ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Кваліфікаційна робота
Перший (бакалаврський) рівень

Засоби паралельних обчислень для прискорення емуляції процесора на прикладі Nintendo GameBoy

Автор:
ст. гр. КІУКІ-21-5
Ординцев М. О.

Керівник:
асистент
Мамчич О. О.

Аналіз предметної області

Переваги:

- зрілість емуляторів;
- наявна екосистема;
- можливість розробки власних програм.

Недоліки:

- при емуляції Nintendo Game Boy відсутнє використання методів паралелізації.

Мета і задачі роботи

Мета: дослідити та оцінити ефективність використання засобів паралельних обчислень для прискорення процесу емуляції процесора на прикладі ігрової консолі Nintendo Game Boy.

Задачі:

- аналіз існуючих підходів до емуляції Game Boy;
- виявленні вузьких місць, які можуть бути оптимізовані за допомогою паралельних обчислень;
- розробка емулятора, що використовує багатопотоковість або інші підходи до паралелізації;
- порівняння реалізованого рішення з традиційним послідовним підходом.

Ігрова консоль Nintendo Game Boy

- Процесор: Sharp LR35902 (гібрид Z80/8080), 4.19 МГц
- Оперативна пам'ять: 8 КБ RAM + 8 КБ відеопам'яті (VRAM)
- Графіка:
  - Роздільна здатність: 160×144 пікселів
  - 4 відтінки сірого
- Апаратна підтримка тайлованої графіки та спрайтів
- Звук: 4 канали — 2 квадратні хвилі, хвильовий канал, шум
- Картриджі: до 8 МБ ROM, можливість батарейкового збереження

Емулятор Nintendo Game Boy

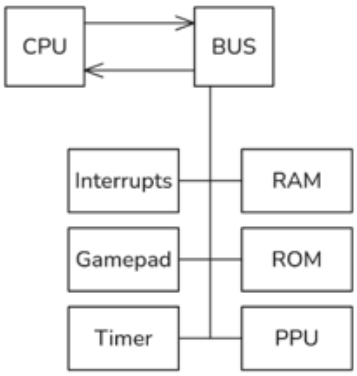
Під час виконання дипломної роботи було створено емулятор ігрової Nintendo Game Boy.

Особливості:

- модульна структура;
- використання мови програмування Rust;
- cycle-accurate реалізація.

Недоліки:

- відсутня реалізація APU;
- відсутність підтримки виконання ROM із додатковими банками пам'яті.



Архітектурні виклики при паралелізації емуляції Nintendo Game Boy

Паралелізація Game Boy здається простою, але його тісно пов'язані компоненти створюють низку складних технічних завдань.

Основні виклики

- точна синхронізація компонентів (CPU, PPU, APU);
- поведінка апаратних переривань і DMA;
- таймінг інструкцій — такт у такт;
- відтворення нестандартних картриджів і банкінгу пам'яті.

Додаткові складності

- залежність багатьох ігор від точного відеотаймінгу;
- складність тестування без реального заліза.

Паралелізація на рівні емульованих компонентів системи

Розподіл компонентів (CPU, PPU, APU) на окремі потоки для імітації їхньої паралельної роботи, як у реальному пристрої.

Переваги

- природне відображення архітектури оригінальної системи;
- потенційно висока продуктивність при правильній реалізації;
- можливість масштабування на системи з великою кількістю ядер.

Недоліки

- складні механізми синхронізації;
- високі накладні витрати на міжпотокову взаємодію;
- ризик виникнення станів гонки та інших проблем багатопотокового програмування.

Паралелізація на рівні допоміжних операцій

Паралелізація допоміжних операцій дозволяє винести ресурсоємні задачі (наприклад, рендеринг) в окремі потоки, зберігаючи при цьому основну логіку емуляції послідовною та точною.

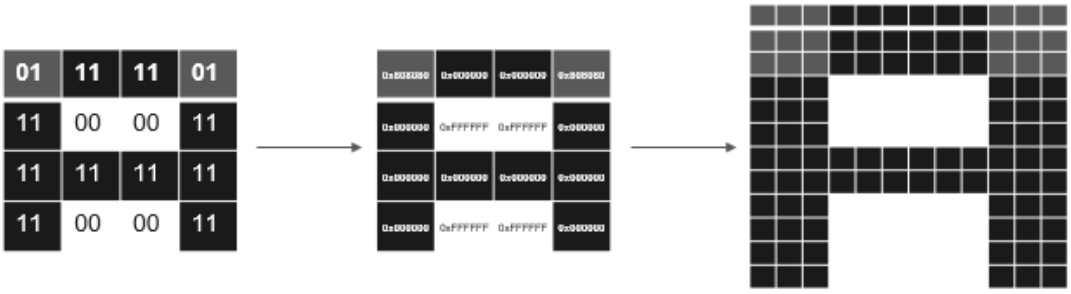
Переваги

- збереження точності емуляції;
- простіша реалізація порівняно з повною паралелізацією;
- ефективне використання багатоядерних процесорів.

Недоліки

- не покращує продуктивність основного ядра емуляції;
- потребує обережного розділення задач;
- підходить не для всіх типів операцій.

Можливість розвантажити основний цикл емуляції



Трансформація з 2bit per pixel формату у RGB

Збільшення зображення

Винесення цих трансформацій у окремий потік

```rust
let (tx: Sender<FrameTask>, rx: Receiver<FrameTask>) = channel::<FrameTask>();
thread_senders.push(tx);
std::thread::spawn(move || {
    while let Ok(task: FrameTask) = rx.recv() {
        let FrameTask {
            start_y: usize,
            end_y: usize,
            buffer: Arc<[u32]>,
            pixel_data: Arc<Mutex<Vec<u8>>>, done_counter: Arc<AtomicUsize>,} = task;
        let mut local: Vec<u8> = vec![0u8; X_RES as usize * (end_y - start_y) * 3];
        for y: usize in start_y..end_y {
            for x: usize in 0..X_RES as usize {
                let index: usize = x + y * X_RES as usize;
                let color: Color = buffer[index].to_color();
                let pixel_index: usize = (y - start_y) * X_RES as usize * 3 + x * 3;
                local[pixel_index] = color.r;
                local[pixel_index + 1] = color.g;
                local[pixel_index + 2] = color.b;
            }
        }
        {
            let mut pd: MutexGuard<'_, Vec<u8>> = pixel_data.lock().unwrap();
            let offset: usize = start_y * X_RES as usize * 3;
            pd[offset..offset + local.len()].copy_from_slice(src: &local);
        }
        done_counter.fetch_add(val: 1, order: Ordering::SeqCst);
    }
});
```

Створення потоків для обробки даних

```rust
let shared_pixel_data: Arc<Mutex<Vec<u8>>> =
 Arc::new(data: Mutex::new(vec![0u8; (X_RES * Y_RES * 3) as usize]));
let shared_buffer: Arc<[GbColor]> = Arc::from(buffer.to_vec().into_boxed_slice());
let done_counter: Arc<AtomicUsize> = Arc::new(data: AtomicUsize::new(0));
let num_threads: usize = self.thread_senders.len();
let chunk_height: usize = Y_RES as usize / num_threads;
for (i: usize, tx: &Sender<FrameTask>) in self.thread_senders.iter().enumerate() {
    let start_y: usize = i * chunk_height;
    let end_y: usize = if i == num_threads - 1 { Y_RES as usize } else {
        (i + 1) * chunk_height
    };
    tx.send(FrameTask {
        start_y,
        end_y,
        buffer: shared_buffer.clone(),
        pixel_data: shared_pixel_data.clone(),
        done_counter: done_counter.clone(),
    }) Result<(), SendError<FrameTask>>
    .unwrap();
}
```

Відправка необхідної для обробки інформації

Опис експериментального стенду

- процесор – AMD Ryzen 7 4800H;
- відеокарта – AMD Radeon Vega 8;
- оперативна пам'ять – 16 Гб;
- операційна система – Windows 10 22H2.

Результати

Назва	Середній FPS до оптимізації	Середній FPS після оптимізації	Приріст (%)
BobtInvite	55.5	88.5	59.5%
Alt Too	60.0	148.0	146.7%
Wyrmhole	49.0	79.5	62.2%
Rex Run	56.5	128.5	127.4%
3D Demo	65.2	148.3	127.5%

Висновки

Результати роботи показали, що використання методів паралелізму можуть пришвидшити навіть такі послідовних системах, де паралелізація наче неможлива.

Але завжди можна зробити швидше:

- Використання методів паралелізації на рівні інструкцій
- Використання методів спекулятивної паралелізації
- Відходження від cycle-accurate емуляції