УДК 004.4

ПРИНЦИПИ ПРОЕКТУВАННЯ GPGPU-ЗАСТОСУВАНЬ НА ОСНОВІ МЕРЕЖ ПЕТРІ

С.Д. Погорілий, Д.Ю. Вітель

Київський національний університет імені Тараса Шевченка,
01601, Київ, вул. Володимирська, 60.
Тел.: (044)521 3590,
E-mail: sdp@univ.net.ua, viteldmitry@gmail.com

Запропоновано низка правил формального представлення виконання GPGPU застосувань за допомогою математичного апарату мереж Петрі. Сформовано моделі виділення та копіювання пам’яті та низка моделей постановки GPU задач в чергу CPU потоками з використанням зазначених правил.

A set of rules for formal GPGPU-application representation in terms of Petri nets was introduced. Models of memory allocation and copying, and CUDA streams scheduling were created using described rules.

Вступ

Процес побудови GPGPU застосувань для сучасних відеоадаптерів – досить складний оскільки вимагає враховувати особливості цільових архітектур. Фірма NVidia пропонує ряд правил створення таких застосувань (Best practices) при використанні платформи CUDA. Етап проектування алгоритму надає змогу суттєво зменшити час розробки останнього та отримати оптимальну за швидкодією його версію.

Тому актуальним є задача побудови моделей сучасних відеоадаптерів з урахуванням особливостей їх роботи. При цьому не обов’язково, щоб дані моделі точно відображали апаратну структуру. Вони мають містити основні правила створення оптимальних застосувань, таким чином обмежуючи розробників від рішень з малою швидкодією ще на етапі проектування.

В роботі за основний апарат проектування паралельних застосувань обрано математичний апарат мереж Петрі. Виокремимо і обґрунтуємо основні задачі (таблиця).

Таблиця

Задача	Актуальність
Створення моделей сучасних відеоадаптерів у термінах мереж Петрі	Дозволить проводити моделювання роботи алгоритму з урахуванням обмежень цільової архітектури
Створення моделей основних шаблонів, що використовуються при побудові GPGPU застосувань	Дозволить будувати алгоритм з основних шаблонів, спрощуючи процес розробки

Створення моделей сучасних відеоадаптерів у термінах мереж Петрі

Основні технології розробки застосувань для GPGPU систем пропонують подібні підходи по створенню оптимальних (максимальних за швидкодією) застосувань. Тому далі розглянуті підходи лише для однієї технології – NVidia CUDA [1–4] – оскільки вона найчастіше використовується у науковій сфері.

Отже при проектуванні GPGPU застосування можна виділити наступні кроки:

- проектування частини, що виконуватиметься на CPU. CPU є керуючим пристроєм при взаємодії з відеоадаптером. Деякі сутності та шаблони цієї взаємодії, моделі яких будуть описані далі;
- потік виконання як конвеєр операції, що виконуються на відеоадаптері;
- синхронні та асинхронні операції з пам’яттю, збільшення швидкодії за рахунок асинхронного виконання різних потоків;
- створення моделей, що описують принципи роботи відеоадаптера:
  - доступ до глобальної пам’яті з урахуванням вирівнювання та послідовності адрес,
  - доступ до спільної пам’яті з урахуванням наявності банків пам’яті.

Модель виконання потоків, розміщення блоків на мультипроцесорах, обмеження апаратних ресурсів, warp як одиниця виконання.

© С.Д. Погорілий, Д.Ю. Вітель, 2014

Методи представлення сутностей за допомогою мереж Петрі

Слід зазначити, що математичний апарат мереж Петрі не накладає ніяких припущень відносно предметної області, до якої він застосовується. Це, в свою чергу, надає свободу при створенні мереж, проте має і негативний фактор: як результат отримані моделі можуть бути неоптимальні за порядком критерій, складні і не ефективні. Тому слід визначити набір правил побудови мереж для конкретної предметної області, вводячи при цьому обмеження на можливі використані конструкції у мережі. Основна ціль – отримані мережі мають бути водночас досить простими, і при цьому відображати наявні в системі механізми, що є важливими для побудови застосувань. При побудові різних мереж однієї предметної області інколи важливо використовувати різні підходи для одних і тих самих чи подібних сутностей (на це впливає контекст використання цієї сутності). Отже наведемо основні правила, що будуть використовуватись далі при побудові мереж (основані на роботах [5–7]):

1. **Представлення потоку виконання**. Виконання інструкцій на CPU чи GPU можна представити у простому випадку міткою без кольору, що вказує на поточну команду. Для одного потоку його мітка в мережі знаходиться у місці перед переходом, що асоційований з наступною командою (чи серією команд) в ньому. Слід зазначити, що інколи необхідно в мережі продемонструвати належність певних даних потоку. Це можна зробити або замінивши безкольорову мітку потоку на кольорову (в багатьох випадках це поганий підхід, оскільки породжує велику кількість кольорів в мережі), або створивши місце-сховище, що має мітку (чи набір міток), асоційованих з потоком. Проте інколи для спрощення структури мережі доцільно замінити місце-сховище на перехід, що відповідає процесу пошуку необхідних даних. До зазначених даних потоку, наприклад може належати його ідентифікатор чи локальні змінні.

2. **Представлення даних**. В процесі моделювання на відміну від процесу виконання досить часто не важливо самі дані, що виникають, переміщуються, змінюються та зникають в системі. Важливішою є саме інформація про дані, тобто метадані: розмір в певних одиницях (байти, біти, слова, …), тип, тип одиниці, ідентифікатор даних (чи адреса в пам'яті) і т. д. В багатьох мережах використовуються саме ці метадані. Проте, якщо алгоритм виявляється залежним від конкретних даних, то в мережі можна використовувати підходи, аналогічні до наведених у попередньому пункті: для отримання значення вводяться або місця-сховища значень даних, або переходи. Також слід зазначити, що такі алгоритми на відеоадаптерах у більшості випадках не є ефективними оскільки їх виконання призводить до розбиття warp-групи на підгрупи, що виконуються послідовно. Важливо уникати узагальнених сховищ даних оскільки останні ускладнюють модель. Так, якщо алгоритм оперує над множиною з трьох векторів доцільно ввести набір місць для кожного з них, а не для усієї множини. Це не тільки спростить мережу у багатьох місцях, але й зменшить кількість метаданих (позбавляємось ідентифікатор вектора).

3. **Ієрархічні мережі**. Далі буде спроектовано ряд мереж, що описують окремі особливості виконання застосувань на відеоадаптері. Деякі з мереж пов'язані з іншими ієрархічними зв'язками (тобто є частинами інших мереж або містять в собі інші мережі). Підмережі складної мережі входять в останню як переходи. В даному випадку важливими є принципи, за якими слід обирати інтерфейс (набір вхідних і вихідних місць у підмережі) взаємодії цих мереж. Цей інтерфейс впливатиме на програмні інтерфейси в подальшому процесі розробки застосувань. Наприклад, розглянемо алгоритм, що приймає на вході 2 параметри – розмір даних і самі данні; і повертає 1 – певне число, що має тип одиниці даних. Входами в таку мережу можна обрати наступні місця: стартове місце для потоку виконання; місце, що має мітку з розміром даних; місце, що має мітку з адресою (ідентифікатором даних). За вихідні місця можна взяти місце виходу потоку виконання, та місце, що містить або результат або його ідентифікатор (адресу) (це місце досить часто можна замінити внутрішнім станом мережі (див. наступний пункт)). Спрощенням в такій мережі є об'єднання вхідних місць для даних в одне з повною метаінформацією про них. Як результат на програмному рівні дане спрощення призведе до того, що алгоритм прийматиме на вхід певний об'єкт, що інкапсулює цю метаінформацію (наприклад, Array, List, Map, Set, …).

4. **Внутрішній стан мережі**. При побудові мережі необхідно, щоб метадані в ній переміщувались якомога менше. Це спростить її структуру, і зробить алгоритм більш очевидним. Так, наприклад, якщо в моделі метадані проходять кожну підмережу майже не змінюючись, доцільно створити набір місць-сховищ для них лише в тих підмережах де вони необхідні.

5. **Рівні абстракції моделі**. Переходи в мережі обираються з урахуванням необхідного рівня абстракції. При деталізації роботи вони можуть бути перетворені в підмережі. Слід намагатись мінімізувати кількість переходів, при цьому зберігаючи особливості роботи системи.

6. **Виключні ситуації**. Побудовані моделі не враховують можливість виникнення помилки етапу виконання: при відсутності CUDA сумісного адаптера, при невірних параметрах запуску ядра, при операціях копіювання, виділення та доступу до пам'яті. Зазначені ситуації лише ускладнять мережу при їх врахуванні. Їх слід враховувати на наступних етапах розробки. На даному ж етапі важливі саме моделі алгоритму та цільової системи.

7. **Створення моделей основних етапів виконання застосування**, а не найбільш узагальненої мо-

делі. Побудова останньої суттєво ускладнює мережу. Так, узагальнюючи процес виділення пам'яті відеоадаптера на невизначену кількість структур даних, отримана мережа включає додаткові переходи і неявні конструкції, що лише ускладнюють розуміння процесу виконання алгоритму. З іншого боку, побудова простих моделей основних шаблонів поведінки дозволяє використовувати останні багато разів в одній мережі, будуючи таким чином повну модель.

8. Набір кольорів та змінних у мережі має бути мінімальним. В різних ситуаціях слід намагатися використовувати вже наявні кольори та змінні.

Моделі виділення та копіювання пам'яті

Почнемо створення моделей з операцій **виділення (allocation) даних** та їх копіювання з RAM до глобальної пам'яті відеоадаптера. Використовуючи зазначені вище правила для операції виділення отримаємо мережу Петрі, що показана на рис. 1.

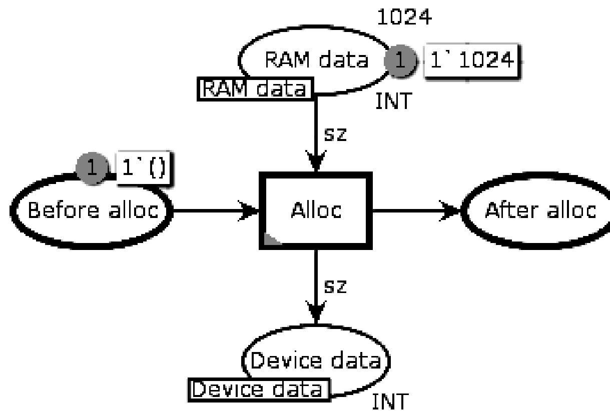


Рис. 1

В даній мережі "Before alloc" і "After alloc" – місця потоку виконання (на рис. 1 і далі такі місця виділені чорним кольором у мережі), мітки яких є безкольоровими (пункт 1). В свою чергу місця "RAM data" і "Device data" є місцями-сховищами для даних в оперативній пам'яті і пам'яті відеоадаптера відповідно (на рис.1 і далі такі місця і їх зв'язки з переходами виділені червоним кольором для RAM даних і фіолетовим для даних на відеоадаптері). Мітки цих місць несуть метаінформацію про дані (в даному випаду про їх розмір). При спрацьовуванні переходу "Alloc" відбувається процес виділення пам'яті розміром, що рівний розміру даних в оперативній пам'яті. Зазначена мережа введена для демонстрації основних принципів, наведених вище. Кожній незалежній структурі даних відповідатиме своє місце сховище. При моделюванні мереж було використано CPN tools, що надає можливість помічати місця маркерами "fusion set" (див. місця-сховища на рис.1 та [8]) і суттєво спрощує таким чином структуру мережі.

Далі розглянемо мережу **синхронного копіювання даних** з RAM на відеоадаптер (рис. 2). Для простоти дана мережа не містить перевірок наявності виділеної пам'яті, ініціалізації GPU, та перевірки розміру даних, що передаються по PCI/PCIe.

Складові частини мережі: потік виконання CPU (виділений чорним), передача по PCI/PCIe (виділена зеленим), потік виконання на GPU (stream 0 по замовчуванню) (виділений синім), місця-сховища для даних в RAM та відеоадаптері. В мережі враховано передачу метаінформації про дані в EXCHANGE_INIT_SIZE перших пакетах. Передача по PCI йде частинами, розмір яких задається константою PCI_SIZE. Дані діляться на пакети переходом "Splitting data into chunks. Sending to buffer" і збираються в одне ціле на стороні адаптера за допомогою переходу "Gathering data from chunks".

При розгляді мережі в рамках апаратно-програмної платформи CUDA слід зазначити, що stream рівний 0 при синхронному копіюванні. Детальніше процес взаємодії потоків відеоадаптера буде розглянуто далі. **Асинхронне копіювання** можливе лише в не нульовому потоці. Його модель подібна до моделі синхронного копіювання. На рис. 3 показана CPU частина, що відрізняється.

В цьому випадку мережа спрощується: CPU вже не очікує закінчення передачі даних по шині.

Потоки відеоадаптера (streams) фактично являють собою чергу команд. Кожен потік має ідентифікатор. Команди в черзі потоку виконуються послідовно. Проте команди різних потоків, окрім нульового, можуть виконуватись паралельно, конкуруючи за ресурси адаптера. Операції, що можуть знаходитись у черзі – операції встановлення чи копіювання пам'яті чи виконання ядра. Оскільки операції потоків на відеоадаптері виконуються незалежно, існує можливість отримати більш оптимальний за швидкодією алгоритм, використовуючи шаблон **Staged concurrent copy**. Нульовий потік виділений: всі операції у черзі нульового потоку очікують виконання операцій в інших потоках і операції в інших потоках очікують виконання операцій нульового потоку.

RAM data
1024
1`1024
In/Out
INT
sz
()
Start exchan
1`()
In
Find data to send in RAM. Pack device address/size into sending package
sz
sz + EXCHANGE_INIT_SIZE*PCI_SIZE
Before exchnage
Data to send
After exchnage
Out
INT
sz
if sz>PCI_SIZE then 1`(sz-PCI_SIZE) else empty
if sz>PCI_SIZE then 1`() else empty
Splitting data into chunks. Sending to buffer
if sz>PCI_SIZE then empty else 1`()
if sz>PCI_SIZE then PCI_SIZE else sz
Send buffer
INT
sz
PCI/PCIe
INT
Packed device address/size in sending data
sz
1`()
()
Before receive
In
Receive buffer
INT
EXCHANGE_INIT_SIZE`PCI_SIZE
sz
if (sz1+sz2)=sz then empty else 1`()
sz2
Gathering data from chunks
sz
Receiving size
sz
Find address in device memory. Unpack device address/size from sending data.
if (sz1+sz2)=sz then empty else 1`sz
INT
if (sz1+sz2)=sz then 1`() else empty
sz1
sz1 + sz2
sz1
0
After receive
Out
Device data
Out
INT

Рис. 2

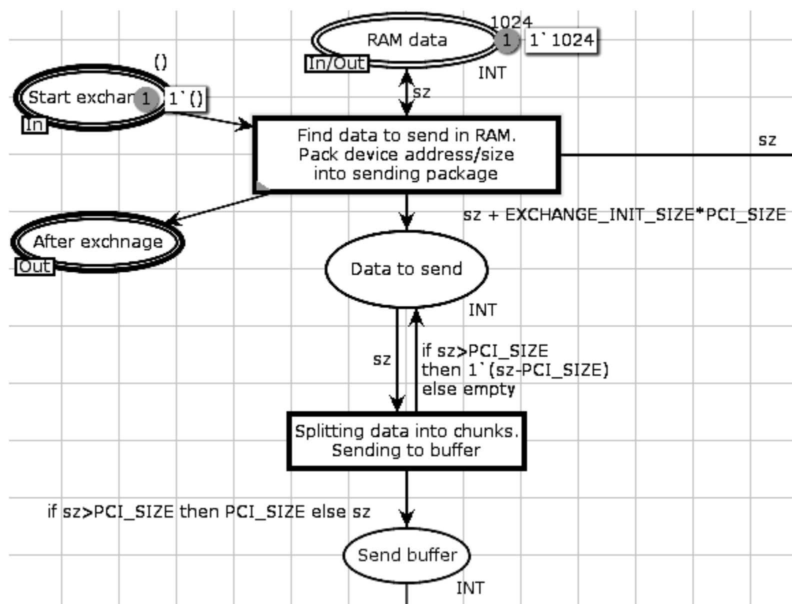


Рис. 3

На рис. 4 показана мережа, що моделює процес постанови завдань у чергу певного потоку GPU. В цій моделі і в наступних операції є асинхронними – CPU не очікує закінчення їх виконання.

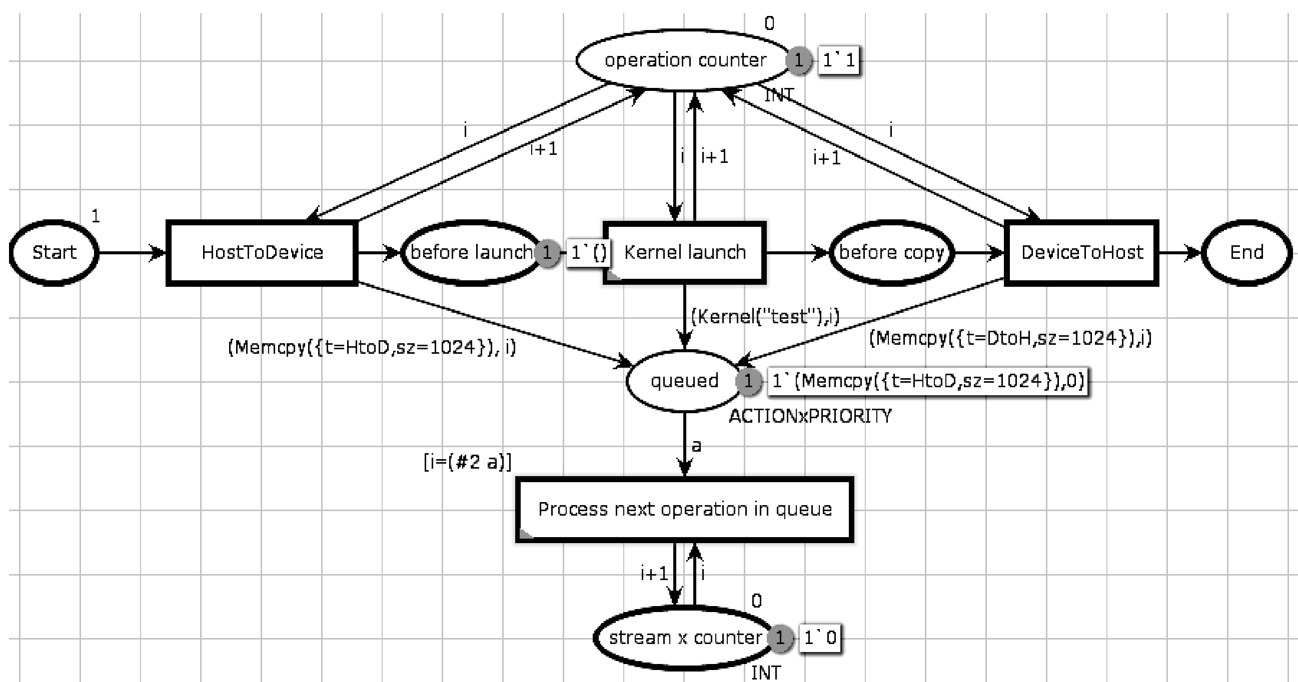


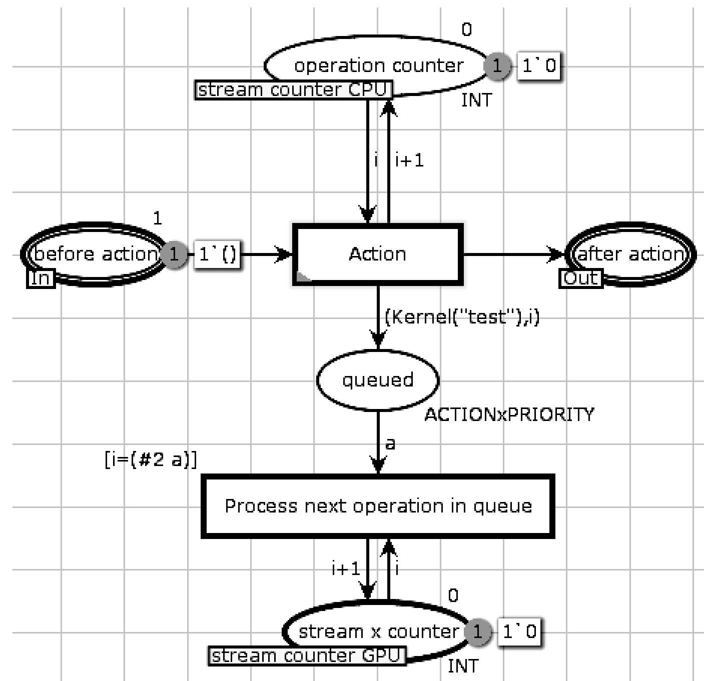
Рис. 4

Дана мережа моделює копіювання даних з RAM у глобальну пам'ять відеоадаптера, виконання ядра (kernel launch) і копіювання даних назад в оперативну пам'ять. Операції представлені у вигляді метаданих: ACTION та ACTIONxPRIORITY. ACTION – колір, що включає усі можливі види операцій у потоці відеоадаптера. Це копіювання пам’яті, що несе в собі інформацію про напрям переміщення даних та їх розмір, та виконання ядра, що ідентифікується за ім’ям (на рис. 4 ядро має назву “test”). ACTIONxPRIORITY додатково включає індекс виконання операції у черзі. Останній задається лічильником в CPU потоці і зчитується для порівняння лічильником GPU потоку. Перехід “Process next operation in queue” є узагальненою для операцій. Його специфікацію у залежності від типу операції буде наведено далі.

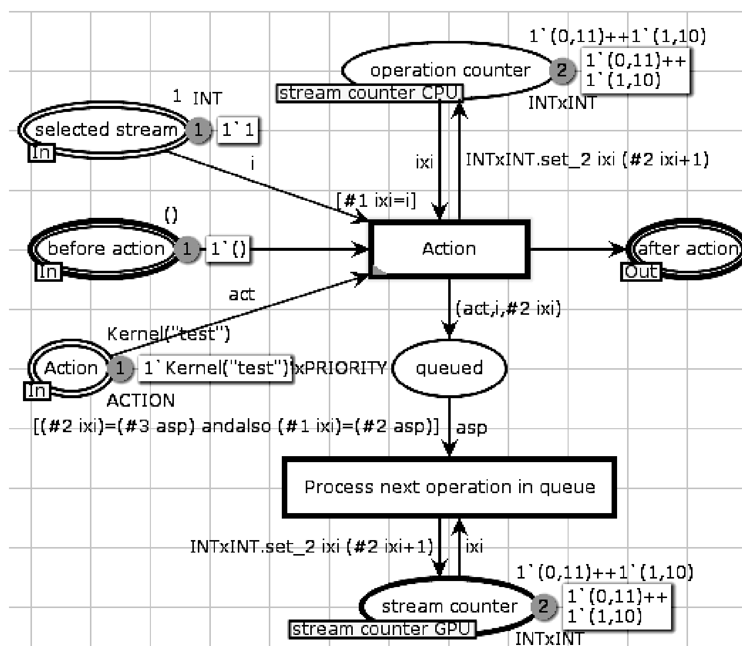
Як NVidia CUDA, так і DirectCompute та OpenCL надають можливість завантажувати ядра (чи шейдери) динамічно (в процесі виконання CPU застосування). Проте в більшості випадків ядро, що виконуватиметься чітко визначене кодом CPU і тому при моделюванні вищезазначеним фактом можна знехтувати. Мережі таких ядер є статичними і можуть бути побудовані аналогічним чином. Інший підхід полягає у можливості введення міток, що самі є мережами Петрі. Фактично будь-яке ядро, що виконується на GPU, завантажується в останній у вигляді даних, що в моделі представлено кольором мітки. Проте таке представлення лише ускладнює процес проектування.

Використовуючи модель на рис. 4 отримуємо мережу на рис. 5, а, що є підмережею постановки нової операції в певний потік GPU в ієрархічній мережі. Рис. 5, б є подальшим її узагальненням на випадок набору потоків. В цьому випадку індекс потоком задається додатково вхідним місцем “selected stream”. Місця “operation counter” і “stream counter” мають містити по одній мітці на кожний активний потік GPU. Ці мітки, в свою чергу, несуть інформацію про номер потоку і кількість або поставлених у чергу операцій (для місця “operation counter”), або кількість виконаних операцій у черзі (для місця “stream counter”). Перехід “Action” помічає операції пріоритетом у черзі GPU потоку. На стороні GPU відбувається послідовний вибір операцій за його значенням.

Модель на рис. 5, б ще не вводить механізм взаємодії потоків. Окрім цього ще одним її недоліком є приховування паралельного виконання операцій у чергах різних потоків. Це виконання реалізоване в переході “Process next operation in queue”, тобто на рівні підмережі зазначеної мережі. Дана ж модель демонструє послідовне виконання переходу з використанням міток різних потоків (тобто порядок операцій у різних потоках не є детермінованим) зі збереженням порядку в чергах кожного потоку. Інша особливість пов'язана із пунктом 2 правил побудови мережі: для набору потоків відеоадаптера було введено лише одне місце-сховище лічильників (що відповідає випадку великої чи невизначеної їх кількості). Модель частково позбавиться від зазначених недоліків і продемонструє інший підхід – набір місць та переходів на кожний потік. Проте до її формування слід провести ряд подальших перетворень.



а



б

Рис. 5

Місця-лічильники операцій (ті, що мають закінчення “counter”) мають бути проініціалізовані. Цей процес відбувається на початку CUDA застосування лише для нульового потоку. Функція cudaCreateStream з CUDA Runtime API виконує його для інших потоків.

Змоделюємо тепер процес взаємодії потоків відеоадаптера, що зазначений на початку розділу. Рис. 4 та 5 б не враховують дану взаємодію. Для цього скористуємось тим фактом, що паралельне виконання операцій у ненульових потоках можна представити як одну операцію в нульовому потоці. Розглянемо простий приклад (рис. 6). Нехай спочатку виконуються n операції у нульовому потоці, потім в потоці i виконуються n_i операцій відповідно, потім k операцій в першому потоці знову і k_i операцій в потоці i. Такий процес аналогічний процесу виконання операцій в черзі. Тобто існує деяка загальна “логічна” черга (global logical queue) виконання операцій, що показана на рис. 6. Тоді мережу на рис. 5, б можна уточнити ввівши додаткові місця-лічильники для неї. На рис. 7 показано мережу Петрі, що враховує взаємодію GPU потоків.

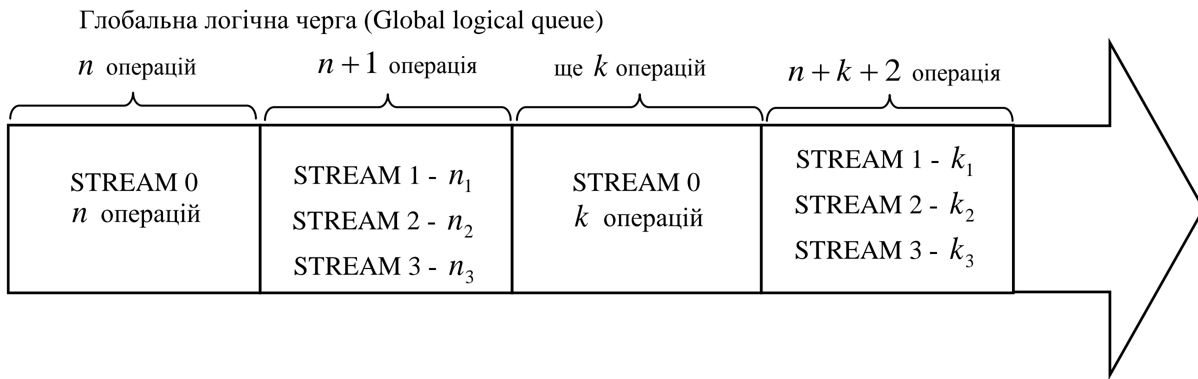


Рис. 6

Рис. 7

В ній відносно попередньої моделі зроблено такі зміни:

1. Введений колір ACTIONxSTREAMxPRIORITIES. Мітки цього кольору мають формат: (action, streamId, (priorityInLQueue, priorityInCurrentStream)), де action – операція, що розміщується в черзі, streamId – ідентифікатор обраного потоку (черги), (priorityInLQueue, priorityInCurrentStream) – пріоритети операції при виконанні. Як на рис. 4, рис. 5, а, б, так і на рис. 7 послідовність операцій у черзі визначається цими пріори-

тетами. Для попередніх моделей (в яких використовувався колір ACTIONxSTREAMxPRIORITY) останні включали лише priorityInCurrentStream – пріоритет операції в обраному потоці. Тобто операції були впорядковані лише в черзі кожного потоку і ніяких припущень щодо послідовності виконання операцій різних потоків не було. В новій моделі пріоритети включають priorityInLQueue, що визначає порядок виконання операції у глобальній логічній черзі (див. рис. 6).

2. Введено додатково чотири місця-лічильники для логічної черги (“Logical stream counter”, “Last operation stream” (виділені зеленим), “Logical stream counter on GPU”, “Last operation stream on GPU”). Таким чином модель реалізує сортування міток у місці “queued” за двома зазначеними пріоритетами. Місця “Last operation stream”, “Last operation stream on GPU” необхідні для відслідковування переходу між появою операцій у нульовому та ненульовому потоках (детальніше вони пояснені при розгляді мережі на рис. 8).

Модель на рис. 7 володіє рядом недоліків, що були зазначені при розгляді моделі на рис. 5, б. Наступна мережа на рис. 8 позбавлена від них.

1. Якщо кількість GPU потоків, що використовується в застосуванні чітко визначена і є невеликою, доцільно розділити місця лічильники на набір місць та переходів однієї структури для кожного потоку (як зазначено в описі представлення даних (див. вище пункт 2 правил побудови моделей). В даній мережі використовується 3 потоки – 0, 1, 2. Нульовий потік є виділеним і тому місця і переходи, що йому відповідають відрізняються від структур інших потоків. Проте перший і другий потоки мають однакові структури в моделі і додати новий (3, 4, 5…) в модель не складно, продублювавши їх.

2. Паралельне виконання операцій для кожної з черг 1, 2, 3…, і послідовне виконання операцій з нульового та інших потоків.

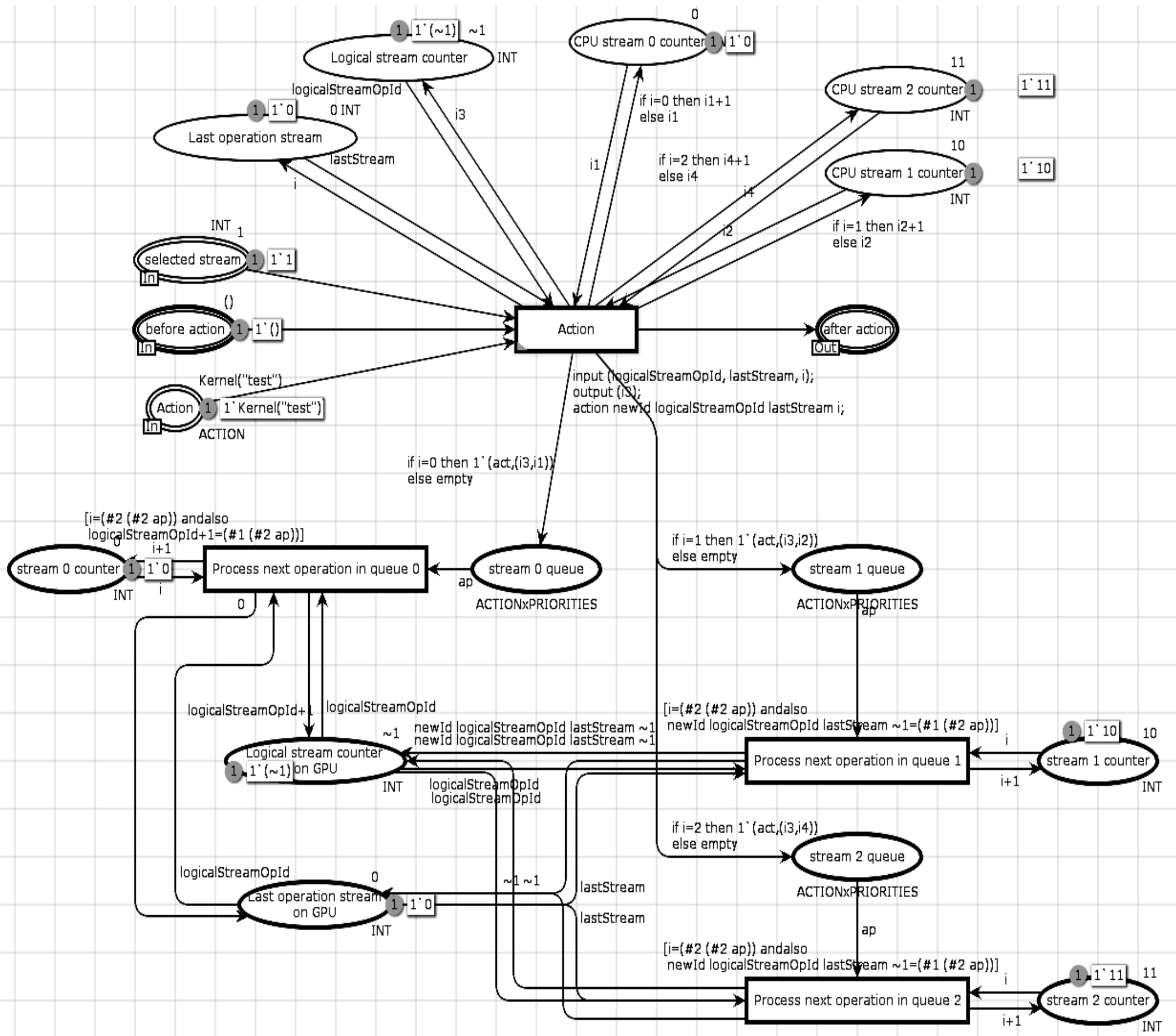


Рис. 8

3. Деякі умови спрацьовування переходів (guards) були спрощені в результаті розділення місць-лічильників. Дані умови тепер відображені в структурі мережі (що робить принципи більш наглядними). Також спростились кольори в мережі (ACTIONxSTREAMxPRIORITIES перетворився в ACTIONxPRIORITIES).

Виділимо подальші задачі:

1. Відображення вибору переходу в залежності від типу операції ACTION: копіюванні даних чи виконання ядра. Даний вибір існує як на стороні CPU (в переході “Action”), так і на стороні GPU потоку (перехід “Process next operation in queue”). Для GPU слід створити єдину структуру для усіх потоків (оскільки цей процес не залежить від їх індексу).

2. Модель з єдиними місцями-лічильниками є більш узагальненою, тому слід реалізувати саме таку ситуацію із збереженням паралельного виконання операцій у чергах ненульових потоків.

3. Необхідно ввести в модель можливість синхронізації CPU та GPU потоків. В цьому випадку CPU очікуватиме завершення операції на GPU.

4. Попередні моделі явно не містять міток потоку виконання (див. пункт 1 правил формування моделей), що не мають кольору. Оскільки GPU потоки виконують одні й ті самі переходи, для того, щоб їх відрізняти ці мітки мають містити інформацію про ідентифікатор потоку. Окрім того, для спрощення структури мережі, вони також несуть додаткову метаінформацію про потік – кількість операцій, що були завершені в ньому.

Висновок

Розглянута актуальність основних задач проектування алгоритмів GPGPU з використанням математичного апарату мереж Петрі.

1. Створення моделей сучасних відеоадаптерів у термінах мереж Петрі.
2. Створення моделей основних шаблонів, що використовуються при побудові GPGPU застосувань.

Запропоновано низка правил формального представлення виконання GPGPU застосувань за допомогою математичного апарату мереж Петрі (див. 8 правил вище). Сформовано моделі виділення та копіювання пам’яті та низка моделей постановки GPU задач в чергу CPU потоками з використанням зазначених правил.

1. *http://*docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
2. *Боресков А.В., Харламов А.А.* Основы работы с технологией CUDA. – ДМК-Пресс, 2010. – 232 с.
3. *CUDA* by Example: An Introduction to General-Purpose GPU Programming, Jason Sanders, Edward Kandrot. Addison-Wesley Professional. Ann Arbor, Michigan, USA, July 2010. – 312 p.
4. *CUDA* Application Design and Development, Rob Farber. Morgan Kaufmann. Waltham, Massachusetts, USA. November 14, 2011. – 336p.
5. *David B. Kirk, Wen-mei Hwu.* “Programming Massively Parallel Processors: A Hands-on Approach”. Published by Elsevier corp.
6. *http://*www.workflowpatterns.com/patterns/
7. *Russell N., ter Hofstede A.H.M., van der Aalst W.M.P., Mulyar N.* Workflow Control-Flow Patterns : A Revised View. BPM Center Report BPM-06-22, BPMcenter.org, 2006.
8. *Погорілий С.Д., Вітель Д.Ю.* Використання мереж Петрі для проектування паралельних застосувань // Проблеми програмування. – 2013. – № 2. – С. 32–40.
9. *Погорілий С.Д., Калита Д.М.* Оптимізація алгоритмів маршрутизації з використанням систем алгоритмічних алгебр // УСиМ. – 2000. – № 4. – С. 20–30.
10. *Pogorilyy S.D., Gusarov A.D.* Paralleling Of Edmonds-Karp Net Flow Algorithm // Appl. Comput. Math. 5. – 2006. – N 2. – P. 121–130.
11. *Levchenko R.I., Sudakov O.O., Pogorilyy S.D.* DDCI: Simple Dynamic Semiautomatic Parallelizing for Heterogeneous Multicomputer Systems // Proceedings of the 5th IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 21 – 23 September 2009, Rende (Cosenza), Italy.