НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Факультет інформатики та обчислювальної техніки
Кафедра технічної кібернетики

«На правах рукопису»

УДК ___004.051____

«До захисту допущено»

Завідувач кафедри
_____ __Ткач М.М.__
(підпис) (ініціали, прізвище)
“___”_____2015 р.

Магістерська дисертація

зі спеціальності __8.05020102 “Комп’ютеризовані та робототехнічні системи”__
(код та назва напряму спеціальності)

на тему: _____Дослідження методів паралельного програмування на PHP____

Виконав: студент _VI_ курсу, групи ___ІК-31м___
(шифр групи)

_____Стельмах Володимир Ігорович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____к.т.н. доц._Крилов Є.В._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант_охорона праці __к.т.н. доц. Праховнік Н.А.___ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2015 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Факультет інформатики та обчислювальної техніки
Кафедра технічної кібернетики

Факультет (інститут)____Інформатики та обчислювальної техніки____
(повна назва)

Освітньо-кваліфікаційний рівень __________«Магістр»__________
(назва ОКР)

Напрям підготовки ________6.050201 "Системна інженерія"_________
(код і назва)

Спеціальність _8.05020102 "Комп'ютеризовані та робототехнічні системи"_
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
__________ __Ткач М.М.__
(підпис) (ініціали, прізвище)

«___»_____________2015 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

___________________Стельмаху Володимиру Ігоровичу___________________
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) ____________ Дослідження методів паралельного програмування на PHP ___

Науковий керівник дисертації_____Крилов Є.В._____к.т.н. _доцент______,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «_10_»_березня_ 2015 р. №__292/2С__

2. Строк подання студентом дисертації________________________________

3. Об'єкт дослідження__________Програма на мові PHP____________________

4. Предмет дослідження_____Оптимізація за рахунок розпаралелювання_____

5. Перелік завдань які необхідно розробити_______________ огляд методик розпаралелювання програмного коду, огляд способів синхронізації паралельної взаємодії, реалізація паралельного виконання в PHP, аналіз та порівняння результатів паралельного виконання, дослідження методики розпаралелювання на основі реального проекту, аналіз виробничого приміщення_____________

6. Орієнтовний перелік ілюстративного матеріалу_______ блок схеми, графіки досліджень, схема виробничого приміщення___________________________

7. Орієнтований перелік публікацій Розробка моделі паралельного програмування на php / Міжвідомчий науково-технічний збірник «Адаптивні системи автоматичного управління» №26. — Київ: Національний технічний університет України “Київський політехнічний інститут”. – 2015

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1. Розділ охорони праці	Праховнік Н.А., доцент		
2. Нормоконтроль	Пасько В.П., доцент		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Ознайомлення із завданням		
2	Вивчення технічної літератури		
3	Реалізація паралельності		
4	Проведення тестування		
5	Аналіз результатів		
6	Дослідження на реальному проекті		
7	Документування		
8	Захист		

Студент _____ Стельмах В.І
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ Крилов Є.В.
(підпис) (ініціали, прізвище)

ВІДОМІСТЬ ДО МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4		Пояснювальна записка	80	
3	А4		Код програми	1	
4	А4		Код програми	1	
5	А4		Код програми	2	
6	А4		Код програми	16	
7	А4		Код програми	10	
8	А3		Демонстраційний плакат	1	
9	А3		Демонстраційний плакат	1	
10	А3		Демонстраційний плакат	1	
11	А3		Демонстраційний плакат	1	
12	А3		Демонстраційний плакат	1	
13	А3		Демонстраційний плакат	1	

ABSTRACT

Work structure and scope. Explanatory note diploma consists of 4 sections, 80 pages, contains 16 figures, 12 tables, 5 applications, 14 sources.

Master dissertation is devoted to research methods of parallel programming in PHP.

Research has shown the possibilities of parallelization of code in PHP to enhance rapid web application. To implement parallel execution messaging model taken as a basis and offered the methods of its use in PHP. To allocate process load has simple scheme of block distribution.

The work was conducted:

- review methods of code parallelization,
- review ways to synchronize parallel interaction,
- asymptotic analysis algorithms parallelization,
- review of load distribution algorithms,
- the parallelism of different methods,
- comparison of parallelization methods,
- implementation of parallelization techniques in a real project.

The result of the study was a class of software to simplify the parallelization of code in PHP.

Keywords:

Parallelization, web applications performance, message passing model, load allocation.

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з 4 розділи, 80 сторінок, містить 16 рисунків, 12 таблиць, 5 додатків, 14 джерел.

Магістерська дисертація присвячена дослідженню методів паралельного програмування на PHP.

Дослідження має показати можливості розпаралелювання програмного коду у PHP для підвищення швидкої веб-додатків. Для реалізації паралельного виконання взято за основу модель передачі повідомлень та запропоновано методи її використання на PHP. Для розподілу навантаження по процесах, запропонована проста схема блочного розподілу.

У ході роботи було проведено:

- огляд методик розпаралелювання коду,
- огляд способів синхронізації паралельної взаємодії,
- асимптотичний аналіз алгоритмів розпаралелювання,
- огляд алгоритмів розподілу навантаження,
- реалізовано паралельність різними методами,
- порівняння методів розпаралелювання,
- впровадження методики розпаралелювання у реальний проект.

Результатом дослідження став програмний клас для спрощення розпаралелювання коду на PHP.

Ключові слова:

Розпаралелювання, швидкодія веб-додатків, модель обміну повідомленнями, розподіл навантаження.

Пояснювальна записка
до магістерської дисертації

на тему: Дослідження методів паралельного програмування на PHP

Київ – 2015 року

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

PHP – Hypertext Preprocessor (Гіпертекстовий препроцесор)

IPC – Interprocess communication (Взаємодія між процесами)

ОС – Операційна система

ЯПФ – Ярусно-пararельна форма

MVC – Model View Controller (Модель вигляд контролер)

ООП – Об’єктно орієнтоване програмування

БД – База даних

СУБД – Система управління базами даних

СКБД – Система керування базами даних

SQL – Structured query language (Мова структурованих запитів)

ODBC – Open Database Connectivity Standard (Стандарт відкритого інтерфейсу зв'язку з базами даних)

ZF – Zend Framework (відкритий об'єктно орієнтований PHP-фреймворк)

ВСТУП

В наш час мережа Інтернет поширюється на всі сфери життя людини. Тому кількість і різноманітність веб-додатків збільшується з кожним днем. Мережа Інтернет створювалась для швидкого та зручного доступу до інформації в будь-який момент часу і будь де. Але з ростом мережі, збільшується і об’єм веб-додатків, що за деяких обставин призводить до неефективної роботи з інформацією.

На поточний момент однією з основних вимог до веб-додатків є їх швидкодія. Адже зі зростанням часу виконання розрахунків (обчислень), і як наслідок зростанням часу завантаження сторінки може скласти негативне враження у відвідувача.

При обробці великих масивів данних, більшість часу витрачається саме на пошук та доступ до конкретного елементу. Наприклад при обробці великої кількості віддалених ресурсів, чи локальних файлів (наприклад зображень).

Таким чином, якщо виконати декомпозицію та модифікацію алгоритму для доступу до ресурсів у параельному режимі можна збільшити ефективність роботи додатку та зменшити час обробки данних.

Стандартних засобів, для розпаралелювання програмного коду, PHP не має.

1. ОГЛЯД МЕТОДИК РОЗПАРАЛЕЛЮВАННЯ ПРОГРАМНОГО КОДУ

2.1. Послідовна і паралельна парадигми програмування

2.1.1. Послідовна парадигма

Послідовна модель припускає, написання звичайної послідовної програми в одній з послідовних моделей програмування для подальшого автоматичного її розпаралелювання компілятором або спеціальними програмними чи апаратними засобами.

Полягає в паралельному виконанні операцій усередині одного потоку виконання. Можливість однопоточного паралелізму визначається архітектурою мікропроцесора, а конкретно його здатністю зчитувати з пам'яті і виконувати одночасно кілька операцій.

Однопотоковий паралелізм володіє своїми перевагами і недоліками. Переваги:

- Відсутність необхідності синхронізації – всі операції виконуються усередині одного потоку, отже в строго певній послідовності.
- Відсутність необхідності підтримки паралелізмом на рівні операційної системи.
- Відсутність необхідності в засобах управління розподіленими ресурсами.

Недоліки:

- Утрудненість використання в алгоритмах з умовними переходами.
- Необхідність адаптації програми для ефективного використання ресурсів мікропроцесора, наприклад, при переході з однієї моделі на іншу.

Існують наступні методи досягнення паралельних обчислень:

- Упаковка даних для групової обробки одиничними інструкціями застосуванням спеціальних методик. Наприклад, можливо скласти дві пари 8-бітних даних 16-бітної операцією виключивши можливість переповнення. Метод має дуже обмежене застосування.

- Векторна обробка. Мікропроцесор має інструкції, що виробляють групові однотипні операції. Однотипні операнди упаковуються в один векторний регістр. Цей метод аналогічний першому, але забезпечення паралельності лежить на мікропроцесорній архітектурі. Векторні регістри як правило мають велику розрядність. Потрібно адаптувати програму для використання векторних інструкцій або застосовувати оптимізуючий компілятор.
- Суперскалярна архітектура. Пристрій управління мікропроцесора самостійно аналізує потік інструкцій на предмет можливості паралельного виконання і управляє декількома функціональними пристроями.
- Мікропроцесор з явним паралелізмом. Метод аналогічний попередньому, але програма для такого процесора містить явні вказівки на те, які операції треба виконувати паралельно. Розпаралелювання обчислень в даному випадку повністю лежить на програмістові або компіляторі.

При вирішенні задачі на послідовній обчислювальній системі (один процесор і одне ядро) прийнято вважати, що ця сукупність складається з п'яти етапів:

1. Постановка завдання.
2. Створення математичної моделі.
3. Розробка алгоритму рішення в рамках створеної математичної моделі.
4. Написання програми, що реалізує алгоритм, в одній з обраних моделей програмування і на обраному алгоритмічній мові. Модель програмування визначає основні ідеї та стиль програмної реалізації, абстрагуючись від алгоритмічної мови і, частково, від апаратних компонентів. Наприклад, модель функціонального програмування, модель об'єктно орієнтованого програмування, модель продукційного програмування і т.д.

5. Робота програми як набору процесів та / або потоків виконання на обчислювальної системі та отримання результатів.[1]

2.1.2. Паралельна парадигма

Паралельні обчислення – спосіб організації комп'ютерних обчислень, при якому програми розробляються як набір взаємодіючих обчислювальних процесів, що працюють паралельно (одночасно). Термін охоплює сукупність питань паралелізму в програмуванні, а також створення ефективно діючих апаратних реалізацій. Теорія паралельних обчислень становить розділ прикладної теорії алгоритмів [5].

Багато задач вимагають обчислень з великою кількістю операцій, які займають значні ресурси навіть сучасної техніки, більш того, можна з упевненістю вважати, що яких би швидкостей не досягла обчислювальна техніка завжди знайдуться завдання, на вирішення яких потрібно значний час. Багато з таких складних завдань вимагають, щоб результат був отриманий за якомога менший час або навіть строго обмежений. До таких завдань, наприклад, відноситься прогнозування погоди, обробка зображень і розпізнавання образів при управлінні технікою. З іншого боку представляє велику технічну проблему зменшення часу виконання кожної операції в мікропроцесорі.

Очевидним способом збільшити швидкість обчислень було б застосування не одного обчислювального пристрою, а декількох, що працюють спільно над вирішенням одного завдання. Такий підхід носить назву паралельних обчислень. Незважаючи на гадану простоту рішення воно є часом вельми нетривіальним завданням з проектування обчислювальної техніки і розробки алгоритмів. Перша проблема криється в тому, що для того, щоб завдання можна було вирішити за допомогою паралельних обчислень алгоритм її вирішення повинен допускати розпаралелювання, мало того, далеко не кожна задача може бути вирішена паралельним алгоритмом. Інший же, не менш важливою

проблемою є побудова системи, на якій би можлива була реалізація паралельних обчислень.

Існують різні способи реалізації паралельних обчислень. Наприклад, кожен обчислювальний процес може бути реалізований у вигляді процесу операційної системи, або ж обчислювальні процеси можуть являти собою набір потоків виконання всередині одного процесу ОС. Паралельні програми можуть фізично виконуватися або послідовно на єдиному процесорі – перемежовуючи по черзі кроки виконання кожного обчислювального процесу, або паралельно – виділяючи кожному обчислювальному процесу один або кілька процесорів (що знаходяться поруч або розподілених в комп'ютерну мережу).

Основна складність при проектуванні паралельних програм – забезпечити правильну послідовність взаємодій між різними обчислювальними процесами, а також координацію ресурсів, що розділяються між процесами.

При вирішенні задачі на паралельній обчислювальній системі (кілька процесорів і / або декілька ядер) в цій сукупності з'являються додаткові етапи. Їх стає вісім:

1. Постановка завдання;
2. Створення математичної моделі;
3. Розробка алгоритму;
4. Декомпозиція алгоритму (decomposition). При паралельній реалізації алгоритму ми припускаємо, що він буде виконаний декількома виконавцями. Для цього потрібно виділити в алгоритмі набори дій, які можуть бути здійснені одночасно, незалежно один від одного – декомпозувати алгоритм. Розрізняють два види декомпозиції: за даними і за обчисленнями. Якщо в алгоритмі схожим чином обробляються великі обсяги даних, то можна спробувати розділити дані на частини – зони відповідальності, кожна з яких допускає незалежну обробку окремим виконавцем, і виявити обчислення, пов'язані з зонами відповідальності. Це – декомпозиція за даними. Інший підхід передбачає поділ обчислень

на зони відповідальності для їх виконання на різних виконавців і визначення даних, пов'язаних з цими обчисленнями. Це – декомпозиція за обчисленнями (функціональна декомпозиція). Декомпозиція можлива не завжди. Існують алгоритми, які принципово не допускають при своїй реалізації участі кількох виконавців;

5. Призначення робіт (assignment). Після успішного завершення етапу декомпозиції весь алгоритм являє собою сукупність множин наборів дій, спрямованих на вирішення підзадач окремими виконавцями. Набори дій одного безлічі допускають одночасне і незалежне виконання. Безлічі можуть містити різну кількість наборів і, відповідно, реалізовуватися на різній кількості виконавців. Не виключено, що частина множин буде містити всього один набір і вимагати всього один процесор (ядро). У реальності кількість наявних ядер завжди обмежена. На даному етапі необхідно визначити, скільки виконавців ви збираєтеся задіяти і як розподілити підзадачі по виконавцях. Основними цілями призначення підзадач є балансування завантаження процесорів (ядер), зменшення обмінів даними між ними і скорочення накладних витрат на виконання самого призначення. За часом способи призначення поділяються на дві категорії:
    - статичні – розподіл виконується на етапі написання, компіляції або старту програми (до реального початку обчислень);
    - динамічні – розподіл здійснюється в процесі виконання.
6. Диригування. Після завершення етапу призначення ми знаходимося в стані композитора, який підготував всі партитури для виконання своєї симфонії. Але нормальне звучання оркестру можливе лише за наявності диригента, який синхронізує діяльність окремих музикантів і вносить у виконання свій стиль. За аналогією з діями диригента назва цього етапу «диригування». Його метою є вибір програмної моделі і визначення необхідної синхронізації роботи виконавців, яка багато в чому буде

залежати від програмної моделі. Зупинимося докладніше на програмних моделях для роботи на паралельних обчислювальних системах. Хоча класифікація і назви програмних моделей до кінця не встановлені, виділимо чотири основні:

- Послідовна модель. Припускає, що ви пишете звичайну послідовну програму в одній з послідовних моделей програмування для подальшого автоматичного її розпаралелювання компілятором або спеціальними програмними засобами. Перевага моделі – нічого зайвого не треба потрібно робити у порівнянні з послідовним варіантом, недолік – автоматичне розпаралелювання має вкрай обмежені можливості.
- Модель передачі повідомлень. Припускає, що працює додаток складається з набору процесів з різними адресними просторами, кожен з яких функціонує на своєму виконавця. Процеси обмінюються даними за допомогою передачі повідомлень через явні операції send/receive. Перевага моделі полягає в тому, що програміст здійснює повний контрольнад вирішенням завдання, недолік – в складності програмування.
- Модель розподіленої пам’яті. Припускає, що додаток складається з набору потоків виконання (thread), що використовують колективні змінні і примітиви синхронізації. Виділяються дві підмоделі: явні потоки виконання – використання системних або бібліотечних викликів для організації роботи потоків і програмування на мові високого рівня з використанням відповідних прагм. Перша підмодель добре переноситься, дає повний контроль над виконанням, але дуже трудомістка. Друга підмодель легка для програмування, але не дає можливості повністю контролювати рішення задачі.

- Модель розділених даних. Припускає, що додаток складається з наборів процесів або потоків, кожен з яких працює зі своїм набором даних, обміну інформацією при роботі немає. Застосовна до обмеженого класу задач.

7. Написання програми, що реалізує алгоритм, в обраної моделі програмування і на обраному алгоритмічній мові;
8. Відображення (mapping). При запуску програми на паралельній комп'ютерній системі необхідно зіставити віртуальним виконавцям, що з'явилися на попередніх етапах парадигми програмування, реальні фізичні пристрої. Залежно від обраної моделі програмування це може здійснюватися як особою, що проводить обчислювальний експеримент, так і операційною системою.

2.2. Способи синхронізації паралельної взаємодії

Для функціонування додатків з паралельними обчисленнями потрібно призначити виконавців та диригента (керівника) що керуватиме роботою виконавців та збиратиме результати.

Розглянемо існуючі способи для роботи на паралельних обчислювальних системах:

2.2.1. Модель передачі повідомлень

Модель передачі повідомлень припускає, що додаток що виконується складається з набору процесів з різними адресними просторами, кожен з яких функціонує на своєму виконавці.

Взаємодія за допомогою передачі повідомлень: на кожному процесорі багатопроцесорної системи запускається однопоточний процес, який обмінюється даними з іншими процесами, що працюють на інших процесорах або ядрах процесора, за допомогою повідомлень. Процеси створюються явно, шляхом виклику відповідної функції операційної системи, а обмін

повідомленнями – за допомогою бібліотеки (наприклад, реалізація протоколу MPI), або за допомогою засобів мови. Обмін повідомленнями може відбуватися асинхронно, або с використанням методу «рандеву», при якому відправник блокований до тих пір, поки його повідомлення не буде доставлене. Асинхронна передача повідомлень може бути надійною (з гарантією доставки) або ненадійною [6].

Паралельні системи, засновані на обміні повідомленнями, найчастіше більш прості для розуміння, ніж системи з пам'яттю, що, і зазвичай розглядаються як більш досконалий метод паралельного програмування. Існує великий вибір математичних теорій для вивчення та аналізу систем з передачею повідомлень, включаючи модель акторів і різні види числень процесів. Обмін повідомленнями може бути ефективно реалізований на симетричних мультипроцесорах як з розділяється когерентної пам'яттю, так і без неї.

У паралелізму з розподіленою пам'яттю і з передачею повідомлень різні характеристики продуктивності. Зазвичай (але не завжди), накладні витрати пам'яті на процес і часу на перемикання завдань у систем з передачею повідомлень нижче, однак передача самих повідомлень більш накладна, ніж виклики процедур. Ці відмінності часто перекриваються іншими факторами, що впливають на продуктивність.

2.2.2. Модель розділеної пам'яті

Модель розділеної пам'яті припускає, що додаток складається з набору потоків виконання (thread), що використовують колективні змінні і примітиви синхронізації.

Модель розділеної пам’яті застосовують для того, щоб збільшити швидкість проходження даних між процесами. У звичайній ситуації обмін інформацією між процесами проходить через ядро. Техніка розділяється пам'яті дозволяє здійснити обмін інформацією не через ядро, а використовуючи деяку частину віртуального адресного простору, куди містяться і звідки зчитуються дані.

Після створення розділяється сегмента пам'яті будь - який з користувальницьких процесів може приєднати його до свого власного віртуального простору і працювати з ним, як із звичайним сегментом пам'яті. Недоліком такого обміну інформацією є відсутність яких би то не було засобів синхронізації, однак для подолання цього недоліку можна використовувати техніку семафорів.

Семафор – об'єкт, що обмежує кількість потоків, які можуть увійти в задану ділянку коду. Визначення введено Едсгером Дейкстрой. Семафори використовуються при передачі даних через пам'ять, що розділяється.

У програмному забезпеченні розподіленою пам'яттю називають:

- Метод міжпроцесної взаємодії (IPC), тобто спосіб обміну даними між програмами, що працюють одночасно. Один процес створює область в оперативній пам'яті, яка може бути доступна для інших процесів.
- Метод економії пам'яті, шляхом прямого звернення до тих вихідними даними, які при звичайному підході є окремими копіями вихідних даних, замість відображення віртуальної пам'яті або описаного методу. Такий підхід звичайно використовується для поділюваних бібліотек.

Оскільки обидва процеси можуть отримати доступ до загальної області пам'яті як до звичайної пам'яті, це дуже швидкий спосіб зв'язку (на відміну від інших механізмів IPC, таких як іменовані канали, UNIX-сокети або CORBA). З іншого боку, такий спосіб менш гнучкий, наприклад, процеси що обмінюються повинні бути запущені на одній машині (з перерахованих методів IPC тільки мережеві сокети, не плутати з сокетами домену UNIX, можуть вести обмін даними через мережу), і необхідно бути уважним, щоб уникнути проблем, при використання розподіленої пам'яті на різних ядрах процесора і апаратної архітектурі без когерентного кеша.

Обмін даними через пам'ять, що розділяється використовується, наприклад, для передачі зображень між додатком і X-сервером на Unix системах, або всередині об'єкта IStream що повертається в бібліотеці COM під Windows.

2.2.3. Модель розділених даних

Модель розділених даних припускає, що додаток складається з наборів процесів або потоків, кожен з яких працює зі своїм набором даних, обміну інформацією при роботі немає. Застосовна до обмеженого класу задач [1].

Оцінивши існуючі методики розпаралелювання зроблено висновок що для використання в PHP підходить лише модель передачі повідомлень, оскільки вона реалізується безпосередньо на високому рівні, в той час як інші моделі потребують інтегрованих в компілятор методів або ж можливість безпосереднього доступу до ресурсів операційної системи, чого PHP не має. Тому для реалізації паралельного виконання скористаємося можливостями веб-сервера, який виділяє окремий процес для кожного нового виклику скрипта.

2.3. Асимптотичний аналіз алгоритмів розпаралелювання

Для багатьох задач математичного моделювання можна сформулювати кілька алгоритмів, що вирішують поставлену задачу. Звичайно, мова йде лише про коректно працюючі алгоритми, тобто такі, які дають правильний результат на досить широкому наборі вхідних даних. Саме з ними ми будемо надалі мати справу, тому питання, пов’язані з коректністю, залишаться за межами нашого розгляду.

Як зрозуміти, чи є обраний алгоритм «достатньо хорошим» для його реалізації або потрібно пошукати що-небудь інше? Як правило, в більшості реальних задач присутні деякі параметри масштабу, пов'язані з обсягом вхідних даних. Наприклад, в задачах сортування масиву інформації таким параметром є розмір початкового масиву. У завданнях чисельного моделювання параметрами масштабу служать розміри розрахункової сітки.

Параметри масштабу завдання впливають на час роботи різних алгоритмів і на обсяг ресурсів, необхідних для їх ефективної реалізації (пам'ять, кількість виконавців і так далі). Для сучасних обчислювальних комплексів на перший план (хоча і не завжди) виходить саме час виконання алгоритму. Чим воно менше, тим краще для користувача. Введемо наступні позначення. Якщо завдання має один параметр масштабу n, то час виконання алгоритму A запишемо так: $TA\ (n)$. Для декількох параметрів масштабу $n_1, ..., n_k$ відповідне позначення буде $TA\ (n_1, ..., n_k)$ або $T\ (n)$. Ми будемо порівнювати різні алгоритми за часом виконання при одних і тих же значеннях параметрів масштабу з дотриманням таких умов (для простоти вважаємо, що у нас один параметр).

1. Оцінка $T\ (n)$ не може бути прив'язана до конкретної обчислювальної системи. Якщо для алгоритму A відома величина $TA\ (n)$ для одного комп'ютера, то, вимірявши для алгоритму B час $TB\ (n)$ на деякій іншій системі, не можна нічого сказати про порівняльну ефективність A і B. Отримані таким способом оцінки можуть свідчити про недоліки або переваги самих комп'ютерних комплексів, а не реалізованих на них алгоритмах. Для отримання коректних оцінок необхідно використовувати деяку єдину теоретичну модель обчислювальної системи.
3. Порівняння $TA\ (n)$ і $TB\ (n)$ на теоретичної моделі при малих значеннях n можливо, але марно. Для малого значення параметра масштабу навіть неефективний алгоритм виконується швидко. Для користувача навряд чи істотна різниця в часах виконання алгоритмів, якщо один з них працює 0.5 секунди, а другий - 0.1 секунди. Порівняння ефективності A і B повинно проводитися при великих значеннях n, в ідеалі при $n \to \infty$.
4. Нас буде цікавити не порівняння значень $TA\ (n)$ і $TB\ (n)$ при великих n, а порівняння їх темпів зростання. Якщо $TA\ (n) = 10^6 \times n^2$, а $TB\ (n) = n^3$, то при $n < 10^6$ алгоритм B ефективніше, але при $n > 10^6$ ефективніше виявиться вже алгоритм A.

Для подальшого викладу нам потрібні деякі стандартні форми запису, використовувані при асимптотичному аналізі поведінки функцій. Припустимо, що є дві функції f і g від цілочисельного невід'ємного аргументу n, що приймають невід'ємні значення. Тоді:

- $f(n) = O(g(n))$ тоді і тільки тоді, коли існують невід'ємні константи c і n_0 такі, що $f(n) \le cg(n)$ при $n \ge n_0$;
- $f(n) = \Omega(g(n))$ тоді і тільки тоді, коли існують невід'ємні константи c і n_0 такі, що $cg(n) \le f(n)$ при $n \ge n_0$;
- $f(n) = \Theta(g(n))$ тоді і тільки тоді, коли існують невід'ємні константи c_1, c_2 і n_0 такі, що $c_1 g(n) \le f(n) \le c_2 g(n)$ при $n \ge n_0$. Відзначимо, що якщо $f(n) = \Theta(g(n))$, *то* $f(n) = O(g(n))$ *і* $f(n) = \Omega(g(n))$.

Нехай $TA(n)$ і $TB(n)$ – час роботи на одній і тій же обчислювальній системі послідовних алгоритмів А і В відповідно, а $T_0(n)$ – теоретична оцінка часу роботи знизу довільного послідовного алгоритму для вирішення того ж самого завдання, тобто $T_0(n) = \Omega(T(n))$ для будь-якого алгоритму. Тоді будемо говорити наступне:

- Якщо $TA(n) = O(TB(n))$, то алгоритм А з поведінки не гірше алгоритму В.
- Якщо $TA(n) = \Omega(TB(n))$, то алгоритм А з поведінки не краща алгоритму В.
- Якщо $TA(n) = \Theta(TB(n))$, то алгоритми А та В по поведінці однакові.
- Якщо $TA(n) = \Theta(T0(n))$, то алгоритм А – оптимальний.

При цьому весь час вимірюються для найгірших вхідних даних.

Для оцінки часу роботи послідовних алгоритмів зазвичай використовується модель обчислювальної системи, що отримала назву оперативної пам'яті (Random Access Machine). Її поведінка регламентується наступними властивостями:

1. У системі є один процесор з одним ядром (один виконавець).
2. Осередки пам'яті для читання і запису доступні в довільному порядку.

3. Час доступу до пам'яті є Θ незалежно від того, є операція доступу читанням або записом.
4. Час виконання основних операцій на виконавця є Θ.

1.4. Нерівномірний розподіл розрахунків

При розподілі робіт між виконавцями актуальною є проблема рівномірного розподілу навантаження. Далеко не всі завдання можуть бути легко розподілені по підзадачах на вузли довільної існуючої системи. Більше того, навіть прості для розпаралелювання задачі часто не можуть бути розподілені рівномірно. Наприклад, така ситуація може виникнути при кількості підзадач, що не кратна числу процесів, а також у разі, якщо наявна обчислювальна система неоднорідна.

Розглянемо, наприклад, можливі варіанти розподілу приблизно однакових за тривалістю незалежних ітерацій циклу. Існує два основні підходи до розподілу навантаження в подібних ситуаціях – блокове і циклічний розподіл [2]. Також існує блочно-циклічний розподіл, який є комбінацією обох підходів. При блоковому розподілі ітерації розподіляються на кожен процес послідовними інтервалами параметра циклу:

```
int nloc = (n + (size - 1)) / size;
for (int i = nloc * rank; i < nloc * (rank + 1) && i < n; i++)
// ... виконання ітерації з номером i
```

При циклічному розподілі всі ітерації розподіляються по процесах послідовним чергуванням, або проріджуванням за номером ітерації з кроком, рівним кількості процесів:

```
for (int i = rank; i < n; i += size)
// ... виконання ітерації з номером i
```

Наприклад, у нас є завдання, що складається з десяти незалежних підзадач однаковою обчислювальної складності і чотири процеси на кластері з чотирьох

вузлів. Якби ми скористалися блоковим розподілом підзадач, ми б отримали розподіл е {{0,1,2} ,{3,4,5},{6,7,8},{9}}. При циклічному розподілі десяти підзадач на чотири процеси ми отримали б {{0,4,8},{1,5,9},{2,6},{3,7}}.

На цьому прикладі видно основна перевага циклічного розподілу перед блоковим – перший забезпечує більш рівномірне завантаження на однорідних системах в тих випадках, коли кількість підзадач не кратна кількості процесів. Інакше кажучи, досягається найменша різниця між зайнятістю окремих процесів. Саме тому щоб уникнути суттєвої нерівномірності завантаження процесів часто вдаються до циклічного розподілу підзадач.

Проте з різних причин блочний розподіл трапляється частіше циклічного, як з точки зору програмування, так і з точки зору продуктивності. Така ситуація найчастіше може виникнути в силу специфіки конкретної задачі. Наприклад, якщо дані для всіх підзадач лежать в послідовному масиві, і нам необхідно розподілити ці дані між вузлами, зручніше і швидше буде посилати дані декількох послідовних підзадач одним безперервним блоком, ніж декількома посилками по одній підзадачі. Також при вирішенні деяких конкретних завдань швидкість обчислень може бути підвищена шляхом використання будь-яких рекурентних співвідношень, що зв'язують сусідні підзадачі, і в цьому випадку нам також набагато зручніше схема блочного розподілу.

Щоб у цьому випадку нам не довелося миритися з більш високою нерівномірністю розподілу навантаження по процесах, ми можемо скористатися наступною простою схемою обчислення кількості підзадач для блочного розподілу.

Припустимо, є цикл в n приблизно однакових за тривалістю n ітерацій і потрібно розподілити його між процесами N, причому n не кратне N. Тоді на основі розподілу із залишком кількість ітерацій може бути представлено через цілі числа a та b у такій формі:

$$n = aN + b, 0 \leq b < N \qquad (1.1)$$

В цьому випадку ширина інтервалу ітерацій для кожного процесу $n_i,\ i = 0,$ ... , $N - 1$ може бути обчислена таким чином:

$$n_i = \begin{cases} a + 1, & i < b \\ a, & i \geq b \end{cases} \quad (1.2)$$

Ліва межа кожного інтервалу $n_i^l,\ i = 0,\ \ldots,\ N - 1$ може бути обчислена на основі тих же співвідношень:

$$n_i^l = \sum_{k=0}^{i} n_k - n_i = ai + \begin{cases} i, i < b \\ b, i \geq b \end{cases} \quad (1.3)$$

При цьому на кожен процес розподіляється інтервал ітерацій

$$n_i^l; n_i^l + n_i \quad (1.4)$$

Описану схему ілюструє наступний код:

```
int nloc, nleft; nloc = n / size + (rank < (n % size) ? 1 : 0);
nleft = (n / size) * rank + (rank < (n % size) ? rank : n % size);
for (int i = nleft; i < nleft + nloc; i++)
// ... виконання ітерації з номером i
```

У разі застосування такого підходу при вирішенні десяти підзадач в чотирьох процесах буде отримано розподіл {{0,1,2},{3,4,5},{6,7},{8,9}}.

У деяких випадках нам може знадобитися розробити програму, яка повинна буде працювати в неоднорідній системі, тобто системі, в яку входять вузли з різною продуктивністю. Наприклад, це може бути мережа з декількох машин з різними характеристиками. У цьому випадку рівномірний за часом розподіл навантаження на вузли обчислювальної системи знову лягає на плечі програміста. У загальному випадку це завдання дуже непросте, проте в деяких окремих випадках, яких досить немало, можна приблизно оцінити продуктивність кожного вузла і розподілити навантаження відповідно до виконаних оцінками. Така оцінка може бути зроблена перед початком обчислень на основі виміру часу виконання усіма вузлами деякої невеликої типовою для всієї задачі підзадачі зразка, багаторазове виконання якої займає переважний час в ході обчислень.

Наприклад, якщо у нас стоїть завдання перемноження двох щільних матриць, перед початком розподілу підзадач кожним обчислювальним вузлом може бути виконаний замір часу виконання операції множення матриці на вектор. Або, приміром, коли у нас стоїть завдання множення матриці на вектор, потрібна оцінка може бути виконана на основі виміру часу обчислення кожним вузлом скалярного добутку.

Після того, як розподіляючий процес отримав результати замірів з усіх вузлів, він може виконати розподіл великої кількості підзадач по вузлах відповідно до отриманих оцінок продуктивності.

Нехай кожен процес виконав свою підзадачу-зразок за час T_i, $i=0,...N-1$. Тоді швидкість виконання таких завдань кожним процесом в одиницю часу становить $1/T_i$. Сумарна швидкість виконання завдань всіма процесами одночасно дорівнює:

$$\sum_{i=0}^{N-1} 1/T_i$$

Тоді, якщо за весь період роботи сукупності процесів повинно бути виконано n завдань, розподілене на конкретний процес кількість має дорівнювати приблизно:

$$n \approx \frac{1/T_i}{\sum_{k=o}^{N-1} 1/T_k} * n \qquad (1.5)$$

Зрозуміло, в цій ситуації не уникнути неточностей. В результаті округлень до цілих чисел може виникнути ситуація, коли:

$$\sum_{i=0}^{N-1} n_i \neq n$$

Проблема може бути вирішена шляхом обчислення в якомусь конкретному процесі, наприклад, нульовому, суми всіх і коригування локального значення n_0 на величину:

$$n - \sum_{i=0}^{N-1} n_i$$

Права n_i^r і ліва n_i^l межі кожного інтервалу можуть бути обчислені з отриманих значень у вигляді часткової суми:

$$n_i^r = \sum_{k=0}^{i} n_k,\ \ n_i^l = n_i^r - n_i,\ \ i = 0, \dots N-1 \qquad (1.6)$$

При цьому з усіх ітерацій [0; n] на кожен процес розподіляється інтервал ітерацій:

$$[n_i^l; n_i^r],\ \ i = 0, \dots N-1$$

Очевидно, подібний спосіб оцінки продуктивності для розподілу навантаження в неоднорідних системах є дуже приблизними, внаслідок чого розбіжності по повному часу обчислень між процесами все одно можуть виникати. Наприклад, може підвести вибір елементарної операції для оцінки. Проте навіть у такій ситуації це розбіжність в неоднорідних системах, як правило, буде набагато нижче, ніж при рівномірному по числу підзадач розподілі навантаження. Інші ж шляху оцінки продуктивності (наприклад, по відношенню тактових частот) можуть ще не відповідати реальності внаслідок відмінностей, наприклад, у внутрішній архітектурі процесорів. Оптимальним тут, ймовірно, є визначення співвідношень обчислювальних швидкостей дослідним шляхом на реальних завданнях з подальшим збереженням цих величин як конфігураційні параметри для використання в подальшому подібних обчисленнях пробна експериментальна схема оцінки близька до методів моделювання на зразок Монте Карло, які також знаходять широке застосування в практиці, незважаючи на потенційну неточність при малій кількості експериментів.

Всі розглянуті шляхи розподілу навантаження були статичними, тобто розподіл навантаження в них відбувався один раз на весь період виконання. На практиці часто також використовується динамічний розподіл. У такій ситуації розподіл чергових підзадач на процеси відбувається динамічно по мірі виконання ними попередніх, приміром, якимсь керуючим процесом. Такий розподіл найбільш зручний для досягнення одночасного завершення виконання завдання, особливо в неоднорідних системах. Однак динамічний розподіл, як

правило, вимагає набагато більше комунікацій між процесами і відповідних втрат часу.

Найчастіше деякого процесу буває простіше виконати якусь роботу самому, ніж передавати дані для виконання цієї роботи іншому процесу і отримувати результат. Наприклад, за наявності не швидкої мережі керуючому процесу може виявитися набагато простіше і швидше обчислити скалярний добуток двох векторів самостійно, ніж доручати його обчислення процесу на іншому вузлі з відповідною передачею даних в обидві сторони.

Внаслідок цього виникає така ситуація, що динамічний розподіл навантаження, при всій своїй перспективності з точки зору можливості управління навантаженням під час виконання, у багатьох ситуаціях виявляється менш швидкодіючі варіантом в порівнянні зі статичним розподілом. Виходом може бути укрупнення блоків операцій, що розподіляються динамічно, тобто збільшення розмірів виділеної за один захід підзадачі з відповідним скороченням комунікацій.

Однак існують завдання, в яких підзадачі не тільки не рівні, але й не детерміновані по тривалості, тобто ми не знаємо наперед, як між собою співвідносяться тривалості виконання підзадач. У таких випадках без динамічного розподілу навантаження обійтися не виходить. Як правило, в таких ситуаціях зручно виділити один процес, який займатиметься розподілом підзадач на інші процеси в міру виконання з подальшим збором результатів.

1.5. Ярусно-паралельна форма програми

Ярусно-паралельна форма (ЯПФ) – розподіл вершин орієнтованого ациклічного графа на підмножини. Кожна з множин називається ярусом ЯПФ, і – його номером, кількість вершин в ярусі – його шириною. Кількість ярусів в ЯПФ називається його висотою, а максимальна ширина його ярусів – шириною ЯПФ. Для ЯПФ графа алгоритму важливим є той факт, що операції, яким відповідають вершини одного ярусу, не залежать одне від одного (не

перебувають у відношенні зв'язку), і тому свідомо існує паралельна реалізація алгоритму, в якій вони можуть бути виконані паралельно на різних пристроях обчислювальної системи. Тому ЯПФ графа алгоритму може бути використана для підготовки такої паралельної реалізації алгоритму.

Мінімальною висотою усіх можливих ЯПФ графа є його критичний шлях. Побудова ЯПФ з висотою, меншою критичного шляху, неможливо.

Якщо в складі ярусу можуть бути вершини, що знаходяться в різних відносинах (наприклад, паралельності або альтернативи, що типово для граф-схем паралельних алгоритмів), ярус називається перетином, а ЯПФ – безліччю перетинів. Наявність більше одного відносини між вершинами перетину істотно ускладнює більшість алгоритмів обробки.

1.5.1. Цілі і механізм побудови ярусно-паралельної форми

Графічне представлення схеми залежностей робіт один від одного називається мережевим графіком робіт. Мережевий графік робіт представляється спрямованим ациклічним графом, тобто ніяка робота не повинна прямо або побічно залежати від самої себе. Наприклад, на рис. 1.1 можна побачити мережевий графік деякої комплексної роботи, що складається з восьми робіт. Прямокутниками тут позначені роботи, стрілками – залежності між ними по вхідних і вихідних даних.

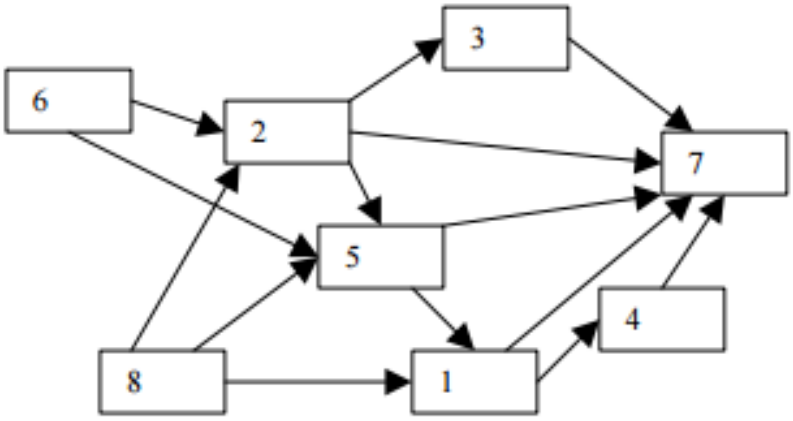

Рис.1.1. Мережевий графік комплексної роботи

У мережевому плануванні використовуються два способу представлення мережевого графіка робіт. У першому випадку, як на рис. 1.1, роботи представляються вершинами графа, залежності – дугами. У другому випадку роботи представляються дугами, вершинами представляються події їх завершення.

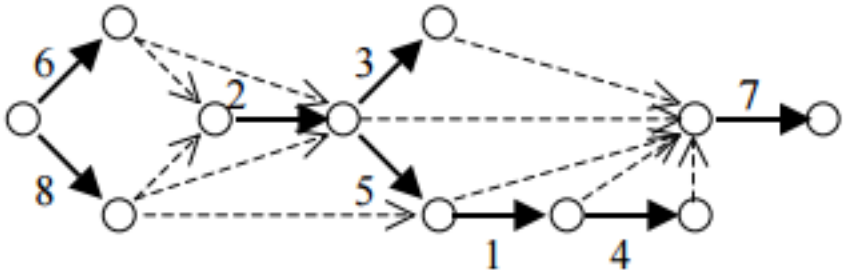

Рис.1.2. Представлення робіт дугами

Приклад уявлення другим способом комплексу робіт, зображеного на рис. 1.1, наведено на рис. 1.2. Тут пунктирними стрілками позначені так звані фіктивні роботи, що не займають ресурсів і характеризують логічний зв'язок. Зрозуміло, на цьому графіку можна опустити фіктивні роботи, що характеризують зв'язку, які неявно випливають з інших, в результаті чого отримуємо заключне уявлення мережевого графіка робіт (рис. 1.3).

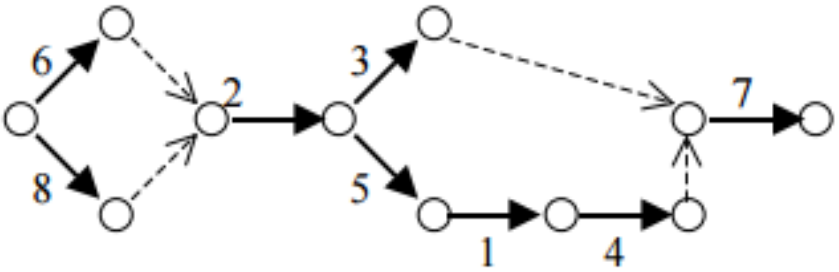

Рис.1.3. Представлення робіт дугами без фіктивних робіт

Обидва способи мають свої переваги в тих чи інших умовах. Другий спосіб зручний для аналізу часових характеристик робіт і не обтяжений зайвими зв'язками, тому широко використовується в мережевому плануванні. Перший

спосіб не потребує використання фіктивних робіт і забезпечує більш високу гнучкість при додаванні неврахованих залежностей. Оскільки ми розглядаємо випадок, коли аналіз часових характеристик не актуальний, ми будемо користуватися першим способом представлення мережевого графіка.

В якості альтернативи графічного представлення комплекс робіт може бути представлений структурною таблицею комплексу робіт, в якій зазначаються роботи, що входять у комплекс, а також залежності кожної роботи від інших.

Таблиця 1.1. Комплекс залежності робіт одна від одної

Номер роботи (виконавця)	Залежності
1	5, 8
2	6, 8
3	2
4	1
5	2, 6, 8
6	-
7	1, 2, 3, 4, 5
8	-

Рішення як таких завдань мережевого планування і управління зводиться до визначення оптимального розподілу ресурсів з метою отримання мінімального часу виконання всього комплексу на основі відомих залежностей часу виконання окремих робіт від кількості виділених їм ресурсів.

При виконанні паралельних обчислень в такій постановці розглядати завдання виявляється важко і часто не має сенсу, оскільки в однорідної системі виявляється нічим управляти – ми не можемо вплинути на швидкість виконання конкретної підзадачі, а в неоднорідній виявляється складно задати залежності величин часу виконання кожної підзадачі.

Нам цікавий лише перший етап вирішення завдання пошуку критичного шляху – впорядкування заданого мережного графіка робіт і побудови ярусно-паралельної форми програми.

Процедура побудови ярусно-паралельної форми (далі - ЯПФ) досить проста і полягає в поданні мережевого графіка у формі послідовних ярусів робіт, в межах кожного з яких роботи можуть виконуватися незалежно один від одного. Побудова ЯПФ легко здійснюється на основі структурної таблиці комплексу робіт і зводиться до додавання стовпця з присвоєними номерами ярусів. Роботами нульового ярусу вважаються всі роботи, виконання яких не залежить від інших. Роботи першого ярусу – ті, які спираються тільки на роботи нульового ярусу. Роботи кожного ярусу з подальших повинні спиратися тільки на роботи більш ранніх стосовно нього ярусів, серед яких повинна бути як мінімум одна робота безпосередньо попереднього ярусу. У таблиці 1.2 можна побачити приклад розподілу по ярусах робіт комплексу, наведеного у таблиці 1.1.

Таблиця 1.2. Розподіл робіт по ярусах

Номер роботи (виконавця)	Залежності	Ярус
1	5, 8	3
2	6, 8	1
3	2	2
4	1	4
5	2, 6, 8	2
6	-	0
7	1, 2, 3, 4, 5	5
8	-	0

Після виконання розподілу по ярусах всіх робіт комплексу, мережевий графік може бути зображений з урахуванням розбиття по ярусах. Приклад розбитого по ярусах комплексу робіт можна побачити на рис. 1.4.

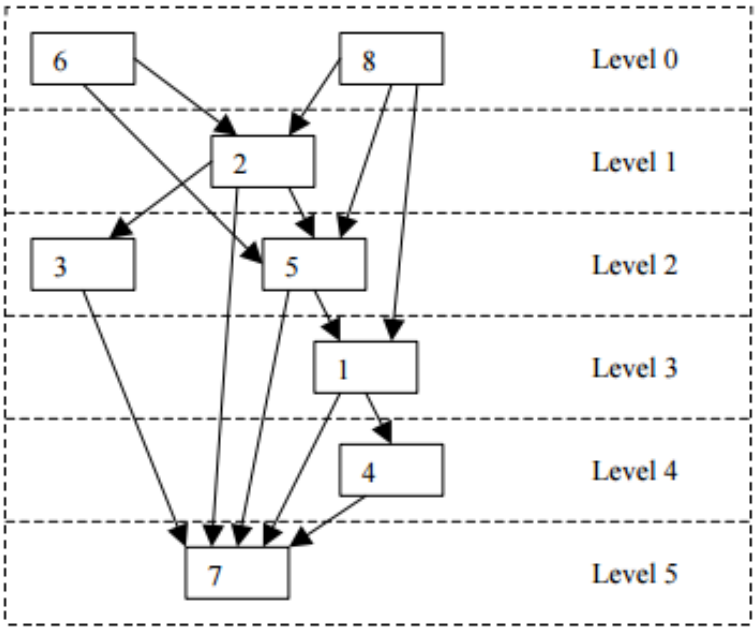

Рис.1.4. Мережевий графік з розбиттям по ярусах

Така форма організації програми носить назву ЯПФ програми. Перевага приведення програми до ЯПФ в можливості паралельного виконання в рамках одного ярусу всіх включених в нього робіт. Кількість робіт в кожному ярусі називається шириною ярусу, кількість ярусів – висотою ЯПФ, максимальна ширина ярусу – шириною ЯПФ програми.

Ці характеристики при деяких припущеннях дозволяють нам судити про властивості побудованої таким чином програми. Зокрема, за умови, що всі роботи приблизно однакові за тривалістю, ширина ЯПФ характеризує необхідну максимальну кількість необхідних паралельних ресурсів під час виконання. У разі наявності необхідної кількості паралельних ресурсів час виконання комплексної роботи буде приблизно дорівнює добутку часу виконання однієї роботи на висоту ЯПФ.

Розпаралелювання програм з використанням ЯПФ погано погоджено з конструкціями існуючих мов програмування, чим обумовлена складність його

виконання і відповідне вкрай рідкісне використання. Дійсно, більшість популярних сьогодні мов програмування, як правило, спочатку призначені для послідовних обчислень і тому погано пристосовані для будь-якого розпаралелювання. Проте слід враховувати, що написання складної програми (а паралельна програма, як правило, є складною) завжди вимагає зусиль, і пошук легких шляхів у даному випадку працює в більшості випадків проти нас. У зв'язку з цим для побудови паралельних програм ми запропонуємо далеко не новий підхід, що полягає у відділенні функціональності використовуваної схеми розпаралелювання, а саме побудови ЯПФ та виконання як такого розпаралелювання, від специфіки конкретної задачі. У цьому випадку написання коду, відповідального за паралельне виконання комплексу робіт, зрозуміло, буде не легше, ніж раніше, однак може бути виконано лише один раз, при цьому використовуватися такий код може при вирішенні широкого кола завдань.

1.6. Висновки до розділу

1. Проведено огляд методик розпаралелювання коду.
2. Оцінено переваги та недоліки послідовної і паралельної парадигми виконання коду.
3. Розглянуто способи синхронізації паралельної взаємодії:
    - модель передачі повідомлень;
    - модель розподіленої пам’яті;
    - модель розділенних данних.
4. Розглянуто асимптотичний аналіз алгоритмів, та способи рівномірного розподілу навантаження між виконавцями. Представлено програму у ярусно-паралельній формі.

2. РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОСТІ В PHP

На сьогоднішній день модель обміну повідомленнями (message passing) є найбільш широко використовуваною моделлю паралельного програмування. Програми цієї моделі, створюють безліч процесів, з кожним з яких асоційовані локальні дані. Кожен процес ідентифікується унікальним ім'ям. Процеси взаємодіють, посилаючи й одержуючи повідомлення.

Модель обмін повідомленнями не накладає обмежень ні на динамічне створення процесів, ні на виконання декількох процесів одним процесором, ні на використання різних програм для різних процесів. Просто, формальні описи систем обміну повідомленнями не розглядають питання, пов'язані з маніпулюванням процесами, Однак, при реалізації таких систем доводиться приймати якесь рішення щодо цього. На практиці склалося так, що більшість систем обміну повідомленнями при запуску паралельної програми створює фіксоване число ідентичних процесів і не дозволяє створювати і руйнувати процеси протягом роботи програми.

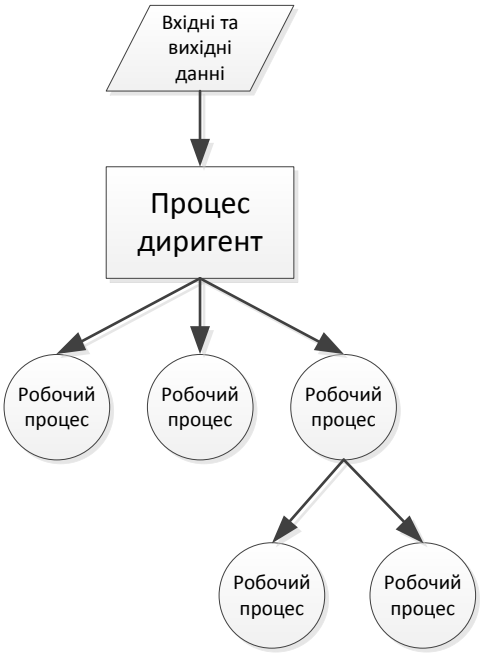

Рис. 2.1. Структура моделі передачі повідомлень

В таких системах кожен процес виконує одну і ту ж програму, але працює з різними даними, тому про такі системи кажуть, що вони реалізують SPMD (single program multiple data – одна програма багато даних) модель програмування. SPMD модель прийнятна і досить зручна для широкого діапазону додатків паралельного програмування, але вона ускладнює розробку деяких типів паралельних алгоритмів.

Модель передачі повідомлень характеризується наступними властивостями:

1. Паралельне обчислення складається з одного або більше одночасно процесів що виконуються, число яких може змінюватися протягом часу виконання програми.
2. Процес – це послідовна програма з локальними даними. Процес має вхідні і вихідні порти, які служать інтерфейсом до середовища процесу.
3. На додаток до звичайних операцій процес може виконувати наступні дії: послати повідомлення через вихідний порт, отримати повідомлення з вхідного порту, створити новий процес і завершити процес.
4. Посилана операція асинхронна – вона завершується відразу, не чекаючи того, коли дані будуть отримані. Операція отримання синхронна: вона блокує процес до моменту надходження повідомлення.
5. Процеси можна розподіляти по фізичним процесорам довільним способами, причому використовуване відображення (розподіл) не впливає на семантику програми. Зокрема, безліч процесів можна відобразити на одиночний процесор [2].

Для розподілу навантаження по процесам, ми можемо скористатися наступною простою схемою обчислення кількості підзадач для блочного розподілу. Припустимо, є цикл в приблизно однакових за тривалістю ітерацій і потрібно розподілити його між N процесами, причому n не кратне N. Тоді на основі розподілу із залишком кількість ітерацій може бути представлено через цілі числа a, b в такій формі:

$$n = aN + b, 0 \leq b < N \tag{2.1}$$

В цьому випадку ширина інтервалу ітерацій для кожного процесу $n_i,\ i = 0,$ ... , $N - 1$ може бути обчислена таким чином:

$$n_i = \begin{cases} a + 1, & i < b \\ a, & i \geq b \end{cases} \tag{2.2}$$

Ліва межа кожного інтервалу $n_i^l,\ i = 0,\ \ldots,\ N - 1$ може бути обчислена на основі тих же співвідношень:

$$n_i^l = \sum_{k=0}^{i} n_k - n_i = ai + \begin{cases} i, i < b \\ b, i \geq b \end{cases} \tag{2.3}$$

При цьому на кожен процес розподіляється інтервал ітерацій

$$n_i^l; n_i^l + n_i \tag{2.4}$$

У дослідженні для реалізації скрипта диригента використаємо 3 різні підходи для комунікації між диригентом і виконавцями, а саме за допомогою: бібліотеки curl, потоків данних (streams) та програмного інтерфейсу сокетів (socket).

У якості виконавців можуть виступати окремі скрипти або скрипт запущений декілька разів але з різними вхідними параметрами. Скрипт диригент викликає виконавців, передає вхідні параметри та отримує результат виконання.

Таким чином диригент та кожен виконавець працюватимуть у різних потоках і операційна система, за наявності ресурсів, зможе забезпечити їх паралельне виконання [4].

Для моделювання розрахунків можна обрати алгоритм розрахунку числа Пі, шляхом чисельного інтегрування.

$$\int_0^1 \frac{4}{1+x^2}\, dx = \pi \tag{2.5}$$

$$\int_0^1 \frac{4}{1+x^2}\, dx = 4 * arctg(x) = 4 * \left(\frac{\pi}{4} - 0\right) = \pi \tag{2.6}$$

Візьмемо найпростіший з методів чисельного інтегрування – метод прямокутників.

Оголосимо деяку кількість n прямокутників на які ми розіб'ємо нашу площу під інтегралом, порахуємо основу:

$$h = 1.0/n \tag{2.7}$$

і всю площу, порахувавши значення функції в кожному прямокутнику і помножимо на основу.

```
sum = 0.0;
for (i = 1; $i <= n; i ++) {
  x    = h * ($i - 0.5);
  sum += (4.0 / (1.0 + x * x));
}
pi = h * sum;
```

Виконавши обрахунок послідовно, отримали результат у 10.894656719318 с.

Далі виконаємо декомпозицію алгоритму декомпозицію використовуючи метод програмної конвеєризації циклів, суть якого полягає в тому, що ітерації циклу поєднуються при виконанні з невеликим зрушенням по відношенню один до одного.

Відповідно попередній код зазнає наступних змін:

```
h   = 1.0 / n;
sum = 0.0;
for (i = rank + 1; i <= n; $i += size) {
  x    = h * (i - 0.5);
  sum += (4.0 / (1.0 + x * x));
}
pi = h * sum;
```

З метою збільшення точності вимірів для кожної кількості виконавців виконаємо по 10 ітерацій і зафіксуємо мінімальний час обрахунку. Мінімальний час використовується через те, що саме у поточний момент процесор був завантажений найменше сторонніми процесами.

Проведемо експериментальні виміри для обчислень з використанням від 1 до 15 виконавців, з проходом у 40,000,000 інтервалів, для кожного з варіантів комунікації.

2.1. Використання бібліотеки curl

cURL – назва проекту і крос-платформового програмного засобу, що служить для передачі даних через Інтернет. cURL – це утиліта для організації вибірки даних з вебу, що надає можливість гнучкого формування запиту із завданням таких параметрів, як cookie, user_agent, referrer і будь-яких інших заголовків. cURL – це додаткова можливість оперувати з файлами на стороні сервера сторінок Інтернету за допомогою параметрів, що можуть бути переданими в рядку URL. За допомогою cURL можна, наприклад, отримати html-сторінку, не використовуючи для цього браузер.

Крім http-запитів, cURL підтримує SMTP, IMAP, POP3, Telnet, FTP, LDAP, RTSP, RTMP та інші мережеві протоколи.

Разом з утилітою cURL, проект паралельно розвиває бібліотеку libcurl, що надає API для задіяння всіх функцій cURL в програмах на таких мовах, як C, Perl, PHP, Python.

Лістинг скрипта диригента з використанням бібліотеки cURL наведено у додатку А.

Завдяки функції curl_multi_init ініціалізується контейнер для окремих з'єднань curl (multi_curl), саме він дозволить нам проводити операції з ними паралельно.

Далі в циклі ініціалізується з'єднання (назвемо його потік) для кожного виконавця з масиву, попутно додаючи його в multi_curl і в масив завдань $tasks. $tasks – масив, в якому ключами є адреси виконавців, а значеннями – відповідні дескриптори curl. Функція curl_multi_add_handle додає до дескриптора багато запитне з'єднання окреме створене з'єднання.

Далі запускається цикл для початку роботи диригента. Функція curl_multi_exec одночасно відправляє на виконання всі оголошені потоки, при цьому в змінну $active заноситься кількість виконуваних потоків.

В основному циклі, відбуваються головні дії. Він виконується до тих пір, поки є незавершені потоки або поки не сталася помилка. Далі викликається

функція curl_multi_select, яка перевіряє готовність якого-небудь з потоків до подальших дій з ним. Потім, функцією curl_multi_info_read отримуємо інформацію про виконавця.

Функція curl_multi_info_read повертає масив, в якому нас цікавлять ключі 'msg' і 'handle'. За 'msg' ми перевіряємо, чи завдання виконано потік, а по 'handle' дізнаємося його дескриптор. Отримавши дескриптор, шукаємо по масиву завдань, до якого виконаця він ставиться і записуємо замість нього отриманий результат, одержувани функцією curl_multi_getcontent.

Тепер, коли даний потік виконав своє завдання, видаляємо його з масиву виконаців, а потім закриваємо.

Після завершення всіх потоків функцією curl_multi_close припиняємо виконання.

Результати експериментального дослідження з використанням бібліотеки cURL наведено у таблиці 2.1.

Таблиця 2.1. Заміри часу виконання розрахунків, бібліотека cURL

Кількість виконавців	**Час виконання, с**
1	12.719146952305
2	7.1828472849114
3	6.6534550692749
4	4.416324937439
5	4.365356220032
6	4.3051870303114
7	4.2451830142731
8	4.1752230947653
9	4.3051452501343
10	4.5287639874023
11	4.6274520814263
12	4.7567330032082

Продовження таблиці 2.1.

Кількість виконавців	Час виконання, с
13	4.9951435844312
14	5.1523973468291
15	5.5612035224124

2.2. Використання stream функцій

Потоки були вперше введені в PHP 4.3.0, як інструмент для роботи з файлами, мережевого обміну, стиснення даних і виконання інших операцій за допомогою одного загального набору функцій. Висловлюючись простими поняттями, потік (stream) – це ресурс (resource), який веде себе, як джерело безперервної послідовності даних. Тобто з потоку можна послідовно читати дані, так само як і записувати в нього. Також можливе переміщатися (fseek()) в різні позиції всередині потоку.

stream_socket функції є частиною streams, за допомогою них можна з’єднуватися або писати / читати дані, не чекаючи завершення попередньої операції.

Лістинг скрипта диригента з використанням бібліотеки streams наведено у додатку Б.

Спочатку створюються три масиви: $rtasks – в нього будуть заноситися виконавці, з яких можна читати дані, $wtasks – для виконавців, готових до запису отримання завдання, і $results – масив для збереження результатів роботи виконавців.

Потім у циклі функцією stream_socket_client відкриваються сокети для кожного з виконавців, при цьому ставиться прапор STREAM_CLIENT_ASYNC_CONNECT, який вказує, що кожне наступне з'єднання треба відкривати не чекаючи завершення відкриття попереднього.

Прапор STREAM_CLIENT_CONNECT вказує, що ми створюємо саме клієнтське підключення.

Дескриптори з'єднань(виконавців) заносяться в масив $wtasks.

while ($wtasks || $rtasks) – початок головного циклу, і відразу масивам $rtasks_ і $wtasks_, переданим у функцію stream_select, присвоюються масиви, в яких зберігаються дескриптори виконавців для читання і запису відповідно. Функція stream_select відстежує наявність даних в потоках і приймає на вхід декілька параметрів: перший – це масив дескрипторів, в яких очікується завершення операції читання, другий – масив з дескрипторами, які очікували завершення запису, третій – для особливих випадків (надходження "out-of-band data " – позасмугових даних) і четвертий параметр – це таймаут. Масиви передаються в функцію за вказівниками, і як тільки функція повертає число більше 0, в них будуть знаходиться виконавці, готові до дій.

У циклі foreach ($wtasks_ as $sh) – перевіряємо, до яких виконавців можна передати дані (треба щось відправити, перш ніж нам прийде відповідь-результат), і якщо такі виконавці є, то видаляємо з масиву для запису поточний дескриптор, поміщаємо його в масив для читання, і відправляємо заголовки, необхідні для отримання контенту.

Далі, foreach ($rtasks_ as $sh), переглядаємо виконавців, що готові для зчитування з них даних, забираємо вміст, видаляємо дескриптор з масиву для читання і закриваємо з'єднання з виконавцем. Результат заносимо в масив $results. Повторюємо ці операції до тих пір, поки у нас залишаються які-небудь відкриті з'єднання.

Після роботи скрипта потрібний нам контент знаходиться в масиві $results.

Результати експериментального дослідження з використанням stream функцій наведено у таблиці 2.2.

Таблиця 2.2. Заміри часу виконання розрахунків, stream функції

Кількість виконавців	**Час виконання, с**
1	11.615467352301
2	6.9838479849124
3	5.5562630693849
4	4.411224937439
5	4.365301415032
6	4.3056370283214
7	4.2451356242731
8	3.982454947663
9	4.3298342301343
10	4.547639874023
11	4.5974320824263
12	4.6967760032082
13	4.851435844312
14	5.1523973468291
15	5.5612035224124

2.3. Використання асинхронних сокетів

Сокети – назва програмного інтерфейсу для забезпечення обміну даними між процесами. Процеси при такому обміні можуть виконуватися як на одній ЕОМ, так і на різних ЕОМ, пов'язаних між собою мережею. Сокет – абстрактний об'єкт, що представляє кінцеву точку з'єднання.

Слід розрізняти клієнтські і серверні сокети. Клієнтські сокети грубо можна порівняти з кінцевими апаратами телефонної мережі, а серверні – з комутаторами. Клієнтський додаток (наприклад, браузер) використовує тільки клієнтські сокети, а сервер (наприклад, веб-сервер, якому браузер посилає запити) – як клієнтські, так і серверні сокети.

Інтерфейс сокетів вперше з'явився в BSD Unix. Програмний інтерфейс сокетів описаний в стандарті POSIX.1 і в тій чи іншій мірі підтримується всіма сучасними операційними системами.

Кожен процес може створити сонет що слухає (серверний сокет) і прив'язати його до якого-небудь порту операційної системи (в UNIX непривілейованих процеси не можуть використовувати порти менше 1024). Хто слухає процес зазвичай знаходиться в циклі очікування, тобто прокидається при появі нового з'єднання. При цьому зберігається можливість перевірити наявність з'єднань на даний момент, встановити тайм-аут для операції і т.д.

Кожен сокет має свою адресу. ОС сімейства UNIX можуть підтримувати багато типів адрес, але обов'язковими є INET-адреса та UNIX-адреса. Якщо прив'язати сокет до UNIX-адреси, то буде створено спеціальний файл (файл сокета) по заданому шляху, через який зможуть повідомлятися будь-які локальні процеси шляхом читання / запису з нього. Сокети типу INET доступні з мережі і вимагають виділення номера порту.

Зазвичай клієнт явно під’єднується до слухача, після чого відбувається читання або запис через його дескриптор передаються дані між ним і сервером.

У нашому випадку сокет буде представляти собою з’єднання між нашою диригентом і виконавцем, з якого потрібно отримати необхідну інформацію.

Лістинг скрипта диригента з використанням сокетів наведено у додатку В.

Можна помітити, що код несильно відрізняється від коду для stream-функцій.

Кожен сокет ініціалізується і переводиться в неблокуючий режим. Потім у циклі проводиться перевірка готових до читання або запису сокетів, і якщо такі існують, проводиться відповідна операція. Всі дії протікають в асинхронному порядку, тобто не чекають закінчення попередньої операції. Наприклад, якщо дані від 3 виконавця надійшли раніше, ніж від першого, то будуть оброблятися саме вони, хоча з’єднання з першим відбулось раніше ніж з третім. Можна

сказати, що все відбувається в кілька "потоків", коли паралельно очікується готовність кожного сокета до якої-небудь дії.

Отже, у нас є масив з адресами виконавців. У рядку foreach ($urls as $url) запускається цикл для ініціалізації з'єднання з кожним виконавцем. У $sh = socket_create(AF_INET, SOCK_STREAM, SOL_TCP) створюється сокет, потім в socket_set_option($sh, SOL_SOCKET, SO_RCVTIMEO, array("sec" => 10, "usec" => 0)); socket_set_option($sh, SOL_SOCKET, SO_SNDTIMEO, array("sec" => 10, "usec" => 0)); рядках ми задаємо таймаут для читання і запису відповідно. У функції socket_set_nonblock задаємо неблокіруючій режим, який перетворює кожен наш сокет в асинхронний. Це необхідно для того, щоб після чергової операції над сокетом не відбувалося очікування її закінчення, а скрипт продовжував роботу. У $wtasks[$url] = $sh; заносимо ініціалізований сокет в масив сокетов для запису.

Основний цикл while ($wtasks || $rtasks). Він триватиме до тих пір, поки у нас залишаються сокети, в які потрібно записати або з яких треба прочитати.

У масив $rtasks_ присвоюємо масив сокетів, з яких повинна бути прочитана інформація. У $ wtasks_ заноситься масив з сокетами, в які необхідно зробити запис.

Головна функція в цьому циклі – socket_select. Вона приймає на вхід масиви сокетів (виконавців), які очікують читання, які очікують запис і масив для виняткових ситуацій, четвертий параметр, переданий в дану функцію - таймаут. Параметри-масиви передаються по посиланню, тобто коли відбудеться очікувана подія (socket_select в цьому випадку поверне число більше 0), в них з'являться відповідні дескриптори.

Якщо знайшлися сокети, доступні для запису, шукаємо адресу, відповідну кожному з готових сокетів, і функцією socket_write записуємо в нього заголовки для отримання відповіді. Так само в unset($wtasks[$url]); видаляємо дескриптор з масиву для запису, і заносимо його в $rtasks[$url] = $sh; в масив для читання, щоб потім отримати відповідь.

Якщо ж є сокети, готові нам відповісти, зчитуємо з них інформацію функцією socket_read.

У масиві $results міститься результат роботи.

Результати експериментального дослідження з використанням сокетів наведено у таблиці 2.3.

Таблиця 2.3. Заміри часу виконання розрахунків, сокети

Кількість виконавців	**Час виконання, с**
1	11.314646959305
2	6.0763478279114
3	4.1122350692749
4	3.936224937439
5	3.5412030220032
6	3.3061890602112
7	3.2371850013733
8	3.1852010917664
9	3.2951879501343
10	3.5212020874023
11	3.6431920814514
12	3.8301930046082
13	3.9851805842514
14	4.1301960468292
15	4.5512030124664

2.4. Бібліотека для реалізації паралельності

В рамках дослідження розроблено бібліотеку, що значно спрощує використання реалізацій паралельності в PHP.

Бібліотека надає наступні можливості:

- Синхронний режим, зі збереженням інтерфейсу при відсутності необхідних розширень для розпаралелювання;

- Багаторазове використання виконавців;
- Повноцінний обмін данними між диригентом і виконавцями (запуск з аргументами і отримання результату після завершення);
- Обробка помилок виконання;
- Максимальна швидкодія.

Лістинг скрипта диригента з використанням сокетів наведено у додатку Г.

Бібліотека надає простий інтерфейс для створення класів-потоків. Які насправді використовують окремі процеси для асинхронної роботи. Можна посилати події з потоку, повертати результати, використовувати один потік безліч разів передаючи йому аргументи запуску або створити пул потоків для виконання завдання не переймаючись тим, що робота відбувається в різних процесах.

Автоматично вибирається розширення для роботи з сокетами. Якщо доступно, то використовується розширення sockets, що дає поліпшення продуктивності. В іншому випадку задіюється stream. Дочірній процес час від часу перевіряє існування батьківського. Якщо він раптом завершиться, то дочірній автоматичний завершиться. Всі ресурси використовуються потоком або пулом потоків автоматично очищаються при виклику деструктора. Але їх можна очистити примусово якщо викликати метод cleanup. У цьому випадку потік / пул більше не можна використовувати. При стандартних налаштуваннях потік ініціалізується заздалегідь, відразу при створенні класу. Якщо встановити параметр prefork в false, то форк буде відбуватися тільки в момент запуску завдання.

Загалом параметрів, що настроюються досить багато. Зміна імені дочірнього процесу після форку (параметр pName конструктора), таймаут на час виконання завдання (timeoutWork), таймаут на максимальний час очікування завдань дочірнім процесом (timeoutMaxWait), таймаут на час пре-ініціалізації

(timeoutInit), розміри буферів для читання сокетів (pipeReadSize , pipeMasterReadSize).

Також присутній режим налагодження в якому виводиться докладна інформація про те, що саме і де виконується.

Таким чином для запуску в окремому «потоці» досить наступного коду:

```
class ExampleThread extends Thread
{
    protected function process()
    {
        // Обчислення тут
    }
}
$thread = new ExampleThread();
$thread->wait()->run();
```

Якщо є все необхідне для повноцінної роботи, то завдання буде виконано асинхронно. Якщо ні, то все буде працювати в синхронному режимі.

Результати експериментального дослідження з використанням бібліотеки розпараралелювання наведено у таблиці 2.4.

Таблиця 2.4. Заміри часу виконання розрахунків, бібліотека

Кількість виконавців	**Час виконання, с**
1	11.414973359321
2	6.1432836279134
3	4.1521340395141
4	3.996214547418
5	3.5812430530864
6	3.3566553382141
7	3.2871276033253
8	3.2137560928664
9	3.6967760032082

Продовження таблиці 2.4.

Кількість виконавців	Час виконання, с
10	3.5583520836823
11	3.6559035813854
12	3.8323740356082
13	3.981740583484
14	4.137639046392
15	4.5614281124852

2.5. Порівняння результатів розпаралелювання

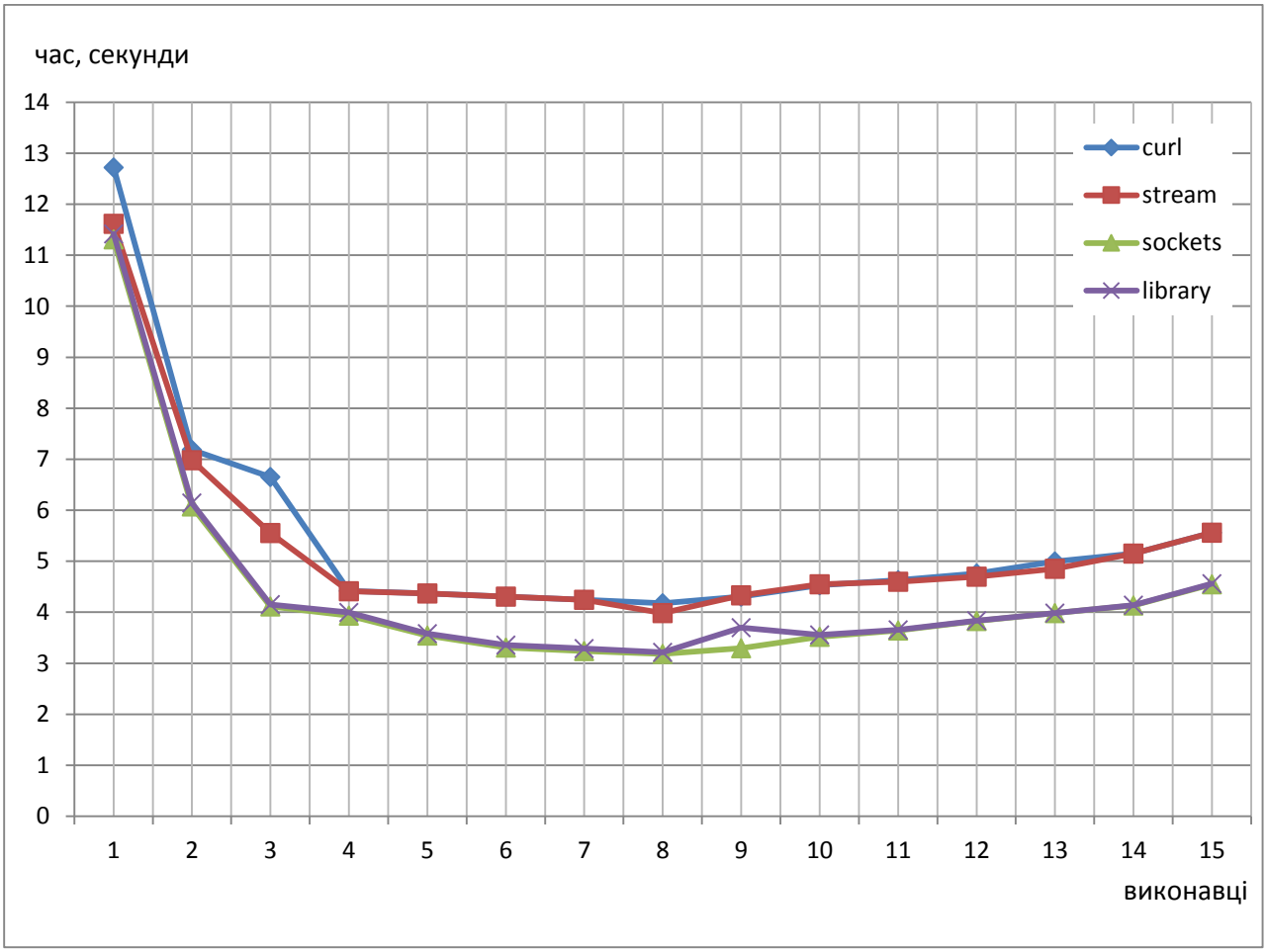

Рис.2.2. Порівняння результатів розпаралелювання різними способами

З результатів експерименту видно що кожен спосіб доволяє виконувати код у паралельному режимі, чітко видно спад часу виконання обрахунків зі

збільшенням кількості виконавців, при цьому розбіжність у часі виконання, між різними способами, невелика.

Таким чином за критерієм ефективності способи реалізації паралельності розташовуються наступним чином:

1. Асинхронні сокети;
2. Бібліотека для реалізації паралельності;
3. Stream функції;
4. Бібліотека curl.

Використовуючи найшвидший спосіб – асинхронні сокети, при розрахунку в 1 потік, задіяне одне ядро процесора і розрахунок триває 11.3 с. Зі збільшення потоків час обрахунку зменшується і на 8 потоках зменшується до 3.1 с, що дає виграш у швидкодії у 72% та доводить спроможність інтерпретатора PHP до вирішення обрахунків з розпаралелюванням виконання завдань. Але при перевищенні кількості ядер час обрахунку знову починає зростати, що доводить недоцільним використання розпаралелювання на одно процесорних (одноядерних) системах.

2.6. Висновки до розділу

1. Доведено спроможність інтерпретатора PHP паралельно виконувати програмний код;
2. Реалізовано паралельне виконання з використанням 3 різних засобів:
    - Бібліотека curl;
    - Stream функції;
    - Асинхронні сокети.
3. Розроблена бібліотека-обгортка для спрощення реалізації паралельності використовуючи досліджені способи;
4. Проведено порівняння результатів ефективності розпаралелювання;

3. ДОСЛІДЖЕННЯ МЕТОДИКИ РОЗПАРАЛЕЛЮВАННЯ НА ОСНОВІ РЕАЛЬНОГО ПРОЕКТУ

Для дослідження методики розпаралелювання обрано підсистему рейтингу порталу «Бізнес-Гід», що була розроблена мною раніше. Наразі підсистема та алгоритм розрахунку є недосконалими. Через архітектурні особливості системи данні для розрахунку рейтингу для кожного підприємства отримуються з бази данних послідовно, проводиться обрахунок, потім запис в базу данних. Оскільки система налічує більше 500000 компаній, то процедура розрахунку займає тривалий час, 1 годину 38 хвилин.

3.1. Архітектура системи

Система Бізнес-Гід – це масштабна, високонавантажена система. Сам же термін "високонавантажена система" при цьому не має чіткого визначення. Але можна сказати, що високонавантажена система це:

- Багатокористувацька система;
  - Тобто в один момент часу з нею працює більш ніж одна людина. Зараз, користувачі порталу – це тисячі і десятки тисяч людей, кількість яких постійно росте.
- Розподілена система;
  - Високонавантажені системи є системами розподіленими, тобто працюють більш ніж на одному сервері. Вимога розподіленості випливає з таких причин:
    - Необхідність обробляти зростаючі обсяги даних;
    - Здатність зберігати роботу системи у випадках відмови частини серверів.
- Постійний розвиток;
  - Якщо додаток являє хоч якийсь інтерес, то навіть якщо нічого не робити, аудиторія зростатиме просто із зростанням аудиторії

Інтернету. Тому характерною рисою високонавантажених систем є не просто велика кількість користувачів, а й динаміка збільшення кількості користувачів.

- Зростаюча кількість користувачів може призвести до того, що така ж динаміка буде і з даними. Тому в контексті високонавантажених систем коректніше говорити не про кількість, а про зростаючу кількість користувачів і даних.

- Інтерактивність.
  - Інтерактивність – одна з основоположних якостей високонавантаженої системи. Інтерактивність означає, що користувач після того як послав запит сидить і чекає відповіді, а люди, як відомо, чекати не люблять. При цьому більшість Інтернет-додатків не є критично важливими в житті людей. Тому, якщо будуть великі затримки з відповідями, відвідувач не буде користуватись системою.

    З цього правила безумовно є винятки. Наприклад, якщо користувач зробив покупку в Інтернеті, повідомивши свою платіжну інформацію третім особам, швидше за все він дочекаєтеся відповіді системи, навіть якщо вона буде відповідати довше хвилини. Але навряд чи повернеться, щоб зробити іншу покупку.

Систему Бізнес-Гід реалізовано на основі трирівневої архітектури, яка передбачає наявність наступних компонентів програми: клієнтський застосунок (тонкий клієнт), підключений до сервера застосунків, який в свою чергу підключений до серверу бази даних.

У порівнянні з клієнт-серверною або файл-серверною архітектурою можна виділити такі переваги трирівневої архітектури:

- Масштабованість.

- Конфігурованість – ізольованість рівнів один від одного дозволяє (при правильному розгортанні архітектури) швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів.
- Високий рівень безпеки.
- Висока надійність.
- Низькі вимоги до швидкості каналу (мережі) між терміналами і сервером за стосунків.
- Низькі вимоги до продуктивності і технічних характеристик терміналів, як наслідок зниження їхньої вартості. Терміналом може виступати не тільки комп'ютер, але і, наприклад, мобільний телефон.

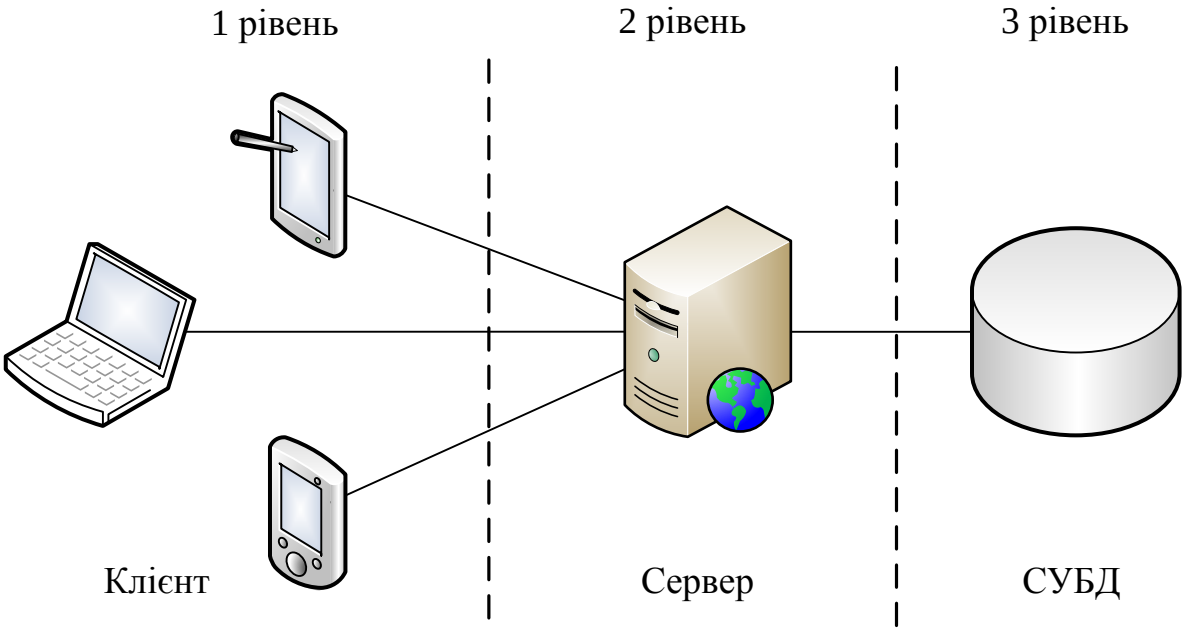

Рис. 3.1. Трирівнева архітектура

3.2. Алгоритм розрахунку рейтингу

Рейтинг підприємства (*Ro*) розраховується за наступною формулою:

$$Ro = P + Rв + Ra$$

Де:

P – Придбаний пакет

Rв – Внутрішній рейтинг

Ra – Рейтинг адміністратора

Рейтинг пакету (*P*), може приймати наступні значення:

★★★★★ - 5000

★★★★ - 4000

★★★ - 3000

★★ - 2000

В залежності від придбаного пакету відповідно.

Внутрішній рейтинг (*Rв*) розраховується із суми трьох критеріїв: активність користувача, кількість доданих товарів, наповненість сайту.

- Активність користувача – частота внесення змін користувачем (додання товарів, послуг, новин, фотографій, статей);
- Кількість внесених товарів або послуг – кількість товарів в каталозі підприємства (максимально можливе число залежить від придбаного пакету);
- Наповненість сайту – повнота внесеної інформації про підприємство (кількість статей, кількість фотографій, кількість новин, кількість товарів).

Наповненість сайту (*НС*) – розраховується за формулою:

$$НС = Статі + Галерея + Новини + Каталог$$

Рейтинг адміністратора (*Ra*) вручну встановлюється адміністратором, для зміни позиції підприємства з особливих причин.

3.3. Алгоритм роботи підсистеми

У методі getConfig() задаються параметри підключення до БД системи.

У конструкторі класу викликається метод getIdEnt() в якому проводиться вибірка, з бази, всіх відвідувачів для яких потрібно перерахувати рейтинг.

Далі для кожного обраного з бази відвідувача провадиться перерахунок рейтингу, за допомогою методу RatingController().

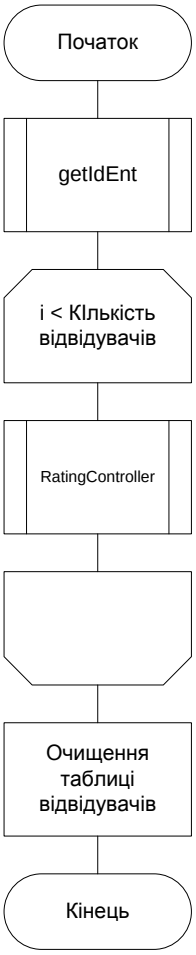

Рис. 3.2. Блок схема роботи підсистеми

RatingController() працює наступним чином:

- Викликається метод getDataRating(), який виконує вибірку з бази загального рейтингу, активності та рейтингу адміністратора, – поточного користувача. Якщо рейтинг адміністратора більше 999 встановлюємо йому значення 999, для коректності виконання розрахунків. Перевірка значення рейтингу адміністратора потрібна тому що це число встановлюється адміністратором вручну, що не виключає помилок введення. Також на початку етапу розробки деяка кількість підприємств уже мала встановлений рейтинг адміністратора більший 999, таким чином при перерахунку ця помилка буде виправлена.

Рис. 3.3. Блок схема методу getDataRating

- За допомогою методу GetPackage() отримується рейтинг пакета поточного користувача. Відповідно збільшивши значення пакету у 1000 разів, таким чином конвертувавши придбаний пакет у числову характеристику рейтингу.

- Методом GetRatingSiteContent() отримується рейтинг заповненості сайту.
- Метод GetRatingSiteContent() розраховує рейтинг в залежності від наповненості сторінок персональних сайтів підприємства (статі, новини, галерея, каталог). GetRatingSiteContent() використовує методи:
    - GetRaitingArticles() – для вибірки з БД всіх статей підприємства і розрахунку рейтингу на основі їх кількості та частоті змін.
    - GetRaitingNews() – для вибірки з БД всіх новин підприємства і розрахунку рейтингу на основі їх кількості та частоті змін.
    - GetRaitingGalery() – для вибірки з БД всіх фото підприємства і розрахунку рейтингу на основі їх кількості та частоті змін.
- Ці методи використовують однаковий метод розрахунку рейтингу, використовуючи різні вхідні дані:
    1. Вибірка даних з БД (максимальна кількість, кількість заповненого).
    2. Розрахунок відсотку заповненого від максимально можливого числа.
    3. Розрахунок числового значення рейтингу шляхом множення на коефіцієнт (0.6).
    4. Повернення результатів розрахунку в основну частину програми для подальших розрахунків.

- Методом GetRaitingProducts() отримуємо рейтинг товарів підприємства. Рейтинг товарів розраховується як відсоткове співвідношення доданих до максимальної кількості товарів у підприємства, помножене на коефіцієнт (3). Коефіцієнт 3 обраний для того щоб збільшити числове значення рейтингу від доданих товарів і послуг, оскільки товари і послуги є більш важливою характеристикою ніж новини, статі та фотогалерея.
- Далі проводиться розрахунок внутрішнього рейтингу підприємства (сума рейтингів активності, товарів та заповнення сайту).
- Якщо рейтинг адміністратора більше ніж внутрішній рейтинг, то внутрішньому рейтингу присвоюється значення рейтингу адміністратора.

- Далі розрахований внутрішній рейтинг та рейтинг пакету сумуються.
- Перевірка існування записів про підприємство в БД, в таблиці рейтингу, якщо запис існує, то проводиться оновлення поточного значення рейтингу, в іншому випадку, проводиться додавання в БД рейтингової інформації про підприємство.

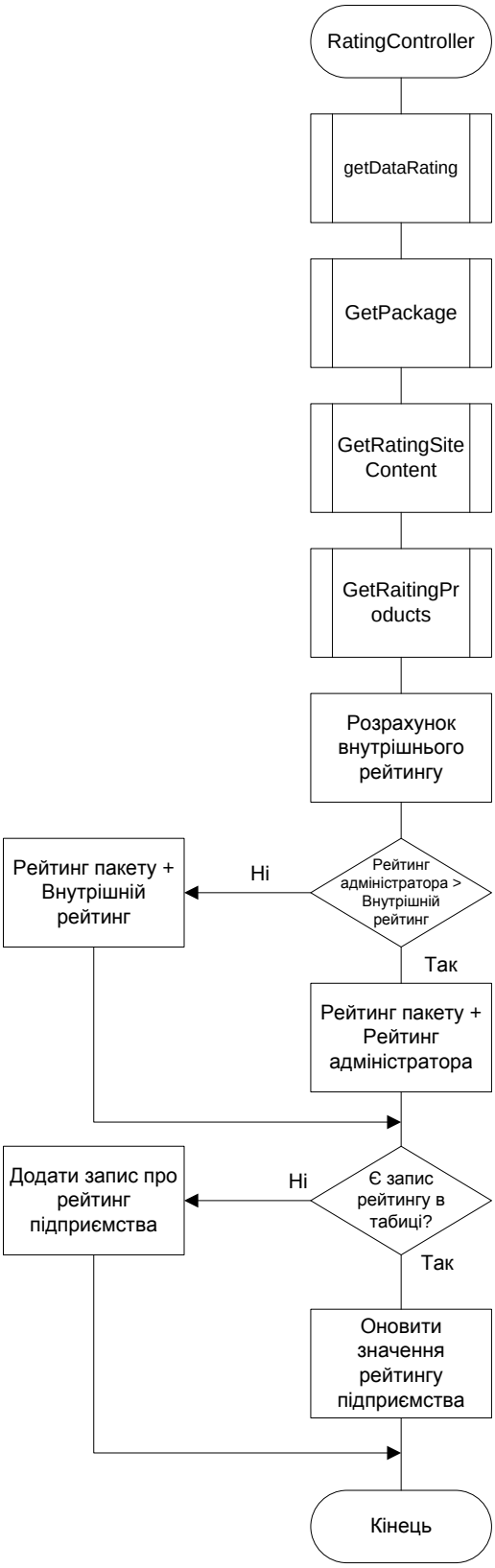

Рис. 3.4. Блок схема RatingController

Після розрахунку проводиться очищення таблиці відвідувачів сайту. На диску створюється текстовий файл з результатами виконання скрипта (кількість розрахованих записів), часом початку і кінця розрахунку, та витраченим часом на виконання.

3.4. Розпаралелювання алгоритму

Здійснивши аналіз алгоритму виявлено що більшість часу витрачається на ініціалізацію з'єднання з БД, записом та завершенням з'єднання. Отже якщо виконувати ці дії паралельно, можна значно збільшити швидкість обрахунку рейтингу підприємств.

Після виконання декомпозиції та розпаралелювання алгоритм набуде наступного вигляду, блок схема роботи диригента зображена на рис.3.5, а виконавця на рис.3.6.

Рис.3.5. Блок схема алгоритму роботи скрипта диригента

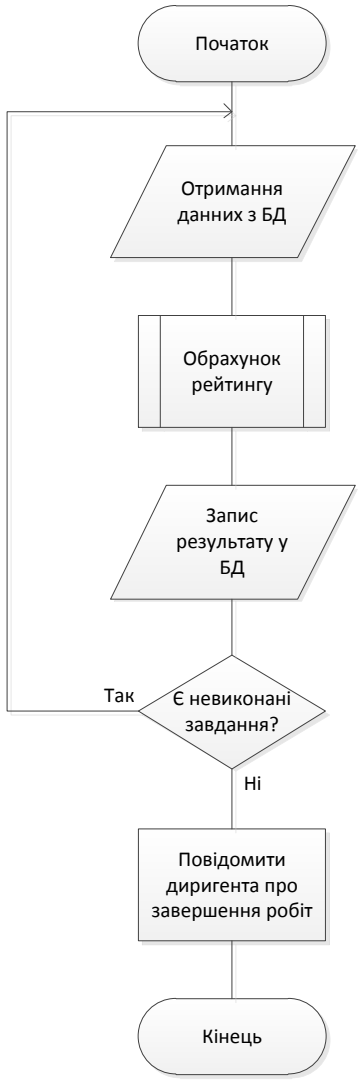

Рис.3.6. Блок схема алгоритму роботи виконавця

Вихідний код розпаралеленого алгоритму обрахунку рейтингу наведено у додатку Д.

3.5. Аналіз результатів

Після виконання розрахунку в лог-файл записуються результати виконання. Завдяки цьому можна дізнатись час який було витрачено на розрахунок та кількість підприємств для яких було перераховано рейтинг.

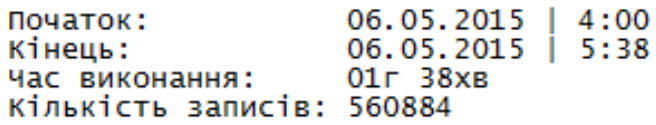

Рис.3.7. Лог файл з результатами послідовного виконання

З логу послідовного виконання видно, що розрахунок розпочато згідно розкладу, о 4:00 та завершено о 5:38 того ж дня. На розрахунок рейтингу 559326 компаній витрачено 1 годину і 38 хвилин.

Після впровадження розпаралеленого алгоритму обрахунку отримано наступний результат зображений на рис.3.8.

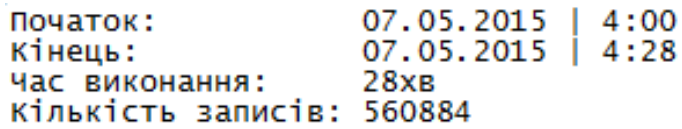

Рис.3.8. Лог файл з результатами паралельного виконання

З логу паралельного виконання видно, що розпаралелений алгоритм працює ефективніше і для обрахунку тієї ж кількості записів витрачено 28 хвилин.

Результати експерименту показали приріст швидкодії у 3.5 рази, що дозволило скоротити час виконання з 1 години 38 хвилин до 28 хвилин.

3.6. Висновки до розділу

1. Проведено аналіз алгоритму роботи існуючої системи, знайдено місця що можуть бути розпаралелені;
2. Модифіковано алгоритм для виконання у паралельному режимі;
3. Впроваджено паралельне обчислення у системі з використанням розробленої бібліотеки;
4. Проведено порівняння результатів роботи початкового та модифікованого алгоритму системи.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Повністю безпечних і нешкідливих виробництв не існує. Завдання охорони праці – звести до мінімуму ймовірність нещасного випадку або захворювання працюючого з одночасним забезпеченням комфортних умов при максимальній продуктивності праці.

Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці.

Нерозривно з охороною праці пов'язані і питання пожежної безпеки.

Для охорони праці важливе значення мають стан повітряного середовища виробничих приміщень, їх освітлення, вентиляція, електромагнітні та радіоактивні випромінювання, вібрація і шум.

Магістерська дисертація виконується на базі розробки нового приміщення підприємства.

4.1. Характеристика робочого приміщення

Характеристика робочого приміщення, в якому сидять користувачі системи тестування, представлена в таблиці 4.1.

Таблиця 4.1. Характеристика робочого приміщення

№	Найменування	Характеристика
1	Ширина	6 м
2	Довжина	4 м
3	Висота	2.8 м
4	Підлога	Ламінат
5	Стіни	Цегла

Продовження таблиці 4.1.

№	Найменування	Характеристика
6	Покриття стін	Шпалери світлих відтінків
7	Природне освітлення	Два вікна з північної сторони
8	Штучне освітлення	12 світильників, що мають 3 гало-генні лампи по 100 Вт
9	Вологість	сухо
10	Отоплення	Водяне
11	Електропроводка	Прихована, трипровідна
12	Вентиляція	Загальна вентиляція
13	Вологість	Не більше 75%

Площа приміщення:

$$S = A*B = 24\ (м2)$$

Об’єм приміщення:

$$V = S \bullet h = 24*2,8 = 67,2\ (м3)$$

Кількість постійних працівників у приміщенні – 3 людини.

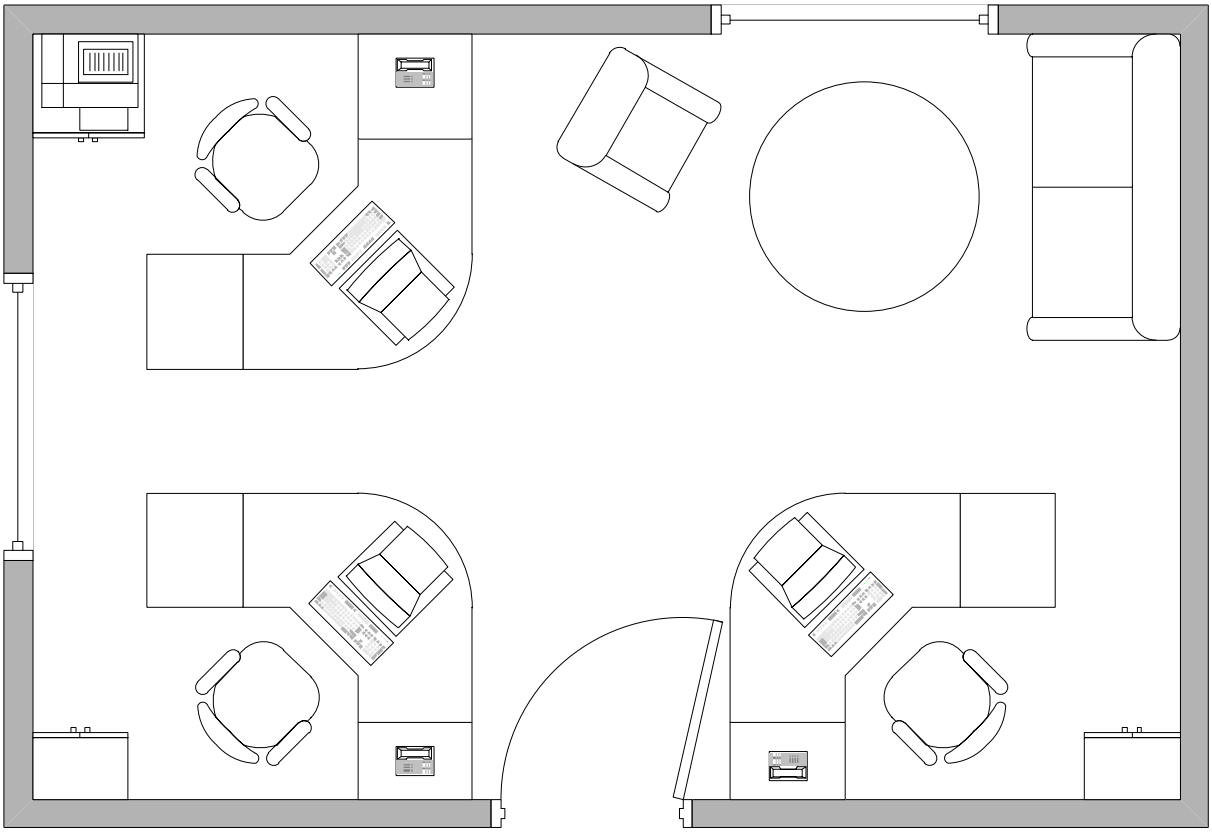

Рисунок 4.1. Організація робочого приміщення

4.1.1. Характеристика робочого місця

Характеристика індивідуального робочого місця користувача представлена в таблиці 4.2.

Таблиця 4.2. Характеристика індивідуального робочого місця

№	Просторові параметри	L, мм
1	Висота сидіння	400-500
2	Висота клавіатури від підлоги	600-750
3	Кут нахилу клавіатура	7-15°
4	Ширина основної клавіатури	не > 400
5	Глибина основної клавіатури	не > 200

Продовження таблиці 4.2.

№	Просторові параметри	L, мм
6	Віддалення клавіатури від краю столу	80-100
7	Висота екрану від рівня підлоги	950-1000
8	Кут нахилу екрану і нормалі	0-30°
9	Віддаленість екрану від краю столу	500-700
10	Висота поверхонь для записів	670-850
11	Площа поверхні для записів	600х400
12	Кут нахилу поверхні для записів	0-100
13	Глибина простору для ніг у колінах	<400
14	Глибина простору на рівні ступень	<600
15	Висота простору для ніг у колінах	<б00
16	Висота простору на рівні ступень	<100
17	Ширина простору для ніг на рівні	<500
18	Висота підставки для ніг	50-130
19	Кут підставки для ніг	0-25
20	Ширина підставки для ніг	300
21	Глибина підставки для ніг	400

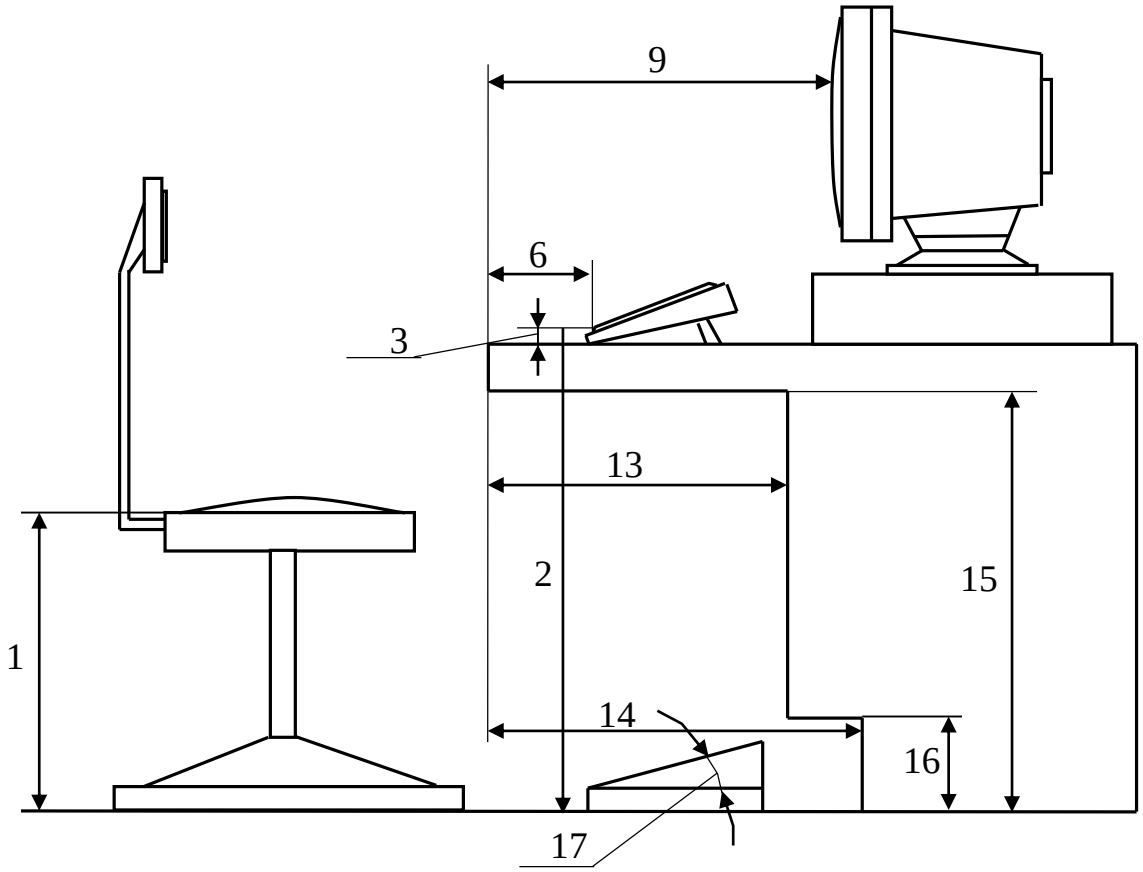

Рисунок 4.2. Організація робочого місця

4.2. Аналіз шкідливих виробничих факторів

4.2.1. Мікроклімат робочого приміщення

Параметри мікроклімату можуть мінятися в широких межах, у той час як необхідною умовою життєдіяльності людини є підтримка постійності температури тіла завдяки терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище. Принцип нормування мікроклімату – створення оптимальних умов для теплообміну тіла людини з навколишнім середовищем.

В даному приміщенні встановлена категорія роботи легка 1а. До неї відносяться роботи, вироблені сидячи, з незначними фізичними вправами.

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вологості в приміщенні. У

приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У санітарних нормах [12] встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються залежно від пори року, характеру трудового процесу і характеру виробничого приміщення (див. табл. 4.3.)

Для забезпечення комфортних умов використовуються як організаційні методи (раціональна організація проведення робіт залежно від пори року і доби, чергування праці і відпочинку), так і технічні засоби (вентиляція, кондиціонування повітря, опалювальна система).

Для регулювання температури в різні пори року, в приміщенні повинні бути встановлені кондиціонер і опалення.

Таблиця 4.3. Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні	22…24 °С
	Відносна вологість	40…60%
	Швидкість руху повітря	до 0,1 м/с
Теплий	Температура повітря в приміщенні	23…25 °С
	Відносна вологість	40…60%
	Швидкість руху повітря	0,1…0,2 м/с

4.2.2. Шум та вібрації

Шум погіршує умови праці здійснюючи шкідливу дію на організм людини. Працюючі в умовах тривалої шумової дії випробовують дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, зниження апетиту, біль у вухах і т.д. Такі порушення в роботі ряду органів і систем організму людини можуть викликати негативні зміни в емоційному стані людини аж до стресових. Під впливом шуму знижується концентрація уваги,

порушуються фізіологічні функції, з'являється втома у зв'язку з підвищеними енергетичними витратами і нервово-психічною напругою, погіршується мовна комутація. Все це знижує працездатність людини і її продуктивність, якість і безпеку праці. Тривала дія інтенсивного шуму (вище 80 дБ (А)) на слух людини приводить до його часткової або повної втрати.

Рівень шуму в залах обробки інформації на обчислювальних машинах – 65дБА. Для зниження рівня шуму стіни і стеля приміщень, де встановлені комп'ютери, можуть бути облицьовані звукопоглинальними матеріалами. Рівень вібрації в приміщеннях обчислювальних центрів може бути понижений шляхом встановлення устаткування на спеціальні віброізолятори.

Таблиця 4.4. Джерела і рівень шуму в робочому приміщенні

Джерело шуму	Рівень шуму, дБА
Жорсткий диск	40
Вентилятор	45
Монітор	17
Клавіатура	10
Принтер	45
Сканер	42

Максимальний час робота принтера за один день – 2,5 години.

Робочий день Т = 8 годин.

Час роботи приводів – півгодини.

$$L_{\Sigma}=10\cdot\lg(10^{4}+10^{4,5}+10^{1,7}+10^{1}+10^{4,5}+10^{4,2})=49,5 \text{ дБА}$$

Гранично допустимі норми встановлені в санітарних нормах виробничого шуму [14]. Що не перевищує норму в 50 дБА. Таким чином, акустичне середовище приміщень знаходиться в межах оптимальних величин.

4.2.3. Аналіз освітленості

Істотне значення для збереження тривалої працездатності, підвищення продуктивності праці і попередження нещасних випадків має забезпечення норм освітленості на робочому місці. Згідно державними будівельними нормами [13] приміщення для роботи з ВДТ (відео дисплейною технікою) повинні мати природне і штучне освітлення.

Таблиця 4.5. Характеристика зорових робіт

Характеристика зорових робіт	Висока точність
Найменший розмір об’єкта розпізнавання, мм	0,3-0,5
Розряд зорової роботи	III (третій)
Підрозряд зорової роботи	г

Для запобігання перевантаження зорового апарату в приміщенні необхідно при його облицюванні використовувати дифузійно-відбиваючі матеріали, з наступними коефіцієнтами відбивання:

стеля – 0,7-0,8

стіни – 0,5-0,6

підлога – 0,3-0,5

Співвідношення яскравості між робочою поверхнею і стінами повинне бути не більш 10:1.

Співвідношення яскравості між робочими поверхнями повинне складати не більш 3:1 - 5:1. Ця умова досягається, тому що робоча поверхня тільки одна – монітор комп'ютера.

Природне освітлення

Роботи, в даному приміщенні, мають проводитись: у світлий час доби при природному освітленні, у темний – при штучному. Природне освітлення здійснюється через два віконних прорізи.

Одне вікно робочого приміщення виходить на південь, інше на північний схід. Тип природного освітлення – бічне двобічне. Коефіцієнт природної освітленості:

$$e_н = e_n \cdot m_N$$

де $e_n = 1$ – значення КПО за таблицями 1 і 2 державних будівельних норм [13];

$m_N = 0.85,\ m_N = 0.9$ – коефіцієнти світлового клімату за таблицею 4 державних будівельних норм [13].

$$e_н = 1 * 0{,}85 = 0{,}85$$
$$e_н = 1 * 0{,}9 = 0{,}9$$

Штучне освітлення.

В відповідності з правилами охорони праці під час експлуатації ЕОМ [11], якщо в приміщенні не вистачає до норми природного освітлення та для освітлення в темний період доби, то воно повинне бути доповнене штучним. Така сукупність природного і штучного освітлення називається ккомбінованим освітленням.

Згідно з державними будівельними нормами [13], для 3-го розряду зорової роботи нормативні значення освітленості становлять:

- для комбінованого освітлення – 400 лк;

Для освітлення приміщення будемо використовувати люмінесцентні лампи ЛД-65.

$E = F_л\ N\ n\ \eta\ /\ S\ k_з\ z$ – фактичне значення освітлення

$F_л = 3750\ лм$ – світловий потік для люмінесцентної лампи ЛД-65.

$N=1$ кількість ламп

i = ab / h (a + b) – індекс приміщення

де a, b та h – ширина, довжина та висота приміщення відповідно

i=5*4,3 /3.2(5+4,3)=0.722

η=34% – коефіцієнт використання світлового потоку при

коефіцієнтах відбиття стелі $\rho_{сл}$=70 та стін $\rho_{сн}$=50 та індексу приміщення i=0,7

n=0,8 – коефіцієнт затінення з урахуванням фіксованих місць праці

$k_з$=1,3 – коефіцієнт запасу

S=21,5 м кв. – площа, що освітлюється

z =1.1 – коефіцієнт нерівномірності освітлення

E= 3750*1*0,8*0,34/21,5*1,3*1,1 = 67,84 (лк) – фактична освітленість однієї лампи ЛД-65.

Отже для освітлення даного приміщення потрібно 400 / 67,84 = 5,89 ≈ 6 ламп ЛД-65. Тому у приміщені розташовано, у ряд, 3 світильники типу ARS/R 265, з дзеркальною екрануючою решіткою, по дві лампи у кожному.

4.2.4. Електробезпека

Приміщення відноситься до класу приміщень без підвищеної небезпеки. Відносна вологість повітря не перевищує 60%, температура не перевищує 25°С. В приміщені присутні споживачі електроенергії: освітлювальні прилади та комп’ютери.

В приміщені використовується прихована проводка, що виключає можливість дотику до оголених проводів. Проводка прокладена кабелем марки ВВП 3*1,5 з мідними струмопровідними жилами; ізоляція виготовлена з ПВХ; кабель плоский з розділеною основою.

Номінальна зміна напруги 220 В; частота до 50 Гц. $I_н$=15 А, $I_ф$=15 А. Вмикачі штучного освітлення виконані з діелектричних матеріалів.

Для забезпечення електробезпеки застосовується занулення. Всі струмопровідні прилади ізольовані. В них використовується подвійне ізоляційне покриття (передня частина системного блоку виконана з пластику, що виключає дотик користувача до металічних частин). Блок живлення в корпусі комп'ютера знаходитись в окремому захисному кожусі, що виключає можливість доступу до високої напруги.

Усі перераховані параметри відповідають правилам охорони праці під час експлуатації ЕОМ [11].

4.2.5. Пожежна безпека

У розглянутому приміщенні знаходяться ЕОМ, в яких дуже висока щільність розміщення елементів електронних схем. Сама ЕОМ являє собою пожежну небезпеку, так як при підвищенні температури окремих вузлів можливо оплавлення ізоляції сполучних проводів, яке веде до короткого замикання, що супроводжується, в свою чергу, іскрінням.

Категорія пожежонебезпеки даного приміщення – Г, помірна пожежонебезпека (негорючі речовини і матеріали в гарячому, розпеченому або розплавленому стані, процес обробки яких супроводжується виділенням променистого тепла, іскор та полум'я, і (або) горючі гази, рідини і тверді речовини, які спалюються або утилізуються в якості палива).

Можливі причини пожежі:

- Перевантаження в електромережі;
- Коротке замикання;
- Руйнування ізоляції провідників.

Система протипожежного захисту:

– Встановлена автоматична пожежна сигналізація на димових повідомлювачах ДІП-1, з розрахунку 2 шт. на кожні 20 м2 площі приміщення, враховуючи високу вартість обладнання, наявність прихованих комунікацій і

специфіку загоряння ЕОМ. Тобто на площу 24 м2 необхідно 3 димових сповіщувача.

– Розміщені 3 вуглекислотних вогнегасники ОУ-5 (ручні) з розрахунку 1 вогнегасник на 10 м2.

Організаційні заходи:

– Проводиться інструктаж персоналу по ТБ;
– Розроблено заходи щодо дій адміністрації на випадок виникнення пожежі;
– Схема евакуації при пожежі поміщена на видному місці;
– Ширина дверного отвору на випадок евакуації 1 м., висота 2 м.

4.3. Дії при виникненні надзвичайних ситуацій

Бізнес-центр, у якому розташоване приміщення, знаходиться у одній з промислових зон міста Києва. Поруч із ним розташовані заводи та їх цехи. Тому виникає небезпека аварії на вибухонебезпечних об’єктах. Розглянемо ситуацію та проведемо оцінювання інженерної та пожежної обстановки під час такої ситуації. (табл. 4.6.)

Таблиця 4.6. Оцінювання інженерної та пожежної обстановки

Вхідні дані		Одиниці вимірювання
Відстань від бізнес-центру до місця аварії (вибуху)	500	метри
Маса вибухової речовини (тротил)	500	тонни
Категорія виробництва з пожежної безпеки	В	
Щільність забудови об’єкту	20	%
Характеристики елементів будівлі бізнес-центру:		
Будівля	З легким металевим каркасом	
Верстати	відсутні	

Продовження таблиці 4.6.

Кабельні лінії	наземні
Контрольно-вимірювальна апаратура	наявна
Границі вогнетривкості несучих стін	0,5 годин
Границі вогнетривкості перегородок	0,25 годин

Розрахунки:

Зона I : $r1 = 17{,}5 \cdot \sqrt[3]{Q}$ *= 17,5 * 7,94=138,95 м*

$\Delta P_ф$ *= 30кПа*

Зона *II: r2 = 1,7 * r1 = 1.7*138.95= 236.22 м*

ΔPII = 1300(r1/r0)^3+50=1300(138.95/500)^3+50=28.6+50=78.6

Висновок**:** об’єкт опиниться за межами цих зон, тобто у зоні повітряної ударної хвилі (зона III)

$$\Delta P_ф = \frac{262}{\sqrt{1 + 7{,}66 \cdot 10^{-5} \cdot \frac{L^3}{Q}} - 1} = 30.1 \text{ кПа}$$

Ступінь руйнування будівлі – *Середня*

Характеристика руйнувань будівлі:

- Руйнування даху;
- Легких внутрішніх перегородок;
- В капітальних стінах з’являються тріщини.

Ступінь руйнування контрольно-вимірювальної апаратури – сильні

Ступінь руйнування кабельних ліній – слабкі

Ступінь ураження людей від прямої дії УХ – слабкі

Характеристика уражень людей: Легка контузія організму, часткова втрата слуху, вивихи кінцівок.

Ступінь вогнестійкості – III ступінь вогнестійкості

Очікувана пожежна обстановка – для виробництва категорії пожежної небезпеки В, ступеня вогнестійкості будівель – III, при надмірному тику 25 кПа

і щільності забудови більше 20% можна очікувати в перші 30 хвилин окремі пожежі з переростанням за 1…2 год в суцільну.

Небезпечна кількість вибухової речовини для уникнення руйнувань будівлі – 40т

Небезпечна кількість вибухової речовини для уникнення будь-яких руйнувань – менше 12 т

Загальні висновки і рекомендації:

На відстані 500 м від бізнес-центру стався вибух тротилу, що призвело до руйнувань будівлі, елементів бізнес-центру, постраждали люди. В першу чергу треба сповістити про НС службу порятунку, а також лікарів. Не дивлячись на те, що загроза для людей є невисокою, слід забезпечики кожного першою невідкладною допомогою.

Серед рекомендацій, спрямованих на зменшення заподіяної шкоди та уражень людей, можуть бути такі:

5. Укріпити будівлю установленням додаткових колон, підкосів;
6. Створити 50% запас контрольно-вимірювальної апаратури;
7. Установити на вікнах захисні металеві сітки, щоб розбите скло не потрапляло в приміщення бізнес-центру;
8. Установити і регулярно контролювати стан вогнегасників та інших протипожежних систем;
9. Порушити питання перед відповідними органами про зменшення запасу вибухонебезпечної речовини до безпечної кількості.

4.4. Висновки до розділу

Виходячи з даного розділу, можна сказати, що приміщення, підходить для роботи співробітників підприємства, має хорошу освітленість і шумоізоляцію. Незважаючи на те, що в ньому досить велика кількість пожежонебезпечних речей, вжито заходів з попередження нещасних випадків. Приміщення відповідає вимогам, які висуваються стандартами.

ВИСНОВКИ

В рамках магістерської дисертації було розглянуто методики розпаралелювання програмного коду та способи синхронізації паралельної взаємодії. Запропоновано методи реалізації розпаралелювання на PHP з використанням:

- бібліотеки curl,
- stream функцій,
- асинхронних функцій.

Проведено тестування швидкодії вирішення однакових задач різними методами та за різних умов, порівняння результатів та визначення оптимального.

В рамках дослідження розроблено бібліотеку для зручності розпаралелювання, що значно спрощує використання реалізацій паралельності в PHP.

Бібліотека надає наступні можливості:

- Синхронний режим, зі збереженням інтерфейсу при відсутності необхідних розширень для розпаралелювання;
- Багаторазове використання виконавців;
- Повноцінний обмін данними між диригентом і виконавцями (запуск з аргументами і отримання результату після завершення);
- Обробка помилок виконання;
- Максимальна швидкодія.

Розпаралелювання було впроваджено і досліджено на реальному проекті, що призвело до збільшення швидкодії у 3.5 рази.

ПЕРЕЛІК ПОСИЛАНЬ

1. Карпов В. Е. Введение в распараллеливание алгоритмов и программ / В. Е. Карпов // Компьютерные исследования и моделирование – 2010 - Т. 2 №3 С. 231–272.
2. Антонов А.С. Введение в паралельные вычисления / А. С. Антонов // Московский государственный университет им. Ломоносова–2002-70 с.
3. Крилов Є.В., Анікін В.К., Шумада А.О. Оптимізація роботи веб-сервера для взаємопов'язаних процесів / Є.В. Крилов, В.К. Анікін, А.О. Шумада // Міжвідомчий науково-технічний збірник «Адаптивні системи автоматичного управління». – 2014. - №1(24). – С.46-52.
4. Федотов И. Е. Некоторые приемы параллельного программирования / И. Е. Федотов // Московский государственный институт радиотехники – 2008–188с.
5. Михалевич В.С. Словарь по кибернетике / В. С. Михалевича. // Главная редакция Украинской Советской Энциклопедии имени М. П. Бажана, 1989. — 751 с.
6. Воеводин В. В. Параллельные вычисления / В.В.Воеводин // СПб: БХВ-Петербург, 2002. — 608 с.
7. Котеров Д.В. Костарев А.Ф. PHP 5 – В подлиннике. БХВ-Петербург, 2007 – 1120 ст.
8. Хоккинс С. Администрирование Web-сервера Apache и руководство по электронной коммерции. Издательский дом «Вильямс», 2001 – 336 ст.
9. Дунаев В.В. Основы Web-дизайна. БХВ-Петербург, 2006 – 512 ст.
10. Методичні вказівки до виконання розділу «Охорона праці» в дипломних проектах для студентів Інституту прикладного та системного аналізу /Уклад.: Демчук Г.В., Луц Т.Є., Праховнік Н.А.К.: НТУУ «КПІ», – 16 с.
11. НПАОП 0.00-1.28-10 Правила охорони праці під час експлуатації ЕОМ
12. ДСН 3.3.6.042-99 Санітарні норми мікроклімату виробничих приміщень
13. ДБН В.2.5-28-2006 Природне і штучне освітлення

14. ДСН 3.3.6.037-99 Санітарні норми виробничого шуму, ультразвуку та інфразвуку
15. ДСанПіН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ЕОМ
16. ДСанПіН 3.3.6.096-2002 Державні санітарні норми і правила при роботі з джерелами електромагнітних полів.
17. НАПБ Б.03.002-2007 Нормы определения категорий помещений, зданий и наружных установок по взрывопожарной и пожарной опасности.
18. НАПБ А.01.001-2004. Правила пожежної безпеки в Україні.
19. ГОСТ 12.1.044-89. Пожаровзрывоопасность веществ и материалов. Номенклатура показателей и методика определения.

ДОДАТКИ ТА ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

ДОДАТОК А

Лістинг програми, бібліотека curl

ДОДАТОК Б

Лістинг програми, streams

ДОДАТОК В

Лістинг програми, сокети

ДОДАТОК Г

Лістинг програми, бібліотека розпаралелювання

ДОДАТОК Д

Лістинг програми, розпаралелена система обрахунку ретингу

ДОДАТОК Е

Ярусно-паралельна форма алгоритму

ДОДАТОК Ж

Результати експерементального розпаралелювання у PHP

ДОДАТОК И

ER-модель бази данних

ДОДАТОК К

Блок схема рейтингової системи

ДОДАТОК Л

Блок схема розпаралеленої рейтингової системи

ДОДАТОК М

Результати дослідження методики розпаралелювання на основі реального проекту