УДК 004.4

ФОРМАЛІЗОВАНИЙ МЕТОД ПРОЕКТУВАННЯ ЗАСТОСУВАНЬ В ТЕХНОЛОГІЇ GPGPU

С.Л. Кривий, С.Д. Погорілий, М.С. Слинько

Запропоновано метод дослідження характеристик систем, що використовують високопродуктивні обчислення, який ґрунтується на апараті транзиційних систем (дискретної моделі обчислень). Запропоновано два варіанти обмежень синхронного добутку цих транзиційних систем, що моделюють підхід, використаний в архітектурі Nvidia CUDA. Описано транзиційні системи, що представляють два типи інструкцій, процес виконання інструкції варпом та роботу планувальника варпу. Виконано формалізацію моделі виконання GPGPU-застосування. Отримано специфікацію вищевказаного підходу та строго доведено його коректність. Специфікацію зведено до двох варіантів мереж Петрі, які дозволяють виявляти помилки проектування в автоматичному або напівавтоматичному режимі.
Ключові слова: Nvidia CUDA, GPGPU, САА, мережі Петрі, транзиційна система.

Предложен метод исследования характеристик систем, использующих высокопроизводительные вычисления, основанный на аппарате транзиционных систем (дискретной модели вычислений). Предложено два варианта ограничений синхронного произведения транзиционных систем, моделирующих подход, использованный в архитектуре Nvidia CUDA. Описаны транзиционные системы, представляющие два типа инструкций, процесс выполнения инструкции варпа и работу планировщика варпа. Выполнена формализация модели выполнения GPGPU-приложения. Получено спецификацию вышеуказанного подхода и строго доказана его корректность. Спецификацию сведено к двум вариантам сетей Петри, которые позволяют выявлять ошибки проектирования в автоматическом или полуавтоматическом режиме.
Ключевые слова: Nvidia CUDA, GPGPU, САА, сети Петри, транзиционная система.

The method of researching systems with high-performance computing support, based on the transition systems apparatus (discrete computational model), is proposed. Two variants of synchronous product limitations of transition systems that model tha Nvidia CUDA approach are proposed. transition systems that represent two types of instructions, process of the warp instruction execution, and the process of warp scheduling were described. GPGPU application execution model was formalized and its correctness was proved. Two variants of the relevant Petri net which allowed automatic or semi-automatic detection of design errors were obtained.
Key words: Nvidia CUDA, GPGPU, SAA, Petri net, transition system.

Вступ

Майбутнє високопродуктивних обчислень (та так званих «exascale» обчислень) лежить в галузі систем з масовим паралелізмом [1]. В свою чергу, використання таких систем унеможливлює інженерне проектування і вимагає наявності формалізованого математичного апарату і методів представлення алгоритму. В роботі основний акцент ставиться на алгоритмічному етапі проектування. Алгоритмічний етап проектування є початковим та найменш технологічно забезпеченим. Але він впливає на подальший процес розробки та визначає її успіх в цілому. Ефективним методом алгоритмічного проектування і дослідження паралельних алгоритмів є використання алгебр алгоритмів, які дозволяють сформувати формули (схеми) алгоритмів, що залежать від різних параметрів, якими можуть виступати програмно-апаратні платформи, парадигми паралельного програмування тощо. Зокрема, В.М. Глушковим було запропоновано математичний апарат систем алгоритмічних (мікропрограмних) алгебр (САА) [2]. Приклад представлення GPGPU застосувань у апараті САА наведено у [3, 4]. Однак, хоч САА і дозволяють осягнути структуру застосування і проводити еквівалентні перетворення схеми застосування без прив'язки до апаратної архітектури, в них відсутня можливість темпорального моделювання та глибокого аналізу. Саме тому в роботі пропонується використання апарату транзиційних систем, що дозволяють створювати різнорівневі абстракції, та можуть бути зведені до мереж Петрі (МП), котрі підтримують темпоральне моделювання.

На початку статті описуються особливості розробки застосувань в технології GPGPU, що є критичними для дослідження їх властивостей та моделювання. Далі описується метод проектування застосувань, що базується на використанні апарату транзиційних систем та їх синхронного добутку. Потім проводиться аналіз властивостей МП, що моделює гетерогенну систему в цілому. В процесі проектування систем визначення синхронного добутку виконується на етапі проектування, а використання МП, як специфікації, є засобом перевірки правильності прийнятих проектних рішень. Метод застосовується для обґрунтування властивостей процесу виконання застосувань на архітектурі Nvidia CUDA.

При проектуванні, за наявності вихідного коду конкретного застосування можливо використати наведену модель для аналітичного обґрунтування його коректності, в тому числі, за допомогою автоматизованих засобів, таких як CPN Tools.

Архітектури сучасних GPU від NVIDIA

Інтенсивний розвиток графічних процесорів (GPU), призвів до створення пристроїв, які, окрім управління графічним дисплеєм, також дозволяють виконувати алгоритми неграфічних задач. CUDA (Computed

Unified Device Architecture) – це архітектура паралельних обчислень, розроблена компанією NVIDIA для спрощення програмування GPGPU за рахунок використання високорівневих API.

Застосування, що працює в гетерогенному середовищі, обладнаному GPU, може бути розділене на наступні частини:

- «хост» – блоки інструкцій, що виконуються на центральному процесорі;
- функції-«ядра» – блоки інструкцій, що виконуються на GPU.

Хост-блоки визначають контекст виконання функцій-ядер, займаються передачею даних між оперативною пам'яттю комп'ютера та пам'яттю GPU. Функції-ядра мають доступ до великої (до 655360 на відеоадаптерах архітектури Pascal) кількості потоків. За таксономією Флінна, принцип роботи CUDA пристроїв є близьким до принципу SIMD (Singe Instruction – Multiple Data), з деякими уточненнями: кожен потік не лише виконує одну й ту ж саму інструкцію над своєю підмножиною даних, але й може мати власний шлях виконання в залежності від заданих умов. Розробники CUDA називають такий підхід SIMT (Single Instruction, Multiple Thread), тобто одна інструкція виконується одночасно багатьма потоками, при чому поведінка кожного окремого потоку нічим не обмежується. У такий спосіб, основною програмною обчислювальною одиницею є потік. Всі GPU потоки структуровані у блоки однакового розміру, котрі, в свою чергу, групуються у сітку [5].

Блок потоків – це набір одночасно виконуваних потоків (насправді, не усі потоки виконуються одночасно, як буде показано нижче). Взаємодія між потоками в блоці досягається за рахунок механізму бар'єрної синхронізації. Також потоки в межах блоку мають доступ до спільної пам'яті. Розмір блоку залежить від апаратної платформи (на сучасних архітектурах, зокрема, Kepler, підтримуються блоки розмірами до 1024 потоків). Кожен блок має ідентифікатор, унікальний в межах сітки, а кожен поток – ідентифікатор, унікальний в межах блоку.

Сітка – це масив блоків, що виконують одну і ту саму функцію-ядро. Кількість блоків у сітці визначається розмірністю даних та апаратною платформою. Обмін даними між блоками зазвичай відбувається з використанням глобальної пам'яті.

Апаратна платформа GPU від Nvidia складається з набору потокових мультипроцесорів (streaming multiprocessor, SM). Блоки потоків розподіляються між SM таким чином, що всі потоки в межах блоку виконуються на одному мультипроцесорі, і в той же час, одному SM може бути поставлено у відповідність більше, ніж 1 блок. В свою чергу, мультипроцесор розділяє блок на групи по 32 потоки. Ця операція виконується для кожного блоку окремо, а утворені групи називаються варпами (warp). Як згадано вище, кожен SM працює за принципом SIMT, тобто всі потоки в межах варпу в один момент часу виконують одну і ту саму інструкцію. Таким чином, фізично паралельно виконуються лише потоки в межах одного варпу. Будь-яке розгалуження в інструкціях (умовні оператори тощо) призводить до того, що кожен шлях виконання буде обчислюватися окремо і послідовно.

Кожен SM працює з багатьма варпами. SM має вбудований планувальник варпу та диспетчер інструкцій (в архітектурі Pascal їх може бути до 4-х). Планувальник обирає варп, інструкція якого може бути виконана, і передає контроль над обчислювальними ядрами його потокам. Якщо варп з будь-яких причин переходить в стан очікування (синхронізація, очікування доступу до пам'яті тощо), планувальник обирає для виконання інший варп, забезпечуючи постійне навантаження ядер SM.

Враховуючи багаторівневість обчислювальної архітектури та велику кількість обчислювальних одиниць (потоків, блоків), дослідження і виправлення помилок проектування в ручному режимі є неможливим. Доцільно застосувати математичну модель загального типу, таку, як транзиційну систему, та використати апарат мереж Петрі для виявлення помилок в автоматичному або напівавтоматичному режимі.

Транзиційні системи та мережі Петрі

Як показано в [6], транзиційна система (ТС) – це система $A = (S, T, \alpha, \beta, s_0)$, де

- S – скінченна або нескінченна множина станів;
- T – скінченна або нескінченна множина переходів;
- α, β – два відображення із T в S, що ставлять у відповідність кожному переходу t два стани $\alpha(t), \beta(t)$, які називають відповідно початком і кінцем переходу t;
- s_0 – початковий стан ТС.

Нехай $A_1, ..., A_n$ – транзиційні системи, де $A_i = (S_i, T_i, \alpha_i, \beta_i, s_0^{'})$, $i = 1, ..., n$. Обмеженням синхронізації називається підмножина Т множини $(T_1 \cup \{\varepsilon\}) \times ... \times (T_n \cup \{\varepsilon\}) \setminus \{(\varepsilon, ..., \varepsilon)\}$, де ε - тотожний перехід, який означає відсутність будь-якої дії в ТС. Елементи із Т називаються глобальними переходами. Якщо $t = (t_1, ..., t_n) \in T$ і $t_i \neq \varepsilon$, то говорять, що ТС A_i бере участь у переході t.

Кортеж $A=(A_1,...,A_n,T)$ називається добутком ТС $A_1,...,A_n$, які називаються компонентами A. Інтуїтивно глобальний перехід $t=(t_1,...,t_n)$ моделює можливі переходи в $A_1,...,A_n$. Якщо перехід $t_i=\varepsilon$, то ТС A_i не приймає участі у глобальному переході t.

Мережа Петрі (МП) (P,T,F,M_0) зображує добуток $A=(A_1,...,A_n,T)$ ТС $A_i=(S_i,T_i,\alpha_i,\beta_i,s_0^{'})$, де $A_i\cap A_j=\varnothing$ при $i\neq j$, $i,j=1,2,...,n$, якщо $P=S_1\cup S_2\cup...\cup S_n$, $T=T$, $F=\{(s,t)\,|\,t_i\neq\varepsilon\,\&\,s=\alpha_i(t_i)\}\cup\{(t,s)\,|\,t_i\neq\varepsilon\,\&\,s=\beta_i(t_i)\}$ для деякого $i\in\{1,2,,...,n\}$, де t_i означає *i*-ту компоненту $t\in T$, $M_0=(s_0^1,s_0^2,...,s_0^n)$.

Семантика добутку ТС і семантика МП, яка його зображує, узгоджуються в тому сенсі, що послідовність глобальних переходів $t_1,...,t_k$ являє собою глобальну історію добутку ТС тоді і тільки тоді, коли вона є допустимою послідовністю спрацювань переходів в МП.

Розробка моделі виконання CUDA-застосування

З точки зору моделювання архітектури обчислень, виконання GPGPU-застосування можна представити у вигляді взаємодії таких систем:

- ТС 1: моделює абстрактну інструкцію;
- ТС 2: моделює варп, що містить набір інструкцій та послідовно їх виконує;
- ТС 3: моделює виконання блоку на потоковому мультипроцесорі.

Розглянемо функціональні можливості цих транзиційних систем.

ТС 1. Залежно від мети моделювання, доцільно змінювати акцент в представленні інструкції у вигляді транзиційної системи. В роботі представлено 2 моделі:

- ТС 1А: представляє собою узагальнену інформаційну інструкцію (рис. 1). Варто використовувати у випадках, коли потрібен прогноз ефективності застосування за рівнем використання певного ресурсу (наприклад, часу);
- ТС 1Б: представляє собою детальну абстракцію обчислювальної інструкції. Варто використовувати у випадках, коли потрібен детальний аналіз «вузького місця» застосування.

ТС 1А. Час виконання різних типів інструкцій може відрізнятися на кілька порядків (наприклад, арифметична інструкція в архітектурі Kepler виконується за 4 обчислювальні такти, а інструкція доступу до глобальної пам'яті може займати від 400 тактів). Абстрактна інформаційна інструкція моделюється простою транзиційною системою $B_A=(\{b_0,b_1,b_2,b_3,b_4,b_5\},\ \{p_1,p_2,p_3,p_4,p_5,p_6\},\alpha,\beta\ b_0)$.

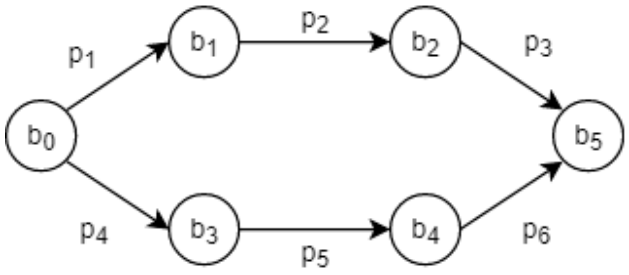

Рис. 1. Представлення інформаційної інструкції у вигляді ТС

Інтерпретація місць та переходів в даній ТС така:

b_0 – фіксація старту виконання;

p_1 – підготовка до виконання арифметичної операції;

b_1 – фіксація старту виконання арифметичної операції;

p_2 – виконання операції;

b_2 – арифметичну операцію виконано;

p_3 – перехід у стан завершення інструкції;

p_4 – формування запиту до пам'яті;

b_3 – фіксація старту запиту до пам'яті;

p_5 – процес отримання даних;

b_4 – фіксація отримання даних;

p_6 – перехід у стан завершення інструкції;

b_5 – інструкцію завершено.

В цій моделі враховуються 2 типи інструкцій – арифметична та інструкція доступу до пам'яті. Втім, час доступу до різних видів пам'яті GPU відрізняється на порядки, тому в реальній моделі може бути доцільним виділення додаткових типів інструкцій, та, відповідно, додаткових місць та переходів за вищевказаним зразком. Також варто зазначити, що кожна реальна інструкція належить лише до одного з типів. З метою спрощення узагальненої моделі, вважаємо, що тип інструкції визначається *випадковим чином*.

ТС 1Б. Обчислювальна інструкція (рис. 2) з акцентом на внутрішню будову моделюється наступною транзиційною системою $B_B = (\{b_0, b_1, b_2, b_3, b_4, b_5\},\ \{p_1, p_2, p_3, p_4, p_5, p_6\}, \alpha, \beta\ b_0)$.

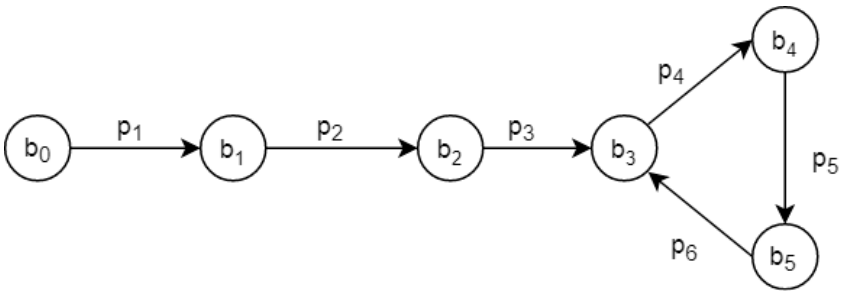

Рис. 2. Представлення обчислювальної інструкції у вигляді ТС

Інтерпретація місць та переходів в даній ТС така:

b_0 – фіксація старту виконання;

p_1 – формування запиту операндів з пам'яті;

b_1 – фіксація старту запиту операндів з пам'яті;

p_2 – завантаження операндів;

b_2 – фіксація отримання операндів;

p_3 – перехід у стан завершення інструкції;

b_3 – інструкцію завершено;

p_4 – підготовка до виконання обчислень;

b_4 – фіксація старту виконання обчислень;

p_5 – процес виконання обчислень;

b_5 – обчислення завершено.

p_6 – перехід у стан завершення інструкції.

ТС 2. Кожен варп містить набір інструкцій, що мають бути виконаними всіма його потоками. За моделлю SIMT інструкції надходять послідовно, а кожна з них виконується одночасно всіма потоками варпу (рис. 3). Такий протокол моделюється транзиційною системою $A = (\{a_0, a_1, a_2, a_3\},\ \{r_1, r_2, r_3, r_4\}, \alpha, \beta\ a_0)$.

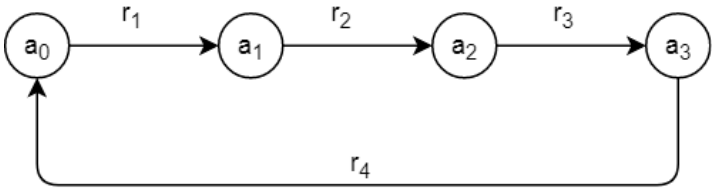

Рис. 3. ТС виконання інструкції варпом

Інтерпретація місць та переходів в даній ТС така:

a_0 – фіксація стану деактивації варпа. Варп готовий до виконання наступної інструкції;

r_1 – активація варпа планувальником (даний варп отримує доступ до ресурсів SM для виконання однієї інструкції);

a_1 – фіксація стану активного варпу;

r_2 – підготовка до виконання інструкції;

a_2 – фіксація стану варпу в процесі виконання інструкції («зайнятого» варпу);

r_3 – отримання підтвердження про виконання інструкції;

a_3 – інструкцію виконано;

r_4 – деактивація варпу (після переходу варп знову стає доступним для вибору планувальником).

ТС 3. Як вказано вище, ТС 3 представляє собою виконання блоку на SM. Основною структурною одиницею в даному процесі за означенням відіграє планувальник варпу (рис. 4). Основні дії на цьому етапі включать вибір варпу та інструкції і забезпечення ексклюзивного доступу до обчислювальних ресурсів на час виконання кожної пари варп-інструкція. Ця пара дій має ітеративно виконуватися, поки всі інструкції не буде виконано на всіх варпах блоку. Такий процес моделюється транзиційною системою $C=(\{c_0, c_1, c_2, c_3\}, \{q_1, q_2, q_3, q_4\}, \alpha, \beta\ c_0)$.

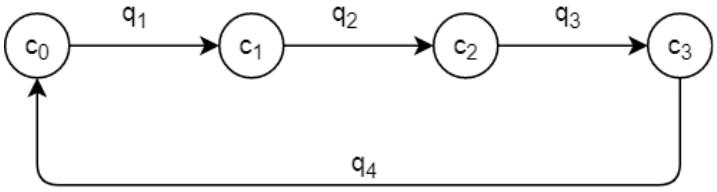

Рис. 4. Представлення роботи планувальника варпу у вигляді ТС

Інтерпретація місць та переходів в даній ТС наступна:

c_0 – фіксація старту планування;

q_1 – вибір варпу та інструкції;

c_1 – обрано пару варп-інструкція;

q_2 – початок виконання обраної інструкції для обраного варпу;

c_2 – фіксація стану очікування виконання інструкції;

q_3 – отримання підтвердження про виконання інструкції;

c_3 – фіксація виконання пари варп-інструкція;

q_4 – перехід до наступної ітерації.

За наявністю сформованих транзиційних систем можна побудувати модель виконання застосування в архітектурі CUDA у вигляді синхронного добутку **ТС 1А, 2, 3** з глобальними переходами:

$$T_1 = \{(\varepsilon, r_1, q_1), (p_1, r_2, q_2), (p_2, \varepsilon, \varepsilon), (p_3, r_3, q_3), (p_4, r_2, q_2)$$

$$(p_5, \varepsilon, \varepsilon), (p_6, r_3, q_3), (\varepsilon, r_4, \varepsilon), (\varepsilon, \varepsilon, q_4)\}. \quad (1)$$

За множиною обмежень синхронного добутку T_1 будуємо мережу Петрі, що моделює сумісну роботу всіх підсистем [6] (рис. 5).

Після побудови мережі Петрі, виникає задача перевірки коректності отриманої моделі. В рамках статті перевіримо живучість такої системи. Характеристика «живучості» означає, що в отриманій моделі всі переходи будуть приймати участь у процесі її функціонування [6]. Якщо певні переходи в МП ніколи не спрацьовують, це означає, що проект системи побудовано *неправильно*, або ж він є *надлишковим*.

Дослідимо МП на живучість у такий спосіб: перевіримо, чи досяжна кінцева розмітка з початкової, і чи всі переходи в цьому процесі спрацьовують. Початкова розмітка має вигляд:

$$M_0 = (1, 1, 0, 0, 0, 0, 0, 0, 1),$$

тобто існує 1 планувальник варпу (токен в місці № 1), 1 інструкція для виконання (токен в місці № 2) та відсутня інструкція в процесі (токен в місці № 9). Кінцева розмітка має вигляд:

$$M_k = (1, 0, 0, 0, 0, 0, 0, 0, 1),$$

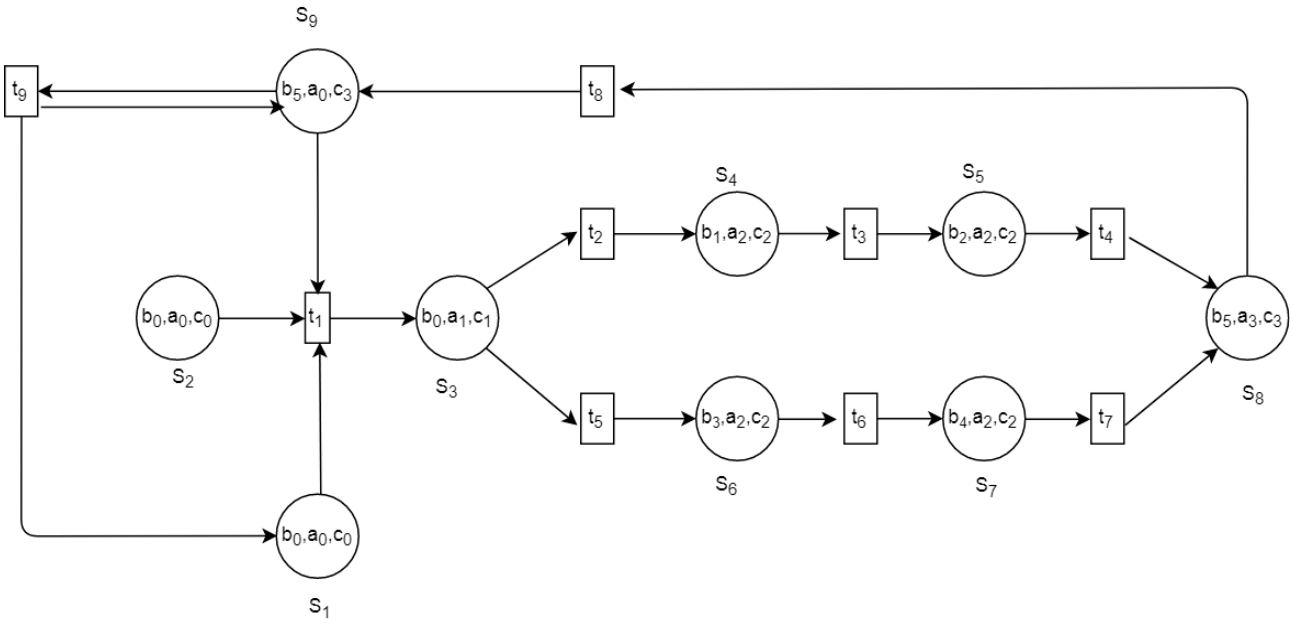

Рис. 4 МП, що представляє синхронний добуток ТС з множиною переходів T_1

тобто всі інструкції виконано, активних варпів немає, планування не відбувається. Знайдемо розв'язки рівняння стану вигляду:

$$\begin{aligned} A \cdot x &= M_k - M_0 \Rightarrow \\ A \cdot x &- (M_k - M_0) = 0, \end{aligned} \tag{2}$$

де A – матриця інцидентності МП.
Для нашої МП рівняння стану має вигляд (табл. 1).

Таблиця 1

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	$-(M_k - M_0)$
S_1	-1	0	0	0	0	0	0	0	1	0
S_2	-1	0	0	0	0	0	0	0	0	1
S_3	1	-1	0	0	-1	0	0	0	0	0
S_4	0	1	-1	0	0	0	0	0	0	0
S_5	0	0	1	-1	0	0	0	0	0	0
S_6	0	0	0	0	1	-1	0	0	0	0
S_7	0	0	0	0	0	1	-1	0	0	0
S_8	0	0	0	1	0	0	1	-1	0	0
S_9	-1	0	0	0	0	0	0	1	0	0

Застосовуючи алгоритм TSS [7] для вирішення рівняння стану з вищевказаною матрицею, отримуємо наступні вирішення рівняння стану з вищевказаною матрицею (табл. 2).

Таблиця 2

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
1	1	1	1	0	0	0	1	1

1	0	0	0	1	1	1	1	1

Як видно з множини розв'язків, всі переходи в МП з вищевказаними початковою і кінцевою розмітками живі (значення, що відповідає кожному переходу, хоча б у одному з розв'язків є позитивним), крім того, в один момент часу виконуються переходи, що відповідають лише одному з можливих типів інструкцій.

Недоліком МП на рис. 5 є неможливість покриття її позитивними інваріантами через наявність місця $\mathbf{S_2}$, яке представляє собою набір інструкцій для виконання, і тому живучість мережі в цілому (а не для конкретних пар початкової та кінцевої розміток) визначити неможливо. Крім того, важливо довести коректність взаємодії ТС 2 та ТС 3 незалежно від форми представлення інструкції. Для аналізу обох питань побудуємо іншу множину обмежень синхронного добутку **ТС 1, 2, 3** (рис. 6), але цього разу як представлення інструкції використаємо **ТС 1Б**:

. (3)

Рис. 6. МП, що представляє синхронний добуток ТС з множиною переходів

Перевіримо живучість МП на рис. 6. В цій мережі можливо знайти інваріанти переходів, незалежно від початкової і кінцевої розмітки. Для цього знайдемо розв'язки рівняння стану . Матриця інцидентності має вигляд (табл. 3).

Таблиця 3

	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	$\mathbf{t_4}$	$\mathbf{t_5}$	$\mathbf{t_6}$	$\mathbf{t_7}$	$\mathbf{t_8}$	$\mathbf{t_9}$
$\mathbf{S_1}$	-1	0	0	0	0	0	0	0	1
$\mathbf{S_2}$	-1	0	0	0	0	0	0	0	1
$\mathbf{S_3}$	1	-1	0	0	0	0	0	0	0
$\mathbf{S_4}$	0	1	-1	0	0	0	0	0	0

S_5	0	0	1	-1	0	0	0	0	0
S_6	0	0	0	1	-1	0	0	0	0
S_7	0	0	0	0	1	-1	0	0	0
S_8	0	0	0	0	0	1	-1	0	0
S_9	0	0	0	0	0	0	1	-1	0
S_{10}	0	0	0	0	0	0	0	1	-1

Застосовуючи алгоритм TSS, отримуємо єдиний розв'язок рівняння стану:

Таблиця 4

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
1	1	1	1	1	1	1	1	1

Це означає, що всі переходи покриваються позитивними інваріантами, і мережа завжди буде живою.

Висновки

Основним результатом статті є узагальнена модель обчислень в архітектурі Nvidia CUDA, що базується на апараті транзиційних систем та мереж Петрі. Створення гетерогенних застосувань з використанням новітніх архітектур відеоадаптерів порушує нагальну проблему їх верифікації, при чому, концептуальні помилки бажано знаходити ще на етапі проектування., що досягається запропонованим підходом.

В роботі пропонується використання математичного апарату транзиційних систем для отримання формалізованих специфікацій систем, що проектуються. Такі специфікації є апаратно-незалежними та інтуїтивно зрозумілими. Крім того, до переваг сформованої моделі належить можливість її трансформування до мережі Петрі, і подальшої верифікації автоматизованими засобами, такими як CPN Tools.

Отримано узагальнену модель виконання застосування з використанням Nvidia CUDA, що надало можливість строго довести коректність підходу, що використовується у вищевказаній архітектурі.

В роботі проведено дослідження двох властивостей системи – досяжності та структурної живучості. Однак поєднання апарату ТС та МП дозволяє також аналізувати контрольованість, структурну обмеженість, несуперечність тощо; дозволяє проводити аналіз циклів, пасток, дедлоків, інваріантів, тобто проводити максимальний набір досліджень для забезпечення надійного функціонування застосування.

Література

1. Погорілий С.Д., Вітель Д.Ю., Верещинський О.А. Новітні архітектури відеоадаптерів. Технологія GPGPU. Частина 1. Реєстрація, зберігання і оброб. даних. 2012. Т. 14, № 4.
2. Pogorilyy S.D. & Shkulipa I. Yu. (2009) A Conception for Creating a System of Parametric Design of Parallel Algorithms and their Software Implementations. Cybernetics and System Analysis, Volume 45, Issue 6 (November 2009). P. 952–958. Springer Science and Business Media Inc. ISSN:1060-0396.
3. Pogorilyy S.D., Slynko M.S. (2016) Research and development of Johnson's algorithm parallel schemes in GPGPU technology. In 10-th international scientific programming conference UKRPROG'2016. Kyiv. P. 105–112.
4. Pogorilyy S.D., Slynko M.S. & Rustamov Y.I. (2017) A formalized method of Johnson's algorithm parallelization suitable for use in GPGPU technology. TWMS Journal of Pure and Applied Mathematics, V. 8, N.1. 2017. P. 12–21.
5. Погорелый С.Д., Бойко Ю.В., Трибрат М.И., Грязнов Д.Б. Анализ методов повышения производительности компьютеров с использованием графических процессоров и программно-аппаратной платформы CUDA. Математичні машини і системи. 2010. № 1. С. 40–54.
6. Бойко Ю.В., Кривий С.Л., Погорілий С.Д. та ін. Методи та новітні підходи до проектування, управління і застосування високопродуктивних ІТ-інфраструктур. – К.: ВПЦ «Київський університет». 2016. 447 с.
7. Кривий С.Л. Лінійні діофантові обмеження та їх застосування. Букрек: Чернівці. 2015. 224 с.

References

1. POGORILYY S.D., VITEL D. Yu. & VERESCHINSKY O.A. (2012) *Modern video adapter architectures. GPGPU technology. Part 1*. Data registration, storage and processing. 14 (4).

2. POGORILYY S.D. & SHKULIPA I. Yu. (2009) *A Conception for Creating a System of Parametric Design of Parallel Algorithms and their Software Implementations*. Cybernetics and System Analysis, Volume 45, Issue 6 (November 2009) pages: 952 - 958. Springer Science and Business Media Inc. ISSN:1060-0396.
3. POGORILYY S.D., SLYNKO M.S. (2016) *Research and development of Johnson's algorithm parallel schemes in GPGPU technology.* In 10-th international scientific programming conference UKRPROG'2016. Kyiv, pp. 105-112.
4. POGORILYY S.D., SLYNKO M.S. & RUSTAMOV Y.I. (2017*) A formalized method of Johnson's algorithm parallelization suitable for use in GPGPU technology*. TWMS Journal of Pure and Applied Mathematics, V.8, N.1, 2017, pp. 12-21.
5. POGORILYY S.D., BOYKO Yu.V., TRYBRAT M.I., GRYAZNOV D.B. (2010) *Analysis of the computer performance improvement methods using graphic processors and CUDA platform*. Mathematical Machines and Systems, N.1, pp. 40-54.
6. BOYKO Yu.V., KRYVYI S.L., POGORILYY S.D. et al. (2016). *Methods and latest approaches to the design, management and application of high-performance IT infrastructures*. Publishing and printing center "Kyiv University".
7. KRYVYI S.L. (2015). *Linear Diophantine limits and their application*. Chernivtsi: "Bukrek" Publishing House.

Про авторів:

Кривий Сергій Лук'янович,
доктор фізико-математичних наук,
професор факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка.
Індекс Хірша (Google Scholar) – 17.
Індекс Хірша (Scopus) – 4.
Кількість наукових публікацій в українських виданнях – 150,
Кількість наукових публікацій в зарубіжних виданнях – 60.
http://orcid.org/0000-0003-4231-0691,

Погорілий Сергій Дем'янович,
доктор технічних наук, професор,
завідувач кафедри комп'ютерної інженерії
факультету радіофізики, електроніки та комп'ютерних систем
Київського національного університету імені Тараса Шевченка.
Індекс Хірша (Google Scholar) – 6.
Індекс Хірша (Scopus) – 2.
Кількість наукових публікацій в українських виданнях – 200.
Кількість наукових публікацій в зарубіжних виданнях – 45.
http://orcid.org/0000-0002-6497-5056,

Слинько Максим Сергійович,
аспірант 2-го року навчання факультету радіофізики,
електроніки та комп'ютерних систем
Київського національного університету імені Тараса Шевченка.
Кількість наукових публікацій в українських виданнях – 2.
Кількість наукових публікацій в зарубіжних виданнях – 1.
http://orcid.org/0000-0001-9667-8729.

Місце роботи авторів:

Київський національний університет імені Тараса Шевченка,
03022, Київ, проспект Академіка Глушкова, 4г.
Тел.: (044) 526 0522,
E-mail: sdp@univ.net.ua,
maxim.slinko@gmail.com