УДК 621.398:007

А.А. ЯРОВИЙ

АНАЛІЗ ФУНКЦІОНУВАННЯ ПРОГРАМНОЇ МОДЕЛІ GPGPU В КОНТЕКСТІ ОРГАНІЗАЦІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ В НЕЙРОПОДІБНИХ ПАРАЛЕЛЬНО-ІЄРАРХІЧНИХ СИСТЕМАХ

Вінницький національний технічний університет,
Хмельницьке шосе, 95, Вінниця, 21021, Україна,
тел.: +380 (432) 580019, E-mail: axa@vinnitsa.com

Анотація. В проведених дослідженнях здійснено аналіз функціонування програмної моделі GPGPU в контексті організації паралельних обчислень в нейроподібних паралельно-ієрархічних системах. В роботі наведено структурну організацію та програмно-апаратні особливості організації паралельних обчислень GPU. Наведено результати розробки програмної бібліотеки для конструювання та моделювання топологій штучних нейронних та нейроподібних мереж, зокрема паралельно-ієрархічних та ієрарх-ієрархічних.

Аннотация. В проведенных исследованиях осуществлен анализ функционирования программной модели GPGPU в контексте организации параллельных вычислений в нейроподобных параллельно-иерархических системах. В работе приведена структурная организация и программно-аппаратные особенности организации параллельных вычислений GPU. Приведены результаты разработки программной библиотеки для конструирования и моделирование топологий искусственных нейронных и нейроподобных сетей, в частности параллельно-иерархических и иерарх-иерархических.

Abstract. The analysis of functioning of program model GPGPU in a context of organization of parallel computations in neural-like parallel-hierarchical systems is carry out in researches. The structural organization and hardware-software features of organization of parallel computations GPU is given in the work. The results of development of program library for designing and modeling of artificial neural and neural-like networks topology, in particular parallel-hierarchical and hierarch-hierarchical are given.

Ключові слова: нейроподібні паралельно-ієрархічні системи, паралельні обчислення, GPGPU, програмування відеоадаптерів, обробка зображень.

ВСТУП

У наш час стрімко зростає попит на комп'ютеризовані системи надвеликої обчислювальні потужності. Це пов'язано не лише із постійним зростанням надвеликих обсягів інформації різної природи, але й зростанням складності у обчислювальному плані. Оперативна реакція та прийняття рішень у реальному часі, обробка відеоданих з великою роздільною здатністю, класифікація і прогнозування швидкозмінних динамічних даних – одні із найбільш актуальних на сьогоднішній день задач. Одним із найефективніших способів опрацювання та обробки великих масивів даних є їх паралельна обробка на основі спеціалізованих системних рішень, зокрема нейроподібних паралельно-ієрархічних систем. Проте, разом з тим, постає задача програмно-апаратної реалізації такого типу систем, а саме обрання адекватної програмно-апаратної платформи для швидкої та ефективної паралельної обробки великих масивів даних [1-3].

У минулих роботах було доведено, що досить вдалим розв'язком такої проблеми є використання відеоадаптерів для обчислень загального призначення (GPGPU) [3,4]. При застосуванні GPGPU дані для організації подальших обчислень повинні подаватися у вигляді текстур – тобто графічного зображення, у кожному пікселі якого у вигляді композиції кольорових каналів RGB подані безпосередні дані для обчислення. Звичайно, це дещо ускладнює обчислювальний процес, зокрема виникає складність реалізації алгоритму певної прикладної задачі у паралельній відносно даних формі (відповідно складно досягти рівня теоретичної швидкодії). Крім того, швидкодія обчислень обмежується ефективністю звертань до пам'яті, адже основними факторами тут є пропускна спроможність локальної пам'яті та затримки при звертанні до основної пам'яті. Для деяких задач може відігравати важливу роль малий обсяг локальної пам'яті (в середньому до 512 Мб та 4 Гб). Також програма, яка використовує відеоадаптер, для максимальної ефективності (утилізації апаратних ресурсів) повинна бути паралельна відносно даних або задач (так звані

Data Parallelism та Task Parallelism). При цьому найчастіше основний блок обчислень програми компілюється у байт-код DirectX 9 чи 10, або у відповідний байт-код ATI CTM IL. Такий байт-код транслюється у спеціальний машинний код (так званий device-specific assembler) перед виконанням. Таким чином, з одного боку необхідно відзначити певну складність програмування відеоадаптерів, а також відсутність єдиного стандарту програмування, що призвело до існування на ринку великої кількості програмних засобів, зокрема ATI Close To Metal Computing Abstraction Layer (ATI CTM CAL) [5], NVidia CUDA [6], GLSL (OpenGL)/HLSL (DirectX), RapidMind Platform [7], Stanford's University Brook for GPU/AMD Brook+ [5] та багатьох інших. Проте, з іншого боку необхідно констатувати, що сучасні масові відеоадаптери за своєю теоретичною швидкодією перевищують сучасні процесори у 10–20 разів, кількість завантажень із пам'яті значно більша, що пояснюється більшою шириною шини та більш високою тактовою частотою пам'яті, а кількість ядер у процесорі є у десятки разів більшою.

МЕТА ДОСЛІДЖЕННЯ

Метою роботи є аналіз функціонування програмної моделі GPGPU (на рівні детального аналізу структурної організації та програмно-апаратних особливостей організації обчислень) в контексті організації паралельних обчислень в нейроподібних паралельно-ієрархічних системах, а також реалізація спеціалізованої програмної бібліотеки, яка б дозволила спростити взаємодію програміста із апаратним забезпеченням, абстрагуватись від специфіки роботи із ATI CTM CAL та дозволила працювати із відеоадаптером як із єдиним об'єктом.

ПОСТАНОВКА ПРОБЛЕМИ. АНАЛІЗ СТРУКТУРНО-ФУНКЦІОНАЛЬНОЇ ОРГАНІЗАЦІЇ ПРОГРАМНОЇ МОДЕЛІ GPGPU

В останньому поколінні графічних апаратних пристроїв (GPU) прийнято стандартну шейдерну архітектуру (шейдер – це програма для одного із ступенів графічного конвеєра, використовувана в тривимірній графіці для визначення остаточних параметрів об'єкту або зображення), таким чином всі графічні функції виконуються на програмованому арифметичному логічному пристрої (ALU), які можуть керувати різними типами програм. Ці програми, які розроблені користувачами, часто називають шейдерними програмами, або ядрами (термін мови програмування Brook+) [8].

При організації обчислень загального призначення, програмована модель таких ALU повинна використовуватись, щоб здійснювати не стільки графічні функції, скільки оперування потоками даних використовуючи віртуальний SIMD (Single Instruction Multiple Data), що обробляє програмні моделі. Тут, термін SIMD використовується на рівні програми, яка функціонує використовуючи всі дані, що обробляються. В моделі "Stream Computing", масиви введених даних, елементи яких зберігаються у пам'яті, розміщені один біля одного у віртуальному SIMD-масиві, що виконує програми шейдера для створення одного чи більше виходів, які потім записуються назад до вихідних масивів у пам'ять. Кожна частина ядра, що працює на віртуальному моменті процесора називається потоком. Розміщення потоків у вихідних позиціях відома під назвою домен виконання [8].

Моделі графічних апаратних пристроїв представляють собою масив потоків на обмеженому накопичувачі фізичних процесорів упорядкування доступних ресурсів у GPU так, щоб кожний елемент віртуального SIMD-масиву був остаточно оброблений, у кожній точці додаткового шейдера, і міг би також паралельно виконуватись, поки програма не завершиться. Спрощений вигляд GPGPU програмної моделі і розміщення потоків в обробці ресурсів GPU зображений на рис. 1 [8].

Необхідно відзначити, що одна із головних особливостей GPU – здатність кожного процесора виконувати паралельні дії. Раніше ядра і потоки тільки використовували скалярні дії. Якщо компілятор виявить паралелізацію в межах ядра, то він спробує оптимізувати цю дію. Наприклад, GPU може виковувати множення кратних чисел і підсумовувати їх.

При організації роботи GPU потрібно розуміти особливості процесів безпосереднього виконання потоків. Сучасні GPU створені мати високу ефективність під час роботи великої кількості потоків у вигляді, що зрозумілий користувачу. GPU використовує велику кількість потоків для зменшення затримок під час доступу до пам'яті, за допомогою перемикання активних потоків процесора до того часу, коли даний потік закінчить очікування доступу до пам'яті, щоб бути завершеним. Час багатопотокової роботи також використовується для виконання багатьох потоків паралельно, і приховати затримку ALU операцій під час передачі. Обидві ці можливості потребують, щоб потік містив велику кількість розрахунків для покращення здатності керування ресурсами для приховання вищезазначених затримок [8].

Елементи програмованих арифметичних логічних пристроїв (ALU) об'єднані в SIMD-блоки. Рисунок 2(а), є спрощеною схемою ядра шейдеру ATI Radeon HD 3870 (спрощено RV670). Звичайно,

різні мікросхемні варіанти мають різні характеристики, але спрощено будуть відповідати одній конструктивній схемі. RV670 має 4 SIMD-блоки. Кожен блок містить 16 шейдерних апаратних пристроїв (SPU – Shader Processing Units). SPU – це фізичні апаратні засоби, що обробляють потоки. SPU організовані, як 5-ти блоковий скалярний процесор, рис. 2(б). В одну VLIW (Very Long Instruction Word) програму можна включити до п‘яти операцій. Скалярні одиниці одночасно можуть обробляти дані з плаваючою комою і операціями з цілими числами. Також, тільки один з п‘яти блоків обробляє операції підвищеної складності (sin, cos, log) або операції подвійної точності. SPU також містить один блок для виконання інструкції переходу.

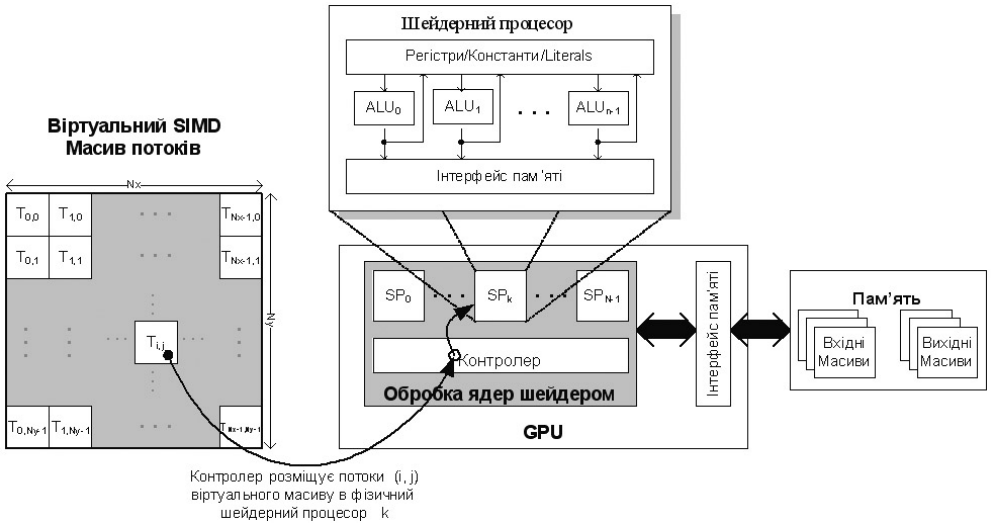

Рис. 1. Структурно-функціональна організація програмної моделі GPGPU [8]

Таким чином, на RV670 є чотири SIMD-блоки, кожен з який містить 16 SPU і кожен SPU організований як 5-ти блоковий скалярний процесор. Таким чином в сумі буде 320 скалярних процесорів на RV670 GPU.

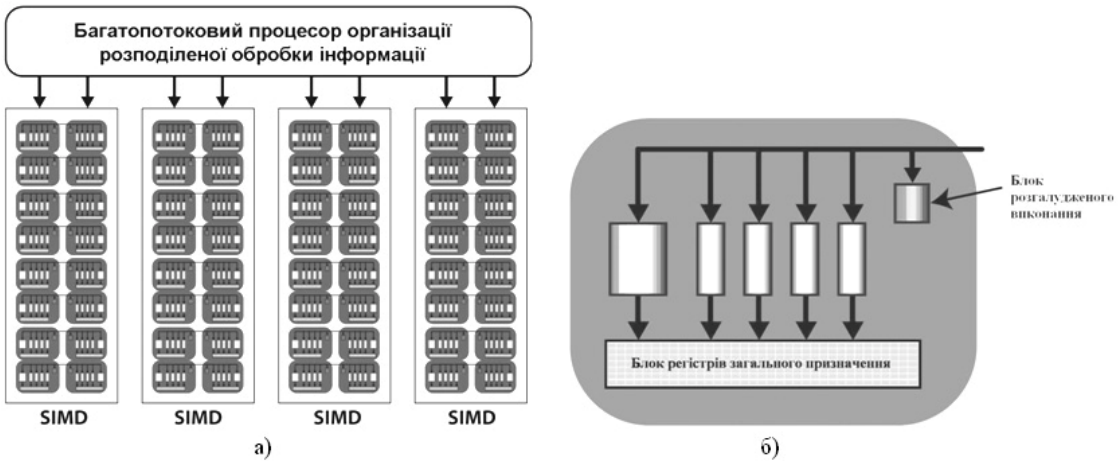

Рис. 2. Структурна схема ядра шейдеру (а) та шейдерного апаратного пристрою (б) [8]

Всі SPU в межах SIMD-блоку виконує одна і таж програма кожного разу. Для зменшення затримки що виникає внаслідок ALU-операцій, багато потоків змінюються по черзі так само, як у SPU, чотири потоки видають чотири VLIW-команди для 4-х циклів. Це означає, що 16 SPU в середині SIMD-блоків обробляють однакові команди, при чому кожен SPU обробляє чотири потоки одночасно. Це характеризує його як ефективного 64-ти розрядного SIMD-процесора. Групу потоків, що обробляються одночасно в SIMD-блоці, називають фронтом хвилі. З того часу як потоки знаходяться у фронті хвилі, вони виконуються одночасно, тому, наприклад, потоки T0…T3 (для RV670) насправді повинні розглядатись, як чотири незалежні фронти хвиль. Таким чином, розміри фронту хвилі зменшуються до 32 і 16 потоків відповідно [8].

Управління потоком керується за допомогою використання всіх необхідних доріжок одночасно, так як і фронт хвилі. Оскільки GPU керується SIMD-методом, то у ситуації, коли обробка потоків в середині фронту хвилі відрізняється, всі доріжки працюють послідовно. Наприклад, якщо ядро містить блок з двома доріжками, тоді фронт хвилі обробить одну доріжку, а потім обробить наступну. Загальний час для обробки блоку дорівнює сумі часу роботи кожної доріжки. Ключовою точкою є те, що якщо навіть тільки один потік у фронті хвилі відхиляється, то решта потоків у фронті хвилі виконає розгалуження, навіть якщо нічого не прописано. Кількість потоків, які повинні виконуватись під час розгалуження часто називається розгалуженням блоку. У апаратних пристроях ATI грануляція блоку має такий самий розмір, що і фронт хвилі [8]. Приклад: Якщо 2 блоки – А і Б мають однаковий час обробки (t) фронту хвилі, тоді загальний час виконання, якщо будь-який з потоків відхиляється, буде 2t.

Цикли виконуються в однаковому порядку, де фронт хвилі займає SIMD-масив так довго, поки потік оброблятиметься. Тому загальний час обробки для фронту хвилі є часом роботи потоку з найдовшим циклом. Приклад: якщо t – час, необхідний для обробки одного повтору циклу в фронті хвилі, всі потоки виконають цикл один раз, за винятком одного потоку, в якому цикл виконається 100 разів, в такому випадку час, що потрібний для виконання фронту хвилі, буде дорівнювати 100t.

SIMD-блоки працюють незалежно один від одного, тому є можливим, що чотири SIMD-блоки в RV670 виконують різні команди. Таким чином, у випадках з блоками і циклами, один фронт хвилі не зупинить роботу інших фронтів хвиль [8].

ОЦІНКА ПРОДУКТИВНОСТІ

Оцінка теоретичної продуктивності апаратного забезпечення є дуже важливим етапом, так як це може допомогти розробникам виявити проблемні аспекти при впровадженні та організації обчислень на GPU.

Для цього розглянемо процедуру розділення шейдера на команди. Адже в подальшому ця інформація буде потрібна для теоретичних оцінок. Іншими частинами інформації, необхідної для оцінювання теоретичної продуктивності, є:

1. # блоки ALU;
2. # текстурні блоки;
3. розмір системної шини;
4. частота ядра;
5. частота пам‘яті.

Для RV670, кількість ALU-блоків буде кількістю SPU (64), які виконують VLIW-команди. Розмір шини пам‘яті – 256 бітів. Частота пам‘яті і ядра буде залежати від використаної плати, і може бути знайдена в технічних специфікаціях плати. Зазвичай Radeon HD 3870 базуються на RV670 з частотою ядра 775 Mhz і частотою пам‘яті 1125 Mhz [8].

Починаємо з шейдеру тільки з ALU командами. Теоретична продуктивність буде [8]:

$$\frac{(\#\ потоки)\times(\#\ VLIW\ \ ALU\ \ команди)}{(ALU\ /\ clk)\times(3D\ \ engine\ \ \ speed)}. \qquad (1)$$

Кількість потоків буде розміром домену виконання. На прикладі RV670 плати одна ALU команда шейдера з доменом 2M потоків (орієнтовний розмір висоти definition frame) будуть обчислюватись так [8]:

$$\frac{(2M\ потоків)\times(1\ \ команда)}{(64\ ALU\ /\ clk)\times(775\ МГц)} = 0.04 \text{ мс}. \qquad (2)$$

Шейдер має тільки одну текстурну команду, тоді теоретична робота буде дорівнювати [8]:

$$\frac{(\#\ потоки)\times(\#\ текстурні\ \_\ команди)}{(текстура\ /\ clk)\times(швидкість\ \_\ 3D\ \_\ акселератора)} = \frac{(2M)\times(1)}{(16)\times(775МГц)} = 0.16 \text{ мс}. \qquad (3)$$

Текстурні блоки запускають частоту ядра і використовують тривимірну швидкість в оцінюванні. Оцінка роботи пам‘яті основана на загальній кількості записаних і прочитаних даних в пам‘ять потоку [8]:

$$\frac{(\#\ потоки)\times(вхідні + вихідні\ \_\ біти\ \_\ потоку)}{(шина)\times(швидкість\ \_\ памяті)}. \qquad (4)$$

В простій програмі (один байт на виході іншого) теоретична робота пам‘яті була б такою [8]:

$$\frac{(2M\ потоків)\times(16\ біт)}{(256\ біт)\times(1125МГц\times 2DDR)} = 0.056\ \text{мс}. \quad (5)$$

На прикладі шейдеру з одним ALU, одним текстурним блоком, одним вхідним і вихідним бітом – теоретичний час роботи буде 0.16 мс. Проте цей шейдер буде обмежений по швидкодії, тому що текстурні блоки є „вузьким місцем” [8].

Звичайно теоретична робота може лише служити вказівками при збільшенні складності шейдера, адже здатність моделювати технічні частини стає дещо важчою. Необхідно відзначити, що представлена модель роботи пам'яті основана на „ідеальних” патернах доступу до пам‘яті (послідовних або високої когерентності). Головна ідеєю є те, що якщо швидкодія в певній задачі близька до теоретичної, то слід внести алгоритмічні зміни.

АВТОМАТИЗАЦІЯ ПРОГРАМУВАННЯ GPGPU НА ОСНОВІ ЗАСТОСУВАННЯ ОБ’ЄКТНО-ОРІЄНТОВАНИХ МЕТОДІВ ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Основними труднощами, які можуть зустрітися при програмуванні відеоадаптерів ATI, є складність написання як власне коду обчислювальних шейдерів, так і коду виклику ATI Close To Metal (CTM). Зокрема, шейдери пишуться на мові ATI Intermediate language (IL), яка є асемблероподібною мовою із жорсткою орієнтацією на графічну специфіку. Для виклику CTM необхідно написати досить велику кількість сервісного коду, який ускладнює реалізацію програми через необхідність чіткого дотримування послідовності виклику функцій CTM. Альтернативою CTM IL є мова AMD/ATI Brook+, яка є розширенням мови C без можливості використання об’єктно-орієнтованого підходу. Програми розробляються на мові C із додатковим синтаксисом, результуюча програма компілюється у код C++, який викликає Brook runtime, який, в свою чергу, викликає CAL runtime. Використання даної мови є неефективним через надмірність генерованого IL-коду.

У даний час у розробці знаходиться відкритий стандарт програмування різних паралельних засобів обчислень, у тому числі відеоадаптерів, під назвою OpenCL [9].

Тому, актуальною є задача позбавлення від деяких із вищезазначених недоліків програмування відеоадаптерів, зокрема спрощення взаємодії програміста із апаратним забезпеченням. В проведених дослідженнях було поставлено за мету розробити спеціалізовану програмну бібліотеку, яка б дозволила програмісту абстрагуватись від специфіки роботи із ATI CTM CAL та дозволила працювати із відеоадаптером як із єдиним об’єктом. Практична цінність даних досліджень полягає у полегшенні написання коду порівняно із типовим сценарієм використання CAL.

У даній роботі розроблена програмна бібліотека, яка являє собою додатковий рівень абстракції над ATI CTM CAL, і призначена для конструювання топологій штучних нейронних і нейроподібних мереж (зокрема паралельно-ієрархічних та ієрарх-ієрархічних) та їх імітаційного моделювання.

Програміст створює лише один об’єкт класу useGPU, з яким проводиться подальша взаємодія. Все що необхідно для видачі завдання на обробку відеоадаптеру – це виділити пам’ять для необхідних вхідних/вихідних даних та констант (функція Allocate), завантажити та скомпілювати програму, написану на IL (функція CreateImage), дочекатись завершення виконання програми (функція Execute) та отримати результати (функція GetResult). Звільнення ресурсів відбувається автоматично у деструкторі класу useGPU. Тобто, загальний порядок використання програмної бібліотеки такий:

Allocate() → CreateImage() → Execute() → GetResult() → ~useGPU().

Необхідно відмітити, що хоча кількість функцій програмної бібліотеки є невеликою, проте сама послідовність виклику є логічно структурованою.

Бажаною для програмістів була б можливість використання при написанні обчислювальних шейдерів об’єктно-орієнтованих принципів. Така можливість реалізована у мові #SL [10]. Проте до її недоліків варто віднести генерацію проміжного HLSL-кода під час виконання програми та орієнтацію на графічну специфіку. Більш доцільним із точки зору навантаження на обчислювальну систему підходом є генерація коду шейдерів на етапі компіляції. Тому планується розвивати дану роботу до рівня платформи програмування відеоадаптерів ATI та забезпечити можливість використання принципів об’єктно-орієнтованого програмування у „C++”- подібній мові.

Оскільки в якості основного призначання запропонованої програмної бібліотеки є конструювання топологій штучних нейронних і нейроподібних паралельно-ієрархічних та ієрарх-ієрархічних мереж для їх імітаційного моделювання, то розглянемо також її основні функціональні можливості в даному контексті. Програмна бібліотека реалізує функції завантаження/збереження

відповідного опису топології мережі у текстових файлах спеціального формату, а також функції для навчання та обробки (проведення сигналу) в нейронній або нейроподібній мережі. Конструювання нейроподібної мережі виконується шляхом об‘єднання між собою шарів нейронних елементів. Шар може містити довільну кількість нейронних елементів, яка обмежується лише максимальним розміром масиву у „.NET framework”, а кількість шарів не обмежується (використовується динамічний список). Таким чином, кількість шарів та нейронних елементів у них обмежуються лише обсягом встановленої оперативної пам‘яті.

Мова реалізації програмної бібліотеки – C# для платформи „MS .NET 2.0”. Функції бібліотеки коректно працюють з різними операційними системами: MS Windows XP (за умови наявності встановленого „MS .NET 2.0”), MS Windows Vista, Linux (за умови встановленої платформи „Mono 1.0”). Для використання програмного продукту, програмна бібліотека має бути відкомпільована з використанням „MS Visual C# Express Edition 2005”. Активація програми – додавання посилання на збірку „NN-Constructor.dll” у проекті та подальша компіляція в MS Visual C# Express Edition 2005.

Робота з програмою відбувається в межах таких основних етапів: завантаження із файлу (або створення користувачем за допомогою відповідних функцій) кількості шарів, зв’язків між ними, кількості нейронних елементів в шарі, зв’язків між нейронними елементами різних шарів нейронної або нейроподібної мережі; обробка вхідної інформації; навчання мережі; збереження топології мережі та результатів моделювання.

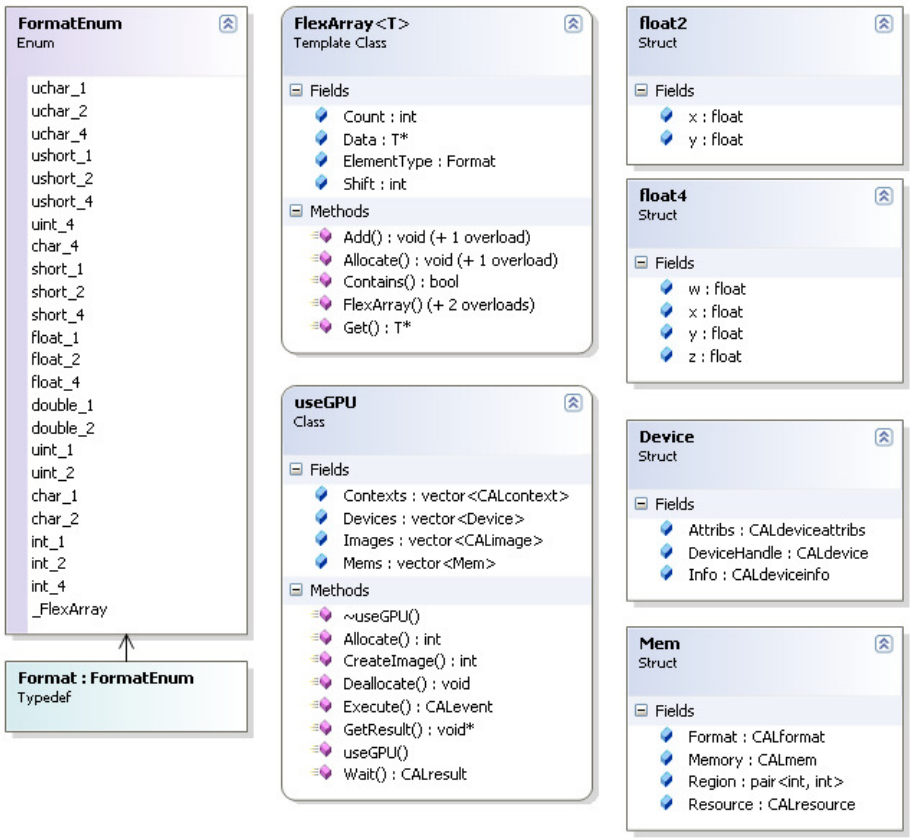

Рис. 3. Класова діаграма програмної бібліотеки „GPU NN-Constructor”

Для роботи програмної бібліотеки необхідна наявність встановленої на комп’ютері операційної системи MS Windows XP/Vista (Linux); наявність MS .NET 2.0 (Mono 1.0); тактова частота процесора не менше 400 MHz; наявність оперативної пам’яті обсягом не менше ніж 128 Mb. Файл програмної бібліотеки займає біля 1 Mb дискового простору.

Розглянемо деякі особливості реалізації алгоритму моделювання нейромереж із урахуванням специфіки програмування паралельних пристроїв. З метою уникнення необхідності реалізації механізму синхронізації паралельних потоків використовуваний алгоритм потребує перетворення формату вхідних даних. Передача імпульсу здійснюється між тактами (такт – сукупність шарів нейронів, сигнал від яких передається одночасно), яка відбувається таким чином:

1. Для кожного нейронного елемента формується одновимірна таблиця, кожний елемент якої являється структурою «номер зв'язаного нейрона у попередньому шарі – вага міжнейронного зв'язку».

2. Таким чином, для кожного шару поточного такту отримується набір таблиць по кількості нейронів у шарі, які характеризують міжнейронні зв'язки.

3. Крім того, для кожного шару формується додаткова одновимірна таблиця, яка містить в собі рівні активації нейронів даного шару.

Таке перетворення дозволяє зберігати дані, необхідні для передачі імпульсу між тактами, у єдиному масиві та завантажувати їх у пам'ять відеоадаптера за один цикл передачі даних. Необхідно відзначити, що така реалізація дозволяє уникнути використання операції довільного запису в пам'ять, яка не підтримується відеоадаптерами ATI на базі мікросхем молодше серії R670, та необхідності реалізації механізму синхронізації між паралельними потоками (яка зменшує швидкість виконання програми).

Розглянемо інші бібліотеки для моделювання нейромереж на відеоадаптерах [3]:

1. NNGPULIB 1.0 (компанія ПАВЛИН, Росія) [11]. У якості апаратної платформи використовується продукція компанії NVidia, починаючи з GeForce 7800. Для доступу до відеоадаптера використовується програмна платформа OpenGL 2.0. Бібліотека дозволяє моделювати лише багатошарові перцептрони та накладає обмеження на вхідні дані (їх кількість має бути кратна 4).

2. Neurala Technology Platform (компанія Neurala, США) [12]. Бібліотека дозволяє моделювати мережі різних топологій, зокрема рекурентні нейронні мережі типу RCF.

3. GNeuron (Raghavendra D Prabhu, Індія) – дозволяє моделювати лише багатошарові перцептрони, для використання відеоадаптера застосовується бібліотека MS Accelerator.

Отже, на основі здійсненого аналізу апаратних платформ у [2-4] та порівняння з існуючими продуктами варто визначити, що розроблена програмна бібліотека (версія для відеоадаптерів) є конкурентоздатною за критеріями кількості топологій, швидкодії та співвідношення «ціна-швидкодія».

ВИСНОВКИ

В ході проведених наукових досліджень було проаналізовано функціонування програмної моделі GPGPU в контексті організації паралельних обчислень в нейроподібних паралельно-ієрархічних системах. А саме, на рівні детального аналізу структурної організації та програмно-апаратних особливостей організації паралельних обчислень в графічних апаратних пристроях (GPU), відзначено можливість та перспективність їх застосування для задач розпізнавання образів та нейроподібної обробки інформації. Також в роботі проаналізовано окремі критерії ефективності функціонування GPU при організації паралельних обчислень, що дає змогу розробляти алгоритмічно складніші шейдерні процесорні структури для GPGPU. На основі наведеного аналізу було поставлено задачу розробки спеціалізованої програмної бібліотеки, яка б дозволила спростити взаємодію програміста із апаратним забезпеченням та працювати із відеоадаптером як із єдиним об'єктом. На основі цього в дослідженні було проведено аналіз існуючих програмних рішень для програмування відеоадаптерів та розроблено програмну бібліотеку, яка пропонує об'єктно-орієнтований підхід до програмування відеоадаптерів ATI. Даний підхід дозволяє суттєво зменшити трудові затрати на написання коду та позбавити програміста від необхідності притримуватись апаратної специфіки при написанні програм. Окреслено шляхи можливого подальшого вдосконалення даної програмної бібліотеки. Також розглянуто функціональні особливості запропонованої програмної бібліотеки в контексті вирішення задачі конструювання топологій штучних нейронних і нейроподібних паралельно-ієрархічних та ієрарх-ієрархічних мереж та їх імітаційного моделювання.

СПИСОК ЛІТЕРАТУРИ

6. Круг П.Г. Нейронные сети и нейрокомпьютеры : учебн. пособие [для студ. высш. учебн. зав. по курсу «Микропроцессоры»] / Круг П.Г. – М.: Издательство МЭИ, 2002. – 176 с. – ISBN 5-7046-0832-9.
7. Апаратна реалізація паралельно-ієрархічної мережі на основі DSP / Кожем'яко В.П., Тимченко Л.І., Яровий А.А., Ремезюк С. : збірник тез доповідей третьої міжнародної науково-технічної конференції [Оптоелектронні інформаційні технології "Фотоніка ОДС–2005"], (Вінниця, 27-28 квітня 2005 р.) – Вінниця: „УНІВЕРСУМ-Вінниця", 2005. – С. 43.
8. Вибір апаратної платформи для реалізації масштабних нейронних та нейроподібних паралельно-ієрархічних мереж [Електронний ресурс] : (IX Міжнародна конференція Контроль і управління в складних системах (КУСС-2008), Вінниця, 21-24 жовтня 2008 року) / А.А. Яровий, Ю.С.

Богомолов, К.Ю. Вознесенський – Режим доступу: http://www.vstu.vinnica.ua/mccs2008/materials/subsection_2.2.pdf.
9. Прикладные аспекты программно-аппаратной реализации нейроподобных параллельно-иерархических систем / Яровой А.А. : Сборник научных трудов. [Научная сессия МИФИ – 2009], [XI Всероссийская научно-техническая конференция «Нейроинформатика-2009»], (Москва, 27-30 января 2009 г.), [В 2 частях. Ч. 2.] – Москва, МИФИ, 2009. – С. 39-48.
10. AMD/ATI StreamComputing SDK [Електронний ресурс] – Режим доступу: http://ati.amd.com/ technology/streamcomputing/index.html.
11. NVidia CUDA [Електронний ресурс] – Режим доступу: http://www.nvidia.com/object/cuda_home.html.
12. RapidMind [Електронний ресурс] – Режим доступу: http://www.rapidmind.net.
13. GPGPU: General Purpose computations on Graphic Processing Unit [Електронний ресурс] – Режим доступу: http://www.gpgpu.org.
14. OpenCL: Open Computing Language [Електронний ресурс] – Режим доступу: http://en.wikipedia.org/ wiki/OpenCL.
15. Объектно-ориентированный подход к шейдерам [Електронний ресурс] – Режим доступу: http://www.dtf.ru/articles/read.php?id=47296&DTFSESSID=fc58ce864752390b052fd34c3fc1f000.
16. Компанія Павлин [Електронний ресурс] – Режим доступу: http://www.pawlin.ru.
17. Neurala Technology Platform [Електронний ресурс] – Режим доступу: http://www.neurala.com/ neurala_amd_press_release_4_8_2008.html.

Надійшла до редакції 27.01.2009р.

ЯРОВИЙ А.А. – к.т.н., доцент кафедри інтелектуальних систем, науковий співробітник кафедри лазерної і оптоелектронної техніки, Вінницький національний технічний університет, Вінниця, Україна.