Схема адаптации классических методов поиска экстремумов к задачам оптимизации GPGPU-программ

Авторы:
Кривов М.А.,
Притула М.Н.,
Иванов П.С.

Переславль-Залесский, 2013

Проблема

- Разработать версию программы для GPU зачастую проще, чем её поддерживать
  - Много производителей GPU

    NVidia, Intel, AMD, ARM, Qualcomm
  - Много поколений GPU

    GT200, Fermi, Kepler1, Kepler2 ...
  - Много API

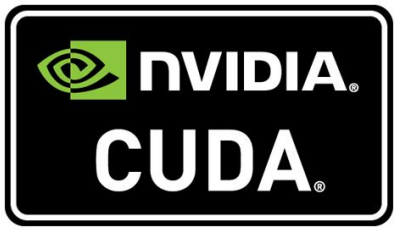
    CUDA, OpenACC, OpenCL, DirectCompute

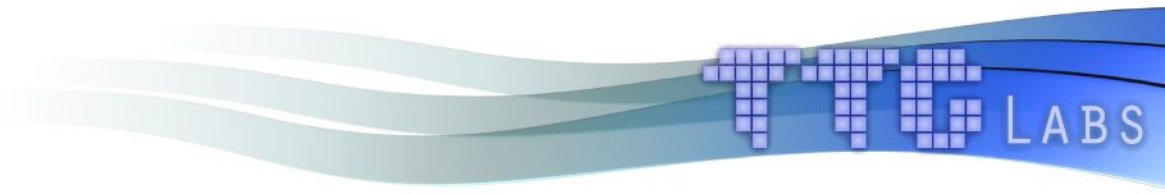
Проблема

```
DoSomething();
```

Проблема

```
cudaDoSomething();
```

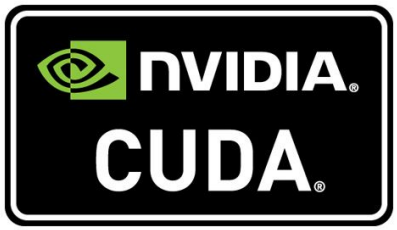




Проблема

Tesla C1060

```
cudaDoSomething_GT200();
```



Проблема

Tesla C1060

```
#if (__CUDA_ARCH__ == 130)
    cudaDoSomething_GT200();
#elif (__CUDA_ARCH__ == 200)
    cudaDoSomething_Fermi();
#elif (__CUDA_ARCH__ == 350)
    cudaDoSomething_Kepler();
#endif
```

Tesla C2050

Tesla K20

NVIDIA CUDA

LABS

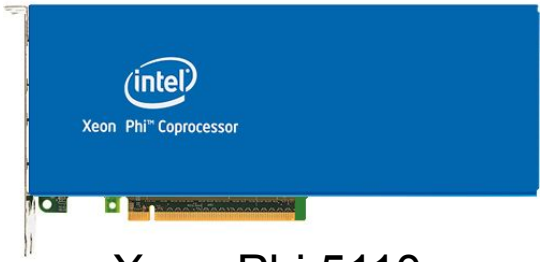
Проблема

Tesla C1060

```
#ifdef USE_CUDA
  #if (__CUDA_ARCH__ == 130)
    cudaDoSomething_GT200();
  #elif (__CUDA_ARCH__ == 200)
    cudaDoSomething_Fermi();
  #elif (__CUDA_ARCH__ == 200)
    cudaDoSomething_Kepler();
  #endif
#elif USE_OMP
    ompDoSomething_KNC();
#endif
```

Tesla C2050

Tesla K20



Xeon Phi 5110p

NVIDIA CUDA

OpenMP

LABS

Проблема

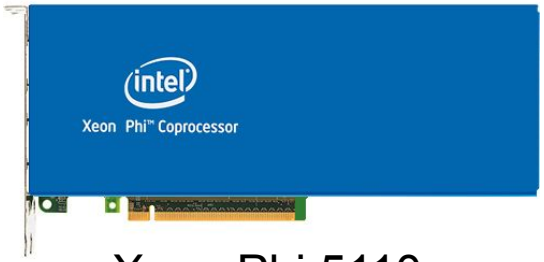
Mali MP-450

Tesla C1060

```
#ifdef USE_CUDA
  #if (__CUDA_ARCH__ == 130)
    cudaDoSomething_GT200();
  #elif (__CUDA_ARCH__ == 200)
    cudaDoSomething_Fermi();
  #elif (__CUDA_ARCH__ == 200)
    cudaDoSomething_Kepler();
  #endif
#elif USE_OMP
    ompDoSomething_KNC();
#elif USE_OCL
    oclDoSomething_Mali();
#endif
```

Tesla C2050

Tesla K20



Xeon Phi 5110p

NVIDIA CUDA

OpenMP

OpenCL

LABS

Решение

- **Идея:** давайте автоматизировать процесс подстройки программы под текущую платформу
  - Делаем одну «универсальную» версию
  - При запуске подстраиваем её под платформу
- **Вопрос:** а как это сделать?
  - Нужно получить возможность проводить оптимизацию
  - Нужно построить модель оптимизации
  - Нужно выбрать алгоритм оптимизации

Решение

- **Идея:** давайте автоматизировать процесс подстройки программы под текущую платформу
  - Делаем одну «универсальную» версию
  - При запуске подстраиваем её под платформу
- **Вопрос:** а как это сделать?
  - Нужно получить возможность проводить оптимизацию
  - Нужно построить модель оптимизации
  - Нужно выбрать алгоритм оптимизации

Примечание: решаем частный случай!

Шаг 1. Получение возможностей для оптимизации

- «Параметризация» программы

  Всё, что может влиять на производительность и зависит от текущей платформы, предлагается свести к набору параметров

- Виды параметров
  - Параметр-перечисление

    $\{ el_1, el_2, \ldots, el_N \}$

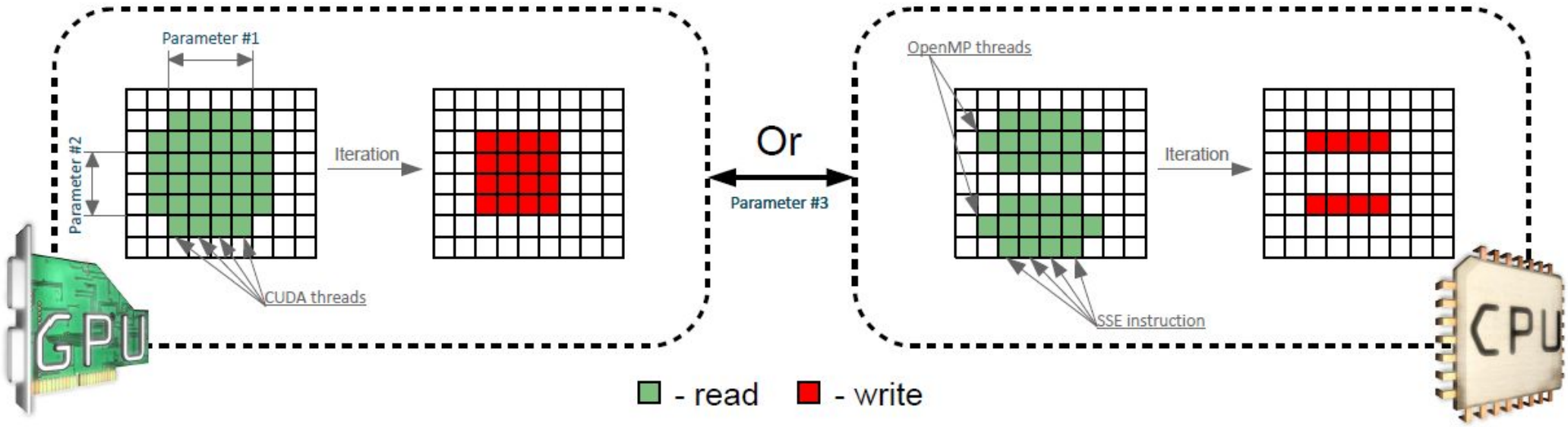
  - Параметр-интервал

    $[lo, hi)$

  - Параметр-разбиение

    $[0, 1] = [0, m_1) \cup [m_1, m_2) \cup \ldots \cup [m_K, 1]$

Шаг 1. Получение возможностей для оптимизации



Пример — решение дифференциального уравнения явной схемой на регулярной сетке

Шаг 2. Построение модели

$$J(x) \rightarrow min$$

Шаг 2. Построение модели

$$J(x) \rightarrow min$$

$J(x)$ – время выполнени одной итерации

Шаг 2. Построение модели

$$J(x) \rightarrow min$$

$J(x)$ – время выполнени одной итерации

$$x = (p_1, p_2, \dots, p_N)$$

Шаг 2. Построение модели

$$J(x) \to min$$

$$J(x) - время\ выполнени\ одной\ итерации$$

$$x = (p_1, p_2, \ldots, p_N)$$

$$\frac{\sum_{i=1}^{M} J(p_1^i, p_2^i, \ldots, p_N^i)}{\sum_{i=1}^{M} J(p_1^0, p_2^0, \ldots, p_N^0)} < 1 + \epsilon$$

Шаг 2. Построение модели

$$J(x) \to min$$

$$J(x) - время\ выполнени\ одной\ итерации$$

$$x = (p_1, p_2, \dots, p_N)$$

$$\frac{\sum_{i=1}^{M} J(p_1^i, p_2^i, \dots, p_N^i)}{\sum_{i=1}^{M} J(p_1^0, p_2^0, \dots, p_N^0)} < 1 + \epsilon$$

$$(p_1, p_2, \dots, p_N) \notin P$$

Шаг 2. Построение модели

$$J(x) \to min$$

$J(x)$ – время выполнени одной итерации

$$x = (p_1, p_2, \dots, p_N)$$

$$\frac{\sum_{i=1}^{M} J(p_1^i, p_2^i, \dots, p_N^i)}{\sum_{i=1}^{M} J(p_1^0, p_2^0, \dots, p_N^0)} < 1 + \epsilon$$

$$(p_1, p_2, \dots, p_N) \notin P$$

$$J(x^i) = \bar{J}(x^i) + \hat{J}(i),$$

где $\hat{J}(i) \to 0$ при $i \to \infty$

Шаг 2. Построение модели

$$J(x) \to min$$

$J(x)$ – время выполнени одной итерации

$$x = (p_1, p_2, \dots, p_N)$$

$$\frac{\sum_{i=1}^{M} J(p_1^i, p_2^i, \dots, p_N^i)}{\sum_{i=1}^{M} J(p_1^0, p_2^0, \dots, p_N^0)} < 1 + \epsilon$$

$$(p_1, p_2, \dots, p_N) \notin P$$

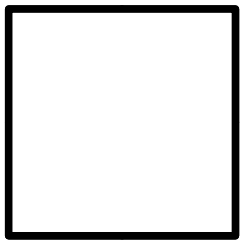
$$J(x^i) = \bar{J}(x^i) + \hat{J}(i),$$

где $\hat{J}(i) \to 0$ при $i \to \infty$

И так далее ...

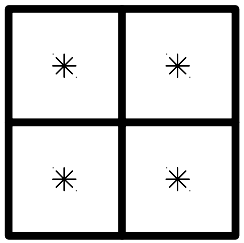
Шаг 3. Выбор алгоритма оптимизации

- Метод «Clicking»



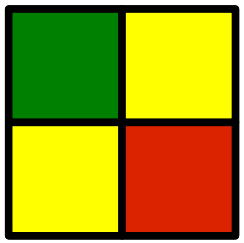
Шаг 3. Выбор алгоритма оптимизации

- Метод «Clicking»



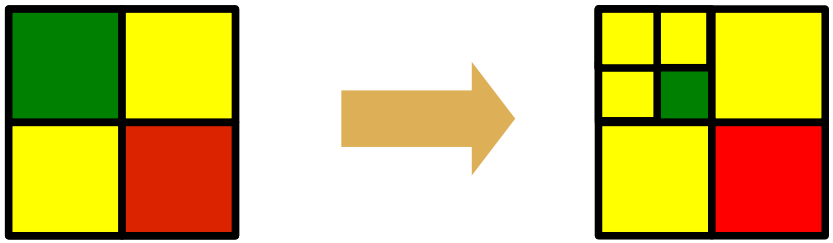
Шаг 3. Выбор алгоритма оптимизации

- Метод «Clicking»



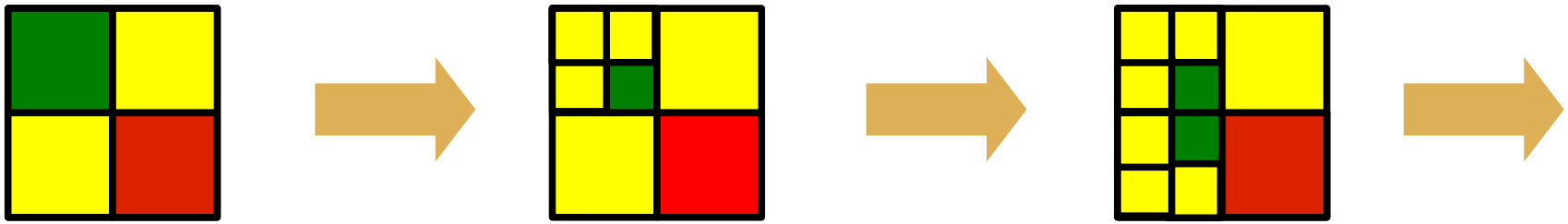
Шаг 3. Выбор алгоритма оптимизации

- Метод «Clicking»



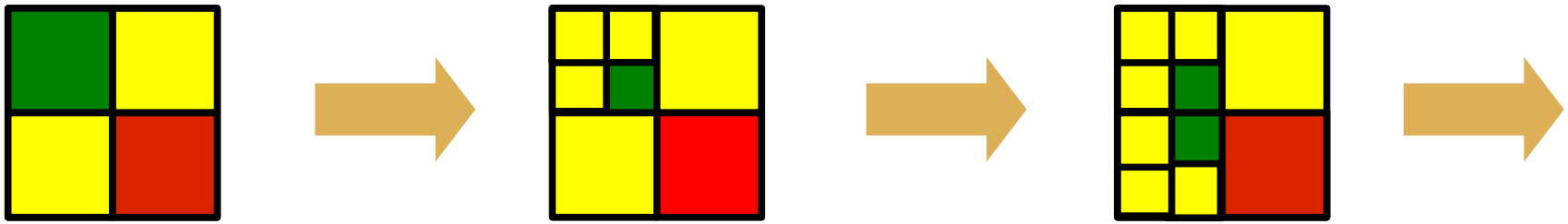
Шаг 3. Выбор алгоритма оптимизации

- Метод «Clicking»



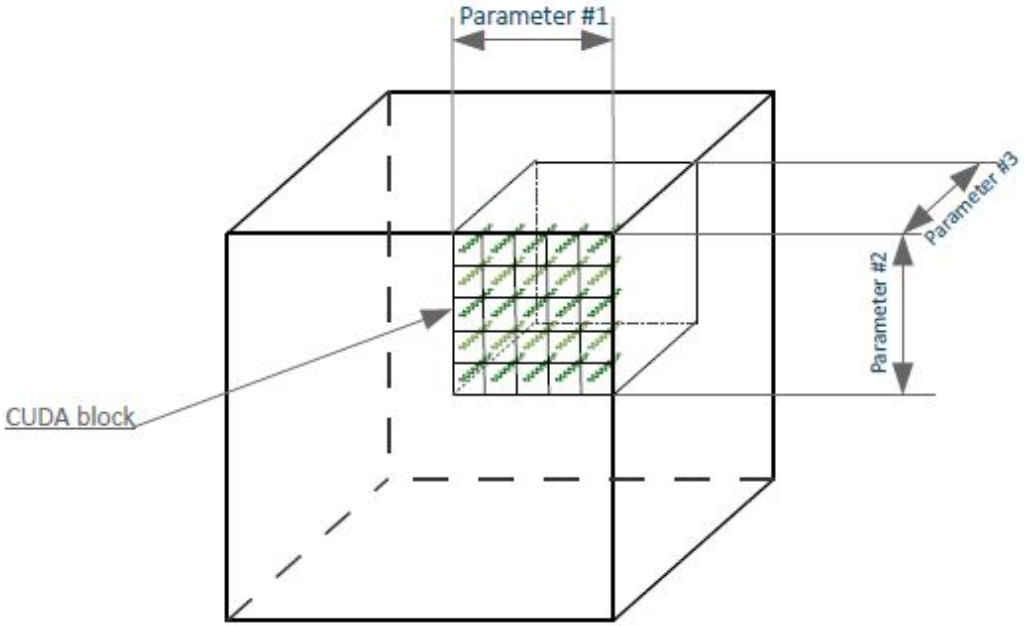
Шаг 3. Выбор алгоритма оптимизации

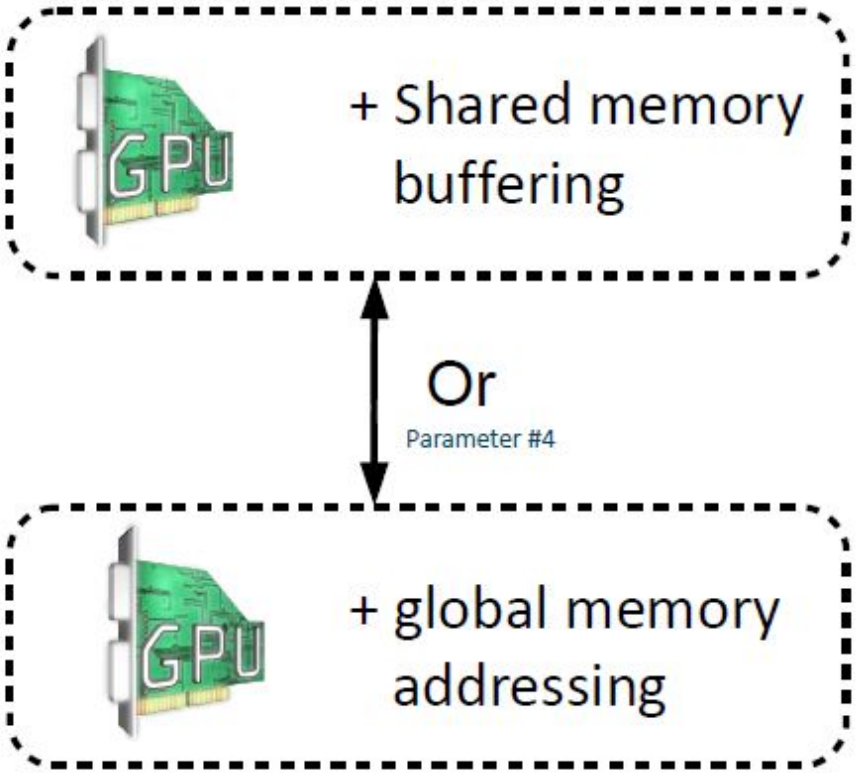
- Метод «Clicking»



- Введение разных режимов работы
  - «Устаканивание»
  - Поиск новых экстремумов
  - Нахождение в локальном экстремуме

Результаты. Уравнение Пуассона





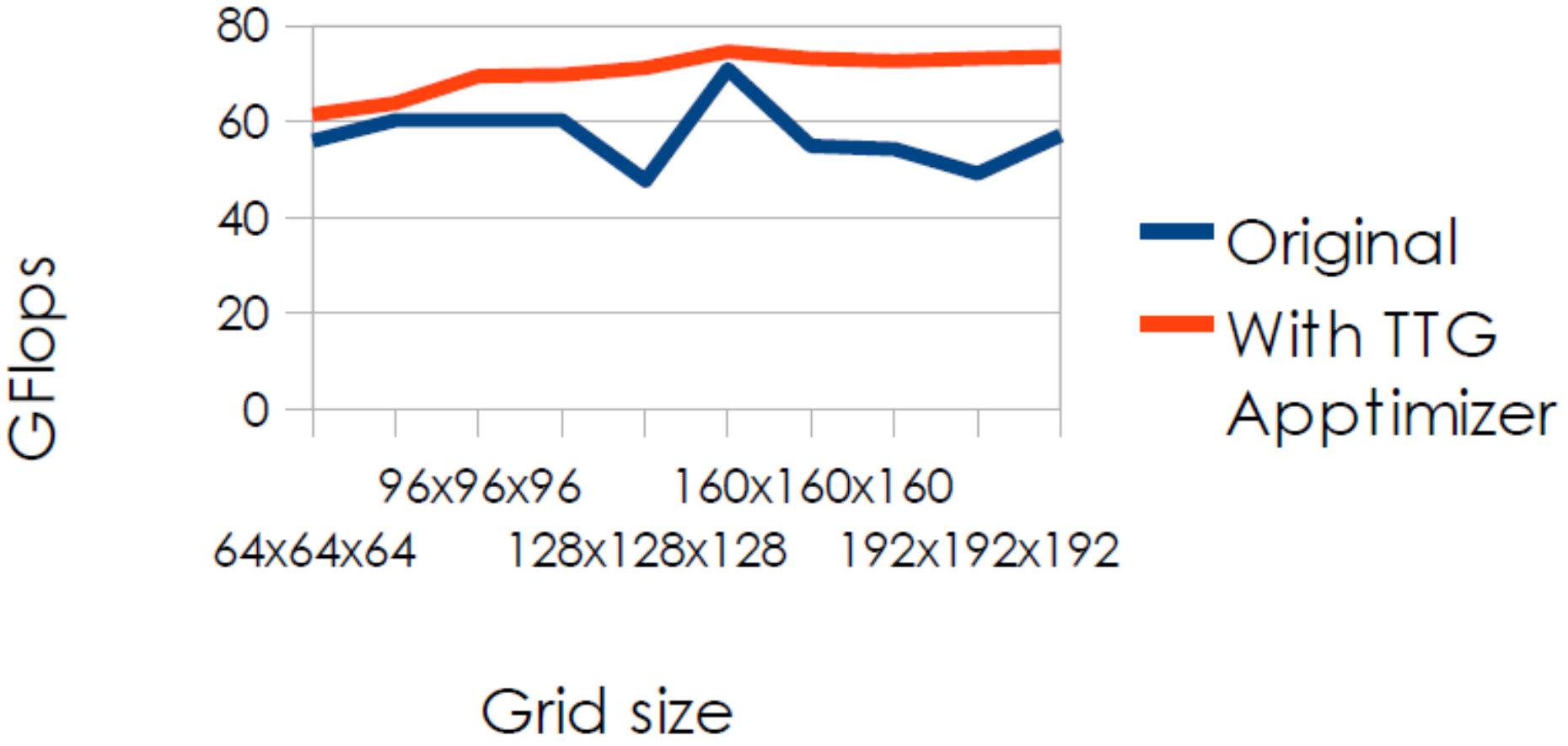
Результаты. Уравнение Пуассона

Grid size	GF 8800	GF 9800	GF 285	GF 480	GF 580	GF 680
64x64x64	9.7 + 10.1%	11.7 + 0.4%	23.1 + 4.1%	54.2 + 3.0%	61.1 + 17.9%	55.9 + 9.9%
96x96x96	10.1 + 18.5%	12.3 + 9.2%	29.9 + 1.9%	59.0 + 22.9%	69.3 + 25.0%	60.3 + 15.2%
128x128x128	10.2 + 16.1%	9.4 + 30.4%	23.1 + 4.2%	55.2 + 25.4%	63.5 + 28.6%	47.7 + 49.2%
160x160x160	10.3 + 21.2%	11.7 + 9.6%	25.5 + 16.8%	61.5 + 22.7%	71.4 + 33.1%	54.9 + 33.2%
192x192x192	10.2 + 21.2%	10.8 + 12.0%	25.8 + 6.1%	60.5 + 22.7%	70.4 + 33.5%	49.1 + 48.9%
Average	+17.4%	+12.3%	+6.6%	+19.4%	+27.6%	+31.3%

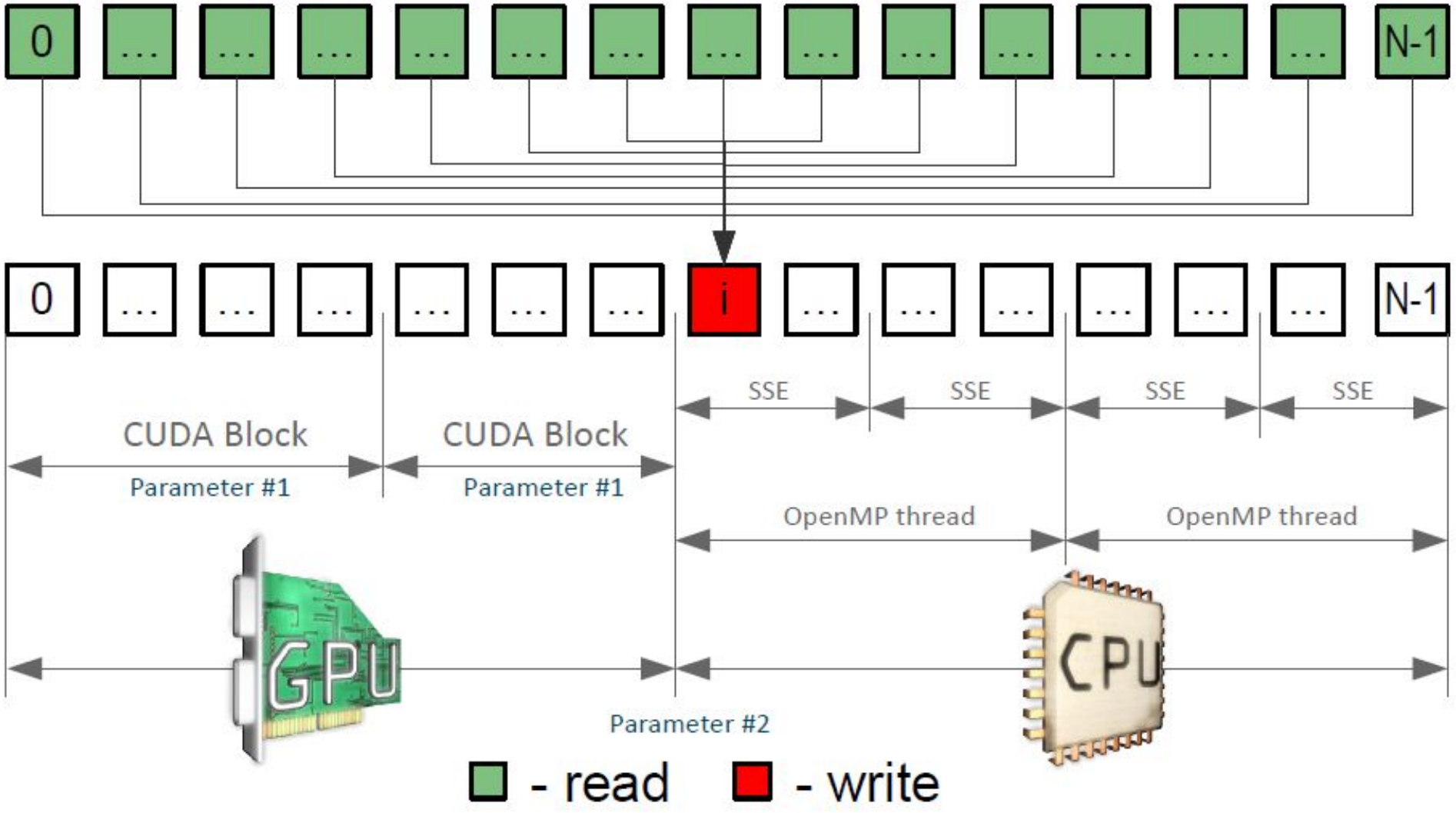
Результаты. Уравнение Пуассона

Grid size	GF 8800	GF 9800	GF 285	GF 480	GF 580	GF 680
64x64x64	9.7 + 10.1%	11.7 + 0.4%	23.1 + 4.1%	54.2 + 3.0%	61.1 + 17.9%	55.9 + 9.9%
96x96x96	10.1 + 18.5%	12.3 + 9.2%	29.9 + 1.9%	59.0 + 22.9%	69.3 + 25.0%	60.3 + 15.2%
128x128x128	10.2 + 16.1%	9.4 + 30.4%	23.1 + 4.2%	55.2 + 25.4%	63.5 + 28.6%	47.7 + 49.2%
160x160x160	10.3 + 21.2%	11.7 + 9.6%	25.5 + 16.8%	61.5 + 22.7%	71.4 + 33.1%	54.9 + 33.2%
192x192x192	10.2 + 21.2%	10.8 + 12.0%	25.8 + 6.1%	60.5 + 22.7%	70.4 + 33.5%	49.1 + 48.9%
Average	+17.4%	+12.3%	+6.6%	+19.4%	+27.6%	+31.3%

Результаты. Уравнение Пуассона



Результаты. Задача N-тел



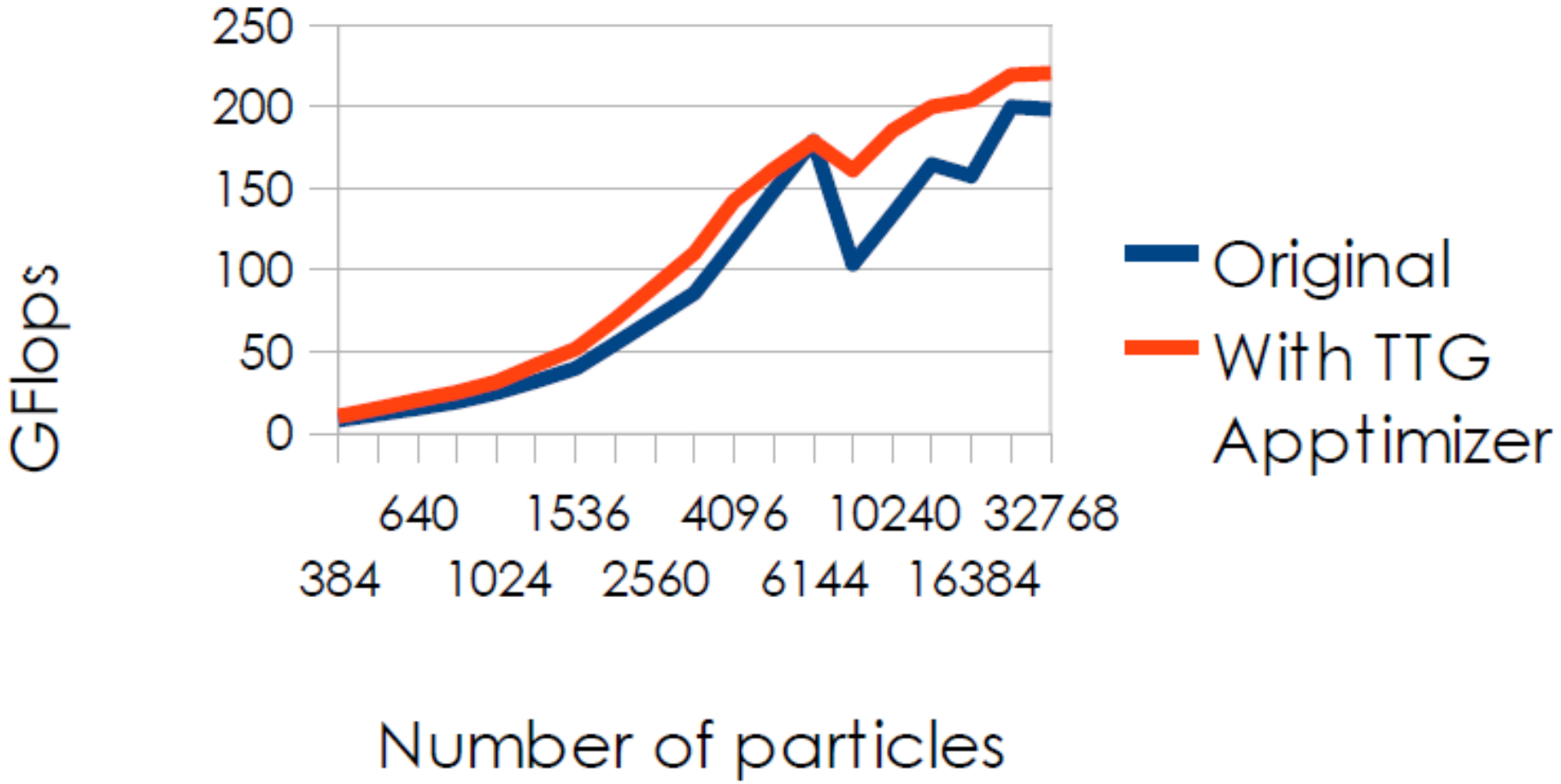
Результаты. Задача N-тел

Particles	GF 8800	GF 9800	GF 285	GF 480	GF 580	GF 680
0.5K	10.8 + 48.1%	13.1 + 56.8%	11.9 + 30.2%	15.3 + 2.8%	19.0 + 3.5%	18.1 + 0.5%
1.0K	22.4 + 48.1%	27.6 + 52.0%	24.8 + 28.3%	30.8 + 2.9%	38.4 + 3.4%	36.8 + 0.7%
2.0K	48.2 + 43.8%	60.5 + 45.7%	55.2 + 27.2%	65.3 + 1.3%	78.2 + 3.3%	75.2 - 0.0%
8.0K	90.9 + 0.0%	112.9 - 1%	103.8 + 55.3%	126.7 + 29.9%	229.9 - 0.1%	256.8 + 4.5%
16.0K	101 - 0.1%	127.7 - 0.0%	157.5 + 29.4%	125.3 +15.2%	252.7 + 0.7%	364.3 + 11.7%
32.0K	103.9 - 0.3%	131.7 - 0.1%	198.5 + 11.1%	160.6 +19.6%	255.3 + 0.7%	391.1 + 5.3%
Average	+23.3%	+25.6%	+30.3%	+12.0%	+1.9%	+3.7%

Результаты. Задача N-тел

Particles	GF 8800	GF 9800	GF 285	GF 480	GF 580	GF 680
0.5K	10.8 + 48.1%	13.1 + 56.8%	11.9 + 30.2%	15.3 + 2.8%	19.0 + 3.5%	18.1 + 0.5%
1.0K	22.4 + 48.1%	27.6 + 52.0%	24.8 + 28.3%	30.8 + 2.9%	38.4 + 3.4%	36.8 + 0.7%
2.0K	48.2 + 43.8%	60.5 + 45.7%	55.2 + 27.2%	65.3 + 1.3%	78.2 + 3.3%	75.2 - 0.0%
8.0K	90.9 + 0.0%	112.9 - 1%	103.8 + 55.3%	126.7 + 29.9%	229.9 - 0.1%	256.8 + 4.5%
16.0K	101 - 0.1%	127.7 - 0.0%	157.5 + 29.4%	125.3 +15.2%	252.7 + 0.7%	364.3 + 11.7%
32.0K	103.9 - 0.3%	131.7 - 0.1%	198.5 + 11.1%	160.6 +19.6%	255.3 + 0.7%	391.1 + 5.3%
Average	+23.3%	+25.6%	+30.3%	+12.0%	+1.9%	+3.7%

Результаты. Задача N-тел



Вопросы?

(m_krivov@ttgLabs.com)