МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження алгоритмів відображення програм на кластери із графічними процесорами
(тема)

Виконав:
Студент 2 курсу, групи ІПЗм-20-2
Федоров Я.О.
(прізвище, ініціали)

Спеціальність 121 Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник проф. Шубін І.Ю.
(посада, прізвище)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)
« ____ » _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Федорова Яна Олександровича
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження алгоритмів відображення програм на кластери із графічними процесорами

затверджена наказом університету від 24.03.2022 р. № 412 Ст

2. Термін подання роботи до екзаменаційної комісії 10 05 2022р.

3. Вихідні дані до роботи *базові алгоритми розпаралелювання програм, побудови складних форм, програмування графічних процесорів, математичні методи моделювання скінченних автоматів*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, опис запропонованих варіантів оптимізації, використовувані методи та алгоритми, опис розробленої програмної системи, опис застосованих програмних рішень, аналіз можливих застосувань*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	31 березня 2022	*виконано*
2.	Огляд існуючих методів	10 квітня 2022.	*виконано*
3.	Розробка алгоритмів, проектування та розробка ПЗ	15 квітня 2022	*виконано*
4.	Підготовка пояснювальної записки	20 квітня 2022	*виконано*
5.	Спецчастина	28 квітня 2022	*виконано*
6.	Підготовка презентації та доповіді	03 травня 2022	*виконано*
7.	Попередній захист	05 травня 2022	*виконано*
8.	Нормоконтроль, рецензування	07 травня 2022.	*виконано*
9.	Занесення роботи в електронний архів	08 травня 2022.	*виконано*
10.	Допуск до захисту в зав. кафедри	11 травня 2022	*виконано*

Дата видачі завдання 28 березня 2022р.

Студент _____
(підпис)

Керівник роботи _____ проф. Шубін І.Ю
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 94 с., 28 рис., 5 табл., 27 джерел.

ПАРАЛЕЛЬНІ ПРОГРАМИ, КЛАСТЕРИ ПРОЦЕСОРІВ, ГРАФІЧНІ ПРОЦЕСОРИ, ДВМ-Н-ПРОГРАМИ.

Об'єктом дослідження є методи розпаралелювання програм.

Метою роботи є дослідження можливостей відображення ДВМ-програм на кластер із прискорювачами обчислень, що забезпечують розподіл, між універсальними багатоядерними процесорами і декількома графічними процесорами.

Методи розробки – методи, методи кластерного аналізу, паралельне програмування.

У результаті з використанням розроблених алгоритмів створена система підтримки виконання ДВМ-програм, що є невід'ємною частиною компіляторів ДВМ-програм.

PARALLEL PROGRAMS, PROCESSOR CLUSTERS, GRAPHIC PROCESSORS, DVM-H-PROGRAMS.

The object of research is the methods of ontological descriptions.

The aim of the work is to study the possibilities of mapping ДВМ-programs to a cluster with computational accelerators that provide distribution between universal multi-core processors and multiple graphics processors.

Development methods – methods, methods of cluster analysis, parallel programming.

As a result, using the developed algorithms created a system to support the execution of H-programs, which is an integral part of the compilers of ДВМ-H-programs.

Умови публікації пояснювальної записки

Я, _____Федоров Ян Олександрович_____
(прізвище, ім'я, по батькові)

студент групи ___ІПЗм-20-2_ здобувач вищої освіти на другому (магістерському) рівні

кафедра програмної інженерії,
(повна назва кафедри)

заявляю: моя кваліфікаційна робота на тему ___Дослідження алгоритмів відображення програм на кластери із графічними процесорами ___,
(назва роботи)

що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ ……..……..8
1 Аналіз стану розв'язання проблеми та обґрунтування
цілей дослідження ……..11
1.1 Огляд високорівневих моделей програмування для прискорювачів ……11
1.2 Алгоритми відображення програм ……...13
1.3 Розроблення розширення ДВМ-моделей ……15
1.4 Організація обчислень та специфікації потоків даних ……18
1.5 Обґрунтування цілей дослідження ……...22
2 Опис проведених теоретичних досліджень …….…..25
2.1 Алгоритми розподілу ресурсів …….…….....25
2.2 Динамічний режим з добором розподілу ……26
2.3 Динамічний режим з використанням підібраної схеми розподілу ……32
2.4 Алгоритми обліку актуального стану змінних ……35
3 Аналіз результатів досліджень 40
3.1 Алгоритми обліку стану актуальності змінних …….……40
3.2 Алгоритм обробки вказівників …….…….……43
3.3 Алгоритми розподілу даних і обчислень ……..45
3.4 Динамічний режим з використанням схем розподілу …….....52
4 Програмна реалізація системи …….....56
4.1 Додатки, що демонструють застосовність програмної моделі …….56
4.2 Моделювання тривимірних алгоритмів …….…….……..66
5 Опис можливості використання отриманих результатів.....69
Висновки ……..73
Перелік джерел посилання …….....74
Додаток А Перелік джерел посилання за науковими напрямами керівника
та науковців кафедри програмної інженерії ……78
Додаток Б Звіт результатів перевірки на унікальність тексту ……..…….....79

Додаток В Слайди презентації ……81
Додаток Г Листінг модуля 89
Додаток Д Апробація роботи……..……...……..91
Додаток Е Експертний висновок результатів перевірки кваліфікаційної
роботи на відповідність оформлення вимогам ДСТУ …….. 93

ВСТУП

Останнім часом з'являється багато обчислювальних кластерів із встановленими у їх вузлах прискорювачами. В основному це графічні процесори компанії NVIDIA. У 2012 році починають з'являтися кластери з прискорювачами іншої архітектури – Xeon Phi від компанії Intel. Так, у списку Top500 [1] найвищих продуктивних суперкомп'ютерів світу, оголошеному в листопаді 2012 р., 62 машини мають у своєму складі прискорювачі, з них 50 машин мають прискорювачі NVIDIA, 7 – Intel, 3 – AMD/ATI, 2 – IBM. Ця тенденція помітно ускладнює процес програмування кластерів, оскільки потребує освоєння на достатньому рівні відразу кількох моделей та мов програмування.

Велика кількість паралельних програм для кластерів розробляються з використанням низькорівневих засобів передачі повідомлень (MPI [1]). MPI-програми важко розробляти, супроводжувати і повторно використовувати при створенні нових програм. Дана проблема ускладнюється тим, що останнім часом з'являється багато обчислювальних кластерів із установленими в їхніх вузлах прискорювачами. В основному, це графічні процесори. Програмістові потрібно тепер освоєння на достатньому рівні відразу декількох моделей і мов програмування. Традиційним підходом можна назвати використання технології MPI для поділу роботи між вузлами кластера, а потім технологій OpenMP [2] і CUDA [3] або OpenCL [4] для завантаження всіх ядер центрального і графічного процесорів.

Тому прикладний програміст прагне одержати інструмент, що автоматично перетворить його послідовну програму в паралельну програму для кластера із прискорювачами, або високорівнева мова паралельного програмування, що забезпечує ефективне використання сучасних паралельних систем. Проведені в 900их роках активні дослідження переконливо показали, що повністю автоматичне розпарелювання для кластерних ЕОМ реальних промислових програм можливо тільки в дуже рідких випадках.

Спочатку дуже високі вимоги до ефективності виконання паралельних програм, а потім і стрімкі зміни в архітектурі паралельних ЕОМ привели до того, що дотепер так і немає загальновизнаної високорівневої мови паралельного програмування, що дозволяє ефективно використовувати можливості сучасних ЕОМ, а також у достатньому ступені абстрагуватися від конкретної конфігурації обчислювальної системи з метою забезпечення ефективного перенесення програми.

Цілями даної роботи є:

– дослідження можливостей відображення ДВМ-програм [5] на кластер із прискорювачами обчислень, що забезпечують розподіл, між універсальними багатоядерними процесорами (ЦПУ) і декількома графічними процесорами (ГПУ);

– розробка алгоритмів розподілу обчислень між ЦПУ і ГПУ і алгоритмів керування переміщенням даних між пам'яттю ЦПУ і пам'яттю ГПУ;

– розробка алгоритму розподілу підзадач між вузлами кластера.

Необхідно запропонувати принципи і алгоритми відображення ДВМ-програм на кластер із прискорювачами, які дозволяють використовувати ДВМ-мови з мінімальним їхнім розширенням. При цьому виходити з того, що у вузлах кластера будуть розміщатися прискорювачі різної архітектури.

Тому що цільовою машиною обраний кластер із багатоядерними процесорами і прискорювачами, то ставиться завдання розробити програмні засоби, що забезпечують використання для обчислень одночасно як ЦПУ, так і ГПУ. У використовуваних низькорівнених засобах програмування для ГПУ (і в нових високорівнених засобах теж) є складності для прикладного програміста, пов'язані з копіюванням даних на прискорювач і назад, тому ставиться завдання по можливості автоматизувати даний процес у розроблювальнім розширенні моделі ДВМ.

Необхідно розробити і впровадити в систему підтримки виконання ДВМ-програм алгоритм розподілу підзадач між вузлами кластера, що забезпечує балансування завантаження обчислювальних ресурсів. Алгоритм повинен

виходити з наявності заданого часу виконання кожної підзадачі на різнім числі процесорів.

Розробити засоби налагодження, що дозволяють знаходити помилки програміста, допущені при розпаралелювані програми на кластер із прискорювачами, а також коректність обчислень, зроблених на графічному процесорі.

З використанням розроблених алгоритмів має бути створена система підтримки виконання ДВМ-Н-програм, що має стати частиною компіляторів ДВМ-Н-програм. Розробка такого компілятора суттєво спростила би створення ефективних програм для кластерів із прискорювачами, здатних автоматично настроюватися на цільову конфігурацію обчислювальної системи. Компілятор з мови ДВМ-Н, що включає в себе систему підтримки виконання ДВМ-Н-програм, входить до складу ДВМ-системи.

З використанням цього компілятора може бути розпаралелюваний на кластер із прискорювачами ряд прикладних обчислювальних завдань.

1 АНАЛІЗ СТАНУ РОЗВ'ЯЗАННЯ ПРОБЛЕМИ ТА ОБҐРУНТУВАННЯ ЦІЛЕЙ ДОСЛІДЖЕННЯ

1.1 Огляд високорівневих моделей програмування для прискорювачів

В 2011 році була запропонована модель ДВМ-Н для програмування кластерів із прискорювачами за допомогою директив. У листопаді 2011 року був запропонований стандарт OpenACC [1] для програмування графічних процесорів за допомогою директив, його друга версія вийшла в червні 2013 року. У липні 2013 року опублікований стандарт Openmp 4.0, у який увійшли засобу для роботи з обчислювальними обладнаннями, що підключаються, володіють власною пам'яттю (співпроцесори, прискорювачі).

Головна відмінність цих підходів полягає в тому, що ДВМ-Н призначений для написання програм для кластерів, у вузлах яких установлені багатоядерні процесори і прискорювачі різних архітектур, що відрізняються як по типу пам'яті (загальна або роздільна), так і по швидкості обмінів між пам'яттю прискорювачів і пам'яттю ЦПУ. Стандарти OpenACC і OpenMP 4.0 призначені для написання програм, що виконуються в окремих вузлах такого кластера.

У специфікаціях ДВМ-Н і OpenACC проглядається орієнтованість на графічні процесори в частині організації паралелізму, керованого кешування даних, відсутності засобів синхронізації і програмного інтерфейсу часу виконання, призначених для використання на прискорювачі. Для Openmp 4.0 це невірно і може бути певною перешкодою для його реалізації для графічних процесорів.

У стандартах Openacc і Openmp 4.0 дуже схожа методика керування переміщенням даних, заснована на вказівці кожного переміщення користувачем і обумовлених користувачем інтервалах (для Openmp 4.0 – тільки статично певних) існування екземпляра змінної на прискорювачі. В ДВМ-Н-мовах застосована інша методика, заснована на вказівниках вхідних і вихідних даних для фрагментів коду, призначених для виконання на прискорювачах (такі фрагменти називаються

обчислювальними регіонами або просто регіонами), що і дозволяє автоматично визначати необхідні переміщення даних залежно від того, на якому обладнанні (прискорювачі або багатоядерні процесори) був виконаний той або інший обчислювальний регіон.

У стандартах Openacc і Openmp 4.0 керування тим, чи виконувати даний регіон на прискорювачі чи ні, завдається вираженнями, що обчислюються під час виконання. В ДВМ-Н такої можливості не має, однак є кілька режимів планування, що визначають, на яких обладнаннях виконувати регіони.

Моделі ДВМ-Н і Openmp 4.0 передбачають використання декількох прискорювачів для однієї програми, але варто відзначити, що в ДВМ-Н це є наслідком наявності вказівок по розпаралелюванню на кластер і не вимагає додаткового керування з боку користувача, а Openmp 4.0 надає набір засобів по використанню декількох прискорювачів з повністю ручним керуванням кожним з них. В OpenACC одночасна робота з декількома прискорювачами не передбачена і припускає додаткове використання технології розпаралелювання для мультипроцесора (наприклад, OpenMP).

Пряме переміщення даних між прискорювачами в стандартах OpenMP 4.0 і Openacc не передбачене, у той час як в ДВМ-Н цей функціонал є завдяки обраній методиці керування переміщеннями даних.

Усі три моделі надають кошти для асинхронного виконання обчислювальних регіонів на прискорювачі. Усі три моделі надають можливість не вказувати повністю списки використовуваних або переміщуваних змінних, застосовуючи консервативні умовчання в цьому випадку.

Для написання програм для кластера із прискорювачами ДВМ-Н надає всі засоби, а при використанні OpenACC або OpenMP 4.0 прийде додати використання комунікаційної бібліотеки, наприклад, MPI.

По частині підтримки циклів із залежностями ДВМ-Н має підтримку приватних змінних, редукційних залежностей, регулярних залежностей по одному або декільком вимірам відразу; Openacc має підтримку тільки приватних змінних і редукційних залежностей; Openmp 4.0 має підтримку приватних змінних і

редукційних залежностей, а також за допомогою вбудованих засобів синхронізації і програмного інтерфейсу часу виконання дозволяє розпаралелювати і цикли з іншими типами залежностей.

1.2 Алгоритми відображення програм

Одним з важливих аспектів функціонування такої програмної моделі, як ДВМ-Н є питання відображення вихідної програми на всі рівні паралелізму і різнорідні обчислювальні обладнання. Важливими завданнями механізму відображення є забезпечення коректного виконання всіх підтримуваних мовою конструкцій на різнорідних обчислювальних обладнаннях, балансування навантаження між обчислювальними обладнаннями, а також вибір оптимального способу виконання кожної ділянки коду на тому або іншому обладнанні.

Паралелізм в ДВМ-Н-програмах проявляється на декількох рівнях:

– розподіл даних і обчислень по Мрі-процесам. Цей рівень задається спеціальними директивами розподілу або перерозподілу даних і специфікаціями паралельних підзадач і циклів;

– розподіл даних і обчислень по обчислювальних обладнаннях при вході в обчислювальний регіон;

– паралельна обробка в рамках конкретного обчислювального обладнання – цей рівень з'являється при вході в паралельний цикл, що перебуває усередині обчислювального регіону.

Наявність цих рівнів дає можливість органічно відобразити програму на кластер із багатоядерними процесорами і прискорювачами у вузлах.

Виконання ДВМ-Н-програми можна представити, як виконання послідовності обчислювальних регіонів і ділянок між ними, які будемо називати позарегіональним простором. Код у позарегіонному просторі виконується на центральному процесорі, тоді як для обчислювальних регіонів можливо їх

виконання на різнорідних обчислювальних обладнаннях. Усередині, так само як і поза регіонами можуть бути як паралельні цикли, так і послідовні ділянки програми.

Виконання ДВМ-Н-програми починається синхронно всіма запущеними процесами в моделі SPMD. На кожний процес виділяється одна основна послідовна нитка виконання.

При вході в обчислювальний регіон кожний процес незалежно виконує додатковий до міжпроцесного розподілу даних, використовуваних цим обчислювальним регіоном, по обчислювальних обладнаннях, обраних для виконання регіону. На цьому етапі проводиться динамічне планування з метою балансування навантаження і мінімізації тимчасових витрат на переміщення даних, пов'язаних з перерозподілом даних.

При вході в паралельний цикл усередині регіону кожний процес розділяє роботу відповідно до розподілу даних по обчислювальних обладнаннях. Потім він вибирає для кожного обладнання метод обробки частини циклу на конкретнім обладнанні, називаний оброблювачем, а також оптимізаційні параметри для обраного оброблювача. На цьому етапі проводиться динамічне настроювання оптимізаційних параметрів, включаючи кількість ЦПУ та ниток для оброблювачів на ЦПУ, а також розмір і форму блоку ниток для CUDA-оброблювачів з метою мінімізації часу виконання на кожнім окремім обладнанні.

Паралельний цикл усередині регіону при виконанні розпадається на кілька частин, кожна з яких обробляється деяким оброблювачем на деякому обчислювальнім обладнанні. На цьому етапі збирається основна інформація для налагодження ефективності ДВМ-Н-програми.

При виході з обчислювального регіону збирається додаткова інформація для цілей динамічного планування виконання регіону, а також може бути здійснене порівняльне налагодження з метою контролю коректності результатів, отриманих при рахунку на прискорювачах.

В ДВМ-Н-програмах усі дані, що розподіляються, розподіляються блоково: кожний розподілений вимір ділиться декількома крапками на відрізки. При цьому

є можливість по кожному виміру розподіленого масиву задавати або рівномірний блоковий розподіл, або розподіл зваженими блоками, тобто з урахуванням заданого вектора ваг. Ці Вказівника безпосередньо виконуються при розподілі даних між Mpi-процесами. Вкладені розподіли, що з'являються при вході в обчислювальний регіон, будуються з використанням цієї інформації, але, у силу різнорідності обчислювальних обладнань, можуть мати, що відрізняється від зовнішньої схему розподілу.

Системою підтримки виконання ДВМ-Н-програм підтримуються три режими розподілу даних і обчислень по обчислювальних обладнаннях у крапках входу в регіони:

– простий статичний режим;

– динамічний режим з добором схеми розподілу;

– динамічний режим з використанням підібраної схеми розподілу.

1.3 Розроблення розширення ДВМ-моделей

Поява багатопроцесорних ЕОМ з розподіленою пам'яттю значно розширило можливості рішення більших завдань, однак різко ускладнило розробку програм. Програміст повинен розподілити дані між процесорами, а програму представити у вигляді сукупності процесів, взаємодіючих між собою за допомогою обміну повідомленнями.

У цей час більшість паралельних програм для кластерів розробляється з використанням низькорівненої моделі передачі повідомлень MPI [1], при використанні якої програміст пише програму, що виконується паралельними процесами. Для поділу роботи між ними, для передачі інформації від одного процесу іншому, а також для синхронізації паралельних процесів надаються бібліотечні виклики. Також використовується модель однобічних комунікацій, представником якої є бібліотека SHMEM [2].

Використовуються також і засновані на директивах високорівненого розширення стандартних мов програмування для програмування кластерів, мови Фортран-ДВМ [3] і Сі-ДВМ [4]. Подібні розширення не є широко використовуваними, однак, становлять інтерес як засіб спрощення програмування кластерів.

З розвитком графічних процесорів стало можливим використовувати їх і для обчислень загального призначення, а в останні роки їх стали встановлювати в суперкомп'ютери для прискорення обчислень .

Для програмування графічних процесорів виробники апаратного забезпечення запропонували дві досить низькорівнені технології – CUDA і OpenCL. Відносна складність і незвичайність (у порівнянні з роботою на ЦПУ) використання графічних процесорів для обчислень загального призначення втримує їх від повсюдного використання [5].

У той же час успіх OpenMP, призначеного для систем із загальною пам'яттю, давав прикладним програмістам надію на появу подібних засобів для використання графічних процесорів, а дослідницьким групам – напрямок для розробок.

Однак поширити стандарт OpenMP на графічні процесори не виходило по наступних причинах: по-перше, наявність окремої пам'яті графічного процесора і необхідності управляти переміщеннями даних на прискорювач і із прискорювача, що ще ускладнюється при підключенні відразу декількох прискорювачів до одній ЕОМ. По-друге, більш складна, ієрархічна модель паралелізму; наявність декількох принципове різних видів пам'яті на прискорювачі. По-третє, обмежені апаратні можливості синхронізації і використання готових бібліотек із прискорювача.

У рамках даного дослідження проаналізоване та викоистане розширення ДВМ-моделі, що дозволяє розробляти програми для кластерів із графічними процесорами. Ця розширена модель названа ДВМ-Н (ДВМ for Heterogeneous systems) [5].

Модель ДВМ-Н є розширенням моделі ДВМ конструкціями, призначеними для двох завдань:

– організація обчислень, специфікації потоків даних.

– керування, що відвантажуються даними, актуальністю. Далі вигляд усіх директив приводиться для мови Фортран-ДВМ-Н.

Виконання ДВМ-Н-програми можна представити, як виконання послідовності обчислювальних регіонів і ділянок між ними, які називають позарегіональним простором. Код у позарегіоному просторі виконується на центральному процесорі, тоді як для обчислювальних регіонів можливо їх виконання на різнорідних обчислювальних обладнаннях. Усередині, так само як і поза регіонами можуть бути як паралельні цикли, так і послідовні ділянки програми.

Виконання ДВМ-Н-програми починається синхронно всіма запущеними процесами в моделі SPMD. На кожний процес виділяється одна основна послідовна нитка виконання. При вході в обчислювальний регіон кожний процес незалежно виконує додатковий до міжпроцесного розподілу даних, використованих цим обчислювальним регіоном, по обчислювальних обладнаннях, що обрані для виконання регіону. На цьому етапі проводиться динамічне планування з метою балансування навантаження і мінімізації тимчасових витрат на переміщення даних, пов'язаних з перерозподілом даних.

При вході в паралельний цикл усередині регіону кожний процес розділяє роботу відповідно до розподілу даних по обчислювальних обладнаннях. Потім він вибирає для кожного обладнання метод обробки частини циклу на конкретнім обладнанні, називаний оброблювачем, а також оптимізаційні параметри для обраного оброблювача. На цьому етапі проводиться динамічне настроювання оптимізаційних параметрів, включаючи кількість ЦПУ та ниток для оброблювачів на ЦПУ, а також розмір і форму блоку ниток для CUDA-оброблювачів з метою мінімізації часу виконання на кожнім окремім обладнанні.

Паралельний цикл усередині регіону при виконанні розпадається на кілька частин, кожна з яких обробляється деяким оброблювачем на деякому

обчислювальнім обладнанні. На цьому етапі збирається основна інформація для налагодження ефективності ДВМ-Н-програми.

При виході з обчислювального регіону збирається додаткова інформація для цілей динамічного планування виконання регіону, а також може бути здійснене порівняльне налагодження з метою контролю коректності результатів, отриманих при розрахунку на графічних прискорювачах.

1.4 Організація обчислень та специфікації потоків даних

Обчислювальний регіон виділяє фрагмент програми з одним входом і одним виходом для можливого виконання на одному або декількох обчислювачах. Регіон може бути виконаний на одному або відразу декількох прискорювачах і/або на багатоядерному процесорі, при цьому на ЦПУ може бути виконаний будь-який регіон, а на можливість використання кожного типу прискорювачів накладаються свої додаткові обмеження на зміст регіону, при цьому зміст хост-секції (спеціального виду секції регіону, яка завжди виконується на ЦПУ) не впливає на визначення можливості виконання регіону на тому або іншому обладнанні.

Наприклад, з використанням Cuda-обладнання може бути виконаний будь-який регіон без використання операцій уведення/висновку, викликів зовнішніх процедур, рекурсивних викликів.

Для керування тем, на яких обчислювачах регіон може виконуватися можна використовувати вказівку `targets`.

Регіон описується в такий спосіб:

```
!DVM$ REGION [clause {, clause}]
   region_inner
! DVM $ END REGION
```

Вкладені (статично або динамічно) регіони не допускаються. region_inner – це нуль або більш наступних один за одним конструкцій, кожна з яких є одним з нижченаведеного:

– паралельний ДВМ-цикл. Простір витків паралельного циклу розділяється відповідно до заданого в директиві паралельного циклу правилом відображення між обчислювачами, обраними для даного регіону;

– послідовна група операторів. Кожний оператор послідовної групи операторів виконується на всіх обчислювачах, обраних для виконання регіону, крім випадку модифікації в ньому розподілених даних – тоді діє правило власних обчислень.

Хост-секція обмежується спеціальними директивами:

```
! DVM$ HOSTSECTION
  hostsection_inner
! DVM$ END HOSTSECTION
```

Секція повідомляє спеціального виду секцію виконання на ЦПУ hostsection_inner – це частина програми з одним входом і одним виходом, ця частина буде виконуватися на ЦПУ в послідовному режимі. Дозволене використання усередині таких секцій директив get_actual. Директиви actual заборонені. Усякі зміни змінних у цій секції можуть пропасти. Такі секції пропонується використовувати в відлагоджених цілях для проміжного контролю значень змінних по ходу виконання регіону. Операції висновку дозволені, виклики зовнішніх процедур дозволені.

У якості clause для обчислювальних регіонів може бути задане:

```
in(subarray_or_scalar {, subarray_or_scalar}),
out(subarray_or_scalar {, subarray_or_scalar}),
inout(subarray_or_scalar {, subarray_or_scalar}),
local(subarray_or_scalar {, subarray_or_scalar}),
inlocal(subarray_or_scalar {, subarray_or_scalar})
```

Вказівник напрямку використання підмасивів і скалярів у регіоні. **in** – по входу в регіон потрібні актуальні дані. out – у регіоні значення змінної змінюється, причому ця зміна може бути використане далі. inout(a) – скорочений

запис одночасно двох вказівок: in(a), out(a). local — у регіоні значення змінної змінюється, але ця зміна не буде використана далі. inlocal(a) — скорочений запис одночасно двох вказівок: in(a), local(a). Якщо для змінної зазначене in, і не зазначене out або local, то вважається, що в таку змінну в регіоні взагалі немає записів і вона не міняється в процесі його виконання.

Вказівник списку типів обчислювачів

targets(target_name {, target_name})

де target_name це CUDA або HOST , на яких передбачається виконувати регіон.

Такий вказівник може бути тільки один в директиві регіону. Даний вказівник обмежує набір типів обчислювальних обладнань, для використання яких регіон буде підготовлений компілятором. Дійсне ж виконання регіону може відбуватися тільки на доступних ДВМ-Н-програмі прискорювачах (або на ЦПУ), кількість і типи яких вказуються при її запуску за допомогою змінних оточення. Визначення конкретних виконавців регіону (із числа доступних обчислювачів, для яких були згенеровані програми регіону) проводиться під час виконання програми.

Вказівник можливості асинхронного виконання регіону async. При запуску регіону в будь-якому режимі (синхронний, асинхронний) очікування завершення раніше запущеного регіону виникає, якщо Вказівниками in, out, local, inout, inlocal задається необхідність змінити дані, використовувані цим (раніше запущеним) регіоном або необхідність використовувати (запис або читання) дані, змінювані цим (раніше запущеним) регіоном (out, inout, local, inlocal).

Керування не перейде на наступний за синхронним регіоном оператор, поки поточний регіон не закінчить виконання. Керування може перейти на наступний за асинхронним регіоном оператор, не чекаючи його завершення (або навіть його старту).

Вказівник всіх використовуваних змінних у регіоні не обов'язковий. При цьому використовувані, але не зазначені в директиві регіону змінні, включаються в регіон в автоматичному режимі компілятором за правилами:

– усі використовувані масиви вважаються використовуваними повністю (не

виділяються підмасиви);

– усяка змінна, яка використовується на читання одержує атрибут in;

– усяка змінна, яка використовується на запис одержує атрибут inout;

– усяка використовувана в регіоні змінна, напрямок використання якої не піддається визначенню, одержує атрибут inout;

– вказівники `local` і `out` в автоматичному режимі не проставляються.

Переміщення даних між обчислювальними регіонами здійснюється відповідно до вказівок у директиві регіону. Керування переміщеннями даних поза регіонами здійснюється з допомогою наступних директив, що виконуються:

```
get_actual[(subarray_or_scalar {, subarray_or_scalar})]
```

Робить усі необхідні відновлення для того, щоб в основній пам'яті (пам'яті ЦПУ) були самі нові дані в зазначеному підмасиві або скалярі (при цьому, можливо, нічого і не слід переміщати). У випадку відсутності в директиви параметрів усі наявні нові дані із прискорювачів листуються в основну пам'ять.

```
actual[(subarray_or_scalar {, subarray_or_scalar})]
```

Повідомляє той факт, що зазначений підмасив або скаляр саму нову версію має в основній пам'яті (пам'яті ЦПУ). При цьому зазначені змінні або елементи масивів, що перебувають у пам'яті прискорювачів уважаються застарілими і перед використанням будуть по необхідності оновлені. У випадку відсутності параметрів усі змінні в пам'яті прискорювачів оголошуються застарілими.

Дані директиви слугують для організації взаємодії покритої регіонами частини програми з іншою частиною програми. Можна виділити дві основні сфери застосування для них:

– використання в процесі інкрементального розпаралелювання;

– використання для підтримки роботи з бібліотеками зовнішніх функцій, у тому числі бібліотек уведення/висновку.

Директиви DISTRIBUTE і ALIGN визначають розподіл даних, у цьому випадку рівномірний блоковий розподіл, який задається для масиву A, а для масиву B задається його вирівнювання на масив A, що означає розподіл масиву B так само, як і масиву A.

Паралельні цикли оформляються за допомогою директиви PARALLEL, у якій також вказуються редукційні операції, операції відновлення тіньових граней (тіньові грані – це розширення локальної частини масиву для організації доступу до розташованих на сусідніх обчислювальних обладнаннях елементів цього масиву). Витки паралельних циклів розділяються між обчислювальними обладнаннями відповідно до правила відображення витків, заданим у директиві PARALLEL.

Паралельні цикли розташовані в обчислювальних регіонах (директиви REGION і END REGION), що приводить до підготовки компілятором виконання цих циклів з використанням графічних процесорів.

У показаному прикладі використовуються директиви актуалізації (GET_ACTUAL) і оголошення актуальності (ACTUAL) для сполучення регіонів із позарегіоним простором. Актуалізація використана для змінних EPS і *B* перед висновком і використанням у позарегіоному просторі, а оголошення актуальності використане для змінної EPS після її модифікації у позарегіоному просторі.

Ця програма може бути виконана на кластері із графічними процесорами, причому між прискорювачами і багатоядерними центральними процесорами будуть розподілені дані і обчислення.

1.5 Обґрунтування задач дослідження

Поява багатопроцесорних ЕОМ з розподіленою пам'яттю значно розширило можливості рішення більших завдань, однак різко ускладнило розробку програм. Програміст повинен розподілити дані між процесорами, а програму представити у вигляді сукупності процесів, взаємодіючих між собою за допомогою обміну повідомленнями.

Необхідно запропонувати принципи і алгоритми відображення ДВМ-програм на кластер із прискорювачами, які дозволяють використовувати ДВМ-

мови з мінімальним їхнім розширенням. При цьому потрібно виходити з того, що у вузлах кластера будуть розміщатися прискорювачі різної архітектури.

Тому що цільовою машиною обраний кластер із багатоядерними процесорами і прискорювачами, то ставиться завдання розробити програмні засоби, що забезпечують використання для обчислень одночасно як ЦПУ, так і ГПУ. У використовуваних низькорівнених засобах програмування для ГПУ (і в нових високорівнених засобах теж) є складності для прикладного програміста, пов'язані з копіюванням даних на прискорювач і назад, тому ставиться завдання по можливості автоматизувати даний процес у розроблювальнім розширенні моделі ДВМ.

Необхідно розробити і впровадити в систему підтримки виконання ДВМ-програм алгоритм розподілу підзадач між вузлами кластера, що забезпечує балансування завантаження обчислювальних ресурсів. Алгоритм повинен виходити з наявності заданого часу виконання кожної підзадачі на різнім числі процесорів.

Розробити засоби налагодження, що дозволяють знаходити помилки програміста, допущені при розпаралелювані програми на кластер із прискорювачами, а також коректність обчислень, зроблених на графічному процесорі:

– розробити алгоритми розподілу підзадач між вузлами кластера, що дозволяють балансувати завантаження обчислювальних вузлів кластера при виконанні багатоблочних програм;

– розробити алгоритми автоматичного переміщення необхідних актуальних даних між пам'яттю ЦПУ і пам'ятями декількох ГПУ;

– створені засоби порівняльного налагодження ДВМ-Н-програм, що базуються на зіставленні результатів одночасного виконання на ЦПУ і на ГПУ тих самих фрагментів програми.

Проведені експерименти з тестами мають показати, що для класу завдань, при рішенні яких використовуються різницеві методи на статичних структурних

сітках, можна писати програми мовою Фортран-ДВМ-Н, які ефективно виконуються на кластерах із прискорювачами.

З використанням розроблених алгоритмів створена система підтримки виконання ДВМ-Н-програм, що є невід'ємною частиною компіляторів ДВМ-Н-програм. Розробка компілятора з мови Фортран-ДВМ-Н суттєво спростила створення ефективних програм для кластерів із прискорювачами, здатних автоматично настроюватися на цільову конфігурацію обчислювальної системи. Компілятор з мови Фортран-ДВМ-Н, що включає в себе систему підтримки виконання ДВМ-Н-Програм, входить до складу ДВМ-системи і використовується при вирішенні задач інженерії програмного забезпечення за технологіями паралельного програмування. З використанням цього компілятора був розпаралелений на кластер із прискорювачами ряд прикладних обчислювальних завдань.

2 ОПИС ПРОВЕДЕНИХ ТЕОРЕТИЧНИХ ДОСЛІДЖЕНЬ

2.1 Алгоритми розподілу ресурсів

У простому статичному режимі в кожному регіоні розподіл проводиться однаково. Користувачем задається вектор відносних продуктивність обчислювальних обладнань, наявних у кожному вузлі кластера (також вони можуть бути грубо визначені автоматично при старті програми), потім ці продуктивності накладаються на параметри міжпроцесного розподілу даних, який по кожному розподіленому виміру може бути розподілене як рівномірно, так і із заданим вектором ваг. У такому режимі зводяться до мінімуму переміщення даних, пов'язані з їхнім перерозподілом, але не враховується різне співвідношення продуктивності обчислювальних обладнань на різних фрагментах програми. У цьому режимі використовуються (при наявності декількох) оброблювачі по-умовчанню, а оптимізаційні параметри одержують наступні значення:

– кількість використовуваних ЦПУ та ниток – максимально доступне поточному процесу;

– метод планування розкладу ЦПУ та ниток – автоматичний (у термінах Openmp);

– розміри і форма CUDA-блоку ниток для кожного циклу вибираються виходячи з кількості вимірів паралельного циклу, кількості і місця розташування вимірів із залежностями, кількості використовуваних апаратних ресурсів на одну CUDA-нитку, кількості наявних апаратних ресурсів графічного процесора і способу їх розподілу по CUDA-ниткам;

– спосіб обробки редукційних операцій в CUDA-оброблювачі вибирається динамічно для кожного циклу виходячи з наявного обсягу глобальної пам'яті графічного процесора – або більш швидкий і більш вимогливий за обсягом пам'яті, або менш швидкий і менш вимогливий за обсягом пам'яті.

В цьому режимі в кожному обчислювальному регіоні розподіл даних і обчислень вибирається на основі історії, що постійно поповнюється, запусків даного регіону і його сусідів, що визначаються динамічно.

2.2 Динамічний режим з добором розподілу

Кожний обчислювальний регіон у даному режимі розглядається у вигляді декількох родинних варіантів запуску, кожний з яких визначається парою (обчислювальний регіон, відповідність даних), де під відповідністю даних розуміється відповідність використовуваних у вихідному коді обчислювального регіону локальних змінних реальним змінним (масивам або скалярам).

Для кожного варіанта запуску регіону визначаються:

– динамічно попередні варіанти запуску регіону, що є постачальниками актуальних вхідних даних, причому враховуються регіони, що і раніше закінчилися, які використовували актуалізовані дані тільки на читання. У якості постачальників актуальних даних також можуть бути директиви ACTUAL і GET_ACTUAL, прирівнювані до регіонів, що виконуються винятково на ЦПУ, що і мають порожнє тіло;

– залежність часу роботи від розподілу даних, причому ухвалюються в увагу всі варіанти запуску того самого обчислювального регіону з обліком їх різних відповідностей даних;

– кількість входжень.

Залежність часу роботи від розподілу даних будується у вигляді табличної функції часу від розподілів розподілених масивів, яка є сумою таких же табличних функцій, побудованих для паралельних циклів, послідовних ділянок і хост-секцій, що втримуються усередині даного варіанта запуску регіону.

Тому що кожний паралельний цикл розподіляється тільки по одній абстрактній машині (розподіленому масиву), те функція залежності часу роботи

паралельного циклу на обладнанні будується як функція одного речовинного аргументу – обсягу обчислень (з урахуванням ваг при розподілі), відданих конкретному обладнанню. Тому що для обробки паралельного циклу навіть для одного обладнання припустима наявність декількох оброблювачів, кожний зі своїми оптимізаційними параметрами, то для паралельних циклів є сімейство табличних функцій часу від даного обсягу обчислень, а підсумкова функція залежності часу від обсягу обчислень, відданих обладнанню для паралельного циклу будується, як мінімум серед відповідного сімейства.

Час обробки послідовних ділянок уважається постійним. Час виконання хост-секцій будується як сума постійної величини, витраченої на виконання операторів хост-секції з тимчасовими витратами, витраченими на GET_ACTUAL, наявні в даній хост-секції.

Для інтерполяції і екстраполяції значень побудовано функції застосовування моделей продуктивності пристрою, в залежності від ресурсу, що віданий пристрою, збільшується за логарифмічному закону.

У цьому режимі періодично включається побудова субоптимальної схеми розподілу даних у всіх варіантах запуску обчислювальних регіонів на основі накопичених відомостей у цілому для програми, аналізуючи цілком послідовність варіантів запуску регіонів і їх характеристики виконання. При побудові таких схем ураховуються як внутрішні показники варіантів запуску регіонів у вигляді залежності часу роботи від розподілу даних, так і послідовність виконання варіантів запуску обчислювальних регіонів з метою мінімізації в тому числі і тимчасових витрат на перерозподіл даних. Після побудови такої схеми, вона застосовується і відбувається подальше нагромадження характеристик і інформації про обчислювальні регіони і паралельних циклах.

У процесі роботи програми в даному режимі вирішуються наступні завдання:

– ідентифікація варіанта запуску регіону;

– збір інформації про потік даних між варіантами запуску регіонів, а також між варіантами запуску регіонів і позарегіональним простором;

– збір інформації про швидкодію обчислювальних обладнань на окремих паралельних циклах;

– побудова глобальної схеми розподілу даних і обчислень по обчислювачах, у якій для кожного варіанта запуску регіону обраний обчислювач.

Ідентифікація варіантів запуску регіону відбувається при вході в обчислювальний регіон за наступною схемою:

– якщо попередній (динамічно попередній) варіант запуску даного регіону був унікальним і разом з тим частина з використовуваних їм змінних нині не існує, то відбувається відновлення інформації про цей варіант запуску (позначка нині не існуючих даних, як локальних), а також ототожнення зі співпадаючим по відповідності даних варіантом запуску (якщо такий є);

– фіксується порядок реєстрації використовуваних у ньому змінних, тобто формується впорядкований список посилань на фактичні змінні разом із зазначеними вирізками для масивів;

– по сімейству родинних варіантів запуску, тобто варіантів запуску одного регіону, проводиться пошук по повному збігові відповідності даних. Якщо збіг знайдений, то поточне виконання регіону зізнається приналежним знайденому варіанту запуску. Інакше створюється новий варіант запуску регіону і поточне виконання регіону зізнається приналежної знову створеному унікальному варіанту запуску.

Збір інформації про потік даних між варіантами запуску обчислювальних регіонів відбувається при вході в обчислювальний регіон відразу після ідентифікації поточного його варіанта запуску в такий спосіб:

– по кожній частині (у випадку масиву) використовуваної на читання змінної визначається джерело її актуального значення – варіант запуску регіону, у якім дана частина змінної була вихідною (або операція оголошення актуальності);

– по кожній частині всіх використовуваних на читання змінних ведеться підрахунок – скільки раз мав місце той або інше джерело її актуального значення.

Збір інформації про швидкодію обчислювачів для кожного варіанта запуску регіону проводиться в такий спосіб:

– регіон запускається на тому обчислювачі, на якім поточний варіант запуску регіону ще не був запущений, якщо такий є;

– якщо такого не є, регіон запускається на тому обчислювачі, на якім поточний варіант запуску виконується найбільше ефективно;

– для кожного паралельного циклу здійснюється вимір швидкодії обраного обчислювача на ньому, а також запам'ятовується інформація про відображення кожного паралельного циклу;

– після завершення виконання регіону відбувається запис отриманих швидкодій з деталізацією по кожному паралельному циклу і послідовній ділянці.

– враховується кількість виконань варіанта запуску регіону, а також середнє значення і середньоквадратичне відхилення для значень ефективності по кожному обчислювачу.

Побудова глобальної схеми розподілу обчислень по обчислювачах, у якій для кожного варіанта запуску регіону обраний обчислювач проводиться у два етапи:

– побудова допоміжного графа;

– вибір обчислювачів для варіантів запуску регіонів по допоміжному графі жадібним алгоритмом.

Побудований граф складається з вершин, які при початковій побудові відповідають варіантам запуску регіонів і ребер, які відповідають зв'язкам за даними між варіантами запуску регіонів. Кожна вершина містить інформацію про сумарний час виконання цієї вершини на кожному з обладнань, що включає відомі на даний момент тимчасові витрати на переміщення даних для забезпечення роботи даної вершини і відомі на даний момент тимчасові витрати на переміщення вихідних даних даної вершини.

Кожне ребро містить інформацію про сумарні тимчасові витрати на переміщення даних уздовж даного ребра для кожної пари обладнань за умови, що ця пара обладнань обрана для виконання інцидентних даному ребру вершин.

Вибір обчислювачів для варіантів запуску регіонів по допоміжному графі жадібним алгоритмом складається із циклічного виконання наступних кроків:

– визначення ваги кожної вершини, як максимальний можливий час її виконання;

– визначення ваги кожного ребра, як максимальні можливі тимчасові витрати на переміщення даних уздовж даного ребра;

– знаходження серед усіх вершин і ребер того об'єкта, чия вага максимальна;

– якщо була знайдена вершина, то даній вершині приписується те обладнання, на якім час її виконання мінімально (відповідно це обладнання вибирається для того набору варіантів запуску регіонів, якому відповідає дана вершина), вершина вилучається із графа. Ті ребра, які були інциденті вилученій вершині зливаються з вершинами, інцидентними їм, що і залишилися в графові за правилом: новий час виконання на обладнанні ухвалюється збільшеним на величину тимчасових витрат на переміщення даних з/на те обладнання, яке було обрано для вилученої вершини;

– якщо було знайдене ребро, то дане ребро вилучається із графа, а інциденті йому вершини зливаються разом, при цьому даній новій вершині відповідає об'єднання наборів варіантів запуску регіонів для, що зливаються вершин. При цьому не допускається утвір кратних ребер – вони також зливаються разом.

У результаті роботи даного алгоритму граф стає порожнім, а кожному варіанту запуску регіону приписується обчислювач. При цьому на останньому кроці роботи алгоритму виходить оцінка часу роботи всієї програми при обраній схемі розподілу.

Крім розподілу даних, зазнає добору також вибір оброблювача для кожного паралельного циклу для кожного обладнання і оптимізаційні параметри оброблювачів такі, як кількість ЦПУ та ниток для ЦПУ-обробників і розмір і форма CUDA-блоку ниток для CUDA-оброблювачів.

Є можливість запису побудованих схем розподілу у файл для використання в наступних запусках програми як у цьому режимі, так і в третьому режимі. Також у якості початкового наближення може бути використаний вектор

продуктивності обчислювальних обладнань у тому ж виді, як і для простого статичного режиму.

З даного режиму можливий перехід у третій режим у будь-якій крапці виконання програми, що забезпечує спрощену схему добору і використання схеми розподілу без необхідності збереження параметрів у файл і повторного запуску програми.

Компілятор з мови Фортран-ДВМ-Н складається із блоку аналізу програми, блоку конвертації, системи підтримки виконання ДВМ-Н-програм (бібліотека LibДВМ-Н).

Аналізатор будує структуру вихідної програми і доповнює задані програмістом Вказівника про режими використання даних, для яких є правила автоматичного визначення.

Конвертер перетворить вихідну програму в еквівалентну їй пари програм, які опираються на систему підтримки виконання ДВМ-Н-програм і вже можуть бути скомпільовані в машинний код стандартними компіляторами (див. рис.2.1).

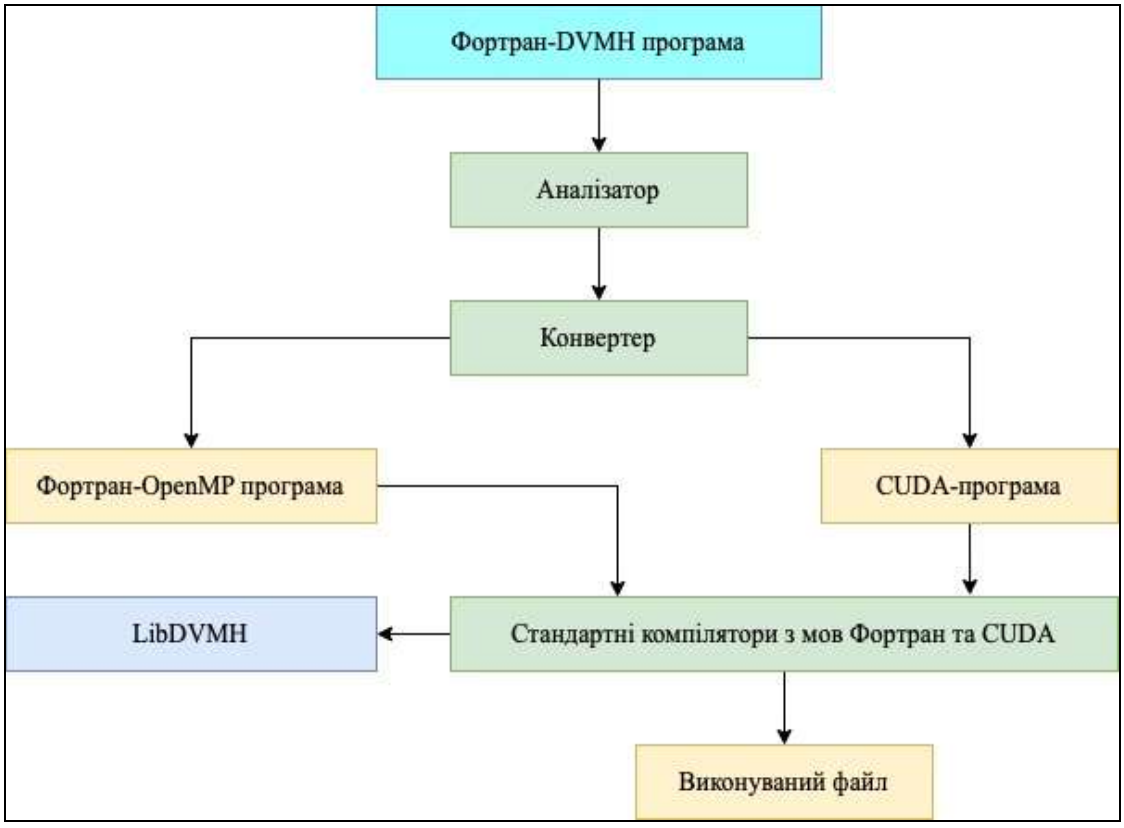

Рисунок 2.1 – Схема роботи компілятора з мови Фортран-ДВМ-Н

LibDVM-H є бібліотекою функцій, яка забезпечує виконання ДВМ-Н-програм. При обробці більшості директив конвертер генерує один або кілька викликів процедур LibDVM-H з параметрами, що відбивають Вказівника користувача в даній директиві.

2.3 Динамічний режим з використанням підібраної схеми розподілу

При компіляції обчислювальних регіонів відбувається їхній поділ на складові частини – паралельні гнізда циклів і послідовні оператори. Ці частини виділяються в окремі підпрограми з використанням відповідних технологій програмування: для ГПУ – CUDA, для ЦПУ – OpenMP.

У динамічному режимі з використанням підібраної схеми розподілу в кожному обчислювальному регіоні розподіл вибирається на основі наданої схеми розподілу, побудованої при роботі програми в другому режимі, причому є можливість як перейти в цей режим безпосередньо із другого, так і використовувати схему розподілу з файлу, отриманого в результаті роботи програми в другому режимі.

У даному режимі працює той же алгоритм ідентифікації варіанта запуску регіону, що і у другому режимі. Це тягне наступні наслідки:

– при використанні схеми розподілу з файлу не гарантується її коректне використання у випадку, якщо параметри програми були змінені, особливо це стосується тих параметрів, які впливають на шлях виконання програми (вибір іншого методу розрахунку, відключення або включення етапів розрахунку);

– перше виконання варіанта запуску регіону при деяких умовах може відбуватися повільніше, чим усі інші. Це пов'язане з апостеріорним ототожненням варіантів запуску регіонів, що працюють із локальними змінними.

Однією з переваг ДВМ-системи є те, що одержувані паралельні програми можуть настроюватися при запуску на кількість виділених для них процесорів.

Така властивість програм дозволяє запускати їх на довільнім числі процесорів, компонувати складні програми з наявних простих програм, підвищити ефективність використання паралельних систем колективного користування за рахунок більш гнучкого розподілу процесорів між окремими програмами. Цією властивістю не мають ДВМ-програми, що використовують механізм підзадач.

Розподіл підзадач по процесорах проводиться вручну і воно найчастіше утруднене внаслідок:

– великої кількості під задач;

– помітного розкиду їх складності;

– необхідності здійснювати розподіл на різну кількість процесорів.

За M позначено кількість процесорів, за N – кількість підзадач. Для кожної підзадачі відомо:

– мінімальна кількість процесорів, яка може бути використане для рахунку підзадачі K_{min};

– максимальна кількість процесорів, яка може бути використане для рахунку підзадачі K_{max};

– функція залежності часу виконання від кількості процесорів time(k) > 0 така, що для всіх k з діапазону $[K_{min}, K_{max - 1}]$ виконане: time(k) * k <= time(k + 1) * (k + 1)

Потрібно скласти оптимальне – з мінімальним фінішним часом – розклад проходження підзадач, де для кожної підзадачі буде зазначений стартовий час, група процесорів для її рахунку. Підзадачі не можуть уважатися одночасно використовуючи той самий процесор. Для всіх процесорів із групи, призначеної для рахунку підзадачі, часи старту рахунку цієї підзадачі збігаються, так само як і часи рахунку (тривалість) підзадачі – синхронне виконання однієї підзадачі.

Як відомо, завдання складання багатопроцесорного розкладу Np-повна, а, виходить, досліджуване завдання Np-важка, тому що є узагальненням Np-повного завдання.

Для більш традиційної постановки, у якій для кожної підзадачі необхідно виділити тільки один процесор, є безліч евристичних алгоритмів. Однак у

літературі не вдалося знайти алгоритм автоматичного розподілу процесорів між підзадачами, що вимагають для свого виконання не одного, а декількох процесорів, що і допускають виконання на різній кількості процесорів.

Основними функціями системи підтримки виконання ДВМ-Н-програм є:

– відображення абстрактної паралельної машини (ідеальної машини, найбільш підходящої для виконання програми) на задану при запуску ґрати процесів і наявний у кожному вузлі набір обчислювальних обладнань;

– створення і знищення розподіленого масиву;

– відображення розподіленого масиву на абстрактну паралельну машину;

– відображення паралельного циклу і паралельних завдань на абстрактну паралельну машину;

– створення і завантаження буферів для доступу до вилучених даних;

– підготовку і організацію паралельного виконання ділянок програми на різнорідних обчислювальних обладнаннях;

– балансування завантаження вузлів кластера і обчислювальних обладнань кожного вузла (статичну і динамічну);

– динамічний добір значень оптимізаційних параметрів;

– керування поточним станом і місцезнаходженням даних, їхніми переміщеннями;

– виконання редукційних операцій;

– відновлення тіньових граней розподілених масивів (тіньові грані – це розширення локальної частини масиву для організації доступу до розташованих на сусідніх обчислювальних обладнаннях елементів цього масиву);

– можливості для функціонального порівняльного налагодження коректності роботи на прискорювачах;

– можливості для налагодження продуктивності.

2.4 Алгоритми обліку актуального стану змінних

При програмуванні графічних процесорів застосовується методика повного їм керування з боку ЦПУ. Це означає, що програміст у першу чергу пише програму для ЦПУ, з якої вже використовує графічний процесор.

Дана ситуація серйозно ускладнюється тим, що графічний процесор не має прямого доступу в основну пам'ять ЦПУ, а розташовує своєю власною швидкодіючою широкосмуговою пам'яттю. Це приводить до того, що програмістові необхідно перед запуском обчислень на ГПУ завантажити використовувані дані на ГПУ, а після проведення обчислень вивантажити результати роботи із ГПУ.

Тому що операції завантаження з пам'яті ЦПУ на пам’ять ГПУ і вивантаження з пам'яті ГПУ на пам’ять ЦПУ повільні (приблизно в 100 разів повільніше доступу до пам’яті ГПУ із ГПУ), то програмістові доводиться застосовувати іншу схему роботи із ГПУ, при якій дані по можливості не вивантажуються, а залишаються в пам'яті ГПУ до наступного їхнього використання.

Є як мінімум два способи керування переміщеннями даних на прискорювач і назад:

– ручне копіювання;

– вказівка вхідних і вихідних даних для фрагментів програми.

При ручнім копіюванні програміст використовує команди копіювання, у яких указує адреси в пам'яті ГПУ і пам'яті ЦПУ, кількість байт, яке необхідно скопіювати і напрямок копіювання (на ГПУ або із ГПУ).

Такий метод є у всіх низькорівнених засобах програмування ГПУ, тому що він є базовим для передачі даних на, що підключається обладнання і одержання даних назад. При використанні даного способу програміст самостійно стежить за тим, де які зміни були зроблені для того, щоб у потрібні моменти часу робити копіювання на прискорювач або назад. Даний спосіб дає максимальну гнучкість

при роботі із прискорювачами, але має і серйозні недоліки, про яких згадується нижче.

При вказівці вхідних і вихідних даних для фрагментів програми способі керування переміщеннями даних, програміст указує вхідні і вихідні дані для фрагментів програми. Кожний такий фрагмент виконується або на ЦПУ, або на ГПУ. При цьому відновлення вхідних даних перед виконанням кожного фрагмента проводяться на підставі інформації про те, на якому обладнанні був виконаний останній фрагмент, для якого ці дані були вихідними.

Даний спосіб дає менший ступінь гнучкості при роботі із прискорювачами, наприклад, у частині сполучення операцій копіювання з обчисленнями.

Ручне керування програмістом усіма переміщеннями даних має ряд недоліків:

– програмістові доводиться бути чітко орієнтованим на те, скільки і яких прискорювачів використовується в програмі, а також чи передбачається одночасне використання ЦПУ для додаткового поділу обчислень. Це може приводити до втрати її універсальності в плані використовуваної цільовий ЕОМ, або ж значному ускладненню з метою підтримки різних конфігурацій устаткування;

– для програм з досить великим ступенем розгалуженості, а також програм, що мають багаторазово виконувані частини, причому при різних вхідних даних і різних шляхах виконання, серйознішає проблемою оптимізація переміщень даних, що може приводити як до зайвих переміщень (у випадку якщо програміст не приклав досить зусиль для оптимізації або навмисно пішов на такі витрати для більшої стійкості програми до подальших модифікацій), так і до помилок, що складно перебувають, проявляються тільки при деяких наборах вхідних даних;

– якщо програміст пише програму, здатну виконуватися на різнім числі вузлів кластера (а саме такими програмами є ДВМ-програми), то забезпечити оптимальне відображення обчислень на багатоядерні процесори і прискорювачі усередині кожного вузла – це завдання практично непосильне для програміста і повинна зважуватися на системному рівні.

Використання методу керування переміщеннями даних, заснованого на завданні вхідних і вихідних даних регіонів, і, як наслідок, повне інформування системи підтримки виконання програм про виконувані модифікації змінних, дозволяє позбутися недоліків повністю ручного керування переміщеннями даних, а також додає корисні можливості такі, як:

– порівняльне функціональне налагодження (виконання того самого регіону з тими самими вихідними даними з використанням ЦПУ і прискорювачів, а потім порівняння значень отриманих вихідних даних);

– багаторазове виконання регіонів з тими самими вихідними даними з метою пошуку значень оптимізаційних параметрів (параметрів, що не впливають на коректність виконання ділянки програми, але, що впливають на час його виконання).

Розроблені алгоритми ґрунтуються на наступних поняттях:

– змінна – скаляр або масив із заданою кількістю вимірів, початковим і кінцевим індексом по кожному виміру, розміром (у байтах) одного елемента;

– представник деякої змінної на деякому обладнанні – екземпляр (можливо, неповний) змінної, розміщений у пам'яті обладнання;

– стан актуальності представника – завдання для всіх елементів представника, чи є вони (елементи) актуальними, тобто чи мають вони останнє привласнене цьому елементу значення;

Для маніпуляцій зі станами актуальності представників змінних уводиться особливий вид функцій – `Pcsn` – як відображення з A_n в N U {0,+inf}.

Нижче перераховані базові операції над ними, де I(a) – індикаторна функція, що ухвалює значення 1 при дійснім вираженні, даному в якості аргументу a; 0 – а якщо ні, то:

– об'єднання двох PCS p1 і p2 називається PCS p3 таке, що p3(x) = max(p1(x), p2(x))

– різниця двох PCS. p1 і p2 називається PCS p3 таке, що p3(x) = I(p1(x)>p2(x))p1(x);

– зниження одного PCS p1 на рівень іншого PCS p2 називається PCS p3 таке,

що p3(x) = I(p1(x)<+inf або p2(x)=0)p1(x) + I(p1(x)=+inf і p2(x)>0)p2(x);

– горизонтальна різниця двох PCS p1 і p2 називається PCS p3 таке, що p3(x) = I(p1(x)!=p2(x))p1(x);

– перетинання двох PCS. p1 і p2 називається PCS p3 таке, що p3(x) = min(p1(x), p2(x));

– перетинання двох PCS p1 і p2 з підвищенням називається PCS p3 таке, що p3(x) = I(0<p1(x)<=p2(x))p2(x)

У процесі роботи ДВМ-Н-програми система підтримки виконання стежить за станом актуальності представників змінних на всіх використовуваних обладнаннях, модифікує його відповідно до вказівок напрямку використання даних в обчислювальних регіонах тіньовими обмінами, що відбуваються, вказівками директив актуалізації і іншими, що відбуваються подіями, що зачіпають дані користувача.

Кожному представникові масиву приписується його стан актуальності у вигляді `Pcsn`, де n – кількість вимірів масиву, значення якого трактуються як ступінь актуальності елементів масиву в представнику, де нуль означає неактуальне значення, `+inf` – актуальне значення, а натуральні значення – деякий проміжний стан, який буде трактуватися як актуальний або неактуальний залежно від контексту. Також елемент масиву, що має ступінь актуальності +inf називається абсолютно актуальним.

З урахуванням схеми роботи з тіньовими гранями, при якій читання з них може відбуватися в будь-якій крапці програми, а відновлення відбувається тільки по спеціальних вказівках користувача, для повної підтримки динамічного режиму поділу роботи між обчислювальними обладнаннями вузла, при якім локальна для даного обладнання частина масиву може змінюватися від регіону до регіону, уведений диференційований ступінь актуальності для елементів масиву.

Для кожного масиву заводиться так званий профіль його тіньових граней. Профіль являє собою Pcsn, для якого вірно, що:

– він дорівнює нулю поза множиною $[-L_1:H_1]x[-L_2:H_2]x...x[-Ln:Hn]$, де Li – ширина нижньої тіньової грані по *i*-ому виміру, а Hi – ширина верхньої тіньової

грані по *i*-му виміру;

– він рівний +inf у крапці (0,0,…,0) і тільки в ній.

Профіль задає вимоги на актуальність елементів тіньових граней цього масиву, прийняті їм значення будемо називати необхідною ступенем актуальності.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

3.1 Алгоритми обліку стану актуальності змінних

Для кожного масиву ведеться лічильник максимального необхідного ступеня актуальності для елементів тіньових граней. Під час роботи програми значення цього лічильника дорівнює максимуму профілю тіньових граней за межами точки (0,0,…,0).

На початку програми локальна частина масиву і його тіньові грані в пам'яті ЦПУ вважаються абсолютно актуальними (ступінь актуальності рівний +inf). Представники на всіх інших обладнаннях уважаються повністю неактуальними (ступінь актуальності рівний 0). Лічильник поточного максимального необхідного ступеня актуальності елементів тіньових граней
для всіх змінних установлюється в одиницю. Профілі тіньових граней ухвалюють значення 1 для всіх елементів тіньових граней.

Усяке виконання вказівок `SHADOW_RENEW` і `SHADOW_COMPUTE` крім іншого збільшує на одиницю лічильник поточному максимальному ступеня актуальності зазначених у директиві змінних-масивів і модифікує їхній профіль тіньових граней, міняючи його в частині зазначених у директиві тіньових граней шляхом завдання їм необхідному ступеня актуальності в поточне значення максимального ступеня актуальності. Також вони приводять до модифікації стану актуальності представників, тому що проводиться оголошення актуальності: для SHADOW_RENEW – у пам'яті ЦПУ, а для SHADOW_COMPUTE – на тому обладнанні, на якому виконується відповідний цикл.

Якщо SHADOW_RENEW відбувається в обчислювальному регіоні, то також проводиться позначка оновленого в пам'яті ЦПУ масиву для його наступного відновлення на обладнаннях регіону перед виконанням наступного паралельного циклу (або послідовної ділянки) даного регіону. При цьому оновлені на обладнаннях регіону будуть тіньові грані, що тільки змінилися (як

наслідок стану, що змінився, актуальності і вимог до ступеня актуальності тіньових елементів).

Усяка актуалізація для представника змінної робиться з урахуванням її профілю тіньових граней. Це означає, що запитувані вимоги актуальності попередньо перетинаються із профілем тіньових граней, розтягнутим на поточну локальну частину. При цьому якщо проведення актуалізації проводиться в ході обробки обчислювального регіону, те локальна частина визначається відповідно до розподілу даних, зробленому при вході в обчислювальний регіон. Для актуалізацій за межами обчислювальних регіонів (а така актуалізація можлива тільки на згадку ЦПУ) використовується локальна частина поточного процесу цілком.

Під розтягуванням профілю тіньових граней мається на увазі формування вимог актуальності *D* на основі профілю тіньових граней *P* за правилом:

$$D(i_1,i_2,\ldots,in)=P(I(i_1<B_1)(i_{1\text{-}B1})+I(i_1>U_1)(i_{1\text{-}U1}),\ I(i_2<B_2)(i_{2\text{-}B2})+I(i_2>U_2)(i_{2\text{-}U2}),\ \ldots,$$
$$I(in<Bn)(in\text{-}bn)+I(in>Un)(in\text{-}un)),$$

де B_i і U_i є нижніми і верхніми границями локальної частини відповідно (включно).

Далі приводяться алгоритми обліку актуального стану даних при різних ситуаціях.

При вході в обчислювальний регіон системі підтримки виконання повідомляється про кожну змінну напрямки її використання для її підмасивів. Це робиться або як результат вказівок користувача, або як результат аналізу компілятором. Для кожного масиву задається відображення з безлічі індексів масиву в безліч {0, IN, OUT, INOUT, LOCAL, INLOCAL}, де: 0 означає, що даний елемент не використовується в регіоні; IN – регіону потрібно актуальне значення і даний елемент може використовуватися тільки на читання; OUT – регіону не потрібно актуальне значення, даний елемент може використовуватися як на читання, так і на запис, причому нове значення може бути використане далі;

INOUT – регіону потрібно актуальне значення, даний елемент може використовуватися як на читання, так і на запис, причому нове значення може бути використане далі; LOCAL – регіону не потрібно актуальне значення, даний елемент може використовуватися як на читання, так і на запис, причому нове значення не може бути використане далі; INLOCAL – регіону потрібно актуальне значення, даний елемент може використовуватися як на читання, так і на запис, причому нове значення не може бути використане далі.

Після визначення розподілу даних по обладнаннях (визначення локальних частин кожного масиву на кожнім обладнанні), система підтримки виконання для кожної використовуваної в регіоні змінної виконує наступні дії:

Для кожного використовуваного обладнання:

– визначення частини змінної, яка повинна перебувати на обладнанні, як локальна частина + тіньові грані;

– якщо на ньому не перебуває необхідна частина змінної , то виділення необхідної кількості пам'яті та перепис актуальних значень із попереднього місця розташування;

– звільнення пам'яті, виділеної під, що раніше перебував на обладнанні частина змінної.

Одержання запитаних (IN, INOUT, INLOCAL частини змінної) актуальних даних з інших обладнань, причому підлягають актуалізації в тому числі і тіньові грані з урахуванням застосування профілю тіньових граней.

Зниження актуальності для OUT, INOUT, LOCAL, INLOCAL частин змінної на всіх обладнаннях.

Для кожного використовуваного обладнання:

– оголошення актуальності для OUT, INOUT, LOCAL, INLOCAL частин змінної на данім обладнанні, причому підлягають оголошенню актуальності тільки локальні для даного обчислювального регіону частини представників;

– обробка запиту актуалізації

Запит актуалізації – це одна з основних операцій, яка в узагальненому виді лежить в основі майже всіх маніпуляцій з даними. Вона для заданих елементів

змінної і для заданого обладнання обновляє ті елементи, які не є в необхідному ступені актуальними, знаходячи в достатньому ступені актуальні елементи змінної на інших обладнаннях.

Алгоритм можна представити в такий спосіб (p – запитуваний ступінь актуальності для елементів, є PCS; u – обладнання, на яке проводиться актуалізація).

Крок 1 – привласнити в p2 різницю p і стану актуальності представника на обладнанні u.

Крок 2 – для кожного обладнання d, відмінного від u:

– привласнити в p1 перетинання з підвищенням p2 і стану актуальності представника на обладнанні d;

– скопіювати з обладнання d на обладнання u усі елементи x, для яких p1(x)>0;

– відняти з p2 p1;

– оновити стан актуальності на обладнанні u, зробивши об'єднання його з p1.

3.2 Алгоритм обробки вказівників

Вказівка SHADOW_RENEW вказується програмістом з метою відновлення тіньових граней масиву. При цьому спочатку випливає актуалізувати елементи, що пересилаються з даного процесу, масиву, а потім урахувати відновлення отриманих від сусідніх процесів елементів. Нижче приводиться алгоритм дій, вироблених безпосередньо до посилки і приймання даних між процесами:

– актуалізувати на пам’яті ЦПУ граничні елементи локальної частини процесу;

– оголосити актуальність для тіньових елементів (щодо локальної частини процесу) у пам'яті ЦПУ;

– обновити профіль тіньових граней, зробивши об'єднання старого профілю

тіньових граней з PCS, що є характеристичною функцією оновлених тіньових елементів, помноженої на збільшений на одиницю лічильник поточному максимальному необхідному ступеня актуальності для тіньових елементів змін-масиву;

– якщо дані дії проводяться в ході роботи регіону, то позначити масив для наступного відновлення перед наступним циклом або послідовною ділянкою. При цьому будуть також оновлені тіньові елементи, що перебувають на іншому обладнанні поточного процесу.

Вказівка `REMOTE_ACCESS` вказується програмістом з метою одержання ефективного доступу до вилучених елементів, що не є в загальному випадку елементами тіньових граней. При вказівці `REMOTE_ACCESS` відбувається:

Актуалізація на пам’яті ЦПУ даних поточного процесу, до яких буде здійснюватися дистанційний доступ.

Створення спеціального буфера вилучених елементів.

Збір із усіх процесів у нього запитаних даних.

Якщо дані дії проводяться в ході роботи регіону, то:

– створення представників буфера вилучених елементів на прискорювачах, що використовуються даним регіоном;

– актуалізація всіх знову створених представників на обладнаннях регіону;

– після закінчення роботи з буфером вилучених елементів знищення представників на прискорювачах.

Вказівник `CONSISTENT` вказується програмістом з метою об'єднання записів у розмножений масив, зроблених у паралельному циклі. При обробці вказівки CONSISTENT у ході виконання регіону відбувається:

– визначення тієї частини розмноженого масиву, яка була записана поточним процесом;

– актуалізація на згадку ЦПУ записаної частини розмноженого масиву;

– оголошення актуальності в пам'яті ЦПУ всього розмноженого масиву цілком. Дана дія проводиться з обліком того, що інші частини масиву, що приводиться в консистентний стан, будуть отримані з інших процесів.

3.3 Алгоритми розподілу даних і обчислень

Одним з важливих аспектів функціонування такої програмної моделі, як ДВМ-Н є питання відображення вихідної програми на всі рівні паралелізму і різнорідні обчислювальні обладнання. Важливими завданнями механізму відображення є забезпечення коректного виконання всіх підтримуваних мовою конструкцій на різнорідних обчислювальних обладнаннях, балансування навантаження між обчислювальними обладнаннями, а також вибір оптимального способу виконання кожної ділянки коду на тому або іншому обладнанні.

Паралелізм в ДВМ-Н-програмах проявляється на декількох рівнях:

– розподіл даних і обчислень по Mpi-процесам. Цей рівень задається спеціальними директивами розподілу або перерозподілу даних і специфікаціями паралельних підзадач і циклів;

– розподіл даних і обчислень по обчислювальних обладнаннях при вході в обчислювальний регіон;

– паралельна обробка в рамках конкретного обчислювального обладнання. Цей рівень з'являється при вході в паралельний цикл, що перебуває усередині обчислювального регіону.

Наявність цих рівнів дає можливість органічно відобразити програму на кластер із багатоядерними процесорами і прискорювачами у вузлах.

В ДВМ-Н-програмах усі дані, що розподіляються, розподіляються за блоками: кожний розподілений вимір ділиться декількома точками на відрізки. При цьому є можливість по кожному виміру розподіленого масиву задавати або рівномірний блоковий розподіл, або розподіл зваженими блоками, тобто з урахуванням заданого вектора ваг. Ці вказівки безпосередньо виконуються при розподілі даних між Mpi-процесами. Вкладені розподіли, що з'являються при вході в обчислювальний регіон, будуються з використанням цієї інформації, але, у силу різнорідності обчислювальних обладнань, можуть мати, що відрізняється від зовнішньої схему розподілу.

Системою підтримки виконання ДВМ-Н-програм підтримуються три режими розподілу даних і обчислень по обчислювальних обладнаннях у крапках входу в регіони:

– простий статичний режим;

– динамічний режим з добором схеми розподілу;

– динамічний режим з використанням підібраної схеми розподілу.

У простому статичному режимі в кожному регіоні розподіл проводиться однаково. Користувачем задається вектор відносних продуктивність обчислювальних обладнань, наявних у кожному вузлі кластера (також вони можуть бути грубо визначені автоматично при старті програми), потім ці продуктивності накладаються на параметри міжпроцесного розподілу даних, який по кожному розподіленому виміру може бути розподілений як рівномірно, так і із заданим вектором ваг.

У такому режимі зводяться до мінімуму переміщення даних, пов'язані з їхнім перерозподілом, але не враховується різне співвідношення продуктивності обчислювальних обладнань на різних фрагментах програми. У цьому режимі використовуються (при наявності декількох) оброблювачі по-умовчанню, а оптимізаційні параметри одержують наступні значення:

– кількість використовуваних ЦПУ та ниток – максимально доступне поточному процесу;

– метод планування розкладу ЦПУ та ниток – автоматичний (у термінах Openmp);

– розміри і форма Cuda-блоку ниток для кожного циклу вибираються виходячи з кількості вимірів паралельного циклу, кількості і місця розташування вимірів із залежностями, кількості використовуваних апаратних ресурсів на одну Cuda-Нитку, кількості наявних апаратних ресурсів графічного процесора і способу їх розподілу по Cuda-ниткам;

– спосіб обробки редукційних операцій в Cuda-оброблювачі вибирається динамічно для кожного циклу виходячи з наявного обсягу глобальної пам'яті графічного процесора – або більш швидкий і більш вимогливий за обсягом

пам'яті, або менш швидкий і менш вимогливий за обсягом пам'яті.

В динамічному режим з добором розподілу в кожному обчислювальному регіоні розподіл даних і обчислень вибирається на основі історії, що постійно поповнюється, запусків даного регіону і його сусідів, що визначаються динамічно.

Кожний обчислювальний регіон у даному режимі розглядається у вигляді декількох родинних варіантів запуску, кожний з яких визначається парою (обчислювальний регіон, відповідність даних), де під відповідністю даних розуміється відповідність використовуваних у вихідному коді обчислювального регіону локальних змінних реальним змінним (масивам або скалярам).

Для кожного варіанта запуску регіону визначаються:

– динамічно попередні варіанти запуску регіону, що є постачальниками актуальних вхідних даних, причому враховуються регіони, що і раніше закінчилися, які використовували актуалізовані дані тільки на читання. У якості постачальників актуальних даних також можуть бути директиви ACTUAL і GET_ACTUAL, прирівнювані до регіонів, що виконуються винятково на ЦПУ, що і мають порожнє тіло;

– залежність часу роботи від розподілу даних, причому ухвалюються в увагу всі варіанти запуску того самого обчислювального регіону з обліком їх різних відповідностей даних;

– кількість входжень.

Залежність часу роботи від розподілу даних будується у вигляді табличної функції часу від розподілів розподілених масивів, яка є сумою таких же табличних функцій, побудованих для паралельних циклів, послідовних ділянок і хост-секцій, що втримуються усередині даного варіанта запуску регіону.

Тому що кожний паралельний цикл розподіляється тільки по одній абстрактній машині (розподіленому масиву), то функція залежності часу роботи паралельного циклу на обладнанні будується як функція одного речовинного аргументу – обсягу обчислень (з урахуванням ваг при розподілі), відданих конкретному обладнанню.

Для обробки паралельного циклу навіть для одного обладнання припустима наявність декількох оброблювачів, кожний зі своїми оптимізаційними параметрами, то для паралельних циклів є сімейство табличних функцій часу від даного обсягу обчислень, а підсумкова функція залежності часу від обсягу обчислень, відданих обладнанню для паралельного циклу будується, як мінімум серед відповідного сімейства.

Час обробки послідовних ділянок уважається постійним. Час виконання хост-секцій будується як сума постійної величини, що витрачена на виконання операторів хост-секції з часовими витратами, витраченими на GET_ACTUAL, наявні в даній хост-секції.

Для інтерполяції і екстраполяції значень побудовані функції застосовування моделі швидкодії, що логарифмічно збільшується, пристроїв.

У цьому режимі періодично включається побудова субоптимальної схеми розподілу даних у всіх варіантах запуску обчислювальних регіонів на основі накопичених відомостей у цілому для програми, аналізуючи цілком послідовність варіантів запуску регіонів і їх характеристики виконання. При побудові таких схем ураховуються як внутрішні показники варіантів запуску регіонів у вигляді залежності часу роботи від розподілу даних, так і послідовність виконання варіантів запуску обчислювальних регіонів з метою мінімізації в тому числі і тимчасових витрат на перерозподіл даних. Після побудови такої схеми, вона застосовується і відбувається подальше нагромадження характеристик і інформації про обчислювальні регіони і паралельних циклах.

У процесі роботи програми в даному режимі вирішуються наступні завдання:

– ідентифікація варіанта запуску регіону;

– збір інформації про потік даних між варіантами запуску регіонів, а також між варіантами запуску регіонів і позарегіональним простором;

– збір інформації про швидкодію обчислювальних обладнань на окремих паралельних циклах;

– побудова глобальної схеми розподілу даних і обчислень по обчислювачах,

у якій для кожного варіанта запуску регіону обраний обчислювач.

Ідентифікація варіантів запуску регіону відбувається при вході в обчислювальний регіон за наступною схемою.

Якщо попередній (динамічно попередній) варіант запуску даного регіону був унікальним і разом з тим частина з використовуваних їм змінних нині не існує, то відбувається відновлення інформації про цей варіант запуску (позначка нині не існуючих даних, як локальних), а також ототожнення зі співпадаючим по відповідності даних варіантом запуску (якщо такий є).

Фіксується порядок реєстрації використовуваних у ньому змінних, тобто формується впорядкований список посилань на фактичні змінні разом із зазначеними вирізками для масивів.

За сімейством родинних варіантів запуску, тобто варіантів запуску одного регіону, проводиться пошук по повному збігові відповідності даних. Якщо збіг знайдений, то поточне виконання регіону зізнається приналежним знайденому варіанту запуску. Інакше створюється новий варіант запуску регіону і поточне виконання регіону зізнається приналежної знову створеному унікальному варіанту запуску.

Збір інформації про потік даних між варіантами запуску обчислювальних регіонів відбувається при вході в обчислювальний регіон відразу після ідентифікації поточного його варіанта запуску в такий спосіб:

По кожній частині (у випадку масиву) використовуваної на читання змінної визначається джерело її актуального значення – варіант запуску регіону, у якім дана частина змінної була вихідною (або операція оголошення актуальності).

По кожній частині всіх використовуваних на читання змінних ведеться підрахунок – скільки раз мав місце той або інше джерело її актуального значення.

Збір інформації про швидкодію обчислювачів для кожного варіанта запуску регіону проводиться в такий спосіб:

Регіон запускається на тому обчислювачі, на якім поточний варіант запуску регіону ще не був запущений, якщо такий є.

Якщо такого не є, регіон запускається на тому обчислювачі, на якім

поточний варіант запуску виконується найбільше ефективно.

Для кожного паралельного циклу здійснюється вимір швидкодії обраного обчислювача на ньому, а також запам'ятовується інформація про відображення кожного паралельного циклу.

Після завершення виконання регіону відбувається запис отриманих швидкодій з деталізацією по кожному паралельному циклу і послідовній ділянці.

Уважається кількість виконань варіанта запуску регіону, а також середнє значення і середньоквадратичне відхилення для значень ефективності по кожному обчислювачу.

Побудова глобальної схеми розподілу обчислень по обчислювачах, у якій для кожного варіанта запуску регіону обраний обчислювач проводиться у два етапи:

– побудова допоміжного графа;

– вибір обчислювачів для варіантів запуску регіонів по допоміжному графі жадібним алгоритмом.

Побудований граф складається з вершин, які при початковій побудові відповідають варіантам запуску регіонів і ребер, які відповідають зв'язкам за даними між варіантами запуску регіонів. Кожна вершина містить інформацію про сумарний час виконання цієї вершини на кожному з обладнань, що включає відомі на даний момент тимчасові витрати на переміщення даних для забезпечення роботи даної вершини і відомі на даний момент тимчасові витрати на переміщення вихідних даних даної вершини.

Кожне ребро містить інформацію про сумарні тимчасові витрати на переміщення даних уздовж даного ребра для кожної пари обладнань за умови, що ця пара обладнань обрана для виконання інцидентних даному ребру вершин.

Вибір обчислювачів для варіантів запуску регіонів по допоміжному графі жадібним алгоритмом складається із циклічного виконання наступних кроків:

Визначення ваги кожної вершини, як максимальний можливий час її виконання.

Визначення ваги кожного ребра, як максимальні можливі тимчасові витрати

на переміщення даних уздовж даного ребра.

Алгоритм знаходження серед усіх вершин і ребер того об'єкта, чия вага максимальна – якщо була знайдена вершина, то даній вершині приписується те обладнання, на якому час її виконання мінімально (відповідно це обладнання вибирається для того набору варіантів запуску регіонів, якому відповідає дана вершина), вершина вилучається із графа. Ті ребра, які були інциденті вилученій вершині зливаються з вершинами, інцидентними їм, що і залишилися в графові за правилом: новий час виконання на обладнанні ухвалюється збільшеним на величину тимчасових витрат на переміщення даних з/на те обладнання, яке було обрано для вилученої вершини.

Якщо було знайдене ребро, то дане ребро вилучається із графа, а інциденті йому вершини зливаються разом, при цьому даній новій вершині відповідає об'єднання наборів варіантів запуску регіонів для, що зливаються вершин. При цьому не допускається утвір кратних ребер – вони також зливаються разом.

У результаті роботи даного алгоритму граф стає порожнім, а кожному варіанту запуску регіону приписується обчислювач. При цьому на останньому кроці роботи алгоритму виходить оцінка часу роботи всієї програми при обраній схемі розподілу.

Крім розподілу даних, зазнає добору також вибір оброблювача для кожного паралельного циклу для кожного обладнання і оптимізаційні параметри оброблювачів такі, як кількість ЦПУ та ниток для ЦПУ-обробників і розмір і форма Cuda-блоку ниток для Cuda-оброблювачів.

Є можливість запису побудованих схем розподілу у файл для використання в наступних запусках програми як у цьому режимі, так і в третьому режимі. Також у якості початкового наближення може бути використаний вектор продуктивності обчислювальних обладнань у тому ж виді, як і для простого статичного режиму.

З даного режиму можливий перехід у третій режим у будь-якій крапці виконання програми, що забезпечує спрощену схему добору і використання схеми

розподілу без необхідності збереження параметрів у файл і повторного запуску програми.

3.4 Динамічний режим з використанням схем розподілу

У динамічному режимі з використанням підібраної схеми розподілу в кожному обчислювальному регіоні розподіл вибирається на основі наданої схеми розподілу, побудованої при роботі програми в другому режимі, причому є можливість як перейти в цей режим безпосередньо із другого, так і використовувати схему розподілу з файлу, отриманого в результаті роботи програми в другому режимі.

У даному режимі працює той же алгоритм ідентифікації варіанта запуску регіону, що і у другому режимі. Це тягне наступні наслідки:

При використанні схеми розподілу з файлу не гарантується її коректне використання у випадку, якщо параметри програми були змінені, особливо це стосується тих параметрів, які впливають на шлях виконання програми (вибір іншого методу розрахунку, відключення або включення етапів розрахунку).

Перше виконання варіанта запуску регіону при деяких умовах може відбуватися повільніше, чим усі інші. Це пов'язане з апостеріорним ототожненням варіантів запуску регіонів, що працюють із локальними змінними.

Однією з переваг ДВМ-системи є те, що одержувані паралельні програми можуть настроюватися при запуску на кількість виділених для них процесорів. Така властивість програм дозволяє запускати їх на довільній кількості процесорів, компонувати складні програми з наявних простих програм, підвищити ефективність використання паралельних систем колективного користування за рахунок більш гнучкого розподілу процесорів між окремими програмами. Цією властивістю не мають ДВМ-програми, що використовують механізм підзадач.

Розподіл підзадач по процесорах проводиться вручну і він найчастіше утруднений внаслідок:

– великої кількості під задач;

– помітного розкиду їх складності;

– необхідності здійснювати розподіл на різну кількість процесорів.

Опис алгоритму. Якщо за M позначити кількість процесорів, за N – кількість підзадач, то для кожної підзадачі відомо:

– мінімальна кількість процесорів, яка може бути використане для рахунку підзадачі K_{min};

– максимальна кількість процесорів, яка може бути використана для рахунку підзадачі K_{max};

– функція залежності часу виконання від кількості процесорів time(k) > 0 така, що для всіх k з діапазону $[K_{min}, K_{max - 1}]$ виконане таким чином

$$time(k) * k <= time(k + 1) * (k + 1).$$

Потрібно скласти оптимальний – з мінімальним фінішним часом – розклад проходження підзадач, де для кожної підзадачі буде зазначений стартовий час, група процесорів для її розрахунку. Підзадачі не можуть бути такими, що одночасно використовують той самий процесор. Для всіх процесорів із групи, призначеної для розрахунку підзадачі, часи старту розрахунку цієї підзадачі збігаються, так само як і часи обчислення (тривалість) підзадачі – синхронне виконання однієї підзадачі.

Як відомо, завдання складання багатопроцесорного розкладу Np-повна, а, досліджуване завдання Np-важка, тому що є узагальненням Np-повного завдання.

Для більш традиційної постановки, у якій для кожної підзадачі необхідно виділити тільки один процесор, є безліч евристичних алгоритмів.

Однак у літературі не вдалося знайти алгоритм автоматичного розподілу процесорів між підзадачами, що вимагають для свого виконання не одного, а декількох процесорів, що і допускають виконання на різній кількості процесорів.

Для опису алгоритму вводяться кілька додаткових позначень:

– Proc(x, d) – множина усіх процесорів, вільних з моменту часу *x* і, як мінімум, до моменту (x + d);

– Proc(x) – відображення таке, що Proc(x)(d) = Proc(x, d) для всіх d і *x*;

– Procs(x) – безліч усіх процесорів, вільних у момент часу *x*;

– Tasks – множина усіх під задач;

– $K_{min}(t)$ – мінімальна кількість процесорів, яка може бути використана для розрахунку підзадачі *t*;

– $K_{max}(t)$ – максимальна кількість процесорів, яка може бути використана для розрахунку підзадачі *t*;

– time(t, k) – час розрахунку підзадачі *t* з використанням *k* процесорів.

Алгоритм розподілу підзадач можна описати в такий спосіб:

Крок 1: Покладемо trestmin – сума по всіх завданнях t з Tasks величин time(t, Kmin(t)) * Kmin(t).

Крок 2: Упорядкувати підзадачі по убуванню величини

$$\texttt{time(t, } K_{min}\texttt{(t))} \texttt{ * } K_{min}\texttt{(t)}$$

тобто списоку `sortedtasks`, де *t* приймає всі значення з Tasks.

Крок 3. Покладено t_{max} і $t_{occupied}$ рівними нулю.

Крок 4 – прибрати завдання *t* з початку списку sortedtasks.

Крок 5 – вилучити завдання *t* зі списку sortedtasks.

Крок 6: зменшити $t_{restmin}$ на величину time(t, K_{min}(t)) * K_{min}(t).

Крок 7: покладається множина notexamined рівною [K_{min}(t), K_{max}(t)]

Крок 8: для кожного моменту часу *x*, що є або початком відліку часу, або моментом зміни множини Procs(x) у порядку зростання виконувати:

– для кожної тривалості *d*, не меншої ніж time(t, k) для деякого *k*, у порядку убування виконувати для кожної кількості процесорів *k*, що належить notexamined і не більшому, ніж потужність Proc(x, d), а також такому, що time(t, k) <= d у порядку зростання виконувати наступне: покласти tsuggested(x, k) рівним максимуму із трьох величин: t_{max}, x + time(t, k), x + (toccupied + trestmin) / k;

– виключити з множини notexamined розглянуті в циклі кількості

процесорів;

– якщо множина notexamined порожня, то перейти до Кроку 9.

Крок 9. Нехай x0 – таке мінімальне, що мінімізує tsuggested(x_0, k) для деякого k.

Крок 10. Нехай k0 – мінімальне з таких, що мінімізують tsuggested(x_0, k_0).

Крок 11 – покласти t_{max} максимумом з t_{max} і x_0 + time(t, k_0).

Крок 12. Збільшимо toccupied на величину time(t, k_0) * k_0.

Крок 13. Зарезервувати групу з k_0 процесорів на час [x_0, x_0 + time(t, k_0)], виділивши підмножину з Proc(x_0, d) з таким максимальним d, що потужність Proc(x_0, d) не менш ніж k_0.

Крок 14: якщо список sortedtasks не порожній, то перейти до Крока 4.

Крок 15. Кінець.

У результаті буде складений повний розклад проходження підзадач на багатопроцесорній системі.

Алгоритм має алгоритмічну складність асимптотично дорівнює O(N * (N + M)), витрати по пам'яті асимптотично дорівнюють O(N + M).

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Додатки, що демонструють застосовність програмної моделі

В якості прикладів використання розроблених алгоритмів розроблено обчислювальні додатки з різних областей фізики, що демонструють застосовність мови ДВМ-Н.

Також потрібно навести результати порівняння продуктивності на тестах з пакета NASA NPB [22] програм ДВМ-Н і програм, що написано з використанням низькорівнених технологій програмування фахівцями з інших дослідницьких груп.

Функції роботи з паралельними циклами і послідовними ділянками усередині регіонів:

DVM-H-loopref loop_create_(DVM-Hregionref *regionref, Loopref *InДВМloop);

*regionref – посилання на об’ємний даний цикл регіон, яку повернув виклик region_create_.

*InDVMloop – посилання на DVM-Описувача паралельного циклу, яку повернув виклик crtpl_. У випадку послідовної ділянки як даного аргументу задається нуль.

Створює описувача паралельного DVM-Н-циклу, посилання на який використовується в подальших викликах. Тут і далі послідовна ділянка вважається часткам случаємо паралельного циклу.

void loop_insred_(DVM-Hloopref *InDVM-Hloop, Redref *Inredrefptr);

*InDVM-Hloop – посилання на описувача паралельного ДВМ-Н-циклу, яку повернув loop_create_.

*Inredrefptr – посилання на описувача редукційної функції, включеної в даний паралельний цикл. Це посилання слід одержати від виклику crtrdf_.

Функція для включення редукції в паралельний ДВМ-Н цикл.

void loop_across_(DVM-Hloopref *InDVM-Hloop, Shadowgroupref *oldgroup, Shadowgroupref *newgroup);

*In DVM -Hloop – посилання на описувача паралельного ДВМ-Н-циклу, яку повернув loop_create_.

*oldgroup – посилання на так звану стару тіньову групу. Дана тіньова група обновляється перед виконанням циклу.

*newgroup – посилання на так звану нову тіньову групу. Дана тіньова група обновляється в ході обчислення паралельного циклу.

Функція для оповіщення LibДВМ-Н про тіньові групи, створені для обробки вказівки ACROSS у директиві паралельного циклу. Т.к. виклик across_ вихідні тіньові групи знищує, то в даний виклику слід передавати їхні копії.

void loop_set_cuda_block_(DVM-Hloopref *InДВМ-Hloop, DVMtype *Inxref, ДВМtype *Inyref, ДВМtype *Inzref);

*InDVM-Hloop – посилання на описувача паралельного ДВМ-Н-Циклу, яку повернув loop_create_.

*Inxref – розмір Cuda-Блоку по координаті X.

*Inyref – розмір Cuda-Блоку по координаті Y.

*Inzref – розмір Cuda-Блоку по координаті Z.

Функція для установки заданої програмістом конфігурації блоку CudanНиток.

void loop_shadow_compute_(DVM-Hloopref *InВ якості прикладів використання розроблених алгоритмів розроблено обчислювальні додатки з різних областей фізики, що демонструють застосовність мови ДВМ-Н.

Hloop, DVMtype DVMdesc[]);

*InДВМ-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яку повернув loop_create_.

DVMdesc – заголовний масив DVM-Масиву, який змінюється в даному циклі.

Функція для повідомлення про те, який масив змінюється при SHADOW_COMPUTE при даному паралельному циклі в регіоні.

Викликається для всіх змінюваних у циклі масивів. Якщо в циклі змінюються всі вихідні і локальні масиви регіону (тобто всі ті, які зареєстровані з атрибутами out або local), то можна або повідомити про них усіх, або взагалі не повідомляти – тоді прийметься консервативна позиція і будуть уважатися змінюваними по SHADOW_COMPUTE усі out і local масиви.

void loop_register_handler_(DVM-Hloopref *InDVM-Hloop, DVMtype *devicetyperef, DVMtype *flagsref, Genfunc f, DVMtype *basescount, DVMtype *paramcount, ...);

*InDVM-Hloop – посилання на описувача паралельного ДВМ-НцЦиклу, яку повернув loop_create_.

*devicetyperef – тип обладнання, для якого годиться даний оброблювач, де ЦПУ=1, CUDA=2.

*flagsref – бітова безліч прапорів. Прапор 1 означає, що оброблювач є паралельним і для нього на розсуд системи підтримки буде виділено трохи (1 або більш) слотів виконання для обладнання. Такий оброблювач повинен сам запросити в системи підтримки своя кількість слотів (функція loop_get_slot_count_). Прапор 2 означає, що оброблювач може виконуватися тільки в основній нитці виконання програми (зокрема це спричиняє той факт, що інших його інстанцій у паралель запущене не буде).

f – посилання на функцію, що є оброблювачем для заданого циклу на обладнаннях заданого типу.

*basescount – кількість переданих з LibDVM-H баз.

*paramcount – кількість додаткових аргументів для передачі у функціюоброблювач. Усі ці додаткові аргументи задаються в кінці списку фактичних аргументів і повинні передаватися винятково за адресою.

Функція зі змінним числом аргументів, призначена для реєстрації оброблювача для паралельного циклу. Для оброблювачів задається тип обладнання, для якого він може бути використаний, його можливості і обмеження у вигляді прапорів.

void loop_perform_(DVM-Hloopref *InDVM-Hloop);

*InDVM-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яку повернув loop_create_.

Команда початку виконання циклу. Якщо цикл перебуває в синхронному регіоні, то в цьому виклику буде відбуватися і очікування кінця циклу.

DVMtype loop_get_device_num_(DVM-Hloopref *InDVM-Hloop);

*InDVM-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яка була передана в оброблювач.

Виклик для одержання номера обладнання, на якім виконує поточний оброблювач поточну порцію циклу.

DVMtype loop_has_element_(DVM-Hloopref *InDVM-Hloop, DVMtype DVMdesc[],DVMtype indexarray[]);

*InDVM-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яка була передана в оброблювач.

DVMdesc – заголовний масив DVM-Масиву, для якого перевіряється приналежність елемента локальної частини.

indexarray – масив глобальних індексів елемента.

Даний виклик є аналогом tstelm_ для виклику в оброблювачах. Повертає 1, якщо елемент належить локальній частині масиву на данім обладнанні в даному регіоні. Повертає 0 інакше. Індекси глобальні.

void loop_fill_bounds_(DVM-Hloopref *InDVM-Hloop, Indextype lowindex[], Indextype highindex[], Indextype stepindex[]);

*InDVM-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яка була передана в оброблювач.

lowindex – масив, у який будуть записані початкові значення.

highindex – масив, у який будуть записані кінцеві значення.

stepindex – масив, у який будуть записані кроки.

Запит на заповнення границь багатомірного циклу (і кроків) для виконання порції циклу оброблювачем.

void loop_fill_local_part_(DVM-Hloopref *InDVM-Hloop, DVMtype ДВMdesc[], Indextype part[]);

*InDVM-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яка була передана в оброблювач.

DVMdesc – заголовний масив DVM-Масиву, для якого запитуються границі локальної частини.

part – вихідний масив, у який будуть записані початкові і кінцеві індекси локальної для поточного регіону на поточнім обладнанні частини.

Запит на заповнення масиву, що описує локальну частину масиву на поточнім обладнанні в поточному регіоні.

void loop_red_init_(ДВМ-Hloopref *InDVM-Hloop, DVMtype *Inrednumref, void *arrayptr, void *locptr);

*InDVM-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яка була передана в оброблювач.

*Inrednumref – номер редукційної змінної в порядку їх реєстрації через loop_insred_.

arrayptr – посилання на локальну змінну, редукційний масив.

locptr – посилання на локальну змінну, locc-масив.

Запит на заповнення редукційного масиву і супутнього йому loccмасиву початковими даними для наступного нагромадження редукції.

DVMtype loop_get_slot_count_(DVM-Hloopref *InDVM-Hloop);

*InDVM-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яка була передана в оброблювач.

Запит на одержання кількості слотів, виділених для виконання порції циклу в даному запуску даного оброблювача

DVMtype loop_get_dependency_mask_(DVM-Hloopref *InDVM-Hloop);

*InDVM-Hloop – посилання на описувача паралельного DVM-НцЦиклу, яка була передана в оброблювач.

Запит маски залежних вимірів паралельного циклу. Молодший розряд відповідає внутрішньому циклу. 1 – є залежність, 0 – немає залежності.

void loop_cuda_register_red(DVM-Hloopref *InDVM-Hloop, DVMtype Inrednum, void **Arrayptr, void **Locptr);

*InDVM-Hloop – посилання на описувача паралельного DVM-НцЦиклу, яка була передана в оброблювач.

Inrednum – номер редукційної змінної в порядку їх реєстрації через loop_insred_.

arrayptr – посилання на локальну змінну для зберігання посилання на редукційний масив у пам'яті ГПУ.

locptr – посилання на локальну змінну для зберігання посилання на loccмасив у пам'яті ГПУ.

Функція для реєстрації змінних, у які будуть поміщені (у процесі ітерування циклу по loop_cuda_do або викликом loop_cuda_red_prepare_) адреси для редукційної змінної і її loccмасиву в пам'яті ГПУ.

void loop_cuda_red_init_(DVM-Hloopref *InDVM-Hloop, ДВМtype Inrednum, void *arrayptr, void *locptr, void **devarrayptr, void **devlocptr);

*InDVM-Hloop – посилання на описувача паралельного DVM-Н-Циклу, яка була передана в оброблювач.

Inrednum – номер редукційної змінної в порядку їх реєстрації через loop_insred_.

arrayptr – посилання на локальну змінну, редукційний масив.

locptr – посилання на локальну змінну, loccмасив.

devarrayptr – посилання на локальну змінну для зберігання посилання на редукційний масив у пам'яті ГПУ.

devlocptr – посилання на локальну змінну для зберігання посилання на loccмасив у пам'яті ГПУ.

Альтернативна loop_red_init_ функція для запиту заповнення редукційного масиву і супутнього йому loccмасиву початковими даними, яка на відміну від loop_red_init_ крім усього іншого створює масиви з початковими значеннями в пам'яті ГПУ.

void loop_cuda_red_prepare_(DVM-Hloopref *InDVM-Hloop, DVMtype *Inrednumref, DVMtype *Incountref, DVMtype *Infillflagref);

*InDVM-Hloop – посилання на описувача паралельного DVM-H-Циклу, яка була передана в оброблювач.

*Inrednumref – номер редукційної змінної в порядку їх реєстрації через loop_insred_.

*Incountref – кількість екземплярів редукційної змінної.

*Infillflagref – ознака, чи необхідно заповнення початковими значеннями виділених змінних.

Виділення пам'яті на задану кількість редукційних змінних (можливо, з ініціалізацією) для наступного проведення редукції.

Cudaindextype *loop_cuda_get_local_part(DVM-Hloopref *InDVM-Hloop, DVMtype DVMdesc[]);

*InДВМ-Hloop – посилання на описувача паралельного DVM-H-Циклу, яка була передана в оброблювач.

DVMdesc – заголовний масив DVM-Масиву, для якого запитуються границі локальної частини.

Усі розглянуті розпаралелювання DVM-H програми можуть працювати на кластері з використанням графічних процесорів або без, а також з використанням загальної пам'яті усередині вузла (розпаралелювання в моделі SMP).

У даному розділі розглядаються додатки з області гідродинаміки – одне двовимірне завдання і одна тривимірна. Також розглядаються додатки з області квантової механіки і мікроелектроніки. Метою їх розгляду є показати застосовність мови Фортран-ДВМ-Н для написання програм для рішення реальних завдань, а також позначити характеристики текстів отриманих паралельних програм, що вказують на зроблені для розпаралелювання зміни вихідної послідовної програми.

Програма «Каверна» призначена для моделювання циркуляційного плину в плоскій квадратній каверні з, що рухається верхньою кришкою у двовимірній постановці в широкому діапазоні як параметрів завдання, так і параметрів чисельного методу.

Послідовна версія програми займає 496 рядків. У ході розробки паралельної програми для даного завдання були перетворені деякі цикли; додані директиви мови Фортран-ДВМ-Н для розподілу даних і обчислень (18 розподілених масивів, 7 обчислювальних регіонів, 28 паралельних циклів), організації доступу до вилучених даних (8 місць), актуалізації (11 місць).

Текст паралельної програми займає 613 рядків.

Приклад конфігураційного файлу, створюваного мережною підсистемою, наведений на рис. 4.1.

```
1.    Default=0x7fff,ipoib : ALL=full;
2.    # Network vnet8001 for customer user1 (2 guids)
3.    vnet8001=0x8001,ipoib:
4.          0x0002c903000d2255=full,
5.          0x0002c90300eb8341=full;
6.    # Network vnet8003 for customer user2 (2 guids)
7.    vnet8003=0x8003,ipoib:
8.          0x0002c903000d2255=full,
9.          0x0002c90377eb8543=full;
```

Рисунок 4.1 – Приклад конфігураційного файлу Opensm

У наведеному прикладі перший рядок конфігураційного файлу описує розділ за замовчуванням, у який входять усі обчислювальні вузли суперкомп'ютера. Другий і шостий рядок є коментарями. Третій і сьомий рядки описують визначення розділів. Рядки 4-5, 8-9 описують набір з GUID портів вузлів усередині розділу і режими доступу.

На відміну від Infiniband-мережі, де використовується централізована система керування логічною топологією (Open-sm менеджер), в Ethernet-мережі суперкомп'ютера для зміни логічної топології необхідно управляти декількома Ethernet-свитчами, кожний з яких управляє логічною топологією окремих блоків з обчислювальними вузлами. Мережева підсистема управляє програмним забезпеченням кожного свитча окремо.

Для керування логічною топологією Ethernet-мережі, необхідно, щоб кожний обчислювальний вузол мав унікальну пару (свитч, порт), що однозначно визначає фізичне підключення порту Ethernet-адаптера вузла з портом свитча.

Опис конфігураційного файлу Opensm [7] презентовано в БНФ і зображене на рис. 4.2.

```
<конфигурационный файл> ::= [<строка конфигурации 1> |…| <строка конфигу-
рации N>]
<строка конфигурации> ::= <определение раздела>:<Список из GUID портов и
режим доступа>;
<определение раздела> ::= <имя раздела>=<pkey раздела>, <флаги >
<флаги> ::= <ipoib>[,<rate>,<mtu>]
<ipoib> ::= <строка ipoib> # флаг ipoib позволяет узлам обмениваться данными
по протоколу ip (ip over ib).
<имя раздела> ::= <строка без пробелов и спецсимволов>
<pkey раздела> ::= <шестнадцатеричное число, идентифицирующее номер раз-
дела>
<список из GUID портов и режим доступа> ::= <GUID порта>=<режим доступа>
<GUID порта> ::= <шестнадцатеричное число, идентифицирующее номер порта
InfiniBand адаптера узла>
<режим доступа> ::= <full>|<limited>
<full> ::= <строка full> # режим доступа full позволяет узлам без ограни-
чений инициировать сетевые обмены
<limited> ::= <строка limited> # режим доступа limited позволяет узлам с
данным режимом инициировать сетевые обмены только с узлами имеющими режим
доступа full
```

Рисунок 4.2 – Опис конфігураційного файлу Opensm у БНФ

У таблицях 4.1 і 4.2 наведено час виконання 200 ітерацій програми «Каверна» на сітці 3200x3200 на оціночно з по скорювачами іншої архітектури – Xeon Phi от компании Intel на різнім числі процесорних ядер і ГПУ (у секундах).

Таблиця 4.1 – Час виконання програми «Каверна» на сітці 3200x3200 на різнім числі процесорних ядер

1	2	4	8	16	64	128	256	400	512	1024
1241,83	631,47	332,36	182,95	100,05	40,20	21,33	11,74	7,11	6,44	3,48

При використанні 1024 ядер програма «Каверна» прискорилася в 357 раз у порівнянні з виконанням на 1 ядрі. При використанні 1 прискорювача програма прискорюється в 17 раз у порівнянні з виконанням програми на 1 ядрі. Максимальне прискорення, отримане з використанням прискорювачів – 390 раз у порівнянні з виконанням програми на 1 ядрі.

Програма «Контейнер» призначена для чисельного моделювання плину грузлої важкої рідини під дією сили ваги в прямокутному контейнері з відкритою

верхньою стінкою і отвором в однієї з бічних стінок у тривимірній постановці в широкому діапазоні як параметрів завдання, так і параметрів чисельного методу.

Таблиця 4.2 – Час виконання програми «Каверна» на сітці 3200x3200 на різнім числі ГПУ

1	2	4	8	16	32	64	128	256	400
73,07	39,34	19,94	11,65	7,17	4,80	3,96	3,45	3,32	3,19

Послідовна версія програми займає 828 рядків. У ході розробки паралельної програми для даного завдання були перетворені деякі цикли; додані директиви мови Фортран-ДВМ-Н для розподілу даних і обчислень (26 розподілених масивів, 5 обчислювальних регіонів, 21 паралельний цикл), організації доступу до вилучених даних (6 місць), актуалізації (7 місць).

Текст паралельної програми займає 942 рядки. У таблицях 4.3 і 4.4 наведені часи виконання програми «Контейнер» на суперкомп'ютері із різній кількості процесорних ядер і ГПУ.

Таблиця 4.3 – Час виконання програми «Контейнер» на різнім числі ядер

Сітка, кількість	4	8	64	128	256	512	1024	2048
ітерацій								
200x200x200 itmax=200	754,01	384,92	49,87	29,90	14,52	8,63	5,63	6,01
400x400x400 itmax=100	-	1202,32	164,02	85,68	43,10	22,53	13,54	7,66
1600x1600x1600 itmax=20	-	-	-	-	-	235,64	117,88	62,68

При використанні 2048 ядер програма «Контейнер» прискорилася в 26, 3 рази в порівнянні з виконанням на 64 ядрах. При використанні 64 прискорювачів програма прискорюється в 11 раз у порівнянні з виконанням програми на 64 ядрах. При використанні 1280 прискорювачів програма прискорюється в 138 раз у порівнянні з виконанням програми на 64 ядрах.

4.2 Моделювання тривимірних алгоритмів

Програма «Стани кубітів» призначена для проведення тривимірних нестаціонарних розрахунків стану кубітів квантового комп'ютера на основі спільного рішення тривимірного рівняння Пуассона і нестаціонарного рівняння Шредінгера для двох електронів в квантовому приладі на основі кремнієвого квантового дроту з обліком спина цих електронів.

Послідовна версія програми займає 683 рядків.

У ході розробки паралельної програми для даного завдання були перетворені деякі цикли; додані директиви мови Фортран-ДВМ-Н для розподілу даних і обчислень (23 розподілених масиву, 5 обчислювальних регіонів, 61 паралельний цикл), організації доступу до вилучених даних (8 місць), актуалізації (5 місць).

Текст паралельної програми займає 1011 рядків.

При використанні 60 ядер програма «Стану кубітів» прискорилася в 4,76 раз у порівнянні з виконанням на 12 ядрах. При використанні 1 прискорювача програма прискорюється в 1,87 раз у порівнянні з виконанням програми на 12 ядрах.

У таблиці 4.5 наведені часи виконання 192 ітерацій програми «Стану кубітів» на сітці 121x121x241 на моделі суперкомп'ютера» на різній кількісті процесорних ядер і ГПУ (у секундах).

Таблиця 4.4 – Час виконання програми «Контейнер» на різнім числі ГПУ

Сітка, кількість ітерацій	1	2	4	32	128	256	512	1024	1280
200x200x200 itmax=200	166,95	86,05	45,77	8,99	3,99	3,26	3,01	3,60	4,32
400x400x400 itmax=100	-	-	168,80	26,09	8,39	4,86	3,20	2,88	3,26
800x800x800 itmax=50	-	-	-	92,20	30,32	13,58	7,56	4,67	4,17
1600x1600x1600 itmax=20	-	-	-	-	-	37,38	20,14	10,74	8,95

Таблиця 4.5 – Час виконання програми «Стану кубітів» на сітці 121x121x241 на різнім числі процесорних ядер і ГПУ

12 ядер	24 ядра	36 ядер	48 ядер	60 ядер	1 ГПУ	2 ГПУ	4 ГПУ	6 ГПУ
190,38	104,20	74,57	56,71	39,96	102,06	59,66	32,97	22,04

Програма «Кристалізація 3D» пов'язана із завданням, у якому лазерний промінь у початковий момент перебуває в крапці (x,y), потім проплавляє матеріал у напрямку осі z і зміщається в напрямку осі x. Коли промінь зміщається, розплавлений матеріал починає кристалізуватися.

Положення променя в кожний момент часу задається деякою функцією.

Послідовна версія програми займає 530 рядків.

У ході розробки паралельної програми для даного завдання були перетворені деякі цикли; додані директиви мови Фортран-ДВМ-Н для розподілу даних і обчислень (21 розподілений масив, 6 обчислювальних регіонів, 27

паралельних циклів), організації доступу до вилучених даних (5 місць), актуалізації (5 місць).

Текст паралельної програми займає 865 рядків. У таблиці 6 наведені часи виконання 100 000 ітерацій програми «Кристалізація 3D» на сітці 103x5x103 на моделі суперкомп'ютера «К-100» на різній кількості процесорних ядер і ГПУ (у секундах).

Таблиця 4.6 – Час виконання програми «Кристалізація 3D» на сітці 103x5x103 на різнім числі процесорних ядер і ГПУ

1 ядро	12 ядер	50 ядер	100 ядер	1 ГПУ
514,4	256,3	89,09	79,97	396,3

При використанні 100 ядер програма «Кристалізація 3D» має прискоритися в 6,43 раз у порівнянні з виконанням на одному ядрі. При використанні 1 прискорювача програма прискорюється в 1,3 рази у порівнянні з виконанням програми на 1 ядрі.

Програма «Спікання 3D» пов'язана із завданням, у якому лазер плавить порошок, що полягає із твердого легкоплавкого компонента. У порошку є пори, заповнені газом. У процесі плавлення легкоплавкого компонента порошку утворюється рідина.

Послідовна версія програми займає 677 рядків.

5 ОПИС МОЖЛИВОСТІ ВИКОРИСТАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

У ході розробки паралельної програми для даного завдання були перетворені деякі цикли; додані директиви мови Фортран-ДВМ-Н для розподілу даних і обчислень (19 розподілених масивів, 4 обчислювальних регіону, 51 паралельний цикл), організації доступу до вилучених даних (3 місця), актуалізації (22 місця).

Текст паралельної програми займає 1236 рядків. У таблиці 8 наведені часи виконання 1 000 ітерацій програми «Спікання 3D» на сітці 103x103x103 на моделі суперкомп'ютера на різній кількісті процесорних ядер і ГПУ (у секундах) (див. табл.. 5.1).

Таблиця 5.1 – Час виконання програми «Спікання 3D» на сітці 103x103x103 на різній кількості процесорних ядер і ГПУ

1 ядро	12 ядер	64 ядра	200 ядер	1 ГПУ	6 ГПУ	12 ГПУ	24 ГПУ
751	192,9	40,23	30,46	50,88	18,47	14,85	12,64

При використанні 200 ядер програма «Спікання 3D» прискорилася в 24,66 раз у порівнянні з виконанням на 1 ядрі. При використанні 1 прискорювача програма прискорюється в 14,76 раз у порівнянні з виконанням програми на одному ядрі.

Також перевіряється поведінка системи при запиті користувачем більшої ніж доступно кількості вузлів.

Тестові завдання «Створення шаблонів» – дані тестові завдання необхідні для перевірки коректності поведінки системи в процесі створення шаблону на основі образа обчислювального вузла.

Тестові завдання «Відкладена установка» – необхідні для перевірки коректності поведінки системи в процесі виконання «відкладеної установки»

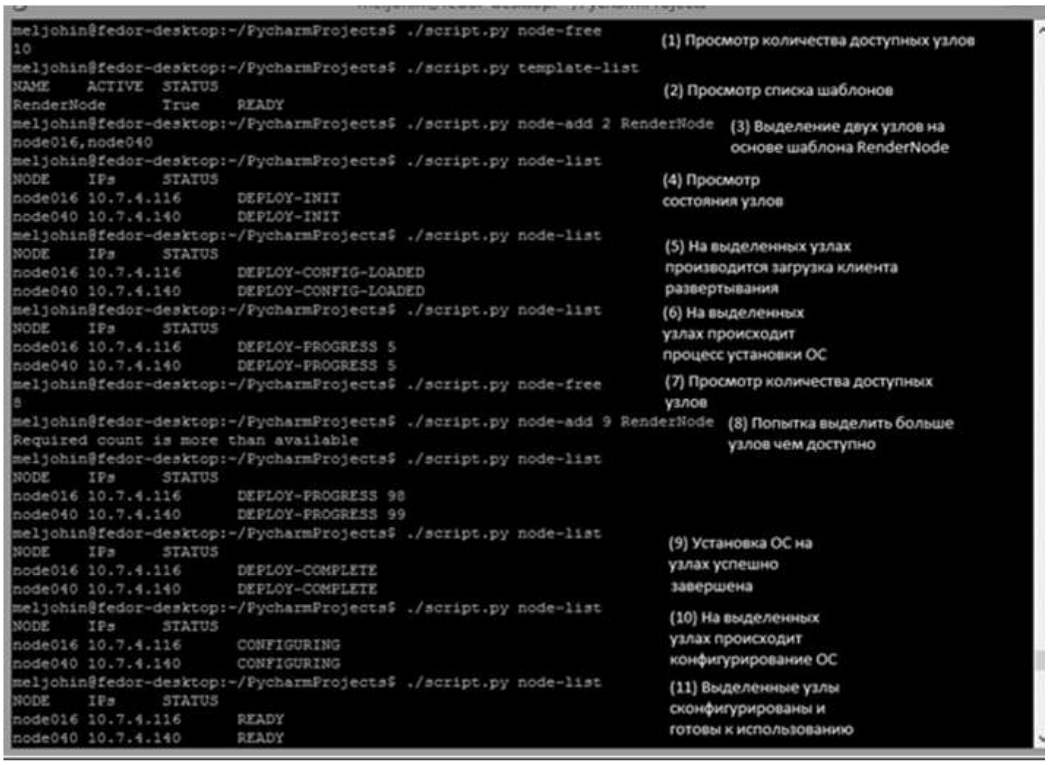

Рисунок 5.1 – Емуляція тестування API «Виділення вузлів»

Інтерфейс даної підсистеми із заданою періодичністю проводить відновлення відображуваної інформації про стан процесу синхронізації образів дисків. У ході проведення тестування у веб-інтерфейсі було видно, що одночасно 30 вузлів робили синхронізацію образів дисків, що відповідає заданій в конфігураційному файлі максимальній кількості вузлів, що одночасно синхронізуються.

За результатами тестування можна зробити висновки про те, що розроблена система буде працювати коректно і функціональність системи відповідає заявленій.

Веб-інтерфейс до діаграми розміщення, відповідно до якої була зроблена установка компонентів системи,

Оскільки функціонування даної системи є критично важливим, був розроблений додатковий модуль діагностики стану системи. Діагностична інформація використовується системою моніторингу Nagios, завдяки чому у випадку апаратного або програмного збою системи, адміністратори будуть сповіщені про це, що дозволить скоротити час відновлення працездатності

системи. Надалі планується підвищення надійності системи у випадку апаратних збоїв головного серверу системи за рахунок створення резервного серверу.

Оскільки доступ до системи здійснюється через мережу Інтернет, то для запобігання спроб добору паролів на обслуговуючий систему сервер була встановлена система fail2ban, що дозволяє блокувати ip-адреси зловмисника при трьох спробах уведення некоректних облікових даних.

Для порівняння програм мовою ДВМ-Н з написаними вручну с використанням низькорівнених технологій програмування програмами по ефективності, що досягається, приводяться прискорення тестів EP, BT, SP, LU з набору NASA NPB при використанні 1 ГПУ в порівнянні згодом роботи на 1 ядрі ЦПУ.

Порівняння проводилося із програмами, написаними дослідниками із Сеульського національного університету. Їхня робота «Performance Characterization of the NAS Parallel Benchmarks in Opencl» опублікована в збірнику праць міжнародної конференції «2011 IEEE International Symposium on Workload Characterization» [23], а вихідні тексти програм перебувають у вільному доступі.

У таблиці 5.2 наведені досягнуті прискорення для тестів EP, BT, SP, LU різних класів на 1 ГПУ стосовно часу роботи на 1 ядрі ЦПУ для програм мовою Фортран-ДВМ-Н і програм мовою С з використанням технології Opencl. Запуски проводилися на кластері моделі суперкомп'ютера «К-100».

Таблиця 5.2 – Прискорення тестів NASA NPB для програм мовою Фортран-ДВМ-Н і програм мовою С з використанням технології Opencl на 1 ГПУ стосовно часу роботи на 1 ядрі ЦПУ

Тест	EP	BT		SP			LU		
Клас	C	A	B	A	B	C	A	B	C
ДВМ-Н	46,24	2,4	2,22	5,88	6,37	7,72	6,47	9,06	11,1
Opencl	44,63	0,89	1,04	3,05	3,36	2,87	5,43	6,72	6,4

ДВМ-Н-версії тестів по ефективності не уступають OpenclвВерсіям, а на тестах, що мають цикли із залежностями, перевершують їх завдяки наявним у компіляторові з мови Фортран-ДВМ-Н оптимізаціям для обробки таких циклів.

Таким чином, розроблені принципи відображення ДВМ-Н -програм на кластери із прискорювачами, що забезпечують динамічний розподіл обчислень між універсальними багатоядерними процесорами (ЦПУ) і декількома графічними процесорами (ГПУ).

Розроблено і реалізовані в системі підтримки виконання ДВМ-Н-програм наступні алгоритми:

– алгоритми розподілу незалежних підзадач між вузлами кластера завантаження, що забезпечують балансування, вузлів;

– алгоритми розподілу витків паралельних циклів усередині вузлів – між ядрами ЦПУ і декількома ГПУ;

– алгоритми автоматичного переміщення необхідних актуальних даних між пам'яттю ЦПУ і пам'ятями декількох ГПУ.

Створено засоби порівняльного налагодження ДВМ-Н-програм, що базуються на зіставленні результатів одночасного виконання на ЦПУ і на ГПУ тих самих фрагментів програми.

Проведено експерименти з тестами і реальними додатками показали, що для класу завдань, при рішенні яких використовуються різницеві методи на статичних структурних сітках, можна писати програми мовою Фортран-ДВМ-Н, які ефективно виконуються на кластерах із прискорювачами.

ВИСНОВКИ

Розроблені принципи відображення ДВМ-Н-програм на кластери із прискорювачами, що забезпечують динамічний розподіл обчислень між універсальними багатоядерними процесорами (ЦПУ) і декількома графічними процесорами (ГПУ).

Розроблені і реалізовані в системі підтримки виконання ДВМ-Н-програм наступні алгоритми:

– алгоритми розподілу незалежних підзадач між вузлами кластера завантаження, що забезпечують балансування, вузлів;

– алгоритми розподілу витків паралельних циклів усередині вузлів – між ядрами ЦПУ і декількома ГПУ;

– алгоритми автоматичного переміщення необхідних актуальних даних між пам'яттю ЦПУ і пам'ятями декількох ГПУ.

Створені засоби порівняльного налагодження ДВМ-Н-програм, що базуються на зіставленні результатів одночасного виконання на ЦПУ і на ГПУ тих самих фрагментів програми.

Проведені програмні експерименти з тестами і реальними додатками показали, що для класу завдань, при рішенні яких використовуються різницеві методи на статичних структурних сітках, можна писати програми мовою Фортран-ДВМ-Н, які ефективно виконуються на кластерах із прискорювачами.

З використанням розроблених алгоритмів створена система підтримки виконання ДВМ-Н-програм, що є невід'ємною частиною компіляторів ДВМ-Н-програм.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЯ

1. Top500 List – November 2018 TOP500 Supercomputer Sites. URL: http://top500.org/list/2012/11/.

2. MPI: The Message-Passing Interface Standard. URL: http://www.mpi-forum.org/docs/mpi-1.1/mpi1-report.pdf.

3. Openmp Application Program Interface. URL: http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.

4. Боресков А.А., Харламов А.А. Основи роботи з технологією CUDA. — М.: Дмк-Пресс, 2010. — 232 стор.

5. High Performance Fortran. URL: http://hpff.rice.edu/ .

6. Konovalov N.A., Krukov V.A., Pogrebtsov A.A., Podderyugina N.V., Sazanov Y.L. Parallel’noe programmirovanie v sisteme ДВМ. Yazyki Fortran-ДВМ i C-ДВМ [Parallel programming in the ДВМ system. Fortran-ДВМ and C-ДВМ languages]. Trudy Mezhdunarodnoy konferencii “Parallel’nye vychisleniya i zadachi upravleniya” (PACO’2001) [Proceedings of International conference “Parallel computations and control problems” (PACO’2001)]. Moscow, 2001. P. 140–154.

7. Konovalov N.A., Krukov V.A., Mihailov S.N., Pogrebtsov A.A. Fortran ДВМ – yazyk razrabotki mobil’nyh parallel’nyh programm [Fortran ДВМ – a language for mobile parallel programs development]. Programmirovanie [Programming]. 1995. No. 1. P. 49–54.

8. Konovalov N.A., Krukov V.A., Sazanov Y.L. C-ДВМ – yazyk razrabotki mobil’nyh parallel’nyh programm [C-ДВМ – a language for mobile parallel programs development]. Programmirovanie [Programming]. 1999. No. 1. P. 54–65.

9. Dolbeau R., Bihan S., Bodin F. HMPP™: A Hybrid Multi-core Parallel Programming Environment URL: http://www.caps-entreprise.com/wp-content/uploads/2012/08/capshmpp-gpgpu-Boston-Workshop-Oct-2007.pdf (accessed 02.12.2012). В.А. Бахтин, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула 2013, т. 2, № 4 55

10. The Portland Group. PGI Accelerator Programming Model for Fortran & C. URL: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf (accessed 02.12.2012).

11. В.А. Бахтин, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула Отображение на кластеры с графическими процессорами ДВМ-Н-программ с регулярными зависимостями по данным https://cyberleninka.ru/ article/n/otobrazhenie-na-klastery-s-graficheskimi-protsessorami-po-dannym/viewer

12. OpenACC. URL: http://www.openacc-standard.org/.

13. Han T.D., Abdelrahman T.S. hiCUDA: High-Level GPGPU Programming. IEEE Transactions on Parallel and Distributed Systems. 2017. Vol. 22, No. 3. P. 78–90.

14. Bakhtin V.A., Klinov M.S., Krukov V.A., Podderyugina N.V., Pritula M.N., Sazanov Y.L. Rasshirenie ДВМ-modeli parallel'nogo programmirovaniya dlya klasterov s geterogennymi uzlami [Extension of the ДВМ parallel programming model for clusters with heterogeneous nodes]. Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta, seriya "Matematicheskoe modelirovanie i programmirovanie". Chelyabinsk, Publishing of the South Ural State University, 2012. No. 18 (277), Issue 12. P. 82–92.

15. Pennycook S.J., Hammond S.D., Jarvis S.A., Mudalige G.R. Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark. ACM SIGMETRICS Performance Evaluation Review – Special issue on the 1st international workshop on performance modeling, benchmarking and simulation of high performance computing systems (PMBS 10). 2019. Vol. 38, Issue 4. P. 23–29.

16. Seo S., Jo G., Lee J. Performance Characterization of the NAS Parallel Benchmarks in OpenCL. 2018 IEEE International Symposium on. Workload Characterization (IISWC). 2011. P. 137–148

17. The Opencl Specification. URL: http://www.khronos.org/registry/cl/ specs/opencl-2.0.pdf.

18. Коновалів Н.А., Гаків В.А., Погребцов А.А., Поддерюгіна Н.В., Сазанів Ю.Л. Паралельне програмування в системі ДВМ. Мови Fortran-ДВМ і С-ДВМ. // Праці Міжнародної конференції «Паралельні обчислення і завдання керування»

(PACO’2001). — Москва, 2013. — с. 140–154.

19.Bakhtin V.A., Krukov V.A., Pritula M.N. Extension of ДВМ parallel programming model for clusters with heterogeneous nodes. // Research conference on information technology. Honoring volume on Pollack Mihaly faculty of engineering and information technology. Seventh international Phd & DLA symposium, жовтень 2018 р., г. Pecs, Hungary. – Komlo: Rotari Press, 2020, с. C17.

20.Seo S., Jo G., Lee J. Performance Characterization of the NAS Parallel Benchmarks in Opencl. // IEEE International Symposium on Workload Characterization (IISWC). — 2021. — p. 137–148.

21.Diamos G., Kerr A., Yalamanchili S., Clark N. Ocelot: A dynamic compiler for bulk-synchronous applications in heterogeneous systems. // Proceedings of The 19th International Conference on Parallel Architectures and Compilation Techniques, Vienna, Austria, 11-15 September, 2021.

22.М.А. Кривов, М.Н. Притула, С.А. Грізан, П.С. Іванов. Оптимізація додатків для гетерогенних архітектур. Проблеми і варіанти рішення. Інформаційні технології і обчислювальні системи, № 2012/03. ISSN 2071-8632. http://www.jitcs.ru/

23. Wolfe M. Implementing the PGI accelerator model. // Proceedings of 3rd Workshop on General Purpose Processing on Graphics Processing Units, Pittsburgh, Pennsylvania, USA, 14 March, 2019.

24. Dave C., Bae H., Min S.-J., Lee S., Eigenmann R., Midkiff S. Cetus: A source-to-source compiler infrastructure for multicores. // IEEE Computer, p. 36–42, 2009.

25. Ren M., Park J.Y., Houston M., Aiken A., Dally W.J. A tuning framework for software-managed memory hierarchies. // Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, New York, NY, USA: ACM, 2018, p. 280–291.

26. Дудар З.В., Склярук Д.О. Дослідження алгоритмів підвищення надійності відеозв'язку / Информационные системы и технологии (ИСТ-2018):

материалы 7-й Международ. Науч.-техн. конф., Харьков-Коблево, 10-15 сентября 2018, – тезисы докладов. – Х.: ХНУРЕ, 2018

27. Shubin, I., Snisar, S., Zhyrnov, V., Slavhorodskyi, V.// Practical Application of Formal Representation of Information for Intelligent Radar Systems 2018 International Scientific-Practical Conference on Problems of Infocommunications Science and Technology, PIC S and T 2018 - Proceedings, 2019, pp. 433-436, 8632103