?
3. Какие параметры из `stats.txt` критичны для оценки производительности CPU?

2 Иерархия кешей: параметры и производительность

Цель: Исследовать влияние размеров, ассоциативности и размера блока кешей L1/L2 на производительность системы.
Задачи:

- Реализовать конфигурацию с раздельными L1I/L1D и общим L2 кешем.
- Варьировать параметры: `size`, `assoc`, `block_size`.
- Запустить memory-intensive бенчмарки, собрать статистику hit/miss.

Теоретическое описание: Кеш-память сокращает среднее время доступа к данным (AMAT). Эффективность определяется размером, ассоциативностью и алгоритмом вытеснения (LRU, Random, NMRU). В gem5 кеш моделируется объектом `Cache`, параметры которого напрямую влияют на `Cache::HitRatio` и `Cache::MissRatio`. Увеличение ассоциативности снижает conflict misses, но увеличивает latency и энергопотребление. **Разобранный пример:** 1. Конфигурация L1: `size='32kB', assoc=8, block_size=64`. 2. Подключение через `system.cpu.dcache = Cache()` и `system.cpu.icache = Cache()`. 3. L2 подключается к `system.membus.slave`. **Программный код:**

```
class L1D(Cache):
    assoc = 8
    size = '32kB'
    block_size = 64
    mshrs = 32
    tgts_per_mshr = 8
    replacement_policy = 'LRU'

system.cpu.dcache = L1D()
system.cpu.dcache.cpu_side = system.cpu.dcache_port
system.cpu.dcache.mem_side = system.membus.cpu_side_ports
```

Индивидуальное задание: Проведите sweep L1D size (16, 32, 64 kB) и assoc (2, 4, 8). Постройте графики IPC и L1 miss rate. Найдите «точку насыщения». **Вопросы для самопроверки:**

1. Как параметр `mshrs` влияет на обработку промахов?
2. Почему увеличение `block_size` не всегда повышает производительность?
3. В каких сценариях предпочтительна политика NMRU вместо LRU?

3 Механизмы предсказания ветвлений

Цель: Сравнить эффективность архитектурных предикторов ветвлений и их влияние на конвейер процессора.
Задачи:

- Настроить различные предикторы: `LocalBP`, `GlobalBP`, `TournamentBP`, `BiModeBP`.
- Запустить workload с высокой плотностью условных переходов.
- Проанализировать pipeline stalls и misprediction rate.

Теоретическое описание: Предсказание ветвлений критично для конвейерных CPU. Ошибка предсказания приводит к очистке конвейера (penalty 10-20 тактов). Динамические предикторы используют историю ветвлений (Global History Register) и паттерны. В gem5 предиктор подключается к CPU через параметр `branchPred`. Современные архитектуры используют Tournament-подход для адаптивного выбора. **Разобранный пример:** 1. Подключение: `cpu.branchPred = TournamentBP()`.
2. Настройка истории: `numGlobalHistBits=14`, `localPredictorSize=2048`.
3. Мониторинг: `system.cpu.branchPred.branchMispredicted`. **Программный код:**

```
from m5.objects import *
system.cpu = DerivO3CPU()
bp = TournamentBP(numLocalBits=8, numGlobalBits=12,
                  localPredictorSize=1024, globalPredictorSize=1024)
system.cpu.branchPred = bp
# Simulation launch and stats analysis
```

Индивидуальное задание: Сравните 4 предиктора на бенчмарке с вложенными циклами. Постройте график: Misprediction Rate vs IPC. Сделайте выводы о точности. **Вопросы для самопроверки:**

1. Как размер GHR влияет на точность предсказания?
2. Почему BTB необходим даже при наличии предиктора?
3. В чём преимущество TournamentBP над GlobalBP?

4 Внеочередное исполнение и суперскалярность

Цель: Изучить влияние параметров O3-ядра на пропускную способность и выявлять архитектурные bottlenecks.
Задачи:

- Модифицировать параметры: `numROBEntries`, `issueWidth`, `numIQEntries`, `LSQ`.
- Запустить CPU-bound workload, собрать метрики utilization.
- Выявить лимитирующий ресурс (структурный, data hazard, branch stall).

Теоретическое описание: Внеочередное исполнение (Out-of-Order) позволяет выполнять инструкции по готовности операндов, обходя задержки памяти. Суперскалярность (issue width > 1) увеличивает параллелизм на уровне инструкций (ILP). Ключевые буферы: ROB (Reorder Buffer), IQ (Issue Queue), LSQ (Load-Store Queue). В gem5 O3-ядро моделируется `DerivO3CPU`, параметры которого напрямую влияют на `simInsts` и `simSeconds`. **Разобранный пример:** 1. `cpu.numROBEntries = 128`, `cpu.issueWidth = 4`.
2. `cpu.numIQEntries = 64`, `cpu.LSQ = 32`.
3. Анализ: `cpu.rob.stall_due_to_int_inst`, `cpu.lsq.stall_due_to_lsq`. **Программный код:**

```
system.cpu = DerivO3CPU()
system.cpu.numROBEntries = 192
system.cpu.issueWidth = 4
system.cpu.numIQEntries = 64
system.cpu.LSQ = 48
system.cpu.numPhysIntRegs = 128
# Statistics: system.cpu.rob.robFull, system.cpu.iew.fetchStall
```

Индивидуальное задание: Проведите sweep issueWidth (2, 4, 8) и ROB size. Найдите оптимальную конфигурацию по IPC/Power. Постройте диаграмму utilization стадий конвейера. **Вопросы для самопроверки:**

1. Почему увеличение ROB не всегда повышает IPC?
2. Как LSQ влияет на обработку memory ordering?
3. В чём разница между IQ и ROB?

5 Протоколы когерентности в SMP-системах

Цель: Моделировать многопроцессорные системы и анализировать трафик когерентности кешей.
Задачи:

- Настроить SMP-систему из 2–8 ядер.
- Сравнить протоколы: `MESI_Two_Level`, `MOESI_Hammer`.
- Измерить количество invalidation-сообщений и scalability.

Теоретическое описание: Когерентность кешей гарантирует единую картину данных в многопроцессорных системах. Протоколы на основе снупинга (snooping) подходят для шин, directory-based – для масштабируемых топологий. В gem5 подсистема Ruby генерирует и моделирует состояния кеш-линий. Переходы состояний (I, S, E, M) генерируют трафик, влияющий на latency и пропускную способность. **Разобранный пример:** 1. Ruby-конфиг: `protocol = 'MESI_Two_Level'`.
2. Подключение когерентной шины: `system.xbar = CoherentXBar()`.
3. Анализ: `Ruby::Invalidations`, `Ruby::DirectoryRequests`. **Программный код:**

```
from m5.objects import *
system.ruby = Ruby(create_system=True, protocol='MESI_Two_Level')
system.num_cpus = 4
# Generation of cache controllers and directories
ruby_sys = system.ruby
# Coherence statistics in stats.txt
```

Индивидуальное задание: Запустите многопоточный benchmark на 2, 4, 8 ядрах. Постройте график IPC vs Cores. Проанализируйте влияние false sharing на invalidation rate. **Вопросы для самопроверки:**

1. Почему directory-based лучше масштабируется, чем snooping?
2. Что такое false sharing и как его минимизировать?
3. Как протокол обрабатывает concurrent write requests?

6 Архитектура NUMA и локализация памяти

Цель: Исследовать влияние размещения ядер/памяти и политик аллокации страниц на latency и bandwidth.
Задачи:

- Настроить 2 NUMA-узла с локальными контроллерами DRAM.
- Привязать потоки к ядрам, включить/выключить page migration.
- Анализировать remote vs local access latency.

Теоретическое описание: NUMA (Non-Uniform Memory Access) отражает асимметрию задержек доступа к памяти в многопроцессорных системах. Локальные обращения быстрее, удалённые проходят через межсоединение. В gem5 NUMA моделируется через `AddrMapper` и независимые `DRAMCtrl`. Локализация данных критична для HPC и серверных workload. **Разобранный пример:** 1. Узлы: `node0 = MemCtrl()`, `node1 = MemCtrl()`.
2. Маппинг адресов: `AddrMapper(range=..., interleaving=False)`.
3. Статистика: `DRAM::accessLatency`, `Numa::RemoteAccesses`. **Программный код:**

```
system.node0 = MemCtrl(dram=DDR3_1600_8x8())
system.node1 = MemCtrl(dram=DDR3_1600_8x8())
system.addr_mapper = AddrMapper(start_addr=0, range=0x80000000,
                                memory=system.node0)
# Thread pinning via workload.cpu_id
```

Индивидуальное задание: Измерьте latency при 100% local vs 50% remote access. Постройте график bandwidth vs remote fraction. Оцените эффект page migration. **Вопросы для самопроверки:**

1. Почему NUMA возникает в реальных серверах?
2. Как OS влияет на локализацию страниц?
3. Когда выгодна UMA, а когда NUMA?

7 Межсоединения в многопроцессорных системах

Цель: Сравнить топологии межсоединений и их влияние на задержки и пропускную способность.
Задачи:

- Настроить `SimpleBus`, `CoherentXBar`, `MeshNetwork`.
- Запустить синтетические memory-intensive workload.
- Измерить network latency, packet utilization, congestion.

Теоретическое описание: Межсоединения определяют масштабируемость системы. Шина проста, но ограничена arbitration overhead. Crossbar обеспечивает параллельные соединения, но требует много логических вентилей. Mesh/NoC масштабируется лучше, но добавляет hop-latency. В gem5 реализованы через `SimpleNetwork` и `Garnet`. **Разобранный пример:** 1. Шина: `system.membus = SimpleBus()`.
2. Кроссбар: `system.xbar = CoherentXBar(width=64)`.
3. Сеть: `SimpleNetwork(routers=4, bandwidth=1600)`.
4. Статистика: `Network::latency`, `Network::utilization`. **Программный код:**

```
from m5.objects import *
system.network = SimpleNetwork(routers=4, ext_links=4,
                               bandwidth='1.6GHz', latency='20ns')
# Connecting CPU and memory through ext_links
# Monitoring: system.network.router[0].output_link_occupancy
```

Индивидуальное задание: Сравните 3 топологии на синтетическом трафике (injection rate 10-90%). Постройте график latency vs injection. Найдите точку congestion.
Вопросы для самопроверки:

1. Почему crossbar не масштабируется линейно?
2. Как arbitration влияет на latency при высокой нагрузке?
3. В чём преимущество адаптивной маршрутизации в NoC?

8 Аппаратная префетчинг и оптимизация памяти

Цель: Оценить эффективность различных механизмов аппаратной предвыборки данных.
Задачи:

- Включить `StridePrefetcher`, `StreamPrefetcher`, `NextLinePrefetcher`.
- Настроить aggressive/conservative параметры.
- Анализировать accuracy, coverage, useless prefetches.

Теоретическое описание: Аппаратный префетчинг скрывает latency памяти, загружая данные до явного запроса. Точность (accuracy) – доля полезных предвыборок, покрытие (coverage) – доля скрытых промахов. Агрессивный префетчинг может занимать кеш-линии и полосу пропускания. В gem5 префетчеры подключаются к L1/L2 кешам.
Разобранный пример: 1. `cpu.dcache.prefetcher = StridePrefetcher()`.
2. Настройка: `prefetch_on_access=True`, `degree=4`.
3. Статистика: `Prefetcher::prefetchHits`, `Prefetcher::uselessPrefetches`. **Программный код:**

```
from m5.objects import *
pf = StridePrefetcher()
pf.prefetch_on_access = True
pf.degree = 2
system.cpu.dcache.prefetcher = pf
# Analysis: cache.pf_misses, cache.pf_hits
```

Индивидуальное задание: Сравните 3 префетчера на последовательном, случайном и stride-доступе. Постройте матрицу: Accuracy vs Coverage. Оцените влияние на IPC.
Вопросы для самопроверки:

1. Почему coverage не всегда коррелирует с IPC?
2. Как избежать pollution кеша useless prefetches?
3. В чём отличие hardware от software prefetching?

9 Моделирование энергопотребления и DVFS

Цель: Изучить trade-off между производительностью и энергоэффективностью при динамическом масштабировании частоты.
Задачи:

- Подключить `PowerModel` к CPU и DRAM.
- Настроить `DVFSHandler` с governor (performance, powersave).
- Симулировать workload на разных частотах, рассчитать EDP.

Теоретическое описание: Энергопотребление = Dynamic + Static. DVFS снижает частоту/напряжение для экономии энергии, но увеличивает время выполнения. EDP (Energy-Delay Product) – метрика эффективности. В gem5 `DVFSHandler` управляет доменами напряжения/частоты, а `PowerModel` оценивает потребление на основе активности. **Разобранный пример:** 1. `dvfs = DVFSHandler(domains=[cpu, dram])`.
2. Частоты: `[1.0, 1.5, 2.0, 2.5] GHz`.
3. Статистика: `CPU::powerState`, `Energy::total`. **Программный код:**

```
from m5.objects import *
system.dvfs = DVFSHandler()
system.dvfs.opp_list = [1.0, 1.2, 1.5, 1.8, 2.0] # GHz
system.dvfs.domains = [system.cpu, system.mem_ctrl]
system.cpu.power_model = [StdCPU()]
# Monitoring: dvfs.current_volt, system.total_power
```

Индивидуальное задание: Постройте график Power vs Performance для 5 частот. Найдите точку минимума EDP. Оцените влияние thermal throttling. **Вопросы для самопроверки:**

1. Почему снижение частоты нелинейно снижает power?
2. Как DVFS влияет на memory-bound vs CPU-bound workload?
3. В чём разница между EDP и ED2P?

10 Гетерогенные вычисления: интеграция акселератора

Цель: Смоделировать систему CPU+специализированный акселератор и проанализировать коммуникации.
Задачи:

- Создать/интегрировать custom SimObject (например, matrix mul unit).
- Реализовать DMA/shared memory и прерывания.
- Измерить speedup, transfer overhead, utilization bottleneck.

Теоретическое описание: Гетерогенные системы объединяют универсальные CPU и специализированные ускорители (GPU, NPU, FPGA). Ключевые вызовы: data movement, synchronization, memory consistency. В gem5 акселераторы реализуются как `SimObject` с `PioPort` или `DmaPort`. Оффлоадинг задач требует управления через драйверы и очереди команд. **Разобранный пример:** 1. Акселератор: `class MatrixAccel(SimObject)`.
2. DMA: `dma = DmaDevice()`.
3. Синхронизация: прерывание `system.cpu.interrupts[0]`.
4. Статистика: `Accel::cycles_busy`, `Dma::transfer_time`. **Программный код:**

```
from m5.objects import *
class MatrixAccel(SimObject):
    type = 'MatrixAccel'
    pio_port = PioPort()
    dma_port = DmaPort()

system.accel = MatrixAccel()
system.accel.pio_port = system.membus.cpu_side_ports
system.accel.dma_port = system.membus.mem_side_ports
# CPU sends commands via MMIO, waits for interrupt
```

Индивидуальное задание: Реализуйте pipeline: CPU → DMA → Accel → Interrupt → CPU. Измерьте overhead передачи данных vs compute time. Постройте график Speedup vs Data Size. **Вопросы для самопроверки:**

1. Почему data movement часто лимитирует гетерогенные системы?
2. Как обеспечить когерентность между CPU и accel кешами?
3. В чём преимущество DMA над PIO?

Список литературы

[1] Binkert N. et al. The gem5 Simulator. ACM SIGARCH Computer Architecture News, 2011. 39(2), pp. 1–7.

[2] Hennessy J.L., Patterson D.A. Computer Architecture: A Quantitative Approach. 6th ed. Morgan Kaufmann, 2017.

[3] Martin M.M. et al. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. SIGMETRICS, 2005.

[4] Song P., Gu Z. Power Modeling in gem5: A Case Study with DVFS. IEEE HPEC, 2020.

[5] Dally W.J., Towles B. Principles and Practices of Interconnection Networks. Morgan Kaufmann, 2003.

[6] Jouppi N.P. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. ISCA, 1990.

[7] Hennessy, J. et al. Domain-Specific Architectures for Deep Neural Networks. Communications of the ACM, 2018.

[8] Official gem5 Documentation. `https://www.gem5.org/documentation/`

[9] SPEC CPU Benchmarks. `https://www.spec.org/cpu/`

[10] Bienia C. et al. The PARSEC Benchmark Suite. PACT, 2008.