

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЗВІТ

з лабораторних робіт №5-6
з дисципліни ”Програмне забезпечення комп’ютерних систем”

Тема: Паралельні комп’ютерні системи

Варіант №5 (81)

Виконав:

Студент 1 курсу, групи ІМ-51мн
Ковальов Олександр

Перевірила:
к.т.н., Русанова Ольга Веніамінівна

Дата здачі: 2.12.2025

КИЇВ – 2025

Мета роботи. Визначення часу виконання арифметичного виразу в заданій ПКС з урахуванням кількості процесорів. Вибір оптимального способу розпаралелювання АВ при його виконанні на заданій ПКС з урахуванням її параметрів.

Вхідні дані: Дерево паралельної форми арифметичного виразу (результат виконання лабораторної роботи №2) та множина дерев (графів) паралельної форми, отриманих у результаті виконання лабораторних робіт №2-4. Крім того, кожний студент узгоджує з викладачем тип ПКС та її параметри, такі як кількість процесорів (шарів конвеєра) і час виконання різних алгебраїчних операцій. За варіантом (81) визначена **векторна система** з 1 суматором, 1 блоком віднімання, 1 блоком множення та 1 блоком ділення.

Завдання: Розробити програмну модель роботи заданої паралельної комп’ютерної системи з урахуванням кількості процесорів (шарів) та часу виконання різних алгебраїчних операцій. При виконанні цього завдання студент може використовувати будь-яку мову програмування. Продемонструвати роботу моделі паралельної системи при виконанні вхідного арифметичного виразу. Визначити час виконання АВ, коефіцієнт прискорення та ефективність роботи системи. З одержаної множини графів паралельних форм вибрati оптимальний(-i) для реалізації в ПКС заданої архітектури. Провести дослідження за направленим пошуком графів АВ, адекватних заданій архітектурі (графу) системи. Для виконання поставлених завдань необхідно промоделювати виконання усіх еквівалентних графів паралельних форм АВ на заданій ПКС. Визначити час їх виконання, коефіцієнти прискорення та ефективності роботи системи. Занести отримані результати у таблицю та вибрati оптимальну форму АВ.

Хід роботи.

В рамках виконання роботи було розроблено та інтегровано підсистему симуляції паралельної комп’ютерної системи (ПКС) векторного типу для аналізу ефективності виконання арифметичних виразів.

Робота розпочалася з реалізації модуля `pcs` (Parallel Computer System), який відповідає за моделювання апаратної частини. Враховуючи отримане завдання, було сконфігуровано модель векторної системи з наступними характеристиками ресурсів: 1 суматор, 1 блок віднімання, 1 блок множення та 1 блок ділення. Дляожної операції встановлено відповідний час виконання у тактах (Ticks) – додавання 1 такт, віднімання 1 так, множення 2 такти, ділення 4 такти.

Ключовим етапом реалізації стала розробка симулятора на базі алгоритму List Scheduling (планування списків). Оскільки абстрактне синтаксичне дерево (AST), отримане в попередніх роботах, є рекурсивною структурою, непристосованою для прямого планування, першим кроком симуляції є конвертація AST у спрямований ацикличний граф завдань (Task Graph). Функція `flatten_ast` рекурсивно обходить дерево та формує плоский список об'єктів `Task`. Кожне завдання отримує унікальний ідентифікатор, тип операції (що відповідає типу функціонального блоку процесора) та список ідентифікаторів завдань-залежностей, які мають бути завершені перед початком поточного. Листові вузли дерева (числа та змінні) обробляються як операції типу `Load` з нульовим часом виконання, що імітує миттєву доступність операндів з пам'яті.

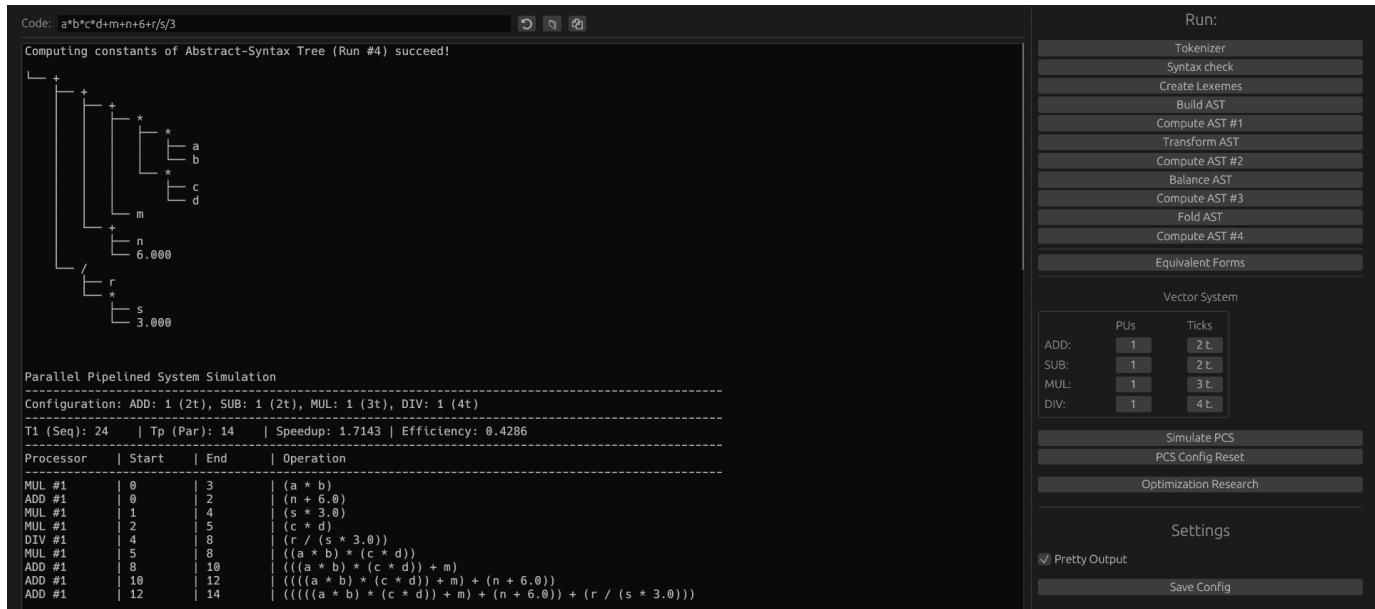
Сам процес симуляції реалізовано як потактовий цикл (tick-based simulation). На кожному такті алгоритм виконує наступні дії:

1. Перевіряє завершення активних завдань та звільняє відповідні ресурси процесора.
2. Формує чергу готових до виконання завдань (`ready_queue`), відбираючи ті вузли графа, для яких задоволені всі залежності по даних (всі попередні операції завершено).
3. Сортує чергу готових завдань за евристикою "найдовший час виконання первісним" (Longest Processing Time first) для оптимізації щільності розкладу.
4. Намагається призначити завдання з черги на вільні функціональні блоки. Якщо відповідний ресурс (наприклад, блок множення) вільний, завдання запускається, а ресурс блокується на кількість тактів, що відповідає вартості операції. В іншому випадку завдання залишається в черзі до наступного такту.

Після завершення симуляції розраховуються ключові метрики ефективності:

- **Послідовний час – T_1** : сума часів виконання всіх арифметичних операцій в однопроцесорному режимі.
- **Паралельний час – T_p** : час закінчення останнього завдання в симульованому розкладі.
- **Коефіцієнт прискорення – S** : відношення $T_1 \div T_p$.

- **Ефективність – E** : відношення S до загальної кількості функціональних блоків.



Згодом функціонал системи було суттєво розширено шляхом впровадження підтримки **конвеєрного виконання операцій (pipelining)**. У попередній реалізації функціональний блок вважався зайнятим протягом усього часу виконання операції (*latency*). В оновленій моделі було змінено логіку керування ресурсами: тепер функціональні блоки здатні приймати нові входні дані на кожному такті (пропускна здатність 1 операція/такт), навіть якщо обробка попередньої операції ще триває на внутрішніх стадіях конвеєра. Це дозволило реалізувати часове перекриття виконання операцій (*instruction overlap*), що значно зменшує загальний час виконання програми (T_p) за наявності незалежних даних.

Для ефективного використання конвеєра було модифіковано алгоритм планування. Під час лінеаризації AST у граф завдань тепер додатково обчислюється **ранг (або шар) кожного вузла**. Ранг визначається як відстань від поточного вузла до найглибшого листка піддерева. Це дозволило застосувати нову евристику сортування черги `ready_queue`: пріоритет надається завданням із меншим рангом (ближчим до листків), а за рівності рангів — операціям із більшою тривалістю виконання. Такий підхід забезпечує своєчасну підготовку операндів для вищих рівнів дерева виразу.

Також було вдосконалено систему моніторингу та звітування. Замість абстрактних назв типів операцій, симулятор тепер формує та відстежує конкретні рядкові представлення підвиразів (наприклад, $(a + b)$ або $T_1 * T_2$). У логах потактової симуляції реалізовано відображення внутрішнього стану конвеєрів:

для кожного активного блоку виводиться інформація про те, яка саме операція виконується та на якій стадії (Stage) вона знаходиться (наприклад, [Stage 2: (a+b)]). Це забезпечує повну прозорість процесу обчислень та дозволяє візуально верифікувати коректність завантаження конвеєрних блоків.

The screenshot shows a software interface for pipeline simulation. On the left, a 'Detailed Pipeline Log (Tick-by-Tick)' window displays the execution of the expression $a*b*c*d+m+n+6+r/s/3$ across 7 ticks. The log shows the state of four units (ADD, DIV, MUL, SUB) at each tick, indicating they are either 'Idle' or processing a stage of the expression. The right side of the interface contains several tabs and configuration panels:

- Run:** Contains buttons for Tokenizer, Syntax check, Create Lexemes, Build AST, Compute AST #1, Transform AST, Compute AST #2, Balance AST, Compute AST #3, Fold AST, Compute AST #4, and Equivalent Forms.
- Vector System:** A table showing PUs (1) and Ticks (2 t., 2 t., 3 t., 4 t.) for ADD, SUB, MUL, and DIV respectively.
- Simulate PCS:** Buttons for PCS Config Reset and Optimization Research.
- Settings:** Includes a 'Pretty Output' checkbox (checked) and a 'Save Config' button.

```

Code: a*b*c*d+m+n+6+r/s/3

Detailed Pipeline Log (Tick-by-Tick):
-----
Tick 01:
Ready Queue: [(a * b), "(s * 3.0)", "(c * d)", "(n + 6.0)"]
ADD #1 : [Stage 1: (n + 6.0)]
DIV #1 : Idle
MUL #1 : [Stage 1: (a * b)]
SUB #1 : Idle

Tick 02:
Ready Queue: [(s * 3.0), "(c * d)"]
ADD #1 : [Stage 2: (n + 6.0)]
DIV #1 : Idle
MUL #1 : [Stage 2: (a * b)] [Stage 1: (s * 3.0)]
SUB #1 : Idle

Tick 03:
Ready Queue: [(c * d)]
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 3: (a * b)] [Stage 2: (s * 3.0)] [Stage 1: (c * d)]
SUB #1 : Idle

Tick 04:
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 3: (s * 3.0)] [Stage 2: (c * d)]
SUB #1 : Idle

Tick 05:
Ready Queue: [(r / (s * 3.0))]
ADD #1 : Idle
DIV #1 : [Stage 1: (r / (s * 3.0))]
MUL #1 : [Stage 3: (c * d)]
SUB #1 : Idle

Tick 06:
Ready Queue: [((a * b) * (c * d))]
ADD #1 : Idle
DIV #1 : [Stage 2: (r / (s * 3.0))]
MUL #1 : [Stage 1: ((a * b) * (...))]
SUB #1 : Idle

Tick 07:
ADD #1 : Idle
DIV #1 : [Stage 3: (r / (s * 3.0))]
MUL #1 : [Stage 2: ((a * b) * (...))]
SUB #1 : Idle

```

Додаткові приклади виразів.

Приклад 1:

Code: $(a-c)*(b-k+1)$

Computing constants of Abstract-Syntax Tree (Run #4) succeed!

```

  └── *
    ├── -
    │   ├── a
    │   └── c
    ├── +
    │   ├── -
    │   │   ├── b
    │   │   └── k
    │   └── 1.000
  
```

Parallel Pipelined System Simulation

Configuration: ADD: 1 (1t), SUB: 1 (1t), MUL: 1 (2t), DIV: 1 (4t)

T1 (Seq): 5 | Tp (Par): 4 | Speedup: 1.2500 | Efficiency: 0.3125

Processor	Start	End	Operation
SUB #1	0	1	$(b - k)$
SUB #1	1	2	$(a - c)$
ADD #1	1	2	$((b - k) + 1.0)$
MUL #1	2	4	$((a - c) * ((b - k) + 1.0))$

Detailed Pipeline Log (Tick-by-Tick):

Vector System

PUs	Ticks
ADD:	1
SUB:	1
MUL:	1
DIV:	1

Run:

- Tokenizer
- Syntax check
- Create Lexemes
- Build AST
- Compute AST #1
- Transform AST
- Compute AST #2
- Balance AST
- Compute AST #3
- Fold AST
- Compute AST #4
- Equivalent Forms

Detailed Pipeline Log (Tick-by-Tick):

Vector System

PUs	Ticks
ADD:	1
SUB:	1
MUL:	1
DIV:	1

Run:

- Simulate PCS
- PCS Config Reset
- Optimization Research

Settings

Pretty Output

Save Config

Tick 01:
Ready Queue: $["(b - k)", "(a - c)"]$
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : Idle
SUB #1 : [Stage 1: $(b - k)$]

Tick 02:
Ready Queue: $["(a - c)", "((b - k) + 1.0)"]$
ADD #1 : [Stage 1: $((b - k) + 1.0)$]
DIV #1 : Idle
MUL #1 : Idle
SUB #1 : [Stage 1: $(a - c)$]

Tick 03:
Ready Queue: $["((a - c) * ((b - k) + 1.0))"]$
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 1: $((a - c) * (...))$]
SUB #1 : Idle

Tick 04:
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 2: $((a - c) * (...))$]
SUB #1 : Idle

Приклад 2:

Code: $a-b*(k-t)-(f-g)*(f+5.9-q)-(f+g)/(d+q-w)$

Computing constants of Abstract-Syntax Tree (Run #4) succeed!

```

  └── -
    ├── a
    ├── +
    │   ├── -
    │   │   ├── b
    │   │   └── k
    │   └── t
    ├── *
    │   ├── -
    │   │   ├── f
    │   │   └── g
    │   └── 5.900
    └── /
      ├── +
      │   ├── -
      │   │   ├── f
      │   │   └── g
      │   └── d
      └── w
  
```

Run:

- Tokenizer
- Syntax check
- Create Lexemes
- Build AST
- Compute AST #1
- Transform AST
- Compute AST #2
- Balance AST
- Compute AST #3
- Fold AST
- Compute AST #4
- Equivalent Forms

Vector System

PUs	Ticks
ADD:	1
SUB:	1
MUL:	1
DIV:	1

Run:

- Simulate PCS
- PCS Config Reset

Lab 5-6

Code: $a-b*(k-t)-(f-g)*(f*5.9-q)-(f+g)/(d+q-w)$

Parallel Pipelined System Simulation

Configuration: ADD: 1 (2t), SUB: 1 (3t), MUL: 1 (3t), DIV: 1 (5t)

T1 (Seq): 37 | Tp (Par): 17 | Speedup: 2.1765 | Efficiency: 0.5441

Processor	Start	End	Operation
SUB #1	0	3	$(f - g)$
MUL #1	0	3	$(f * 5.9)$
ADD #1	0	2	$(f + g)$
SUB #1	1	4	$(k - t)$
ADD #1	1	3	$(d + q)$
SUB #1	3	6	$((f * 5.9) - q)$
MUL #1	4	7	$(b * (k - t))$
SUB #1	4	7	$((d + q) - w)$
MUL #1	6	9	$((f - g) * ((f * 5.9) - q))$
DIV #1	7	12	$((f + g) / ((d + q) - w))$
ADD #1	9	11	$((b * (k - t)) + ((f - g) * ((f * 5.9) - q)))$
ADD #1	12	14	$(((b * (k - t)) + ((f - g) * ((f * 5.9) - q))) + ((f + g) / ((d + q) - w)))$
SUB #1	14	17	$(a - (((b * (k - t)) + ((f - g) * ((f * 5.9) - q))) + ((f + g) / ((d + q) - w))))$

Detailed Pipeline Log (Tick-by-Tick):

Tick 01:

```
Ready Queue: ["(f - g)", "(f * 5.9)", "(k - t)", "(f + g)", "(d + q)"]
ADD #1 : [Stage 1: (f + g)]
DIV #1 : Idle
MUL #1 : [Stage 1: (f * 5.9)]
SUB #1 : [Stage 1: (f - g)]
```

Tick 02:

```
Ready Queue: ["(k - t)", "(d + q)"]
ADD #1 : [Stage 2: (f + g)] [Stage 1: (d + q)]
DIV #1 : Idle
MUL #1 : [Stage 2: (f * 5.9)]
SUB #1 : [Stage 2: (f - g)] [Stage 1: (k - t)]
```

Tick 03:

```
ADD #1 : [Stage 2: (d + q)]
DIV #1 : Idle
MUL #1 : [Stage 3: (f * 5.9)]
SUB #1 : [Stage 3: (f - g)] [Stage 2: (k - t)]
```

Tick 04:

```
Ready Queue: ["((f * 5.9) - q)", "((d + q) - w)"]
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : Idle
SUB #1 : [Stage 3: (k - t)] [Stage 1: ((f * 5.9) - q)]
```

Tick 05:

```
Ready Queue: ["((b * (k - t))", "((d + q) - w)"]
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 1: (b * (k - t))]
SUB #1 : [Stage 2: ((f * 5.9) - q)] [Stage 1: ((d + q) - w)]
```

Tick 06:

```
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 2: (b * (k - t))]
SUB #1 : [Stage 3: ((f * 5.9) - q)] [Stage 2: ((d + q) - w)]
```

Tick 07:

```
Ready Queue: ["((f - g) * ((f * 5.9) - q))"]
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 3: (b * (k - t))] [Stage 1: ((f - g) * (...))]
SUB #1 : [Stage 3: ((d + q) - w)]
```

Run:

- Tokenizer
- Syntax check
- Create Lexemes
- Build AST
- Compute AST #1
- Transform AST
- Compute AST #2
- Balance AST
- Compute AST #3
- Fold AST
- Compute AST #4
- Equivalent Forms

PUs	Ticks
ADD:	1 2t.
SUB:	1 3t.
MUL:	1 3t.
DIV:	1 5t.

Lab 5-6

Code: $a-b*(k-t)-(f-g)*(f*5.9-q)-(f+g)/(d+q-w)$

Run:

- Tokenizer
- Syntax check
- Create Lexemes
- Build AST
- Compute AST #1
- Transform AST
- Compute AST #2
- Balance AST
- Compute AST #3
- Fold AST
- Compute AST #4
- Equivalent Forms

PUs	Ticks
ADD:	1 2t.
SUB:	1 3t.
MUL:	1 3t.
DIV:	1 5t.

Simulate PCS

PCS Config Reset

Optimization Research

Settings

Pretty Output

Save Config

Tick 01:

```
Ready Queue: ["(f - g)", "(f * 5.9)", "(k - t)", "(f + g)", "(d + q)"]
ADD #1 : [Stage 1: (f + g)]
DIV #1 : Idle
MUL #1 : [Stage 1: (f * 5.9)]
SUB #1 : [Stage 1: (f - g)]
```

Tick 02:

```
Ready Queue: ["(k - t)", "(d + q)"]
ADD #1 : [Stage 2: (f + g)] [Stage 1: (d + q)]
DIV #1 : Idle
MUL #1 : [Stage 2: (f * 5.9)]
SUB #1 : [Stage 2: (f - g)] [Stage 1: (k - t)]
```

Tick 03:

```
ADD #1 : [Stage 2: (d + q)]
DIV #1 : Idle
MUL #1 : [Stage 3: (f * 5.9)]
SUB #1 : [Stage 3: (f - g)] [Stage 2: (k - t)]
```

Tick 04:

```
Ready Queue: ["((f * 5.9) - q)", "((d + q) - w)"]
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : Idle
SUB #1 : [Stage 3: (k - t)] [Stage 1: ((f * 5.9) - q)]
```

Tick 05:

```
Ready Queue: ["((b * (k - t))", "((d + q) - w)"]
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 1: (b * (k - t))]
SUB #1 : [Stage 2: ((f * 5.9) - q)] [Stage 1: ((d + q) - w)]
```

Tick 06:

```
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 2: (b * (k - t))]
SUB #1 : [Stage 3: ((f * 5.9) - q)] [Stage 2: ((d + q) - w)]
```

Tick 07:

```
Ready Queue: ["((f - g) * ((f * 5.9) - q))"]
ADD #1 : Idle
DIV #1 : Idle
MUL #1 : [Stage 3: (b * (k - t))] [Stage 1: ((f - g) * (...))]
SUB #1 : [Stage 3: ((d + q) - w)]
```

Lab 5-6

Code: $a-b*(k-t)-(f-g)*(f*5.9-q)-(f+g)/(d+q-w)$

Run:

- Tokenizer
- Syntax check
- Create Lexemes
- Build AST
- Compute AST #1
- Transform AST
- Compute AST #2
- Balance AST
- Compute AST #3
- Fold AST
- Compute AST #4
- Equivalent Forms

PUs	Ticks
ADD:	1 2t.
SUB:	1 3t.
MUL:	1 3t.
DIV:	1 5t.

Simulate PCS

PCS Config Reset

Optimization Research

Settings

Pretty Output

Save Config

Tick 08:

```
Ready Queue: ["((f + g) / ((d + q) - w))"]
ADD #1 : Idle
DIV #1 : [Stage 1: ((f + g) / (...)]
MUL #1 : [Stage 2: ((f - g) * (...)]
SUB #1 : Idle
```

Tick 09:

```
ADD #1 : Idle
DIV #1 : [Stage 2: ((f + g) / (...)]
MUL #1 : [Stage 3: ((f - g) * (...)]
SUB #1 : Idle
```

Tick 10:

```
Ready Queue: ["((b * (k - t)) + ((f - g) * ((f * 5.9) - q)))"]
ADD #1 : [Stage 1: ((b * (k - t)) / (...)]
DIV #1 : [Stage 3: ((f + g) / (...)]
MUL #1 : Idle
SUB #1 : Idle
```

Tick 11:

```
ADD #1 : [Stage 4: ((b * (k - t)) / (...)]
DIV #1 : Idle
MUL #1 : Idle
SUB #1 : Idle
```

Tick 12:

```
ADD #1 : Idle
DIV #1 : [Stage 5: ((f + g) / (...)]
MUL #1 : Idle
SUB #1 : Idle
```

Tick 13:

```
Ready Queue: ["(((b * (k - t)) + ((f - g) * ((f * 5.9) - q))) + ((f + g) / ((d + q) - w)))"]
ADD #1 : [Stage 1: (((b * (k - t)) + ((f - g) * ((f * 5.9) - q))) / (...)]
DIV #1 : Idle
MUL #1 : Idle
SUB #1 : Idle
```

Tick 14:

```
ADD #1 : [Stage 2: (((b * (k - t)) + ((f - g) * ((f * 5.9) - q))) / (...)]
DIV #1 : Idle
MUL #1 : Idle
SUB #1 : Idle
```

Tick 15:

```
Ready Queue: ["(a - (((b * (k - t)) + ((f - g) * ((f * 5.9) - q))) + ((f + g) / ((d + q) - w)))"]
ADD #1 : Idle
```

Для виконання завдань Лабораторної роботи №6 було реалізовано модуль `research`, який автоматизує пошук оптимальної форми виразу. Цей модуль використовує напрацювання лабораторних робіт №3-4 (генерацію еквівалентних форм) у поєднанні з симулятором з лабораторної №5. Алгоритм дослідження працює наступним чином:

1. Отримує базове оптимізоване дерево виразу.
2. Генерує повну множину алгебраїчно еквівалентних форм, застосовуючи асоціативний та дистрибутивний закони.
3. Для кожної згенерованої форми запускається окремий екземпляр симулятора `VectorSystemSimulator` із заданими параметрами архітектури.
4. Збираються результати всіх симуляцій (T_p , S , E) дляожної форми.

Результати дослідження виводяться у вигляді зведененої таблиці, де дляожної форми зазначено її текстове представлення та розраховані метрики. Система автоматично визначає оптимальну форму за критерієм мінімального часу виконання (T_p). У випадку, коли декілька форм мають одинаковий мінімальний час, перевага надається тій, що має меншу сумарну обчислювальну складність (T_1), що дозволяє обрати найбільш енергоефективний варіант серед найшвидших.

Для візуалізації результатів у графічний інтерфейс додано компоненти, що відображають детальну діаграму виконання по тактах, показуючи стан кожного функціонального блоку та черги готових завдань у будь-який момент часу.

The screenshot shows the 'Optimization Research' application window. On the left, the input code is `a*c + a*d + b*c + b*d`. The goal is to find the optimal parallel form for the given architecture. The system configuration is: Add(1), Sub(1), Mul(1), Div(1) | Costs: A=2, S=2, M=3, D=4. The results table lists 8 different expression forms (Snippets) along with their T1, Tp, Kp (Spd), and Ep (Eff) values. The optimal form found is ID #7, which is $((a + b) * (c + d))$, with metrics: T1 = 7, Tp = 6 ticks, Speedup = 1.1667, Efficiency = 0.2917. On the right, there is a 'Run' sidebar with various options like Tokenizer, Syntax check, Create Lexemes, Build AST, etc., and a Vector System configuration section with PUs and Ticks for ADD, SUB, MUL, and DIV operations.

ID	Form (Snippet)	T1	Tp	Kp (Spd)	Ep (Eff)
0	$((a * c) + (a * d)) + ((b * c) + (b * d))$	18	10	1.8000	0.4500
1	$(((c + d) * a) + (b * c)) + (b * d))$	15	9	1.6667	0.4167
2	$(((a + b) * c) + (a * d)) + (b * d))$	15	9	1.6667	0.4167
3	$(((a + b) * d) + (a * c)) + (b * c))$	15	9	1.6667	0.4167
4	$(((c + d) * b) + (a * c)) + (a * d))$	15	9	1.6667	0.4167
5	$((c * d) * a) + ((c + d) * b))$	12	8	1.5000	0.3750
6	$((a + b) * c) + ((a + b) * d))$	12	8	1.5000	0.3750
7	$((a + b) * (c + d))$	7	6	1.1667	0.2917

Optimal Form Found: ID #7
Expression: $((a + b) * (c + d))$
Metrics: T1 = 7, Tp = 6 ticks, Speedup = 1.1667, Efficiency = 0.2917

Додаткові приклади виразів.

Приклад 1:

Code: $(a-c)*(b-k+1)$

Optimization Research
Goal: Find the optimal parallel form for the given architecture.

System Config: Add(1), Sub(1), Mul(1), Div(1) | Costs: A=1, S=1, M=2, D=4

ID	Form (Snippet)	T1	Tp	Kp (Spd)	Ep (Eff)
0	$((b - k) + 1.00) * (a - c)$	5	4	1.2500	0.3125
1	$((a - c) * (b - k)) + (a - c)$	6	6	1.0000	0.2500
2	$((a - c) * b) - ((a - c) * k)) + (a - c)$	9	6	1.5000	0.3750
3	$((a - c) * (b - k)) + a - c$	6	6	1.0000	0.2500
4	$((a * b) - ((a - c) * k) + (b * c))$	11	7	1.5714	0.3929
5	$((a * b) - ((a - c) * k) - (c * k))$	11	6	1.8333	0.4583
6	$((a - c) * b) - ((a - c) * k) + a$	9	7	1.2857	0.3214
7	$((a * b) - ((a * k) + (b * c)))$	13	8	1.6250	0.4062
8	$((a * b) - ((a - c) * k) + (b * c))$	11	8	1.3750	0.3438
9	$((a - c) * b) - ((a * k) - (c * k))$	11	7	1.5714	0.3929
10	$((a * b) - ((a * k) - (c * k)))$	13	8	1.6250	0.4062
11	$((a * b) - ((a * k) + (b * c)))$	13	10	1.3000	0.3250
12	$((a - c) * b) - ((a - c) * k) + (b * c))$	11	7	1.5714	0.3929
13	$((a - c) * b) - ((a * k)) + (c * k))$	11	8	1.3750	0.3438
14	$((k - (1.00 + b)) * c) + (a * b))$	11	7	1.5714	0.3929
15	$((c - a) * k) + (a * b)) + a$	11	7	1.5714	0.3929
16	$((a * b) + (a - c)) - (b * c))$	13	8	1.6250	0.4062
17	$((b - k) + 1.00) * a) + ((k - (1.00 + b)) * c)$	9	6	1.5000	0.3750
18	$((1.00 - k) * a) + ((a - c) * b))$	11	7	1.5714	0.3929
19	$((a - c) * b) + ((c - a) * k))$	9	7	1.2857	0.3214
20	$((a - c) * b) + ((k - (1.00 + b)) * c))$	11	7	1.5714	0.3929
21	$((a - c) * b) + (a - c)) + (c * k))$	11	7	1.5714	0.3929
22	$((1.00 + b) * a) + ((c - a) * k))$	11	7	1.5714	0.3929
23	$(((-b) - 1.00) * c) + ((c - a) * k))$	12	7	1.7143	0.4286
24	$((c - a) * k) + (a - c)) + (a * b))$	11	7	1.5714	0.3929
25	$((b - k) * a) + (a - c)) + (c * k))$	11	7	1.5714	0.3929
26	$((k - b) * c) + (a - c)) + (a * b))$	11	7	1.5714	0.3929
27	$((1.00 - k) * a) + ((k - 1.00) * c))$	11	7	1.5714	0.3929
28	$((c - a) * k) + (a - c)) + ((a - c) * b))$	9	5	1.8000	0.4500
29	$((1.00 + b) * (a - c)) - (a * k))$	10	6	1.6667	0.4167
30	$(((-b) - 1.00) * c) + ((1.00 + b) * a))$	12	7	1.7143	0.4286
31	$((a - c) * b) + (a - c)) + ((c - a) * k))$	9	5	1.8000	0.4500
32	$((b - k) * a) + ((k - b) * c)) + (a - c))$	9	7	1.2857	0.3214
33	$((1.00 + b) * (a - c)) + ((c - a) * k))$	8	5	1.6000	0.4000

Optimal Form Found: ID #0
Expression: $((b - k) + 1.00) * (a - c)$
Metrics: T1 = 5, Tp = 4 ticks, Speedup = 1.2500, Efficiency = 0.3125

Приклад 2:

Code: $a \cdot b \cdot (k-t) \cdot (f-g) \cdot (f+g) / (d+q-w)$

Optimization Research
Goal: Find the optimal parallel form for the given architecture.

System Config: Add(1), Sub(1), Mul(1), Div(1) | Costs: A=2, S=3, M=3, D=5

ID	Form (Snippet)	T1	Tp	Kp (Spd)	Ep (Eff)
0	$(a - (((((5.90 * f) - q) * (f - g)) + (f * g)) * b) * k) * t) / (d + q - w)$	37	16	2.3125	0.5781
1	$(a - (((((5.90 * f) - q) * (f - g)) + (f * g)) * b) * k) * t) / (d + q - w)$	40	17	2.3529	0.5882
2	$(a - (((((f * g) * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)$	43	20	2.1500	0.5375
3	$(a - (((((5.90 * f) - q) * (f - g)) + (f * g)) * b) * k) * t) / (d + q - w)$	47	18	2.6111	0.6528
4	$(a - (((((f - g) * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)$	46	19	2.4211	0.6053
5	$(a - (((((5.90 * f) - q) * (f - g)) + (f * g)) * b) * k) * t) / (d + q - w)$	50	19	2.6316	0.6579
6	$(a - (((((f * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)$	48	19	2.5263	0.6316
7	$(a - (((((f - g) * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)$	46	20	2.3000	0.5750
8	$(a - (((((f - g) * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)$	53	22	2.4091	0.6023
9	$(a - (((((p * k) - (b * t)) + ((f * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	51	19	2.6842	0.6711
10	$(a - (((((f - g) * f) * 5.90) + ((b * t) * f) * k) * t) / (d + q - w)) / (d + q - w)$	49	20	2.4500	0.6125
11	$(a - (((((f - g) * f) * 5.90) + ((b * t) * f) * k) * t) / (d + q - w)) / (d + q - w)$	56	21	2.6667	0.6667
12	$(a - (((((f * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	51	20	2.5500	0.6375
13	$(a - (((((f * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	58	21	2.7619	0.6905
14	$(a - (((((f - g) * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	56	21	2.6667	0.6667
15	$(a - (((((f * k) - (b * t)) + ((f * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	54	21	2.5714	0.6429
16	$(a - (((((f * k) - (b * t)) + ((f * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	61	21	2.9048	0.7262
17	$(a - (((((f - g) * f) * 5.90) + ((b * t) * f) * k) * t) / (d + q - w)) / (d + q - w)$	59	22	2.6818	0.6705
18	$(a - (((((f * f) * 5.90) + ((k - t) * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	61	21	2.9048	0.7262
19	$(a - (((((f * f) * 5.90) + ((b * t) * f) * k) * t) / (d + q - w)) / (d + q - w)$	64	23	2.7826	0.6957
20	$(a - (((((f * f) * 5.90) + ((b * t) * f) * k) * t) / (d + q - w)) / (d + q - w)$	64	24	2.6667	0.6667
21	$(a - (((((t - k) * b) + a) * ((f * g) * f) * k) * t) / (d + q - w)) / (d + q - w)$	62	25	2.4800	0.6200
22	$(a - (((((t - k) * b) + a) * ((f * g) * f) * k) * t) / (d + q - w)) / (d + q - w)$	62	31	2.0000	0.5000
23	$(a - (((((5.90 * f) * (g - f)) * a) + ...)) / (d + q - w)) / (d + q - w)$	59	26	2.2892	0.5673
24	$(a - (((((5.90 * f) * (g - f)) * a) + ...)) / (d + q - w)) / (d + q - w)$	62	29	2.1379	0.5345
25	$(a - (((((f - g) * q) * a) + ((b * t) * f) * k) * t) / (d + q - w)) / (d + q - w)$	62	25	2.4800	0.6200
26	$(a - (((((f * g) * 5.90) * a) + ((q - (5.90 * f)) * b) * k) * t) / (d + q - w)) / (d + q - w)$	58	22	2.6364	0.6591
27	$(a - (((((5.90 * f) * (g - f)) * a) + ((t - k) * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	55	18	3.0556	0.7639
28	$(a - (((((5.90 * f) * (g - f)) * a) + ((t - k) * f) * b) * k) * t) / (d + q - w)) / (d + q - w)$	58	22	2.6364	0.6591
29	$(a - (((((f - g) * q) * a) + ((t - k) * b) * k) * t) / (d + q - w)) / (d + q - w)$	58	19	3.0526	0.7632
30	$(a - (((((5.90 * f) * (g - f)) * a) + ((q - (5.90 * f)) * b) * k) * t) / (d + q - w)) / (d + q - w)$	58	23	2.5217	0.6304
31	$(a - (((((5.90 * f) * (g - f)) * a) + ((f - g) * q) * b) * k) * t) / (d + q - w)) / (d + q - w)$	55	20	2.7500	0.6875
32	$(a - (((((5.90 * f) * (g - f)) * a) + ((q - (5.90 * f)) * b) * k) * t) / (d + q - w)) / (d + q - w)$	56	20	2.8000	0.7000
33	$(a - (((((5.90 * f) * (g - f)) * a) + ((f - g) * q) * b) * k) * t) / (d + q - w)) / (d + q - w)$	53	18	2.9444	0.7361
34	$(a - (((((5.90 * f) * (g - f)) * a) + ((t - k) * b) * k) * t) / (d + q - w)) / (d + q - w)$	56	20	2.8000	0.7000
35	$(a - (((((5.90 * f) * (g - f)) * a) + ((t - k) * b) * k) * t) / (d + q - w)) / (d + q - w)$	53	18	2.9444	0.7361

Optimal Form Found: ID #0
Expression: $(a - (((((5.90 * f) - q) * (f - g)) + ((k - t) * b)) + ((f + g) / ((d + q - w)))) / (d + q - w)$
Metrics: T1 = 37, Tp = 16 ticks, Speedup = 2.3125, Efficiency = 0.5781

Висновок.

У ході виконання лабораторної роботи було успішно спроектовано та програмно реалізовано систему симуляції роботи паралельної комп’ютерної системи (ПКС) векторного типу. Основною метою роботи було дослідження ефективності виконання арифметичних виразів на архітектурі з обмеженою кількістю функціональних блоків (суматорів, помножувачів тощо) та різною тривалістю виконання операцій.

В результаті розробки було створено комплексне програмне забезпечення мовою Rust, яке вирішує задачу відображення високорівневого абстрактного синтаксичного дерева (AST) на апаратні ресурси системи. Ключовим досягненням роботи стала відмова від спрощеної блокуючої моделі виконання на користь **повноцінної конвеєрної (pipelined) архітектури**.

Реалізований алгоритм симуляції враховує дві ключові часові характеристики функціональних блоків: затримка (*latency* – час проходження операції через усі стадії блоку) та пропускну здатність. Це дозволило моделювати ситуацію, коли функціональний блок (наприклад, помножувач із затримкою 2 такти) здатний приймати нові вхідні дані на кожному такті, не чекаючи завершення попередньої операції. Такий підхід забезпечив можливість часового перекриття команд (*instruction overlap*), що суттєво підвищило теоретичну швидкодію системи та коефіцієнт прискорення (S).

Для забезпечення коректного та ефективного планування завдань у конвеєрному режимі було вдосконалено алгоритм List Scheduling. Зокрема, впроваджено систему **ранжування вузлів (layering)**. Кожній підоперації у графі завдань присвоюється ранг, що відповідає її віддаленості від листових вузлів. Це дозволило застосувати евристику планування, яка надає пріоритет операціям нижчих рівнів, забезпечуючи своєчасну підготовку даних для наступних стадій обчислень та мінімізуючи простоту конвеєра через залежності по даних.

Важливою частиною роботи стала реалізація модуля автоматизованого дослідження (Researcher). Цей модуль генерує множину алгебраїчно еквівалентних форм вхідного виразу (використовуючи асоціативний та дистрибутивний закони) та проводить симуляцію кожної з них на заданій конфігурації ПКС. Порівняльний аналіз отриманих метрик (T_1 , T_p , S , E) дозволив на практиці підтвердити, що різні форми запису одного й того ж математичного виразу можуть мати суттєво різний час виконання на паралельній архітектурі. Система автоматично визначає форму, що забезпечує мінімальний час виконання (T_p) та найвищу ефективність використання ресурсів.

Для верифікації коректності роботи планувальника було розроблено детальну систему логування. Вихідні дані симуляції містять потактову діаграму стану кожного функціонального блоку, де відображається не просто факт зайнятості пристрою, а конкретна операція (у символному вигляді, наприклад, $a+b$) та стадія її виконання всередині конвеєра.

Таким чином, розроблена система дозволяє не лише моделювати роботу векторної ПКС, але й виконувати автоматичну оптимізацію програмного коду під конкретну апаратну архітектуру, що є критично важливим завданням у проектуванні високопродуктивних обчислювальних систем.

Лістинг.

pcs/vector.rs

```
1  use crate::compiler::ast::tree::{
2      AbstractSyntaxTree, AstNode, BinaryOperationKind, UnaryOperationKind,
3  };
4
5  use crate::compiler::pcs::{SystemConfiguration, TimeConfiguration};
6  use crate::compiler::reports::Reporter;
7  use crate::utils::StringBuffer;
8  use std::collections::{HashMap, HashSet};
9
10 #[derive(Debug, Clone, Copy, PartialEq, Eq, Hash)]
11 pub enum OperationType {
12     Add,
13     Sub,
14     Mul,
15     Div,
16     Load, // For variables and numbers (immediate execution)
17 }
18
19 impl OperationType {
20     fn execution_time(&self, time_config: &TimeConfiguration) -> usize {
21         match self {
22             Self::Add => time_config.add,
23             Self::Sub => time_config.sub,
24             Self::Mul => time_config.mul,
25             Self::Div => time_config.div,
26             Self::Load => 0,
27         }
28     }
29
30     fn from_binary(kind: &BinaryOperationKind) -> Self {
31         match kind {
32             BinaryOperationKind::Plus => Self::Add,
33             BinaryOperationKind::Minus => Self::Sub,
34             BinaryOperationKind::Multiply => Self::Mul,
35             BinaryOperationKind::Divide => Self::Div,
36             _ => OperationType::Load, // Logical ops are treated as immediate/loads for this lab
37         }
38     }
39
40     impl std::fmt::Display for OperationType {
41         fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
42             let str = match self {
43                 Self::Add => "ADD_BLOCK",
44                 Self::Sub => "SUB_BLOCK",
45                 Self::Mul => "MUL_BLOCK",
46                 Self::Div => "DIV_BLOCK",
47                 Self::Load => "LOAD",
```

```

48     };
49     write!(f, "{}", str)
50 }
51 }
52
53 #[derive(Debug, Clone)]
54 struct Task {
55     id: usize,
56     operation_type: OperationType,
57     dependencies: Vec<usize>,
58     display_name: String,
59 }
60
61 #[derive(Debug, Clone)]
62 pub struct ScheduledTask {
63     pub task_id: usize,
64     pub name: String,
65     pub start_time: usize,
66     pub end_time: usize,
67     pub processor: OperationType,
68 }
69
70 #[derive(Debug, Clone)]
71 pub struct TickLog {
72     pub tick: usize,
73     // Map: Processor Name -> Task Name (or "Idle")
74     pub processor_states: HashMap<String, String>,
75     pub ready_queue: Vec<String>,
76 }
77
78 pub struct SimulationResult {
79     pub schedule: Vec<ScheduledTask>,
80     pub tick_logs: Vec<TickLog>,
81     pub tl: usize,
82     pub tp: usize,
83     pub speedup: f64,
84     pub efficiency: f64,
85
86     pub configuration: SystemConfiguration,
87 }
88
89 pub struct VectorSystemSimulator<'a> {
90     ast: &'a AbstractSyntaxTree,
91     configuration: &'a SystemConfiguration,
92 }
93
94 impl<'a> VectorSystemSimulator<'a> {
95     pub fn new(
96         ast: &'a AbstractSyntaxTree, configuration: &'a SystemConfiguration,
97     ) -> Self {

```

```

98     Self { ast, configuration }
99 }
100
101 pub fn simulate(&self) -> SimulationResult {
102     // Convert AST to Task Graph
103     let (tasks, _) = Self::flatten_ast(self.ast);
104
105     // Simulate execution
106     let (schedule, tick_logs) = self.run_list_scheduling(tasks.clone());
107
108     // Calculate metrics
109     // Tp = End time of the last task
110     let tp = schedule.iter().map(|t| t.end_time).max().unwrap_or(0);
111
112     // T1 = Sum of execution times of all computational tasks (excluding Loads)
113     let t1: usize = schedule.iter().map(|t| t.end_time - t.start_time).sum();
114
115     let total_processors = self.configuration.processors.total();
116
117     // Avoid division by zero
118     let speedup = if tp > 0 { t1 as f64 / tp as f64 } else { 0.0 };
119     let efficiency = if total_processors > 0 {
120         speedup / total_processors as f64
121     } else {
122         0.0
123     };
124
125     SimulationResult {
126         schedule,
127         tick_logs,
128         t1,
129         tp,
130         speedup,
131         efficiency,
132
133         configuration: self.configuration.clone(),
134     }
135 }
136
137     /// Converts the recursive AST into a flat HashMap of Tasks with dependencies.
138 fn flatten_ast(ast: &AbstractSyntaxTree) -> (HashMap<usize, Task>, usize) {
139     let mut tasks = HashMap::new();
140     let mut id_counter = 0;
141     let root_id = Self::traverse_node(&ast.peek, &mut tasks, &mut id_counter);
142     (tasks, root_id)
143 }
144
145 fn traverse_node(
146     node: &AstNode, tasks: &mut HashMap<usize, Task>, counter: &mut usize,
147 ) -> usize {

```

```

148     let current_id = *counter;
149     *counter += 1;
150
151     match node {
152         AstNode::BinaryOperation {
153             operation,
154             left,
155             right,
156         } => {
157             let left_id = Self::traverse_node(left, tasks, counter);
158             let right_id = Self::traverse_node(right, tasks, counter);
159
160             tasks.insert(
161                 current_id,
162                 Task {
163                     id: current_id,
164                     operation_type: OperationType::from_binary(operation),
165                     dependencies: vec![left_id, right_id],
166                     display_name: operation.to_string(),
167                 },
168             );
169         },
170         AstNode::UnaryOperation {
171             operation,
172             expression,
173         } => {
174             let child_id = Self::traverse_node(expression, tasks, counter);
175
176             // Map Unary Minus to Subtraction block (0 - expr)
177             let operation_type = match operation {
178                 UnaryOperationKind::Minus => OperationType::Sub,
179                 _ => OperationType::Load, // Negation '!' treated as instant
180             };
181
182             tasks.insert(
183                 current_id,
184                 Task {
185                     id: current_id,
186                     operation_type,
187                     dependencies: vec![child_id],
188                     display_name: format!("Unary{}", operation),
189                 },
190             );
191         },
192         AstNode::Number(n) => {
193             tasks.insert(
194                 current_id,
195                 Task {
196                     id: current_id,
197                     operation_type: OperationType::Load,

```

```

198     dependencies: vec![],  

199     display_name: format!(" {:.1}", n),  

200     },  

201     );  

202 },  

203 AstNode::Identifier(s) => {  

204     tasks.insert(  

205         current_id,  

206         Task {  

207             id: current_id,  

208             operation_type: OperationType::Load,  

209             dependencies: vec![],  

210             display_name: s.clone(),  

211             },  

212         );  

213 },  

214 // Function calls and Array access treated as Load (black box)  

215 AstNode::FunctionCall { name, .. } => {  

216     tasks.insert(  

217         current_id,  

218         Task {  

219             id: current_id,  

220             operation_type: OperationType::Load,  

221             dependencies: vec![], // Simplifying: arguments handled inside but treated as one unit  

here  

222             display_name: format!("{}()", name),  

223             },  

224         );  

225 },  

226 AstNode::ArrayAccess { identifier, .. } => {  

227     tasks.insert(  

228         current_id,  

229         Task {  

230             id: current_id,  

231             operation_type: OperationType::Load,  

232             dependencies: vec![],  

233             display_name: format!("{}[..]", identifier),  

234             },  

235         );  

236 },  

237 _ => {  

238     tasks.insert(  

239         current_id,  

240         Task {  

241             id: current_id,  

242             operation_type: OperationType::Load,  

243             dependencies: vec![],  

244             display_name: "?".to_string(),  

245             },  

246         );

```

```

247     },
248 }
249 current_id
250 }
251
252 /// List Scheduling Algorithm
253 fn run_list_scheduling(
254     &self,
255     tasks: HashMap<usize, Task>,
256 ) -> (Vec<ScheduledTask>, Vec<TickLog>) {
257     let mut final_schedule = Vec::new();
258     let mut tick_logs = Vec::new();
259
260     // Initialize Processor states: Map<OpType, Vec<BusyUntilTick>>
261     // The Vec represents the pool of processors of that type.
262     let mut processors: HashMap<OperationType, Vec<usize>> = HashMap::new();
263     processors.insert(
264         OperationType::Add,
265         vec![0; self.configuration.processors.add],
266     );
267     processors.insert(
268         OperationType::Sub,
269         vec![0; self.configuration.processors.sub],
270     );
271     processors.insert(
272         OperationType::Mul,
273         vec![0; self.configuration.processors.mul],
274     );
275     processors.insert(
276         OperationType::Div,
277         vec![0; self.configuration.processors.div],
278     );
279
280     // "Load" operations don't need processors, they are instant.
281
282     // Task states
283     let mut task_finish_time: HashMap<usize, usize> = HashMap::new();
284     let mut completed_tasks: HashSet<usize> = HashSet::new();
285
286     // Pre-process "Load" tasks (variables/constants) as completed at T=0
287     for task in tasks.values() {
288         if task.operation_type == OperationType::Load {
289             task_finish_time.insert(task.id, 0);
290             completed_tasks.insert(task.id);
291         }
292     }
293
294     let mut current_tick = 0;
295     let mut active_tasks: HashMap<usize, (usize, OperationType)> = HashMap::new(); // TaskId -> (
296     EndTime, ProcType)
297

```

```

296     // Main Loop
297     loop {
298         // A. Check for completed tasks in this tick
299         let mut just_finished = Vec::new();
300         // We need to collect keys to remove to avoid borrowing issues
301         let finished_ids: Vec<usize> = active_tasks
302             .iter()
303             .filter(|( _, (end_time, _))| *end_time <= current_tick)
304             .map(|(id, _)| *id)
305             .collect();
306
307         for id in finished_ids {
308             active_tasks.remove(&id);
309             completed_tasks.insert(id);
310             just_finished.push(id);
311         }
312
313         // If all tasks are done, break
314         if completed_tasks.len() == tasks.len() {
315             break;
316         }
317
318         // B. Find Ready Tasks
319         let mut ready_queue: Vec<&Task> = tasks.values()
320             .filter(|t| !completed_tasks.contains(&t.id)) // Not completed
321             .filter(|t| !active_tasks.contains_key(&t.id)) // Not currently running
322             .filter(|t| {
323                 // All dependencies must be completed
324                 t.dependencies.iter().all(|dep| completed_tasks.contains(dep))
325             })
326             .collect();
327
328         // Heuristic: Sort by operation type cost (Longest Processing Time first) or just ID
329         ready_queue.sort_by(|a, b| {
330             let time_a = a.operation_type.execution_time(&self.configuration.time);
331             let time_b = b.operation_type.execution_time(&self.configuration.time);
332             if time_a != time_b {
333                 time_b.cmp(&time_a) // Higher cost first
334             } else {
335                 a.id.cmp(&b.id)
336             }
337         });
338
339         // Log snapshot preparation
340         let mut current_tick_log = TickLog {
341             tick: current_tick,
342             processor_states: HashMap::new(),
343             ready_queue: ready_queue.iter().map(|t| t.display_name.clone()).collect(),
344         };
345

```

```

346     // C. Assign Ready Tasks to Free Processors
347     for task in ready_queue {
348         if let Some(proc_pool) = processors.get_mut(&task.operation_type) {
349             // Find a processor that is free at current_tick
350             if let Some(proc_idx) = proc_pool
351                 .iter()
352                 .position(|&busy_until| busy_until <= current_tick)
353             {
354                 // Schedule!
355                 let duration =
356                     task.operation_type.execution_time(&self.configuration.time);
357                 let start = current_tick;
358                 let end = start + duration;
359
360                 // Mark processor busy
361                 proc_pool[proc_idx] = end;
362
363                 // Add to schedule
364                 final_schedule.push(ScheduledTask {
365                     task_id: task.id,
366                     name: task.display_name.clone(),
367                     start_time: start,
368                     end_time: end,
369                     processor: task.operation_type,
370                 });
371
372                 // Add to active tasks
373                 active_tasks.insert(task.id, (end, task.operation_type));
374
375                 // We also record finish time for dependency checking later
376                 task_finish_time.insert(task.id, end);
377             }
378         }
379     }
380
381     // D. Populate Tick Log with Processor Status
382     // Helper to find what task is running on a specific processor type
383     // Note: This simple model assumes 1 processor of each type.
384     // If COUNT > 1, we would need to track which specific processor index is used.
385     for operation_type in processors.keys() {
386         if operation_type.eq(&OperationType::Load) {
387             continue;
388         }
389
390         let op_name = operation_type.to_string();
391         let mut status = "Idle".to_string();
392
393         // Find if any active task matches this op_type
394         // Since we have 1 proc per type, we just check if there is ANY active task of this type
395         if let Some((id, _)) =

```

```

396     active_tasks.iter().find(|(_ , (_ , t))| t == operation_type)
397     && let Some(task) = tasks.get(id)
398     {
399         status = format!("Processing '{}'", task.display_name);
400     }
401
402     current_tick_log.processor_states.insert(op_name, status);
403 }
404 tick_logs.push(current_tick_log);
405
406 // E. Advance Time
407 current_tick += 1;
408
409 // Safety break
410 if current_tick > 10000 {
411     break;
412 }
413 }
414
415 (final_schedule, tick_logs)
416 }
417 }
418
419 impl Reporter {
420 pub fn pcs_simulation(&self, result: &SimulationResult) -> String {
421     let mut buffer = StringBuffer::default();
422
423     buffer.add_line("Parallel Computer System Simulation Results:".to_string());
424     buffer.add_line("-".repeat(60));
425     buffer.add_line(format!(
426         "Configuration: Add({}), Sub({}), Mul({}), Div({})",
427         result.configuration.processors.add,
428         result.configuration.processors.sub,
429         result.configuration.processors.mul,
430         result.configuration.processors.div,
431     ));
432     buffer.add_line(format!(
433         "Costs (Ticks): Add={}, Sub={}, Mul={}, Div={}",
434         result.configuration.time.add,
435         result.configuration.time.sub,
436         result.configuration.time.mul,
437         result.configuration.time.div,
438     ));
439     buffer.add_line("-".repeat(60));
440
441     // Metrics
442     buffer.add_line(format!("Sequential Time (T1): {}", result.t1));
443     buffer.add_line(format!("Parallel Time (Tp):   {}", result.tp));
444     buffer.add_line(format!("Speedup (Ky):      {:.4}", result.speedup));
445     buffer.add_line(format!("Efficiency (E):    {:.4}", result.efficiency));

```

```

446     buffer.add_line("-".repeat(60));
447
448     // Schedule Table
449     buffer.add_line(format!(
450         "{:<12} | {:<10} | {:<10} | {:<15}",
451         "Processor", "Start", "End", "Operation"
452     ));
453     buffer.add_line("-".repeat(60));
454
455     // Sort by start time for better readability
456     let mut sorted_schedule = result.schedule.clone();
457     sorted_schedule.sort_by_key(|t| t.start_time);
458
459     for task in sorted_schedule {
460         buffer.add_line(format!(
461             "{:<12} | {:<10} | {:<10} | {:<15}",
462             task.processor.to_string(),
463             task.start_time,
464             task.end_time,
465             task.name
466         ));
467     }
468
469     buffer.add_line("\n".to_string());
470     buffer.add_line("Tick-by-Tick Execution Log:".to_string());
471     buffer.add_line("-".repeat(60));
472
473     for log in &result.tick_logs {
474         buffer.add_line(format!("Tick {:02}", log.tick));
475
476         // Show Queue
477         if log.ready_queue.is_empty() {
478             buffer.add_line(" Queue: [Empty]".to_string());
479         } else {
480             buffer.add_line(format!(" Queue: {:?}", log.ready_queue));
481         }
482
483         // Show Processors
484         // Sort keys for consistent output
485         let mut keys: Vec<&String> = log.processor_states.keys().collect();
486         keys.sort();
487
488         for proc in keys {
489             let state = match log.processor_states.get(proc) {
490                 Some(state) => state,
491                 None => unreachable!("Non-existent processor in log"),
492             };
493             buffer.add_line(format!(" {:<10}: {}", proc, state));
494         }
495         buffer.add_line("".to_string());

```

```

496     }
497
498     buffer.get()
499 }
500 }
501

```

pcs/research.rs

```

1  use crate::compiler::ast::tree::AbstractSyntaxTree;
2  use crate::compiler::pcs::SystemConfiguration;
3  use crate::compiler::pcs::vector::{SimulationResult, VectorSystemSimulator};
4  use crate::compiler::reports::Reporter;
5  use crate::utils::StringBuffer;
6
7  pub struct Researcher<'a> {
8      forms: &'a Vec<AbstractSyntaxTree>,
9      configuration: &'a SystemConfiguration,
10 }
11
12 pub struct OptimizationReport {
13     pub index: usize,
14     pub canonical_string: String,
15     pub result: SimulationResult,
16 }
17
18 impl<'a> Researcher<'a> {
19     pub fn new(
20         equivalent_forms: &'a Vec<AbstractSyntaxTree>,
21         system_configuration: &'a SystemConfiguration,
22     ) -> Self {
23         Self {
24             forms: equivalent_forms,
25             configuration: system_configuration,
26         }
27     }
28
29     pub fn run(&self) -> Result<Vec<OptimizationReport>, String> {
30         let mut results = Vec::new();
31
32         for (index, form) in self.forms.iter().enumerate() {
33             let simulator = VectorSystemSimulator::new(form, self.configuration);
34             let result = simulator.simulate();
35             results.push(OptimizationReport {
36                 index,
37                 canonical_string: form.to_canonical_string(),
38                 result,
39             });
40         }
41
42         Ok(results)

```

```

43     }
44 }
45
46 impl Reporter {
47     pub fn generate_optimization_report(reports: &[OptimizationReport]) -> String {
48         let mut buffer = StringBuffer::default();
49
50         buffer.add_line("Optimization Research".to_string());
51         buffer.add_line(
52             "Goal: Find the optimal parallel form for the given architecture."
53             .to_string(),
54         );
55         buffer.add_line("-".repeat(100));
56
57         // First line contains system configuration
58         match reports.first() {
59             Some(first_report) => {
60                 let configuration = &first_report.result.configuration;
61                 let processors = &configuration.processors;
62                 let time = &configuration.time;
63
64                 buffer.add_line(format!(
65                     "System Config: Add({}), Sub({}), Mul({}), Div({}) | Costs: A={}, S={}, M={}, D={}",
66                     processors.add,
67                     processors.sub,
68                     processors.mul,
69                     processors.div,
70                     time.add,
71                     time.sub,
72                     time.mul,
73                     time.div,
74                 ));
75
76                 buffer.add_line("-".repeat(100));
77             },
78             None => {
79                 buffer.add_line("No optimization results available.".to_string());
80                 return buffer.get();
81             },
82         }
83
84         // Table header
85         buffer.add_line(format!(
86             "{:<4} | {:<40} | {:<5} | {:<5} | {:<8} | {:<8}",
87             "ID", "Form (Snippet)", "T1", "Tp", "Kp (Spd)", "Ep (Eff)"
88         ));
89         buffer.add_line("-".repeat(100));
90
91         let mut best_tp = usize::MAX;
92         let mut best_efficiency = f64::MAX;

```

```

93     let mut best_index = 0;
94
95     // Table rows
96     for report in reports {
97         let result = &report.result;
98         let form_str = &report.canonical_string;
99
100        // Shortening variable names for clarity
101        let short_form = if form_str.len() > 37 {
102            format!("{}...", &form_str[0..37])
103        } else {
104            form_str.clone()
105        };
106        buffer.add_line(format!(
107            "{:<4} | {:<40} | {:<5} | {:<5} | {:<8.4} | {:<8.4}",
108            report.index,
109            short_form,
110            result.t1,
111            result.tp,
112            result.speedup,
113            result.efficiency
114        ));
115
116        // Searching for optimal form
117        if result.tp < best_tp {
118            best_tp = result.tp;
119            best_index = report.index;
120        } else if result.tp == best_tp && result.efficiency > best_efficiency {
121            best_tp = result.tp;
122            best_index = report.index;
123            best_efficiency = result.efficiency;
124        }
125    }
126
127    buffer.add_line("-".repeat(100));
128
129    // Conclusion with the best form
130    let best_report = &reports[best_index];
131    let best_result = &best_report.result;
132    buffer.add_line(format!("\nOptimal Form Found: ID #{}", best_index));
133    buffer.add_line(format!("Expression: {}", best_report.canonical_string));
134    buffer.add_line(format!(
135        "Metrics: Tp = {} ticks, Speedup = {:.4}, Efficiency = {:.4}",
136        best_result.tp, best_result.speedup, best_result.efficiency
137    ));
138
139    buffer.get()
140 }
141
142

```