DOI https://doi.org/10.15407/usim.2018.04.0046
УДК 004.4

С.Д. ПОГОРІЛИЙ, доктор технічних наук, професор, завідувач кафедри,
Київський національний університет імені Тараса Шевченка
03022, Київ, проспект Академіка Глушкова, 4г,
sdp@univ.net.ua,
ORCID ID - 0000-0002-6497-5056

С.Л. КРИВИЙ, доктор фізико-математичних наук, професор
Київський національний університет імені Тараса Шевченка
03022, Київ, проспект Академіка Глушкова, 4г,
sl.krivoi@gmail.com,
ORCID ID - 0000-0003-4231-0691

М.С. СЛИНЬКО, аспірант ,
Київський національний університет імені Тараса Шевченка
03022, Київ, проспект Академіка Глушкова, 4г,
maxim.slinko@gmail.com,
ORCID ID - 0000-0001-9667-8729

ПРОЕКТУВАННЯ ТА МОДЕЛЬНЕ ОБҐРУНТУВАННЯ ЗАСТОСУВАНЬ НА ОСНОВІ ВІДЕОАДАПТЕРІВ

Досліджено модельне обгрунтування властивостей архітектур відеоадаптерів виробництва NVIDIA. Розглянуто послідовність застосування апарату транзиційних систем для побудови високорівневої специфікації та апарату мереж Петрі для верифікації моделі застосування. Проведено формалізацію узагальненої моделі обчислень в архітектурі NVIDIA CUDA та досліджено її на предмет наявності дедлоків, пасток, властивості обмеженості та відсутності мертвих місць і переходів.

Ключові слова: *транзиційні система, NVidia CUDA, мережі Петрі, модельне проектування.*

Вступ

Основною особливістю новітніх графічних відеоадаптерів, які використовують графічні процесори (*Graphical Processing Unit —GPU)*, є наявність набору потокових мультипроцесорів (*Streaming multiprocessor, SM*), що використовувалися раніше лише в алгоритмах і задачах, пов'язаних з обробкою графічних зображень. Технологія обчислень загального призначення на графічних процесорах (*GPGPU*) ґрунтується на використанні сукупності процесорів *GPU*, що працюють паралельно, для обробки даних алгоритмами загального призначення (наукових або інших, але не обов'язково пов'язаних з обробкою зображень). Новітні архітектури графічних відеоадаптерів від фірми *Nvidia* включають *Pascal*, *Volta*, *Amper*, *Тюрінг* [1]. На цей день *Volta* — це найпотужніша архітектура *GPU*, яка є індикатором досягнень в області високопродуктивних обчислень штучного інтелекту (відеоадаптер *GTX TITAN Z*, побудований на двох потужних ядрах *GK*110, здатний забезпечити пікову продуктивність до восьми терафлопс, а кожне ядро може реалізувати 2880 потокових процесорів, що в сумі дає 5760 потокових процесорів).

Компанія *Nvidia*, крім того, має серію відеоадаптерів, орієнтованих на наукові застосування і використання у високопродуктивних (кластерних) обчисленнях. Ці *GPU* позбавлені деяких специфічних для графіки функцій і широко застосовуються у науковій сфері. Це призвело до значного збільшення кількості суперкомп'ютерів з відеоадаптерами від Nvidia, що входять до складу top 500 найбільш продуктивних комп'ютерів світу [2].

На даний час тенденції в галузі високопродуктивних обчислень (*HPC*) зміщуються від кластерів, що складаються з модулів загального призначення, до більш спеціалізованих компонентів-акселераторів (іншими словами, від універсальних *CPU* до інших блоків — *GPU*, *FPGA* тощо), тобто до менш функціональних та менш енерговитратних модулей. Акселератори, на відміну від універсальних *CPU*, не можуть керувати операційною системою абощо, і для операцій вводу-виводу або планування задач використовують зовнішні системи. Їх переваги у продуктивності полягають повністю у тому, що ці елементи використовуються у великих групах одночасно. В даній статті розглядаються методи модельного обґрунтування властивостей архітектур *GPU* акселераторів.

З позиції програмування, успіх найбільш поширених *GPGPU* технологій (зокрема, *CUDA*) полягає в тому, що вони приховують *SIMD* аспект апаратного забезпечення *GPU*. Як правило, в більшості випадків розробник має справу з термінами окремих потоків, які працюють зі скалярними даними, а не варпами, що працюють з векторами.

Створення систем високого рівня складності із масовим паралелізмом на основі відеоадаптерів потребує розробки новітніх наукових методів обґрунтування як таких систем, так і застосувань для них.

Розв'язання задач паралелізації для такого класу систем унеможливлює інженерне проектування і потребує наявності формальних методів математичного апарату і методів представлення алгоритму. Ефективним методом алгоритмічного дослідження властивостей паралельних алгоритмів є модельне обґрунтування. Одним з варіантів реалізації такого обґрунтування є використання апарату алгоритмічних алгебр, які дозволяють сформулювати схеми алгоритмів у вигляді алгебраїчних виразів, залежних від різних параметрів, якими можуть бути програмно-апаратні платформи, парадигми паралельного програмування тощо. В статті основним формальним математичним засобом дослідження є транзиційні системи (ТС) [3,4] та їх композиції, що дозволяє створювати моделі на різних рівнях абстракції, а їх властивості можна досліджувати шляхом трансляції в мережі Петрі (МП) [5,6] та описувати формулами темпоральної логіки [7].

Модельне обґрунтування та верифікація

Основним методом підвищення надійності програм при традиційних методах розробки є тестування. Цей метод може вказати на наявність помилок, але не може надати гарантії їх відсутності. Крім того, тестування не може виявити типові помилки синхронізації паралельних програм: в них можуть роками зберігатися помилки, які проявляються після тривалої експлуатації як реакція на виниклу специфічну комбінацію численних факторів, наприклад, непередбачуваних швидкостей виконання окремих паралельних процесів. Однак, якщо властивості системи можна виразити формально, наприклад, у вигляді формули математичної логіки, то аналіз цієї властивості може бути проведений формальними методами верифікації. Верифікація системи складається з наступних частин:

- Побудова математичної моделі системи, що проектується, та опис її властивостей на вибраному рівні абстракції.
- Подання властивостей, що аналізуються, у вигляді формального тексту (званого специфікацією).
- Застосування відповідних формальних методів доведення виконуваності чи не виконуваності на моделі системи властивостей, які перевіряються.

Як правило, математична модель дискретної системи являє собою граф, вершини яко-

го відповідають станам (або класам станів), в яких може перебувати система в різні моменти часу; а ребрам — переходам між станами, які можуть мати позначки, що зображують дії чи події, які виконує система. Функціонування системи при такому зображенні представляється послідовностями переходів від одного стану до іншого. Якщо ребро має позначку, то ця позначка являє собою дію системи, що виконується при переході від стану на початку ребра до стану в його кінці.

Вибір рівня абстракції для зображення системи залежить від багатьох факторів (алгоритмічна розв'язуваність, астрономічні розміри моделі, відсутність ефективних методів формального аналізу властивостей). У зв'язку з цим неформальні правила такого вибору зводяться до таких: модель системи не має бути надмірно детальною, тому що зайва складність моделі може викликати суттєві обчислювальні проблеми при її формальному аналізі; модель системи не має бути надмірно спрощеною, оскільки вона повинна відображати ті аспекти системи, які мають стосунок до властивостей, що перевіряються, і зберігати всі властивості системи, що проектується, які представляють інтерес для аналізу.

Основним методом формального доведення того, що модель не задовольняє своїй специфікації, використовується метод перевірки на моделі (*model checking* (*MC*), [7, 8]). Далі в роботі розглядається метод перевірки відповідності моделі своїй специфікації, який використовує апарат транзиційних систем та мереж Петрі. Нагадаємо означення розміченої ТС, а означення МП відоме і за необхідності його можна знайти в [5,6].

Означення 1. Транзиційною системою називається п'ятірка $A=(S,T,\alpha,\beta,s_0)$, 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

$t \in T$ два стани $\alpha(t)$ і $\beta(t)$, які називають відповідно початком і кінцем переходу t.

s_0 — початковий стан.

Перехід t з початком s і кінцем s' записують так: $s \to s'$.

Інколи переходи мають спільний початок або кінець, або спільний і початок і кінець. Це значить, що відображення α, β: $T \to S$ не обов'язково є ін'єктивними відображеннями. *ТС A* називається скінченною, якщо множини S і T скінченні. Якщо в множині станів виділено початковий стан, то така ТС позначається п'ятірка $A = (S, T, \alpha, \beta, a_0)$ і називається *ініціальною*.

Нехай X — деякий алфавіт. Розміченою транзиційною системою (РТС) називається упорядкована шістка $A = (S, T, \alpha, \beta, s_0, h)$, де $(S, T, \alpha, \beta, s_0)$ ТС, а h — відображення із T в X, яке ставить кожному переходу t його позначку $h(t)$ із X. РТС називається скінченною, якщо множини S, T, X скінченні. Позначку переходу $h(t)$ також називають дією, а сам перехід записують як $s \xrightarrow{h(t)} s'$. До позначок переходів часто додається спеціальна позначка τ, яка означає внутрішню дію в системі, що не видима на даному рівні моделювання.

Використання РТС як моделі системи дозволяє створювати абстракції різних рівнів (за рівнем деталізації, за вибором формальної логічної мови для опису специфікацій тощо), композиції яких можуть транслюватися і досліджуватися їх засобами. Предметом дослідження обрано верифікацію додатків для графічних відеоадаптерів на прикладі відеоадаптерів компанії *Nvidia*. Ця область ідеально ілюструє неможливість верифікації в ручному режимі, оскільки кількість потоків, що виділяються для розв'язання завдання, вимірюється сотнями тисяч (в архітектурах *Pascal/Volta*).

Новітні архітектури відеоадаптерів 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 Ar-*

chitecture) — це архітектура паралельних обчислень, розроблена компанією *Nvidia* для спрощення програмування *GPGPU* за рахунок використання високорівневих *API*.

З моменту впровадження пілотної платформи *CUDA* минуло більше 10 років, тому кожне нове покоління *GPU Nvidia* забезпечувало кращу продуктивність додатків (наприклад, в операціях із плаваючою точкою), підвищувало енергоефективність, додавало важливі нові обчислювальні функції та спрощувало програмування графічного процесора.

Сьогодні процесори *Nvidia* виступають як прискорювачі при високопродуктивних обчисленнях, в центрах обробки даних та прикладних програм у машинному навчанні. *GPU* від *Nvidia* стали провідними обчислювальними пристроями, що створили в певному сенсі революцію в штучному інтелекті (*AI*). Наразі технологія *GPGPU* пришвидшує системи глибинного навчання; системи високоточного розпізнавання тексту, голосу та інших медіаданих; використовуються у сферах молекулярного моделювання, моделювання медичних препаратів, медичного діагностування, фінансового моделювання та ін.

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*.

Всі архітектури *GPU Nvidia* виконують інструкції у групах по 32 потоки (відомі як варпи), використовуючи *SIMT* модель (*Single Instruction*, *Multiple Thread*), тобто одна інструкція виконується одночасно багатьма потоками, при чому поведінка кожного окремого потоку нічим не обмежується. Втім, архітектури до Pascal включно мають спільний для всіх потоків варпу програмний лічильник та маску, яка визначає, які потоки активні в будь-який момент часу. Це означає, що у випадку програмного розгалуження, у кожному з шляхів виконання застосовано лише частину потоків, а решта деактивується, тобто абсолютної паралельності досягти не вдасться. Після сходження шляхів виконання потоки варпу знову виконуються одночасно.

Така модель виконання відкидає необхідність відслідковування стану кожного потоку окремо. Однак відслідковування лише варпу в цілому означає зменшення рівня паралелізму за наявності розгалужень шляху виконання обчислень, як описано вище.

В свою чергу, це призводить до неможливості обміну даними між потоками одного варпу, що знаходяться на різних етапах виконання, або обчислюють інструкції в різних гілках виконання, тобто потоки різних варпів виконують інструкції паралельно, але потоки одного варпу виконують частину інструкцій послідовно.

Так, алгоритми, що потребують обміну даними на високому рівні деталізації та використовують засоби синхронізації (наприклад, м'ютекси), можуть призвести до дедлоків. Тому на *Nvidia GPU* архітектури *Pascal* чи нижче, тобто на більш ніж 80 відсотків пристроїв (архітектура *Volta* вийшла у 2017, а *Turing* — лише у 2018 р.), розробникам доводиться покладатися на алгоритми з мінімальним блокуванням.

Одним з нововведень новітньої архітектури *Volta* є незалежне планування потоків, що зберігає лічильники інструкцій та стек викликів для кожного потоку окремо для оптимального використання ресурсів, або, щоб дозволити одному потоку чекати отримання даних від іншого. Щоб збільшити рівень паралелізації, *Volta* включає в себе оптимізатор, що визначає, як групувати активні потоки в межах варпу у *SIMT*-модулі. Це зберігає високу продуктивність виконання *SIMT*, як і в попередніх *GPU Nvidia*, але з набагато більшою гнучкістю: потоки тепер можуть виконувати різні шляхи розгалуження в межах варпу. Проілюструємо методи верифікації на прикладі аналізу простої моделі системи.

Модель виконання CUDA-застосування

В роботі [9] представлено варіанти узагальненої моделі обчислень в архітектурі *Nvidia CUDA*, що базується на моделях розмічених транзиційних систем та мереж Петрі. Нагадаємо, що в процесі декомпозиції було виділено наступні підсистеми:

- РТС $A=(S_1 = \{a_0, a_1, a_2, a_3\}, T_1, \alpha_1, \beta_1, a_0, h_1)$, що представляє варп, який містить набір інструкцій та послідовно їх виконує, де $X_1=\{r_1, r_2, r_3, r_4\}$;
- РТС $B = (S_2=\{b_0, b_1, b_2, b_3, b_4, b_5\}, T_2, \alpha_2, \beta_2, b_0, h_2)$, що представляє узагальнену інформаційну інструкцію, де $X_2=\{p_1, p_2, p_3, p_4, p_5, p_6\}$;
- РТС $C = (S_3=\{c_0, c_1, c_2, c_3\}, T_3, \alpha_3, \beta_3, c_0, h_3)$, що являє собою виконання блоку на SM, $X_3 = \{q_1, q_2, q_3, q_4\}$. Функції позначок переходів h_1, h_2, h_3 показані нижче на діаграмах.

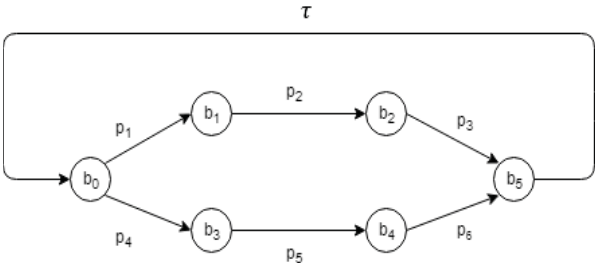

Рис. 1. Зображення інформаційної інструкції у вигляді РТС

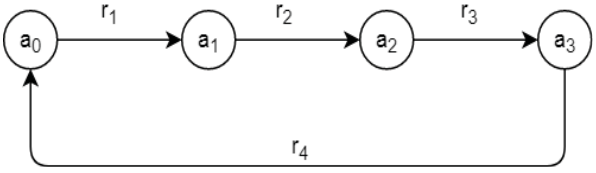

Рис. 2. РТС виконання інструкції варпом

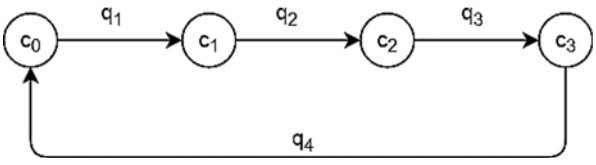

Рис. 3. Зображення роботи планувальника варпу у вигляді РТС

Семантика позначок переходів не даному рівні абстракції не розглядається.

Основна діяльність на цьому етапі передбачає планування, вибір варпу та інструкції для нього і забезпечення ексклюзивного доступу до обчислювальних ресурсів на час виконання кожної пари варп-інструкцій.

Інтеграція ТС в цілісну систему, яка організовує роботу всіх ТС, виконується в залежності від потреб взаємодії складових (синхронна, асинхронна, паралельна, послідовна). Ці способи взаємодії узагальнюються операцією синхронного добутку ТС. Наведемо означення синхронного добутку ТС.

Нехай $A = A_1 \times A_2 \times ... \times A_n$ — декартів добуток ТС, де $A_i = (S_i, T_i, \mathsf{a}, \mathsf{b}, a_0^i)$, $i = 1,...,n$.

Означення 2. Обмеженням синхронізації називається підмножина Т множини $(T_1 \cup \mathsf{e}) \times (T_2 \cup \mathsf{e}) \times ... \times (T_n \cup \mathsf{e})$, де e — тотожна дія, яка означає відсутність якої-небудь дії в ТС. Елементи із T також називаються глобальними переходами. Якщо $t = (t_1,...,t_n) \in T$ і $t_i \notin \mathsf{e}$, то говорять, що ТС A_i бере участь у переході t. Кортеж $A = (A_1,..., A_n, \mathrm{T})$ називається синхронним добутком ТС $A = A_1 \times A_2 \times ... \times A_n$, а ТС $A_1,..., A_n$ називають компонентами ТС А.

Якщо ТС розмічена, то множині глобальних переходів Т відповідає множина позначок І переходів із Т. Тобто $I \subseteq X_1 \times ... \times X_n$ де X_i — алфавіт РТС A_i, $i = 1,...,n$. Довільний елемент із множини І називається вектором синхронізації РТС $A = (A_1,..., A_n, \mathrm{T})$.

Для наведених ТС будуємо модель застосування в архітектурі *CUDA* у вигляді синхронного добутку з глобальними переходами:

$$\begin{aligned} T = \{&t_1 = (\tau, r_1, q_1), t_2 = (p_1, r_2, q_2), \\ &t_3 = (p_2, \varepsilon, \varepsilon), t_4 = (p_3, r_3, q_3), \\ &t_5 = (p_4, r_2, q_2), t_6 = (p_5, \varepsilon, \varepsilon), \\ &t_7 = (p_6, r_3, q_3), t_8 = (\varepsilon, r_4, \varepsilon), \\ &t_9 = (\varepsilon, \varepsilon, q_4)\} \end{aligned} \quad (1)$$

Отже, отримуємо високорівневу специфікацію системи (модель загального типу). Маючи вказану модель, можливо приступати до процесу її верифікації. Основними моде-

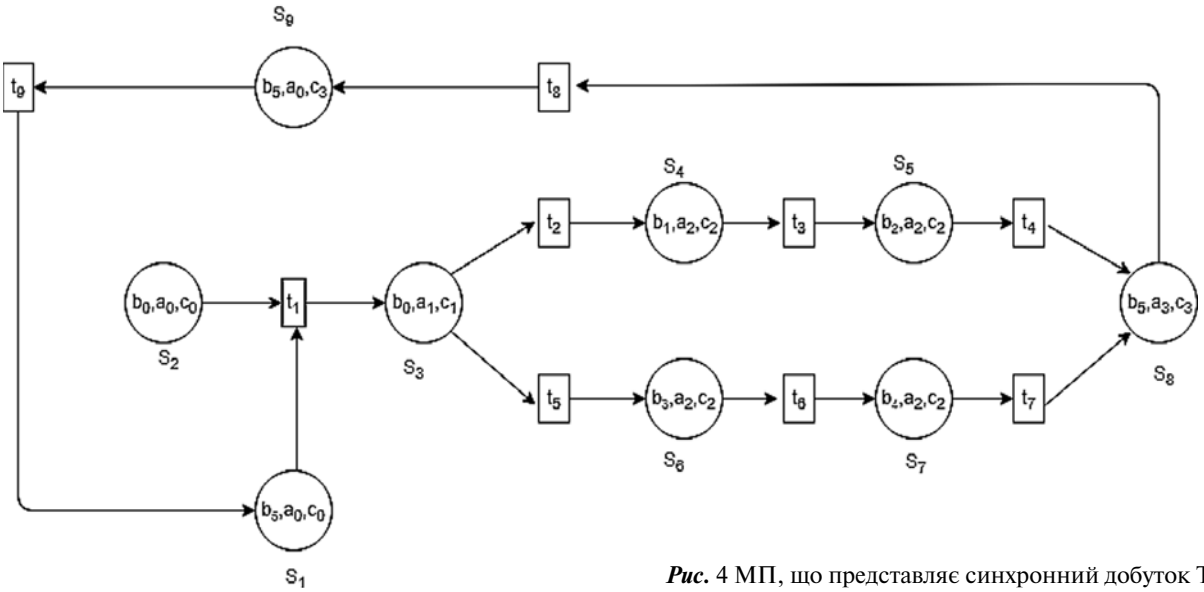

Рис. 4 МП, що представляє синхронний добуток ТС

лями такого процесу є автоматні та мережеві моделі. У даній статті розглядаються мережеві моделі, а саме — мережі Петрі, для яких існує широкий апарат методів аналізу. В [9, 10] описано, що семантика добутку ТС і семантика МП, яка його моделює, узгоджені в тому сенсі, що послідовність глобальних переходів $t_1,...,t_k$ являє собою глобальну історію добутку ТС А тоді і тільки тоді, коли вона є допустимою послідовністю виконання переходів в МП. Відповідно, елементи множини Т стають переходами МП, а глобальні стани ТС (набір станів кожної з ТС, що беруть участь у синхронному добутку до або після глобального переходу) стають місцями отриманої мережі. За множиною обмежень синхронного добутку Т будуємо МП, що моделює спільну роботу всіх підсистем:

Семантично місця та переходи отриманої МП описують наступні концепції:

- s_1 — фіксація старту планування. Варп деактивовано, інструкцію не обрано;
- s_2 — місце отримання токенів-інструкцій;
- s_3 — активовано варп, обрано інструкцію для виконання;
- s_4 — варп в процесі виконання арифметичної операції. Планувальник варпу в стадії очікування;
- s_5 — арифметичну операцію виконано. Планувальник варпу в стадії очікування;
- s_6 — варп в процесі виконання запиту до пам'яті. Планувальник варпу в стадії очікування;
- s_7 — фіксація отримання даних з пам'яті *GPU*. Планувальник варпу в стадії очікування;
- s_8 — фіксація виконання пари варп-інструкція. Відповідно до обчислювальної архітектури, поточний варп має бути деактивованим; планувальник обирає наступну пару варп-інструкція;
- s_9 — варп деактивовано, пристрій готовий до наступної ітерації планування;
- t_1 — активація конкретного варпа планувальником (даний варп отримує доступ до ресурсів SM для виконання однієї інструкції), вибір інструкції для виконання;
- t_2 — старт виконання обраної інструкції для обраного варпу — на рівні планувальника, підготовка до виконання арифметичної операції на рівні інструкції;
- t_3 — виконання арифметичної операції;
- t_4 — отримання підтвердження про виконання арифметичної операції;
- t_5 — старт виконання обраної інструкції для обраного варпу — на рівні планувальника, формування запиту до пам'яті на рівні інструкції;

▪ t_6 — процес отримання даних з пам'яті;

▪ t_7 — отримання підтвердження про виконання запиту до пам'яті;

▪ t_8 — деактивація варпу (після переходу варп знову стає доступним для вибору планувальником);

▪ t_9 — перехід до наступної ітерації планування.

Побудована МП служить моделлю, на якій перевіряється правильність визначення синхронного добутку. В зв'язку з цим виникає задача перевірки властивостей коректності отриманої моделі синхронного добутку. Розглянемо аналіз МП на предмет наявності в ній дедлоків та пасток, а також властивості обмеженості та відсутності мертвих місць і переходів.

Семантично стан дедлоку — це досяжна розмітка мережі, з якої неможливий жодний перехід. Нехай для деякої МП (P,T,F,M_0) маємо підмножину $Q \subseteq P$. Множину Q називають *дедлоком* (сифоном) тоді і тільки тоді, коли $\bullet Q \subseteq Q \bullet$ та *пасткою* тоді і тільки тоді, коли $Q \bullet \subseteq \bullet Q$.

Як показано в [4], МП структурно жива тоді і тільки тоді, коли кожен її дедлок має пастку. Дедлоки та пастки можуть бути знайдені в МП шляхом розв'язання системи логічних рівнянь, що описують наступні властивості:

▪ якщо дедлок Q є пустою множиною, то він завжди залишається таким в процесі функціонування МП;

▪ якщо принаймні одне з місць пастки одержало фішку, то пастка постійно залишається непустою в процесі функціонування МП.

Отже, можна уточнити попереднє твердження, вказавши, що МП (P, T, F, M_0) жива тоді і тільки тоді, коли кожен дедлок цієї мережі має пастку, відзначену початковою розміткою M_0. Розглянемо МП на рис. 4. Система логічних залежностей для пошуку дедлоків має такий вигляд:

$$\begin{array}{ll} s_1 \to s_9, & s_3 \to s_1 \vee s_2, \\ s_4 \to s_3, \quad \ldots & s_5 \to s_4, \\ s_6 \to s_3, & s_7 \to s_6, \\ s_8 \to s_5 \wedge s_7, & s_9 \to s_8. \end{array} \qquad (2)$$

СЛОДН, яка відповідає цій системі логічних залежностей (2) можна подати у вигляді наступної системи нерівностей:

$$s = \left|\begin{array}{ccccccccc|c} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \geq 0 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & \geq 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & \geq 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & \geq 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & \geq 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & \geq 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & \geq 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & \geq 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & \geq 0 \end{array}\right|$$

Базисними розв'язками системи S є x = = (1,0,1,1,1,1,1,1,1) y = (0,1,0,0,0,0,0,0,0), z = (1,1,1,1,1,1,1,1,1), t = (0,1,1,1,1,1,1,1,0) яким відповідає множина дедлоків даної МП, серед яких перший дедлок є комбінацією базисних:

$$\begin{array}{l} D_1 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}, \\ D_2 = \{s_1, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}, \\ D_3 = \{s_2\}, \\ D_4 = \{s_2, s_3, s_4, s_5, s_6, s_7, s_8\}. \end{array} \qquad (3)$$

Нагадаємо, що як дедлоки, так і пастки мають наступні властивості: дедлок називається *базисним*, якщо він не може бути представленим у вигляді об'єднання інших дедлоків; дедлок називається *мінімальним*, якщо він не містить інших дедлоків; всі дедлоки МП можна породити шляхом об'єднання базисних дедлоків.

Так, дедлоки D_2, D_3, D_4 є базисними, а дедлок D_3 — мінімальним. Зауважимо, що третій дедлок відповідає місцю s_2, яке є особливим в даній МП тим, що в нього не ведуть переходи. Крім того, четвертий дедлок також існує лише внаслідок особливого статусу місця s_2. Тому ці дедлоки можна вилучити з розгляду.

Перейдемо до дослідження пасток МП, зображеної на рис. 4. Система логічних залежностей для пошуку пасток має вигляд

$$\begin{array}{ll} s_1 \to s_3, & s_2 \to s_3, \\ s_3 \to s_4 \wedge s_6, & s_4 \to s_5, \\ s_5 \to s_8, \quad \ldots & s_6 \to s_7, \\ s_7 \to s_8, & s_8 \to s_9, \\ s_9 \to s_1. & \end{array} \qquad (4)$$

СЛОДН, яка відповідає цій системі логічних залежностей (4), можна представити у

вигляді наступної таблиці:

$$S = \left|\begin{array}{cccccccccc} -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \geq 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \geq 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & \geq 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & \geq 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & \geq 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & \geq 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & \geq 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & \geq 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & \geq 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & \geq 0 \end{array}\right|$$

Система нерівностей S має розв'язки x = = (1,1,1,1,1,1,1,1,1), y = (1,0,1,1,1,1,1,1,1), яким відповідає множина пасток МП:

$$Tr_1 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\},$$
$$Tr_2 = \{s_1, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}. \quad (5)$$

Оскільки мережа Петрі є результатом вільного вибору, то для таких мереж справедливе твердження: *МП вільного вибору жива тоді і тільки тоді, коли кожний дедлок такої МП містить пастку, позначену початковою розміткою*.

Як видно з наведеної МП і отриманих результатів, кожен її базисний дедлок (3) включає в себе щонайменше одну з позначених початковою розміткою пасток (5), а тому МП є живою.

Розглянемо питання обмеженості та наявності мертвих переходів в МП. Для цього складемо рівняння стану $A \cdot x + M_0 - M_k = A \cdot x + d = 0$, де M_0 = (1,1,0,0,0,0,0,0,0), M_k = (1,0,0,0,0,0,0,0,0), d = $-(M_k - M_0)$.

	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	0	0	0	0	0	0	0	1	-1	0

Застосовуючи *TSS*-алгоритм [10] для розв'язання рівняння стану з вищевказаною матрицею, отримуємо наступні розв'язки:

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.

Аналіз обмеженості виконується шляхом розв'язання системи рівнянь вигляду $A^T x = 0$, де A^T — матриця, транспонована до матриці A. Система $A^T x = 0$ має єдиний розв'язок x = (1, 0, 1, 1, 1, 1, 1, 1, 1), у якому всі місця МП покриваються позитивними значеннями, за винятком місця s_2. Але очевидно, що в цьому місці, накопичуватися токени нескінченно не можуть, оскільки в цьому місці число токенів скінченне (особливий статус місця s_2). Отже, МП обмежена і в ній відсутні недосяжні місця.

Висновки

Досліджено використання математичного апарату транзиційних систем, який дозволив отримати формалізовану специфікацію системи, що аналізується. До переваг сформованої моделі належить можливість представити синхроний добуток мережею Петрі, що дозволяє виконувати подальшу верифікацію автоматизованими засобами.

Подано можливості дослідження характеристик моделі, утвореної поєднанням апаратів транзиційних систем та мереж Петрі.

Проведено аналіз моделі на наявність дедлоків та пасток і відсутність мертвих переходів та місць. Отримано множини базисних та мінімальних дедлоків та пасток, показано, що кожен дедлок включає в себе щонайменше одну з пасток, тобто отримана модель є живою. Створений підхід дозволяє спростити та скоротити процеси верифікації і тестування багатопоточних застосувань в комп'ютерних системах з відеоадаптерами.

СПИСОК ЛІТЕРАТУРИ

1. Nvidia Data Center | Nvidia (2018). Тип доступу — URL: https://www.nvidia.com/en-us/data-center/ (accessed November 11, 2018).
2. TOP500 Lists | TOP500 Supercomputer Sites (2018). Тип доступу — URL: https://www.top500.org/lists_(accessed November 11, 2018).
3. Arnold A. Finite Transition Systems: Semantics of Communicating Systems. — Paris: Prentice Hall.-1994. — 177 p.
4. Kryvyi S. L., Boyko Y. V., Pogorilyy S. D., Boretskyi O. F., Glybovets M. M. Design of Grid Structures on the Basis of Transition Systems with the Substantiation of the Correctness of Their Operation. Cybernetics and Systems Analysis. January 2017, Volume 53, Issue 1, pp 105—114. Springer Science+Business Media New York 2017.
5. Котов В. Е. Сети Петри. М.: Наука. — 1984. — 157 с.
6. Murata T. Petri nets: properties, analysis and applications. In Proc. of the IEEE.77:541.80 -1989.
7. Ben-Ari M. Mathematical Logic for Computer Science. Prentice Hall International (UK) Ltd. -1993. — 305 p.
8. CLARKE E.M, Jr., GRUMBERG O., PELED D.A (1999) Model Checking, MIT Press, ISBN 0-262-03270-8.
9. Кривий С.Л., Погорілий С.Д., Слинько М.С. Формалізований метод проектування застосувань в технології GPGPU. Матеріали одинадцятої міжнародної науково-практичної конференції з програмування УкрПРОГ'2018. Проблеми програмування, 2018.
10. Кривий С.Л. Лінійні діофантові обмеження та їх застосування. — Букрек: Чернівці. — 2015. — 224 с.

Надійшла 26.11.2018

REFERENCES

1. Nvidia Data Center | Nvidia. https://www.nvidia.com/en-us/data-center/ (accessed November 11, 2018).
2. TOP500 Lists | TOP500 Supercomputer Sites. https://www.top500.org/lists Nvidia Data Center | Nvidia. https://www.nvidia.com/en-us/data-center/ (accessed November 11, 2018)
3. Arnold A.,1994. Finite Transition Systems: Semantics of Communicating Systems. Paris: Prentice Hall. 177 p.
4. Kryvyi S. L., Boyko Y. V., Pogorilyy S. D., Boretskyi O. F., Glybovets M. M., 2017. Design of Grid Structures on the Basis of Transition Systems with the Substantiation of the Correctness of Their Operation. Cybernetics and Systems Analysis. January 2017, Volume 53, Issue 1, pp 105—114. Springer Science+Business Media New York
5. Kotov V.E., 1984. Petri nets. Moscow: Nauka, 157 p.
6. Murata T., 1989. Petri nets: properties, analysis and applications. In Proc. of the IEEE.77:541.80.
7. Ben-Ari M., 1993. Mathematical Logic for Computer Science. Prentice Hall International (UK) Ltd. — 305 p.
8. Clarke E.M, Jr., Grumberg O., Peled D.A., 1999. Model Checking, MIT Press, ISBN 0-262-03270-8.
9. Kryvyi S.L., Pogorilyy S.D., Slynko M.S., 2018. Transition systems as method of designing applications in GPGPU technology. Proceedings of the 11-th international scientific and practical conference on programming UkrPROG'2018.
10. Kryvyi S.L. 2015. Linear Diophantine constraints and their application. Chernivtsi: "Bukrek" Publishing House.

Received 26.11.2018

Sergiy Pogorilyy, Doctor of Technical Sciences, Professor, Head of Department,
Taras Shevchenko National University of Kyiv, Glushkov ave., 4g, 03022, Kyiv, Ukraine.
sdp@univ.net.ua,
ORCID ID - 0000-0002-6497-5056

Sergii Kryvyi, Doctor of Physical and Mathematical Sciences, Professor,
Taras Shevchenko National University of Kyiv, Glushkov ave., 4g, 03022, Kyiv, Ukraine.
sl.krivoi@gmail.com
ORCID ID - 0000-0003-4231-0691

Maksym Slynko, PhD student,
Taras Shevchenko National University of Kyiv, Glushkov ave., 4g, 03022, Kyiv, Ukraine.
maxim.slinko@gmail.com
ORCID ID - 0000-0001-9667-8729

MODEL JUSTIFICATION OF GPU-BASED APPLICATIONS

Introduction. Nowadays high performance computing (HPC) trends are shifted from using cluster systems consisting of the general-purpose modules to more specialized accelerator components (*GPUs*, *FPGAs* etc.). Development of high-level complexity systems with massive parallelism based on *GPU* accelerators requires evolution of the scientifically based methods of application design and verification.

Purpose. The purpose of this article is to present the new method of *GPU*-based application development based on the model design.

Methods. The method of application design and verification, which allows creating abstrac-tions of the different level by using the labeled transition systems apparatus, compositions of which may be converted to Petri nets and analyzed by the relevant means, is proposed. This me-thod allows to research the properties of the system, which can't be analyzed manually, since the number of threads allocated for executing the code is measured by hundreds of thousands (in *Pascal/Volta/Turing* architectures).

Result. A formalized specification of the analyzed system could be obtained due to the transi-tion system apparatus usage. The model is analyzed for the deadlocks (siphons) and traps and the absence of dead transitions and places. The sets of basis and minimum deadlocks and traps are obtained. Model aliveness is proved due to the fact that each deadlock includes at least one of the traps.

Conclusions. The proposed method allows to create the models of the different abstraction le-vels, which allows carrying out the further analysis by automated means. The possibilities of studying the model characteristics created by the transition systems combination and *Petri Net*-works apparatuses are presented.

Kewwords: *transition systems, Nvidia CUDA, Petri Networks, the model design.*

С.Д. Погорілий, доктор технічних наук, професор, завідувач кафедри,
Київський національний університет імені Тараса Шевченка
03022, Київ, проспект Академіка Глушкова, 4г.
sdp@univ.net.ua,
ORCID ID - 0000-0002-6497-5056

С.Л. Кривий, доктор фізико-математичних наук, професор
Київський національний університет імені Тараса Шевченка
03022, Київ, проспект Академіка Глушкова, 4г.
sl.krivoi@gmail.com
ORCID ID - 0000-0003-4231-0691

М.С. Слинько, аспирант ,
Киевский национальный университет имени Тараса Шевченко
03022, Киев, проспект Академика Глушкова, 4г.
maxim.slinko@gmail.com
ORCID ID - 0000-0001-9667-8729

МОДЕЛЬНОЕ ОБОСНОВАНИЕ ПРИЛОЖЕНИЙ НА ОСНОВЕ ВИДЕОАДАПТЕРОВ

Введение. В настоящее время тенденции в области высокопроизводительных вычи слений (HPC) смещаются от использования кластерных систем, состоящих из модулей общего назначения к более специализированным компонентам-акселераторам (*GPU*, *FPGA* и т.д.). Создание систем высокого уровня сложности с массовым параллелизмом на основе *GPU*-акселераторов требует разработки новых научно обоснованных методов проектирования и верификации приложений.

Цель. Презентация нового метода создания приложений с использованием видеоадаптеров, основанного на модельном проектировании.

Методы. Предложен метод проектирования и верификации приложений, который позволяет создавать абстракции различных уровней, используя аппарат размеченных транзиционных систем, композиции которых могут транслироваться в сети Петри и исследоваться средствами СП. Предложенный метод позволяет исследовать свойства системы, анализ которых в ручном режиме невозможен, так как количество потоков, выделяемых для решения задачи, измеряется сотнями тысяч (в архитектурах *Pascal / Volta*).

Результат. С помощью аппарата транзицийних систем удалось получить формализованную спецификацию анализируемой системы. Благодаря ее представлению сетью Петри проведен анализ модели на наличие дедлоков и ловушек, а также на отсутствие мертвых переходов и мест. Получены множества базисных и минимальных дедлоков и ловушек, показано, что каждый дедлок включает в себя по меньшей мере одну из ловушек, то есть полученная модель является живой.

Выводы. Предложенный метод позволяет формировать модели разного уровня абстракции и выполнять дальнейшую верификацию автоматизированными средствами. Представлены возможности исследования характеристик модели, созданной при помощи аппаратов транзиционных систем и сетей Петри. Созданный подход позволяет упростить и сократить процессы верификации и тестирования многопоточных приложений в компьютерных системах с видеоадаптерами.

Ключевые слова: *транзиционные системы, Nvidia CUDA, сети Петрі, модельное проектирование*