ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Модель аналізу спектральних даних астрономічних об'єктів

ВИКОНАВ:
ст.гр. Спм-22-3
Ніколаєнко Д.С.

КЕРІВНИК:
доц. каф. ЕОМ
Філімончук Т.В.

ХАРКІВ 2024

Метою даної роботи є побудова моделі обробки спектральних даних астрономічних об’єктів з використанням паралельних обчислень.

Об’єктом дослідження є процес обробки спектральних даних, які було отримано від астрономічних об’єктів.

Предметом дослідження є існуючі на даний час моделі та методи обробки спектральних даних.

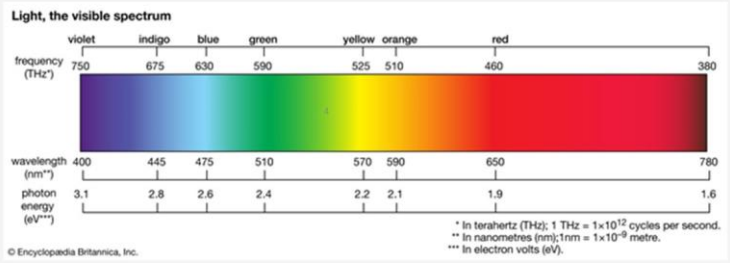
Задачі кваліфікаційної роботи

- проаналізувати предметну область та ознайомитися з актуальним програмним забезпеченням;
- виявити недоліки сучасної моделі аналізу спектральних даних;
- ознайомитися з поняттям паралельних обчислень та їх основам;
- побудувати модель аналізу спектральних даних з урахуванням недоліків актуальної моделі;
- на основі результатів роботи моделі зробити висновки щодо використання паралелізму під час аналізу спектральних даних астрономічних об’єктів.

3

Спектр – це графік або діаграма, що показує, як інтенсивність світла (або іншого випромінювання) змінюється в залежності від довжини хвилі або частоти. Кожен тип випромінювання має свої унікальні властивості та використовується для різних цілей у науці та техніці.

Спектральний аналіз є однією з ключових методик у науках про Землю, астрономії, фізиці та хімії. Він дозволяє вивчати властивості матеріалів та об'єктів через їхні спектри, тобто розподіл інтенсивності випромінювання або поглинання по різних довжинах хвиль.



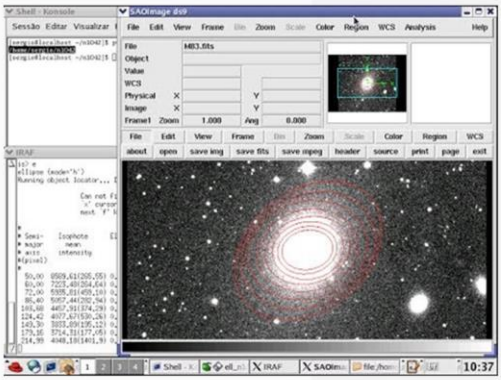
4

Актуальні засоби обробки

На даний час найпопулярнішим середовищем для аналізу та обробки спектральних даних астрономічних об’єктів є середовище **Image Reduction and Analysis Facility (IRAF)**, яке було створено у 80-х роках для швидкої та оптимальної обробки даних, отриманих від астрономічних об’єктів.

Мовою програмування середовища IRAF є **Python**, а для математичних обчислень використовується бібліотека **Astropy**.

Середовище являє собою потужний програмний пакет, розроблений для використання в астрономії та астрофізиці.



5

Формат зберігання даних FITS

Формат **FITS** є стандартом для зберігання, передачі та обробки астрономічних даних. Цей формат був розроблений Національною обсерваторією оптичної астрономії США (NOAO) у кінці 1970-х років і з тих пір став невід'ємною частиною інфраструктури астрономічних досліджень. FITS використовується для зберігання різноманітних типів даних, включаючи зображення, спектри, таблиці та багатовимірні масиви даних. Його гнучкість, надійність та здатність зберігати метадані разом з основними даними роблять його ідеальним форматом для астрономічних спостережень.

6

6

Паралельні обчислення

Паралельні обчислення є важливою складовою сучасних наукових досліджень та технологій. Їх суть полягає у розподілі задачі на менші підзадачі, які можуть виконуватися одночасно на декількох процесорах або ядрах процесора. Це дозволяє ефективніше використовувати обчислювальні ресурси та скорочувати час виконання задачі.

Для розробки моделі з паралельним алгоритмом було обрано середовище CUDA.

7

Програмна архітектура CUDA

CUDA – це програмно-апаратна архітектура паралельних обчислень, яка дозволяє істотно збільшити обчислювальну продуктивність завдяки використанню графічних процесорів NVIDIA. При використанні технології слід орієнтуватися на наступну термінологію:

- пристрій (device) – сама відеокарта, графічний процесор (GPU), який виконує команди центрального процесора;
- хост (host) – центральний процесор (CPU), який запускає різні завдання на пристрої, виділяє пам'ять;
- ядро (kernel) – завдання, що буде виконуватися на GPU.

8

Сучасна модель аналізу спектральних даних

$$MP_{SD} = \{D, S [IRAF, Astropy], SA\},$$

D – датасет спектральних даних;
S – програмне забезпечення, що обробляє датасет даних;
SA – послідовний алгоритм обробки датасету.

Основним недоліком моделі є використання застарілого програмного забезпечення та бібліотек, котрі не містять в собі змогу обробляти дані розподілено.

9

Аналіз складових моделі спектральних даних

Розроблено нову модель аналізу спектральних даних з використанням паралельних обчислень на основі архітектури CUDA:

$$MP_{SD} = \{D, S [CUDA, OpenCV], DA\},$$

D – датасет спектральних даних;
S – ПЗ, що обробляє датасет даних;
DA – розподілені алгоритми обробки датасету.

10

Доступ до спектральних даних

Для реалізації алгоритмів було обрано спектральні дані формату FITS галактики Андромеда з архівів NASA, які забезпечують високу ступінь структурованості даних, що робить їх зручними для використання.

Для первинного аналізу даних архіви надають інструменти для візуалізації, що дозволяють попередньо переглянути та обробити дані без необхідності завантаження на локальний комп'ютер.

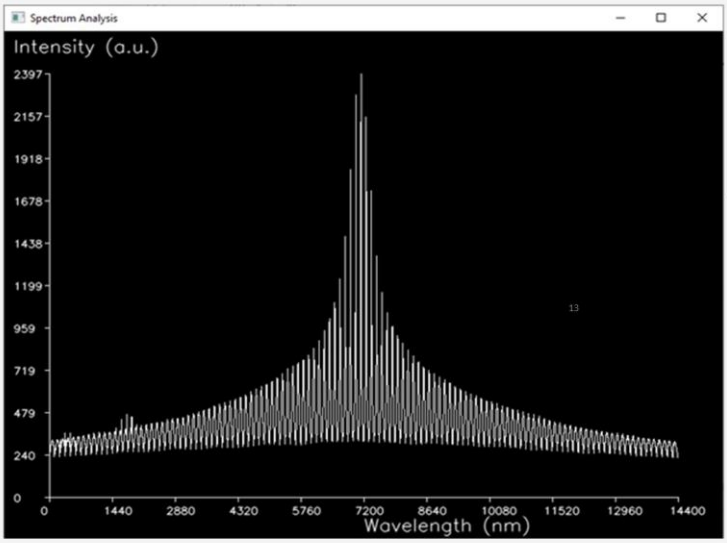
11

Методи послідовного та паралельного аналізу спектральних даних

```
void FindMaxIntensityByProcessor(const float* spectrum, float& maxIntensity, int spectrumSize)
{
    maxIntensity = 0.0f;

    for (int i = 0; i < spectrumSize; ++i)
        if (spectrum[i] > maxIntensity)
            maxIntensity = spectrum[i];
}

__global__ void FindMaxIntensityByGraphicsCard(float* spectrum, float* maxIntensity, int spectrumSize)
{
    extern __shared__ float shared_data[];

    int tid = threadIdx.x;
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    if (idx < spectrumSize)
        shared_data[tid] = spectrum[idx];
    else
        shared_data[tid] = -FLT_MAX;

    __syncthreads();

    for (int stride = blockDim.x / 2; stride > 0; stride >>= 1)
    {
        if (tid < stride)
            shared_data[tid] = fmaxf(shared_data[tid], shared_data[tid + stride]);

        __syncthreads();
    }

    if (tid == 0)
        atomicMax((int*)maxIntensity, __float_as_int(shared_data[0]));
}
```

12

Результати аналізу



CPU and GPU Results

```
CPU Max intensity: 2397.13
CPU Elapsed time: 7.67e-05 seconds
CPU Operations per second: 1.87744e+08
GPU Max intensity: 2397.13
GPU Elapsed time: 0.0012743 seconds
GPU Operations per second: 1.13003e+07
```

13

Результати аналізу

SpectrumSizeMultiplicator	Час виконання на CPU в секундах	Час виконання на GPU в секундах	Кількість операцій в секунду на CPU	Кількість операцій в секунду на GPU
1	0.0000767	0.0012743	187744000	11303000
1024	0.0203643	0.0126303	724091000	1167480000
2046	0.0306026	0.0271126	962742000	1086670000
4092	0.0784251	0.0475633	751351000	1238870000
8096	0.126231	0.091471	923562000	1274530000

14

Висновки

- Для великих обсягів даних GPU демонструє значну перевагу у швидкості обробки. Це пов'язано з високою паралельною обчислювальною потужністю GPU, яка дозволяє одночасно обробляти великі масиви даних. У таких випадках накладні витрати на передачу даних і запуск ядра стають незначущими у порівнянні з часом, який витрачається на обробку даних.

- Результати експериментів підтверджують необхідність використання паралельних обчислень на GPU для ефективної обробки великих обсягів спектральних даних в астрономії. Використання GPU дозволяє значно скоротити час обробки даних, що є критично важливим для прискорення наукових досліджень та відкриття нових астрономічних явищ.

- Майбутні дослідження в цій галузі можуть включати подальше вдосконалення алгоритмів паралельної обробки, розширення функціоналу інструментів, що використовуються, а також адаптацію моделі для нових областей застосування в астрономії.

15

ДОДАТОК Б

Лістинг коду

Лістинг Б.1 – Метод графічного відображення інтенсивності спектральних даних

```
void DisplaySpectrumData(const float* spectrum, int spectrumSize)
{
    int image_width = 800;
    int image_height = 600;
    int margin = 50;
    cv::Mat graph = cv::Mat::zeros(image_height, image_width, CV_8UC3);
    float maxIntensity = *std::max_element(spectrum, spectrum +
spectrumSize);
    for (int i = 0; i < spectrumSize - 1; ++i)
    {
        float intensity1 = spectrum[i];
        float intensity2 = spectrum[i + 1];
        int x1 = static_cast<int>(static_cast<float>(i) / spectrumSize *
(image_width - 2 * margin) + margin);
        int y1 = static_cast<int>((1.0f - intensity1 / maxIntensity) *
(image_height - 2 * margin) + margin);
        int x2 = static_cast<int>(static_cast<float>(i + 1) / spectrumSize *
(image_width - 2 * margin) + margin);
        int y2 = static_cast<int>((1.0f - intensity2 / maxIntensity) *
(image_height - 2 * margin) + margin);
        cv::line(graph, cv::Point(x1, y1), cv::Point(x2, y2), cv::Scalar(255,
255, 255), 1);
    }
    cv::line(graph, cv::Point(margin, margin), cv::Point(margin, image_height
- margin), cv::Scalar(255, 255, 255), 1);
    cv::line(graph, cv::Point(margin, image_height - margin),
cv::Point(image_width - margin, image_height - margin), cv::Scalar(255, 255,
255), 1);
    cv::putText(graph, "Intensity (a.u.)", cv::Point(10, margin / 2),
cv::FONT_HERSHEY_SIMPLEX, 0.7, cv::Scalar(255, 255, 255), 1, cv::LINE_AA);
    cv::putText(graph, "Wavelength (nm)", cv::Point(image_width / 2,
image_height - 10), cv::FONT_HERSHEY_SIMPLEX, 0.7, cv::Scalar(255, 255, 255),
1, cv::LINE_AA);
    int num_ticks_x = 10;
    for (int i = 0; i <= num_ticks_x; ++i)
    {
        int x = margin + i * (image_width - 2 * margin) / num_ticks_x;
        cv::line(graph, cv::Point(x, image_height - margin), cv::Point(x,
image_height - margin + 5), cv::Scalar(255, 255, 255), 1);
        std::stringstream ss;
        ss << i * (spectrumSize / num_ticks_x);
        cv::putText(graph, ss.str(), cv::Point(x - 10, image_height - margin
+ 20), cv::FONT_HERSHEY_SIMPLEX, 0.4, cv::Scalar(255, 255, 255), 1,
cv::LINE_AA);
    }
    int num_ticks_y = 10;
    for (int i = 0; i <= num_ticks_y; ++i)
    {
        int y = margin + i * (image_height - 2 * margin) / num_ticks_y;
        cv::line(graph, cv::Point(margin - 5, image_height - y),
cv::Point(margin, image_height - y), cv::Scalar(255, 255, 255), 1);
        std::stringstream ss;
```

```
        ss << static_cast<int>(std::round(i * (maxIntensity / num_ticks_y)));
        cv::putText(graph, ss.str(), cv::Point(10, image_height - y + 5),
cv::FONT_HERSHEY_SIMPLEX, 0.4, cv::Scalar(255, 255, 255), 1, cv::LINE_AA);
    }
    cv::imshow("Spectrum Analysis", graph);
}
```

Лістинг Б.2 – Метод відображення результатів аналізу

```
void DisplayResults(const std::string& results)
{
    double font_scale = 0.5;
    int font_face = cv::FONT_HERSHEY_SIMPLEX;
    int baseline = 0;
    int thickness = 1;
    cv::Size text_size = cv::getTextSize(results, font_face, font_scale,
thickness, &baseline);
    int image_width = 800;
    int image_height = text_size.height * 10 + 40;
    cv::Mat output = cv::Mat::zeros(image_height, image_width, CV_8UC3);
    std::istringstream iss(results);
    std::string line;
    int y = 30;
    while (std::getline(iss, line))
    {
        cv::putText(output, line, cv::Point(10, y), font_face, font_scale,
cv::Scalar(255, 255, 255), thickness, cv::LINE_AA);
        y += text_size.height + 10;
    }
    cv::imshow("CPU and GPU Results", output);
}
```

Лістинг Б.3 – Головний метод програми

```
int main()
{
    fitsfile* fptr;
    int status = 0;
    long naxes[3] = { 1, 1, 1 };
    float* h_spectrum;
    float* d_spectrum, * d_max_intensity;
    int bitpix, naxis;
    bool useIncreasedSize = false;
    const char* fits_path = "C:\\Demon files\\University\\6-th Course Second
Semester\\SpectrumAnalysis\\x64\\Debug\\spectrum.fits";
    if (fits_open_file(&fptr, fits_path, READONLY, &status))
    {
        fits_report_error(stderr, status);
        return status;
    }
    if (fits_get_img_param(fptr, 3, &bitpix, &naxis, naxes, &status))
    {
        fits_report_error(stderr, status);
        fits_close_file(fptr, &status);
        return status;
    }
    long totalSize = 1;
    for (int i = 0; i < naxis; ++i)
        totalSize *= naxes[i];
    int spectrumSize = useIncreasedSize ? totalSize *
SpectrumSizeMultiplicator : totalSize;
    h_spectrum = (float*)malloc(spectrumSize * sizeof(float));
```

```
    if (h_spectrum == nullptr)
    {
        std::cerr << "Failed to allocate host memory" << std::endl;
        fits_close_file(fptr, &status);
        return -1;
    }
    if (fits_read_img(fptr, TFLOAT, 1, totalSize, nullptr, h_spectrum,
nullptr, &status))
    {
        fits_report_error(stderr, status);
        fits_close_file(fptr, &status);
        free(h_spectrum);
        return status;
    }
    fits_close_file(fptr, &status);
    if (useIncreasedSize)
        for (int i = totalSize; i < spectrumSize; ++i)
            h_spectrum[i] = h_spectrum[i % totalSize];
    float max_intensity_cpu = 0.0f;
    auto start_cpu = std::chrono::high_resolution_clock::now();
    FindMaxIntensityByProcessor(h_spectrum, max_intensity_cpu, spectrumSize);
    auto end_cpu = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed_cpu = end_cpu - start_cpu;
    double ops_per_sec_cpu = spectrumSize / elapsed_cpu.count();
    std::stringstream ss;
    ss << "CPU Max intensity: " << max_intensity_cpu << std::endl;
    ss << "CPU Elapsed time: " << elapsed_cpu.count() << " seconds" <<
std::endl;
    ss << "CPU Operations per second: " << ops_per_sec_cpu << std::endl;
    cudaMalloc((void**)&d_spectrum, spectrumSize * sizeof(float));
    cudaMalloc((void**)&d_max_intensity, sizeof(float));
    cudaMemcpy(d_spectrum, h_spectrum, spectrumSize * sizeof(float),
cudaMemcpyHostToDevice);
    float initial_max = -FLT_MAX;
    cudaMemcpy(d_max_intensity, &initial_max, sizeof(float),
cudaMemcpyHostToDevice);
    int blockSize = 256;
    int numBlocks = (spectrumSize + blockSize - 1) / blockSize;
    auto start_gpu = std::chrono::high_resolution_clock::now();
    FindMaxIntensityByGraphicsCard <<<numBlocks, blockSize, blockSize *
sizeof(float) >>> (d_spectrum, d_max_intensity, spectrumSize);
    cudaDeviceSynchronize();
    auto end_gpu = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed_gpu = end_gpu - start_gpu;
    double ops_per_sec_gpu = spectrumSize / elapsed_gpu.count();
    cudaMemcpy(&max_intensity_cpu, d_max_intensity, sizeof(float),
cudaMemcpyDeviceToHost);
    ss << "GPU Max intensity: " << max_intensity_cpu << std::endl;
    ss << "GPU Elapsed time: " << elapsed_gpu.count() << " seconds" <<
std::endl;
    ss << "GPU Operations per second: " << ops_per_sec_gpu << std::endl;
    cudaFree(d_spectrum);
    cudaFree(d_max_intensity);
    DisplaySpectrumData(h_spectrum, spectrumSize);
    DisplayResults(ss.str());
    free(h_spectrum);
    cv::waitKey(0);
    return 0;
}
```