Формальное моделирование конвейера современного графического процессора

Гранкин Юрий Константинович

Алтайский государственный технический университет им. И. И. Ползунова

Научный руководитель:
Старолетов Сергей Михайлович

Магистерская диссертация — 2026

Актуальность темы

- **Рост популярности GPU** — массовое применение в HPC, машинном обучении, научных вычислениях
- **Модель SIMT** — массовый параллелизм через варпы (32 потока)
- **Дивергенция** — условные ветви вызывают последовательное выполнение
- **Сложность настройки** — зависимость от аппаратных и программных параметров

Зачем формальные методы?

- Симуляция медленная и не даёт гарантий
- Машинное обучение — «чёрный ящик»
- **Model Checking (SPIN):** полная верификация всех путей выполнения и формальные гарантии

Постановка задачи исследования

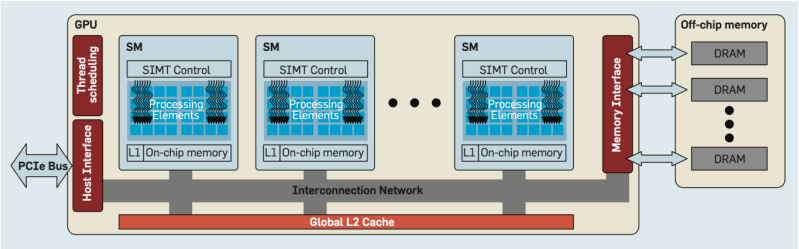
Цель работы: разработать формальную модель конвейера GPGPU для верификации корректности и поиска оптимальных параметров выполнения расходящихся ядер.

Задачи:

1. Проанализировать PTX-код и извлечь граф потока управления
2. Разработать формальную модель компонентов GPU (варпы, планировщик, SIMT-стек)
3. Реализовать модель на языке Promela
4. Провести верификацию с помощью верификатора SPIN
5. Выполнить поиск оптимальных параметров конфигурации[1]

[1] Д. А. Кондратьев, С. М. Старолетов, И. В. Шошина, А. В. Красненкова, К. В. Зиборов, Н. В. Шилов, Н. О. Гаранина, Т. Ю. Черганов. Конкурс по формальной верификации VeHa-2024: двухлетний опыт и перспективы. Труды ИСП РАН, 2025, том 37, вып. 1, стр. 159–184. – DOI: 10.15514 / ISPRAS - 2025 - 37(1) - 10.

Архитектура GPGPU и SIMT-модель выполнения



SIMT-модель:

- Варпы — группы из 32 потоков
- Единый поток инструкций
- Индивидуальные регистры
- Аппаратная синхронизация

Проблема дивергенции:

- Условные переходы
- Последовательное выполнение ветвей
- SIMT-стек для реконвергенции
- Снижение производительности

Promela/SPIN: инструментарий формальной верификации

SPIN (Simple Promela Interpreter)

- **Режимы:** симулятор, верификатор моделей
- **Рабочий процесс:** Спецификация $\rightarrow$ Симуляция $\rightarrow$ LTL $\rightarrow$ Верификация

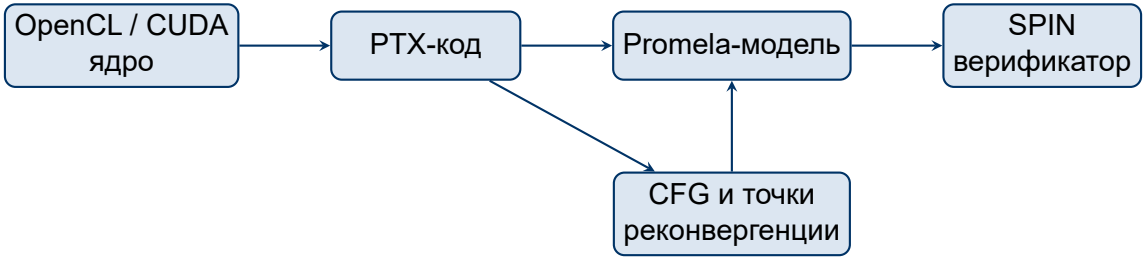
Promela (Process Meta-Language)

- **Фокус:** синхронизация и координация
- **Возможности:** `proctype`, `chan`, недетерминизм, атомарные блоки
- **Основа:** CSP + акторная модель

Почему для моделирования GPU?

- Моделирует параллельные процессы (варпы, SM)
- Обрабатывает синхронизацию (барьеры, SIMT-стек)
- Недетерминизм (попадание/промах кэша)
- Формальные гарантии через контрпримеры

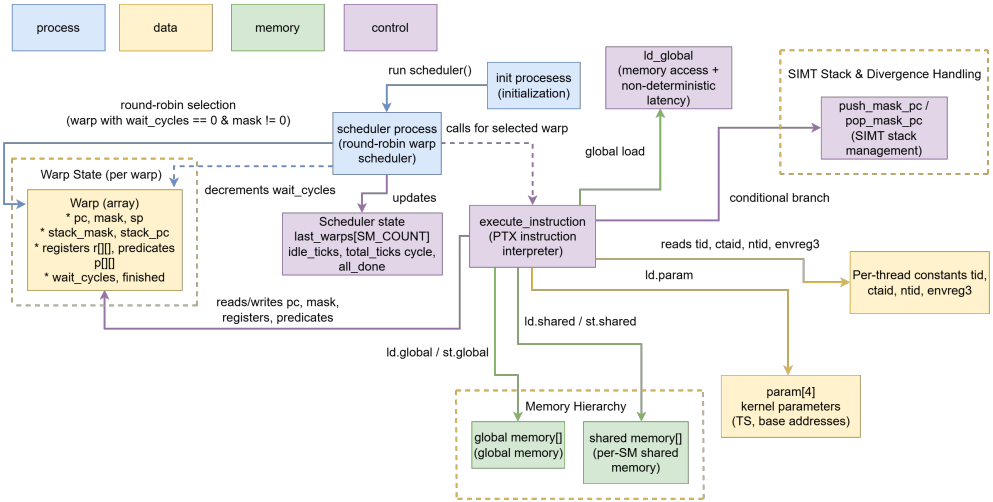
Архитектура предлагаемой модели



Ключевые компоненты

Планировщик варпов, SIMT-стек, моделирование задержек памяти, LTL-свойства.

Компоненты архитектуры



PTX — промежуточное представление GPU

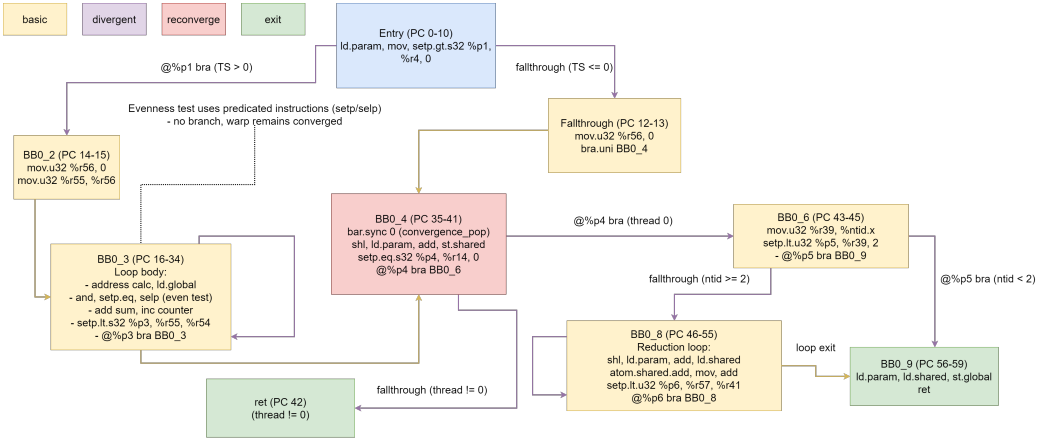
Ядро OpenCL

```
kernel void sum_even(
  global int *glob,
  global int *output,
  local int *loc,
  int TS)
{
  int my_elem = get_local_id(0);
  int my_unit = get_global_id(0) /
    get_local_size(0);
  int sum = 0;
  for (int i = 0; i < TS; i++) {
    int idx = i + get_global_id(0)*TS;
    if (glob[idx] % 2 == 0)
      sum += glob[idx];
  }
  barrier(CLK_LOCAL_MEM_FENCE);
  loc[my_elem] = sum;
  if (my_elem == 0) {
    for (int i = 1;
      i < get_local_size(0); i++)
      atomic_add(loc, loc[i]);
    output[my_unit] = loc[0];
  }
}
```

PTX-код

```
.entry sum_even(
.param .u32 .ptr .global .align 4
  sum_even_param_0,
.param .u32 .ptr .global .align 4
  sum_even_param_1,
.param .u32 .ptr .shared .align 4
  sum_even_param_2,
.param .u32 sum_even_param_3)
{
.reg .pred %p<7>;
.reg .s32 %r<58>;
ld.param.u32 %r4, [sum_even_param_3];
mov.u32 %r14, %tid.x;
div.u32 %r6, %r21, %r19;
setp.gt.s32 %p1, %r4, 0;
@%p1 bra BB0_2;
ld.global.u32 %r34, [%r33];
and.b32 %r35, %r34, 1;
setp.eq.s32 %p2, %r35, 0;
selp.b32 %r36, %r34, 0, %p2;
add.s32 %r56, %r56, %r36;
bar.sync 0;
st.shared.u32 [%r38], %r56;
st.global.u32 [%r48], %r46;
ret;
}
```

Граф потока управления (CFG) ядра `sum_even` (извлечён из PTX)



Основные константы и структуры модели

Константы модели:

```
#define SM_COUNT 2
#define WARP_SIZE 2
#define THREADS_PER_BLOCK 4
#define WARPS_PER_SM 2
#define TOTAL_WARPS 4
#define REG_COUNT 58
#define PRED_COUNT 7
#define MASK_STACK_DEPTH 8
```

Структура варпа:

```
typedef Warp {
  int pc;
  int mask;
  int stack_mask[8];
  int stack_pc[8];
  int sp;
  int r[...];
  int p[...];
  int wait_cycles;
  bit finished;
};
```

Глобальные массивы

```
Warp warps[TOTAL_WARPS], global_mem[1024], shared_mem[512]
```

Моделирование ключевых операций

Загрузка из памяти (недетерминизм)

```
inline ld_global(warp, thread,
  dest_reg, addr_reg) {
  atomic {
    addr = R(warp, thread, addr_reg);
    R(warp, thread, dest_reg) =
      global_mem[addr/4];
    if
    :: true -> warps[warp].wait_cycles = 1;
    :: true -> warps[warp].wait_cycles = 10;
    fi;
  }
}
```

SIMT-стек (дивергенция)

```
inline push_mask_pc(warp,
  saved_mask, saved_pc) {
  atomic {
    warps[warp].sp++;
    assert(warps[warp].sp < 8);
    warps[warp].stack_mask[
      warps[warp].sp] = saved_mask;
    warps[warp].stack_pc[
      warps[warp].sp] = saved_pc;
  }
}
```

Безусловный переход

```
inline bra_uni(warp, target_pc) {
  atomic {
    warps[warp].pc = target_pc;
    printf("Warp %d: BRA.UNI -> target %d\n",
      warp, target_pc);
  }
}
```

Восстановление маски и PC

```
inline pop_mask_pc(warp) {
  atomic {
    assert(warps[warp].sp >= 0);
    warps[warp].mask =
      warps[warp].stack_mask[
        warps[warp].sp];
    warps[warp].pc =
      warps[warp].stack_pc[
        warps[warp].sp];
    warps[warp].sp--;
  }
}
```

Интерпретатор PTX-инструкций

PTX-инструкции

```
// PC=0: Загрузка TS
ld.param.u32 %r4,
  [sum_even_param_3];

// PC=1: Загрузка tid.x
mov.u32 %r14, %tid.x;

// PC=11: Дивергенция
setp.gt.s32 %p1, %r4, 0;
@%p1 bra BB0_2;

// Цикл: проверка чётности
ld.global.u32 %r34, [%r33];
and.b32 %r35, %r34, 1;
setp.eq.s32 %p2, %r35, 0;
selp.b32 %r36, %r34, 0, %p2;
add.s32 %r56, %r56, %r36;
```

Promela-модель

```
inline execute_instruction(warp) {
  int tm, fm;
  atomic {
    if
    :: warps[warp].pc == 0 ->
      for (i : 0 .. WARP_SIZE-1) {
        if :: (warps[warp].mask & (1<<i)) ->
          R(warp, i, 4) = param[3];
        :: else -> skip; fi;
      }
      warps[warp].pc = 1;
    :: warps[warp].pc == 1 ->
      for (i : 0 .. WARP_SIZE-1) {
        if :: (warps[warp].mask & (1<<i)) ->
          R(warp, i, 14) =
            tid[warp*TOTAL_WARPS+i];
        :: else -> skip; fi;
      }
      warps[warp].pc = 2;
    :: warps[warp].pc == 11 ->
      atomic { tm = 0; fm = 0;
        for (i : 0 .. WARP_SIZE-1) {
          if :: (warps[warp].mask & (1<<i)) ->
            if :: (P(warp, i, 1) != 0) ->
              tm = tm | (1<<i);
            :: else ->
              fm = fm | (1<<i); fi;
          :: else -> skip; fi;
        }
        warps[warp].mask = tm;
        warps[warp].pc = 14;
      }
    :: else -> break;
    fi;
  }
}
```

Планировщик варпов (Round-Robin)

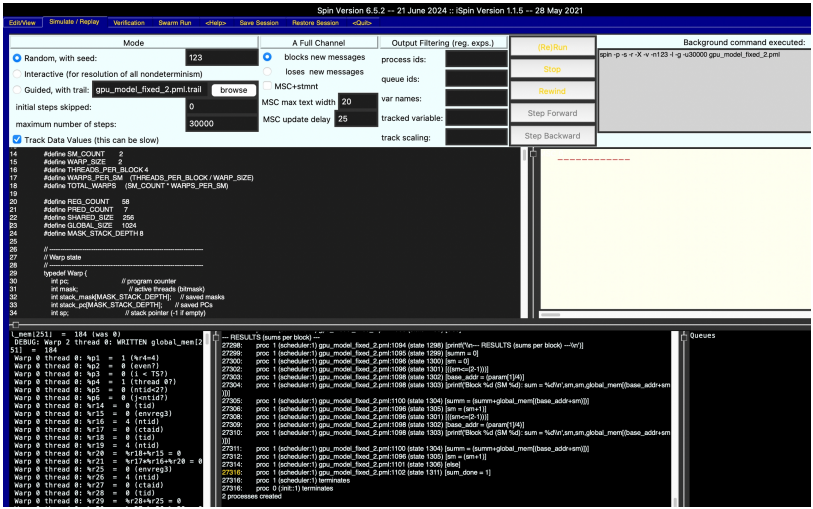
Алгоритм планирования:

1. Циклический опрос всех SM
2. Для каждого SM — поиск готового варпа (Round-Robin)
3. Проверка готовности: `wait_cycles == 0` и `mask != 0`
4. Выполнение одной инструкции
5. Обновление указателя `last_warp[sm]`
6. Уменьшение `wait_cycles` для заблокированных варпов

Процесс планировщик:

```
proctype scheduler() {
  do :: atomic {
    cycle++;
    for (sm : 0 .. SM_COUNT-1) {
      for (i : 0 .. WARPS_PER_SM-1) {
        w = (start + i) % WARPS_PER_SM;
        warp = sm * WARPS_PER_SM + w;
        if ::(warps[warp].wait_cycles==0 &&
          warps[warp].mask!=0) -> {
            execute_instruction(warp);
            last_warp[sm] = (w + 1) % WARPS_PER_SM;
            break;
          } ::else -> skip
          fi
      }
    }
    for (warp : 0 .. TOTAL_WARPS-1) {
      if ::(warps[warp].wait_cycles > 0) ->
        warps[warp].wait_cycles--;
      ::else -> skip
      fi
    }
  } od;
}
```

Визуализация выполнения модели в iSPIN



Верификация корректности (LTL-свойства)

Используем линейную временную логику (LTL) для спецификации требований.

Задача: ядро `sum_even` — сумма чётных элементов массива.

Входные данные: массив из 32 элементов (0, 1, 2, …, 31).

Функциональная корректность (свойство `sum_ok`): Проверяет, что итоговая сумма равна ожидаемому значению (240 для тестовых входных данных).

```
ltl sum_ok { [] (all_done → (global_mem[250] == 56 &&
                global_mem[251] == 184)) }
```

Результат: **ДОКАЗАНО** верификатором SPIN.

Подход к автотюнингу

Через поиск контрпримеров: если SPIN находит выполнение с меньшим числом тактов, чем текущий минимум T, конфигурация улучшается. Итеративно уменьшаем T.

Экспериментальная оценка

Статистика модели:

- **Исследовано состояний:** $\approx 1{,}53 \times 10^6$
- **Переходов:** $\approx 1{,}56 \times 10^6$
- **Вектор состояния:** 15 268 байт
- **Глубина:** 16 144 шага

Использование памяти:

- С `-DCOLLAPSE`: **5,5 ГБ**
- Без Collapse: $\approx 22{,}3$ ГБ

Результаты

- **Доказанные свойства:** `sum_ok`
- **Итоговые суммы:** Блок 0: 56, Блок 1: 184 (Всего: 240)
- **Адекватность:** проверена на соответствие PTX-семантике

Заключение и перспективы дальнейших исследований

Основные результаты:

- **Разработана** формальная модель конвейера GPGPU на языке Promela
- **Реализован** механизм обработки дивергенции через SIMT-стек
- **Проведена** верификация корректности с помощью SPIN
- **Предложен** подход к автотюнингу на основе контрпримеров

Перспективы:

- **Расширение модели:** иерархия памяти (кэш L1/L2)
- **Автоматизация:** трансляция PTX $\rightarrow$ Promela
- **Масштабирование:** к более крупным ядрам
- **Применение:** анализ энергопотребления

Основные результаты и публикации по теме исследования

Публикации по теме диссертации:

1. **Гранкин Ю.К., Бигеев Д.Л., Боярский А.Ю.** Обзор архитектур графических процессоров и конвейера исполнения параллельных инструкций // Современные цифровые технологии: Материалы IV Всероссийской научно-практической конференции, Барнаул, 03 июня 2025 года. — Барнаул: АлтГТУ, 2025. — С. 205-209. — EDN: ZWGKKU.
2. **Гранкин Ю.К., Старолетов С.М.** Формальное моделирование конвейера GPGPU для дивергентных ядер // Труды Института системного программирования РАН. — *(принята к публикации, доклад представлен на конференции SYRCOSE 2026)*.