Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Программирование для Intel Xeon Phi

Лабораторная работа №6
Оптимизация вычислительно трудоемкого программного модуля для архитектуры Intel Xeon Phi. Метод Монте-Карло

При поддержке компании Intel

Горшков А.В.
Кафедра математического обеспечения ЭВМ

Содержание

Введение

- В настоящее время в медицинских исследованиях существует потребность в развитии новых безопасных и доступных методов диагностики, поскольку используемые традиционные методы (МРТ, КТ, ПЭТ) имеют ряд ограничений.
- Классом наиболее перспективных методов диагностики, которые могут применяться как в сочетании с существующими методами, так и в некоторых случаях вместо них, являются оптические методы.
- В лабораторной работе рассматривается алгоритм моделирования для одного из таких методов – оптической диффузионной спектроскопии (ОДС).

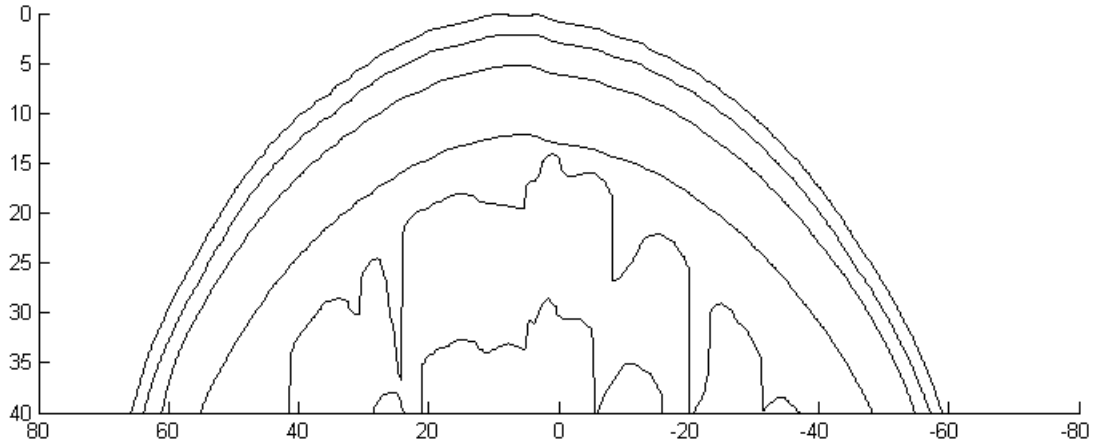
Цели работы

- **Обозначить основные направления и описать техники оптимизации алгоритма моделирования распространения излучения в сложных биологических тканях методом Монте-Карло для эффективного использования сопроцессоров Intel Xeon Phi:**
  - Изучение базовых принципов и особенностей алгоритма моделирования переноса излучения.
  - Прямой перенос алгоритма на сопроцессор Intel Xeon Phi.
  - Выявление «узких мест» в алгоритме с использованием соответствующих инструментов профилировки.
  - Выполнение оптимизации алгоритма с последующей проверкой результатов его производительности.

Тестовая инфраструктура

Процессор	Intel Xeon E5-2690 (2.9 GHz, 8 ядер)
Сопроцессор	Intel Xeon Phi 7110X (61 ядро, 244 потока)
Память	64 Gb
Операционная система	Linux CentOS 6.2
Компилятор, профилировщик, отладчик	Intel C/C++ Compiler 14

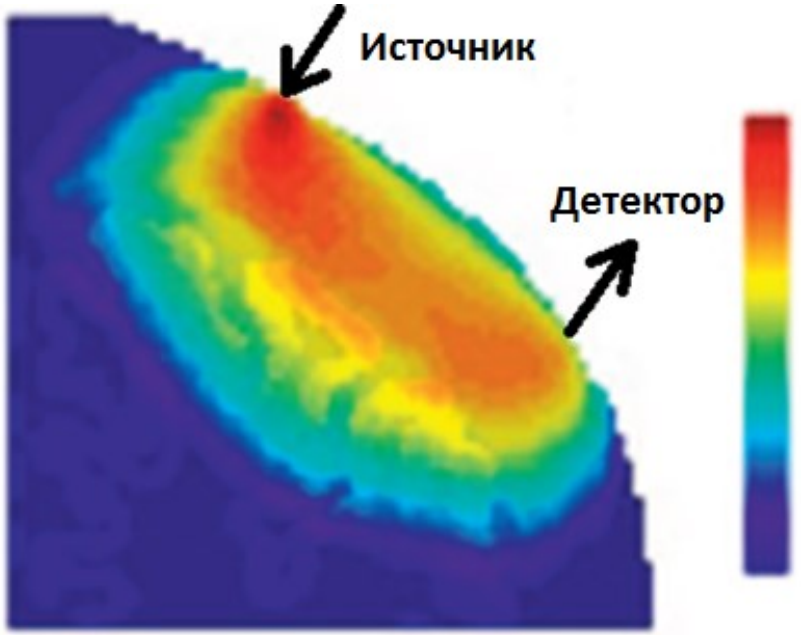
Алгоритм Монте-Карло моделирования переноса излучения…



- Рассматривается объект в трехмерном пространстве, состоящий из набора слоев.
- Каждый слой описывает определенный тип биологической ткани, обладающей набором оптических характеристик.
- Оптические характеристики постоянны в рамках слоя.
- Например, если моделировать перенос излучения в голове человека, то в ней можно выделить такие слои, как кожа головы, жировая ткань, череп, цереброспинальная жидкость, серое и белое вещество головного мозга.

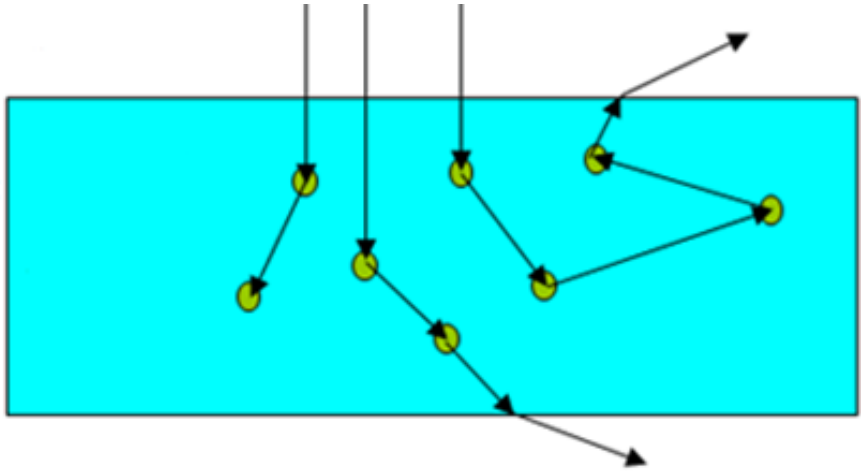
Алгоритм Монте-Карло моделирования переноса излучения…

- Каждый слой характеризуется набором границ, описываемых триангулированными поверхностями.
- Источник излучения представляет собой бесконечно тонкий луч фотонов и описывается направлением и положением в трехмерном пространстве.
- Детектор - некоторая замкнутая область на поверхности исследуемого объекта, которая способна улавливать проходящие через нее фотоны.



Алгоритм Монте-Карло моделирования переноса излучения…

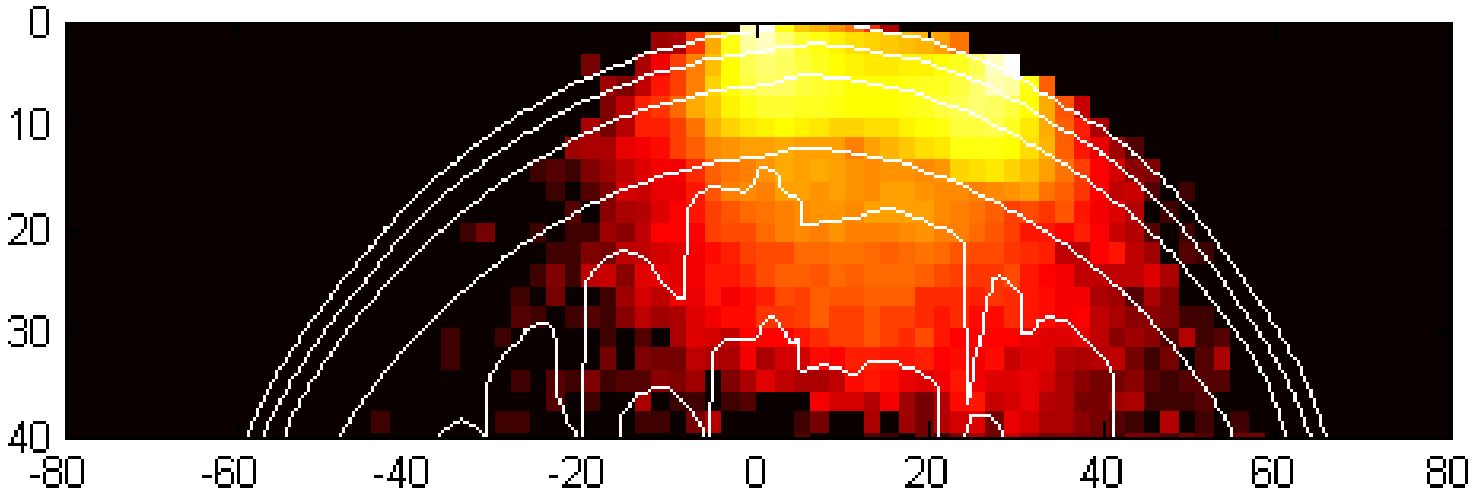
- Моделируется поведение набора фотонов в биоткани.
- Фотоны с одинаковой траекторией движения объединяются в пакет, вес пакета принимается равным 1.
- Пакет начинает движение от источника излучения.



- На каждом шаге трассировки:
  - Случайно выбирается новое направление движения пакета;
  - Случайно выбирается длина свободного пробега пакета фотонов;
  - Часть фотонов пакета поглощается средой (вес пакета уменьшается).
- Трассировка завершается, если пакет вылетел за границы среды, либо если вес пакета становится меньше минимального.

Алгоритм Монте-Карло моделирования переноса излучения

- Результатами моделирования являются:
  - интенсивность рассеянного назад излучения на детекторах (сигнал на детекторах);
  - фотонные карты траекторий для каждого детектора (для фотонов, попавших из источника на детектор);
  - общая карта траекторий (для всех фотонов).



Описание базовой версии программы…

- Исходные коды различных версий программы находятся в каталоге **src**.
- Исследуемый программный пакет состоит из трех компонент:
  - Библиотеки для чтения XML файлов **TinyXML** [http://www.grinninglizard.com/tinyxml/];
  - Библиотеки, содержащей набор функций для трассировки фотона в среде (**xmcml**);
  - Исполняемого модуля, в задачи которого входит чтение параметров задачи, параллельный запуск процесса моделирования и запись результатов работы программы в файл (**xmcmlLauncher**).

Описание базовой версии программы…

- Для компиляции программы в операционных системах семейства Windows следует воспользоваться проектом **Visual Studio 2010**.
- При работе на ОС Linux компиляция осуществляется скриптами ***cpu_make.sh*** (компиляция для CPU) и ***mic_make.sh*** (компиляция для Intel Xeon Phi в режиме работы только на сопроцессоре).

Описание базовой версии программы…

- Входными файлами программы являются:
  - XML файл с описанием входных параметров задачи. В качестве примера к программе прилагается файл ***brain_915.xml***, описывающий биоткани головы человека при длине волны источника излучения в 915 нм.
  - SURFACE файл с описанием геометрии границ биотканей. В качестве примера к программе прилагается файл ***planes.surface***. В данном случае реальная геометрия тканей головы человека аппроксимируется набором параллельных плоскостей. Файл имеет бинарный формат.

Описание базовой версии программы…

- Для исследования производительности и проведения оптимизации программы имеет смысл варьировать только следующие параметры задачи (содержатся в XML файле):
  - **NumberOfPhotons** – количество трассируемых фотонов. От этого параметра напрямую зависит время работы программы. Зависимость линейна.
  - **Area.PartitionNumber** – содержит количество разбиений сетки по осям X, Y и Z. Сетка используется для хранения фотонных карт траекторий. Чем больше ее размер, тем больше памяти необходимо программе для работы.

Описание базовой версии программы…

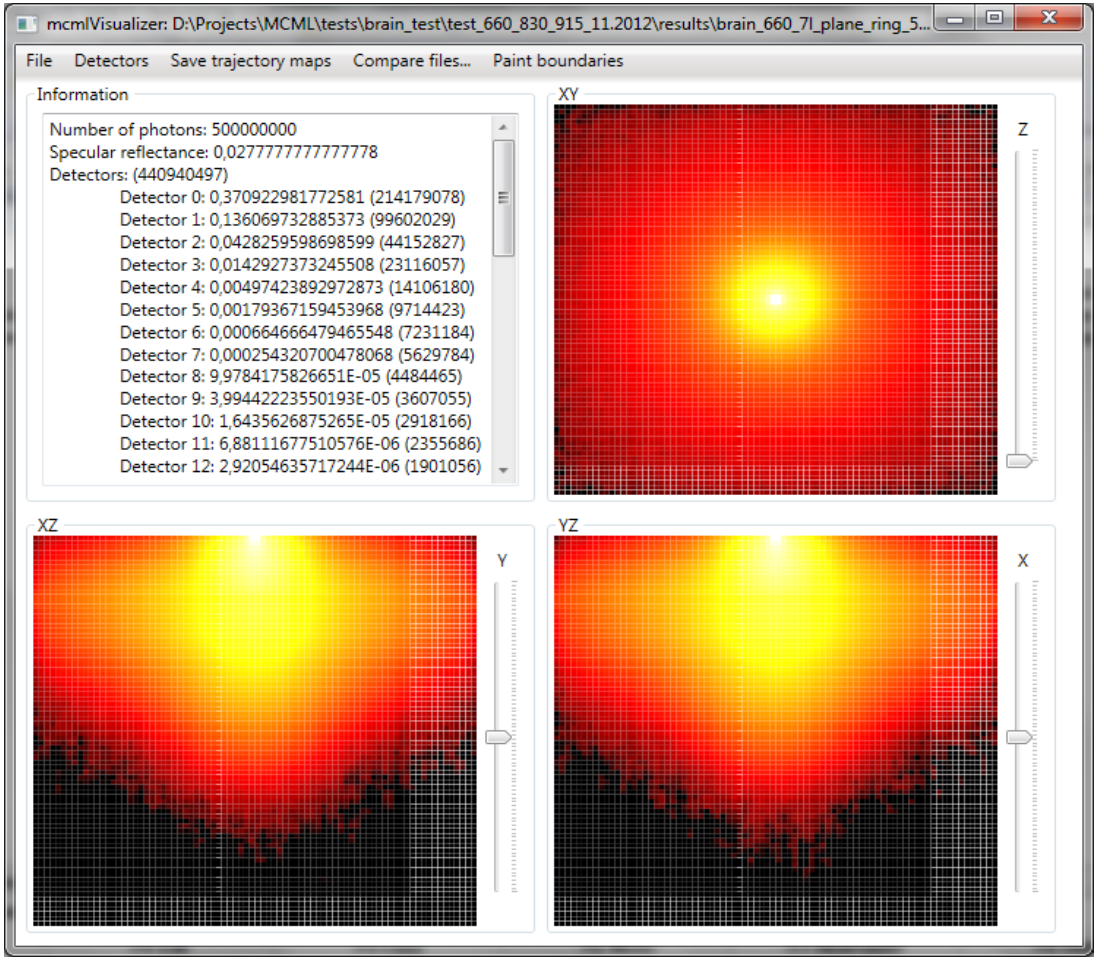
- При запуске программы используются следующие параметры командной строки (запуск программы без параметров приводит к выводу соответствующей справки):

```
xmcmlLauncher.exe
      -i <ИмяXMLФайла>.xml
      -s <ИмяSURFACEФайла>.surface
      -o <ИмяФайлаСРезультатамиМоделирования>.mcml.out
      -nthreads <КоличествоПараллельныхПотоков>
```

- Для просмотра результатов моделирования следует использовать программу **mcmlVisualizer**. Программа использует .NET Framework 4.0 и предназначена для работы на ОС Windows.

Описание базовой версии программы

- Помимо визуализации результатов, программа **mcmlVisualizer** позволяет проводить сравнение двух *.mcml.out файлов на предмет их совпадения.
- Обычное побитовое сравнение не всегда является показательным, поэтому для этих целей лучше использовать предложенную программу.



Прямой перенос базовой версии: особенности программы…

- Распараллеливание ведется по фотонам, каждый поток обсчитывает свой набор траекторий (*./Base/xmcmlLauncher/launcher_omp.cpp*):

```
void LaunchOMP(InputInfo* input, OutputInfo* output,
    MCG59* randomGenerator, int numThreads)
{
    omp_set_num_threads(numThreads);

    …

    #pragma omp parallel
    {
        int threadId = omp_get_thread_num();

        for (uint64 i = threadId;
            i < input->numberOfPhotons; i += numThreads)
        {
            ComputePhoton(specularReflectance, input,
                &(threadOutputs[threadId]),
                &(randomGenerator[threadId]), trajectory);
        }
    }
    …
}
```

Прямой перенос базовой версии: особенности программы…

- Результатами трассировки являются:
  - Величина сигнала на детекторе – массив **weightInDetector**, количество элементов равно числу детекторов (в примере – 10), размер – 80 Б.
  - Общая фотонная карта траекторий – массив **commonTrajectory**, количество элементов равно числу ячеек сетки (в примере – 100*100*50=500 000 элементов), размер – 4 МБ.
  - Фотонные карты траекторий для каждого детектора – массив **detectorTrajectory**. Для каждого из детекторов хранится массив, аналогичный **commonTrajectory**, того же размера. Общий размер данных для 10 детекторов – 40 МБ.

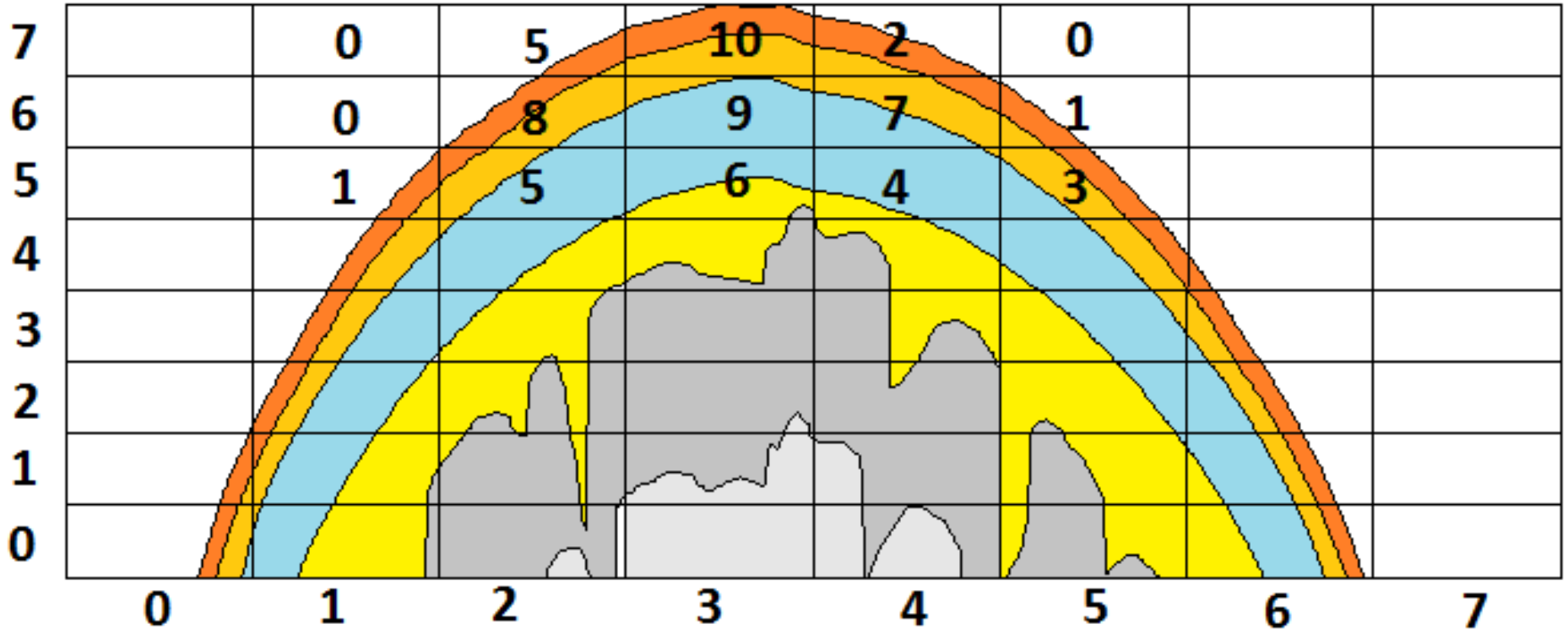
Прямой перенос базовой версии: особенности программы…

- Результаты трассировки сохраняются в структуре OutputInfo (*./Base/xmcml/mcml_kernel_types.h*):

```
typedef struct __OutputInfo
{
    uint64 numberOfPhotons;
    double specularReflectance;
    uint64* commonTrajectory;
    int gridSize;
    double* weightInDetector;
    DetectorTrajectory* detectorTrajectory;
    int numberOfDetectors;
} OutputInfo;
```

Прямой перенос базовой версии: особенности программы…

- Фотонные карты траекторий хранятся в виде трехмерной сетки следующего вида:



Прямой перенос базовой версии: особенности программы…

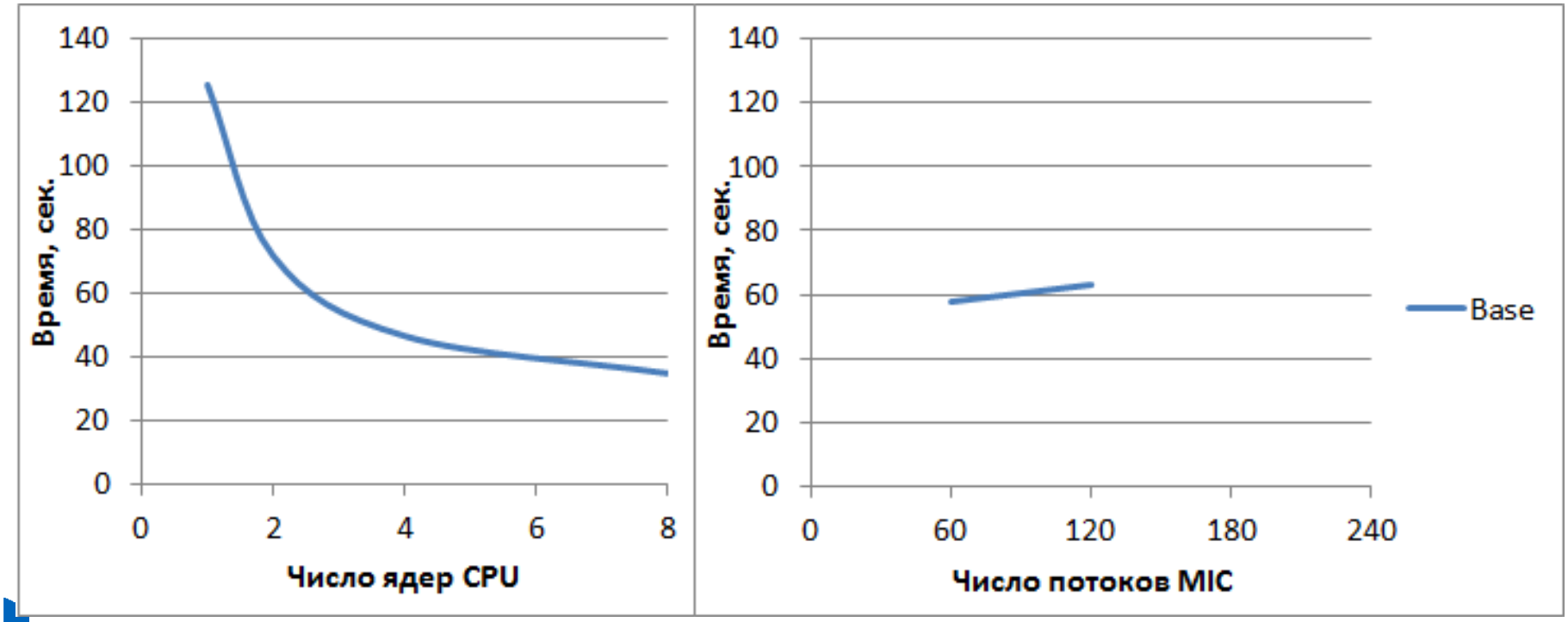
- Каждому фотону требуется доступ к массивам результатов для их обновления, а значит необходимо либо использовать синхронный доступ к данным на запись, либо воспользоваться техникой дублирования данных.
- В исходной версии программы применяется второй подход: создаются копии результирующих массивов для каждого потока исполнения, а после окончания расчетов данные этих копий суммируются.
  - Объем дополнительной памяти на CPU (8 потоков): 350 МБ
  - Объем дополнительной памяти на MIC (240 потоков): более 10 ГБ **(!)**

Прямой перенос базовой версии: особенности программы

- Еще одна особенность алгоритма состоит в необходимости хранения траектории каждого отдельного фотона в процессе его трассировки.
- В исходной версии выделяется массив для хранения траектории для каждого потока. С точки зрения структур данных используется все та же трехмерная сетка, соответственно размер массива равен 4 МБ на поток.
- Перед началом трассировки каждого отдельного фотона этот массив обнуляется.

Прямой перенос базовой версии...

- Для переноса используется режим работы только на сопроцессоре.
- Первый шаг – компиляция программы под Intel Xeon Phi.
  - Воспользуйтесь скриптом ***mic_make.sh***.

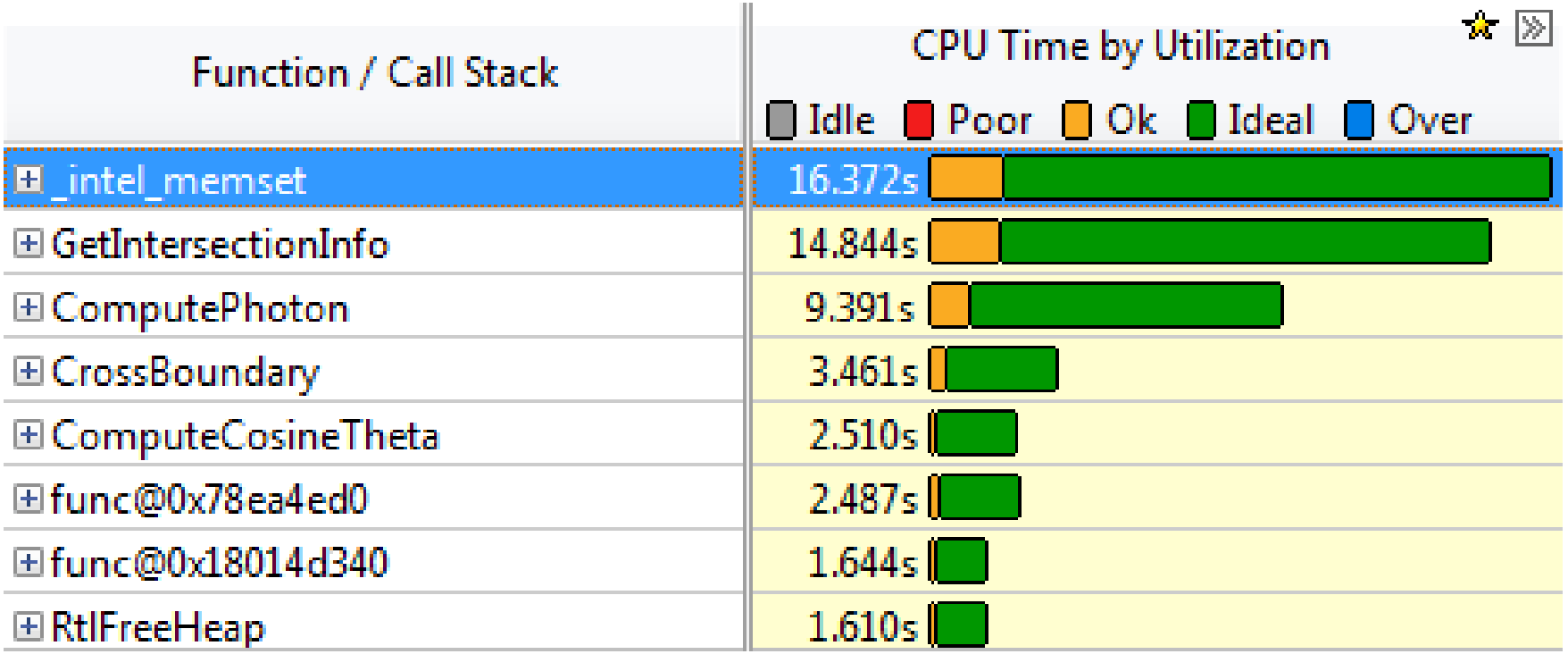


Прямой перенос базовой версии

- Время работы программы на сопроцессоре в 2 раза больше, чем на CPU.
- Запуск программы в 240 потоков на Intel Xeon Phi невозможен в связи с недостатком памяти на сопроцессоре.

Оптимизация 1: обновление структуры данных для хранения траектории фотона…

- Результаты профилировки базовой версии с помощью Intel VTune Amplifier XE:

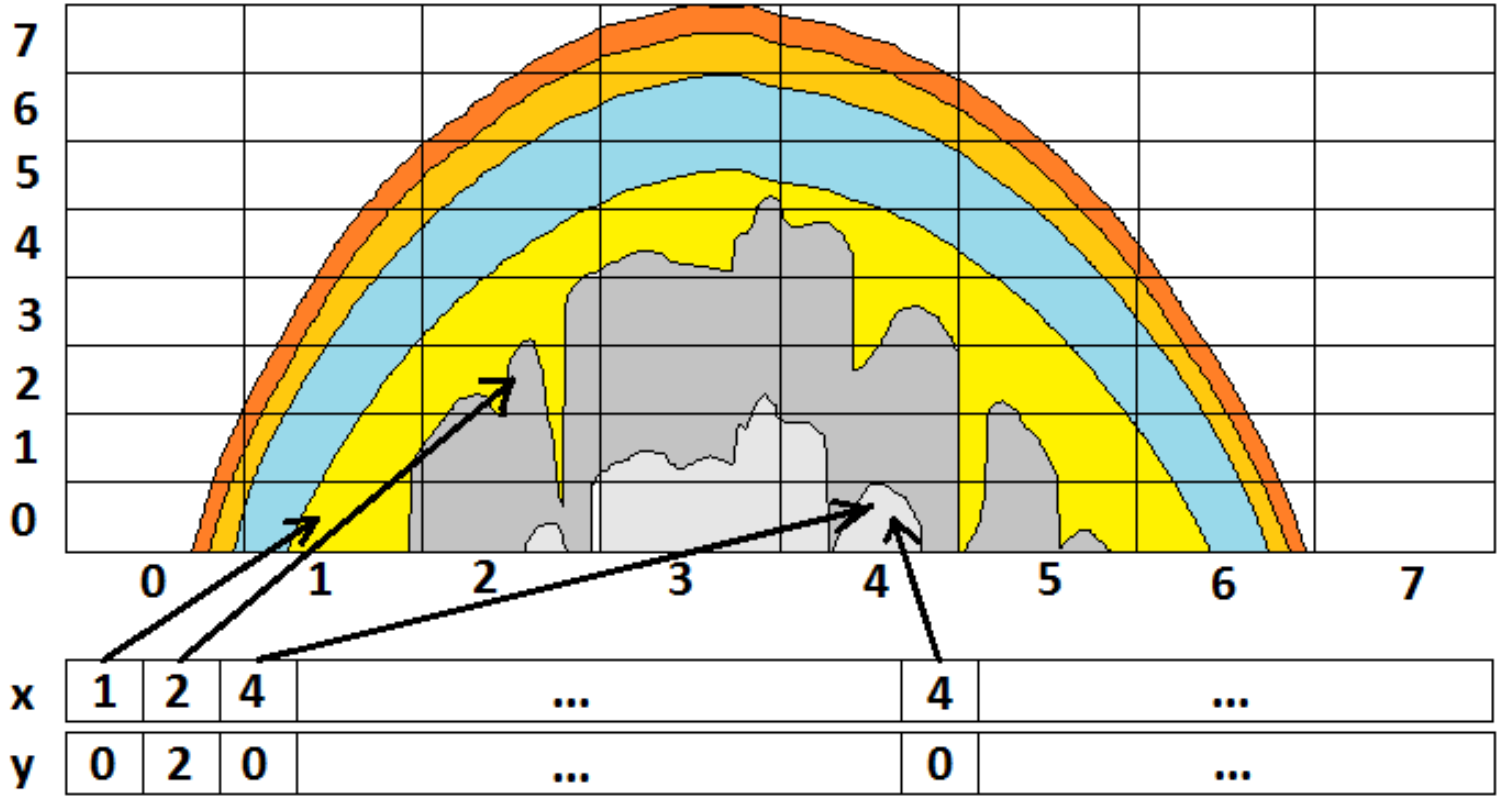


Оптимизация 1: обновление структуры данных для хранения траектории фотона…

- Наибольшее время занимает функция memset(), которая используется в момент начала процесса трассировки фотона для обнуления памяти, предназначенной для хранения траектории его движения (функция ComputePhoton(), *./Base/xmcml/mcml_kernel.cpp*).
- Кроме необходимости обнуления памяти на каждой итерации, текущий подход обладает еще несколькими недостатками. А именно, размер массива слишком велик по сравнению с размером L2 кэш памяти, и доступ к элементам массива осуществляется в случайном порядке (в силу случайности траектории фотона).

Оптимизация 1: обновление структуры данных для хранения траектории фотона…

- Для устранения описанных выше недостатков предлагается хранить не число посещений данной ячейки для всей сетки, а список координат посещенных ячеек:



Оптимизация 1: обновление структуры данных для хранения траектории фотона…

- Для хранения траектории фотона в виде списка координат вводится новый тип данных (*./Opt1/xmcml/mcml_kernel_types.h*):

```
#define MAX_TRAJECTORY_SIZE 2048

typedef struct __PhotonTrajectory
{
    byte x[MAX_TRAJECTORY_SIZE];
    byte y[MAX_TRAJECTORY_SIZE];
    byte z[MAX_TRAJECTORY_SIZE];
    uint size;
} PhotonTrjectory;
```

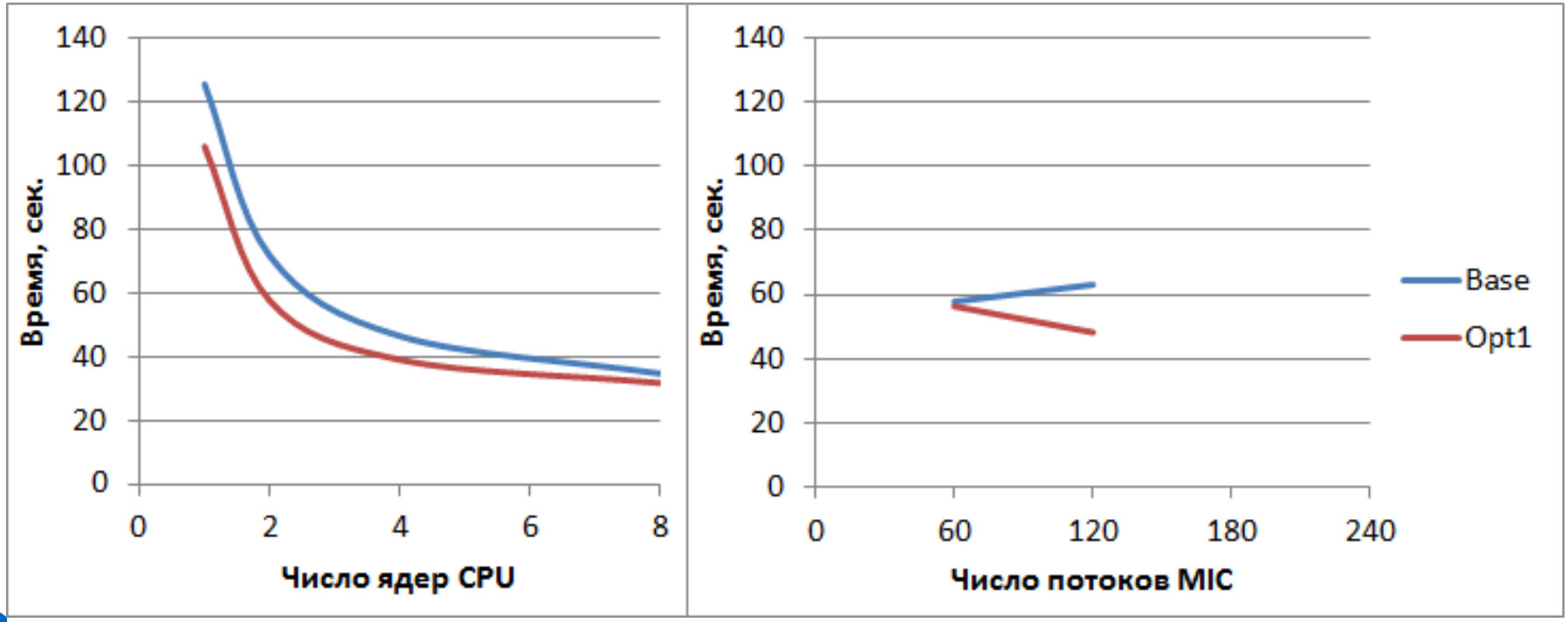
- Экспериментальные данные показывают, что максимальная длина траектории фотона в данном примере не превосходит 2048 шагов (эта величина существенно зависит от параметров биотканей и размера сетки). Размер структуры данных в этом случае составляет всего 6 КБ.

Оптимизация 1: обновление структуры данных для хранения траектории фотона…

- Для того чтобы использовать эту новую структуру данных, в программу необходимо внести некоторые изменения:
  - в функции LaunchOMP() меняем часть, связанную с выделением памяти под траекторию (*./Opt1/xmcmlLauncher/launcher_omp.cpp*).
  - в функции ComputePhoton() избавляемся от обнуления памяти (*./Opt1/xmcml/mcml_kernel.cpp*).
  - вводим новый тип данных – byte3 (*./Opt1/xmcml/mcml_types.h*).
  - вводим новую функцию – GetAreaIndexVector() (*./Opt1/xmcml/mcml_kernel.cpp*).
  - исправляем код функции UpdatePhotonTrajectory() (*./Opt1/xmcml/mcml_kernel.cpp*).
  - необходимо полностью обновить функцию UpdateDetectorTrajectory() (*./Opt1/xmcml/mcml_kernel.cpp*).

Оптимизация 1: обновление структуры данных для хранения траектории фотона

- Ускорение на CPU
  - В 1 поток – 18%;
  - В 8 потоков – 9%.
- Ускорение на MIC в 120 потоков – 31%.



Оптимизация 2: отказ от использования дублирующих массивов…

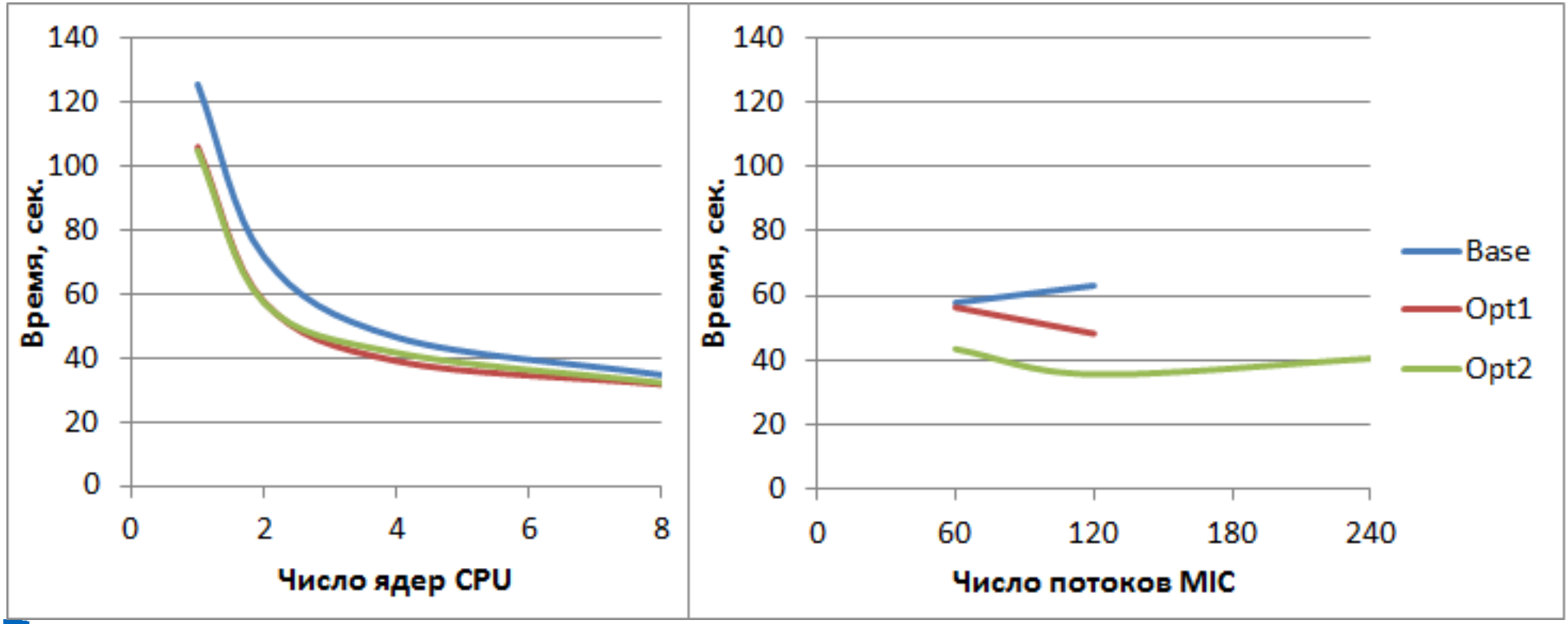
- Основной недостаток текущей версии программы состоит в том, что мы не можем использовать все возможности сопроцессора – **число потоков, на которых мы можем запустить программу, ограничено объемом памяти сопроцессора**.
- В нашем случае дублирование выполнено для двух типов результатов: общей фотонной карты траекторий и фотонных карт для каждого из детекторов:
  - Общая фотонная карта траекторий: размер – 4 МБ на поток; число одновременных операций доступа велико => **отказ от дублирования нецелесообразен**.
  - Фотонные карты для детекторов: размер – 40 МБ на поток; вероятность, что фотоны из разных потоков одновременно попадут на детектор мала => **отказ от дублирования целесообразен.**

Оптимизация 2: отказ от использования дублирующих массивов…

- Для реализации предложенной идеи следует внести изменения в код программы:
  - основные изменения коснутся функции параллельного запуска процесса моделирования LaunchOMP() (*./Opt2/xmcmlLauncher/launcher_omp.cpp*);
  - вместо функций InitOutput() и FreeOutput() используются InitThreadOutput() и FreeThreadOutput() (*./Opt2/xmcmlLauncher/launcher_omp.cpp*);
  - избавившись от дублирования, необходимо позаботиться о синхронизации в функциях UpdateDetectorTrajectory() и UpdateWeightInDetector() (*./Opt2/xmcml/mcml_kernel.cpp*).

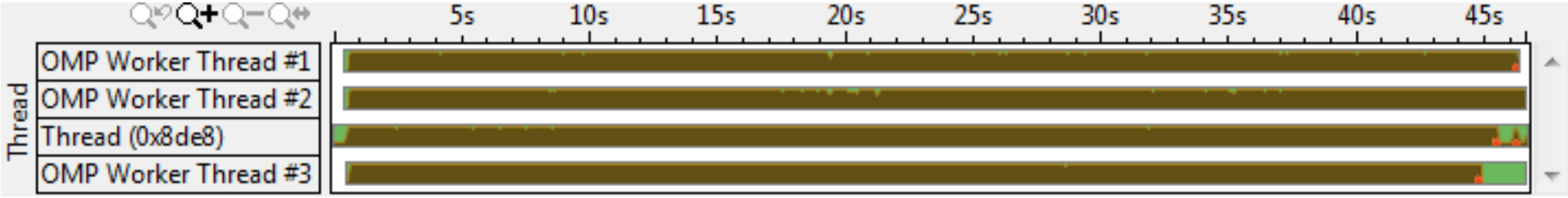
Оптимизация 2: отказ от использования дублирующих массивов

- Время работы на CPU практически не изменилось.
- Ускорение на MIC в 120 потоков – 35%.
- Появилась возможность запуска программы в 240 потоков.



Оптимизация 3: балансировка нагрузки…

- Профилировка программы с использованием Intel VTune Amplifier XE выявила еще одну особенность используемого алгоритма – разное время выполнения отдельных потоков:



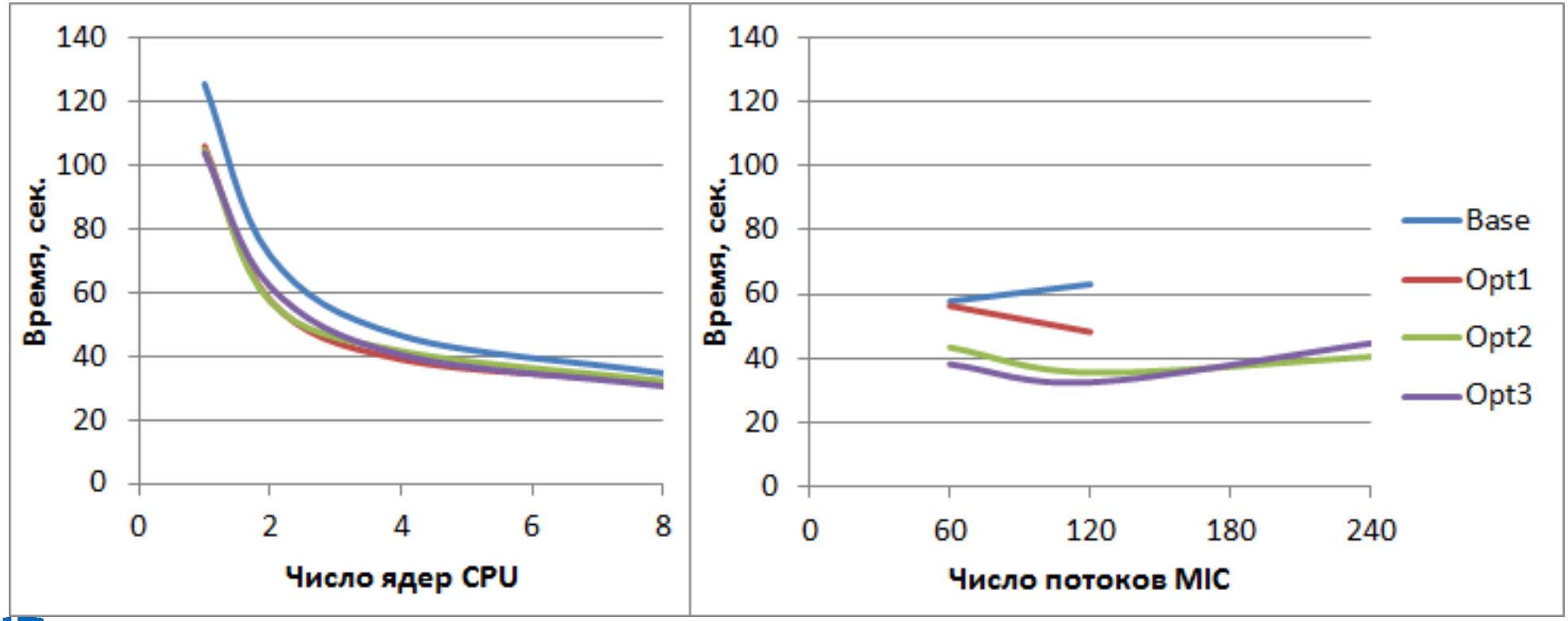
Оптимизация 3: балансировка нагрузки…

- Меняем статическую схему балансировки нагрузки на динамическую (*./Opt3/xmcmlLauncher/launcher_omp.cpp*):

```
void LaunchOMP(InputInfo* input, OutputInfo* output, MCG59*
randomGenerator, int numThreads)
{
    …
    #pragma omp parallel for schedule(dynamic)
    for (uint64 i = 0; i < input->numberOfPhotons; ++i)
    {
        int threadId = omp_get_thread_num();
        ComputePhoton(specularReflectance, input,
            &(threadOutputs[threadId]),
            &(randomGenerator[threadId]),
            &(trajectory[threadId]));
    }
    …
}
```

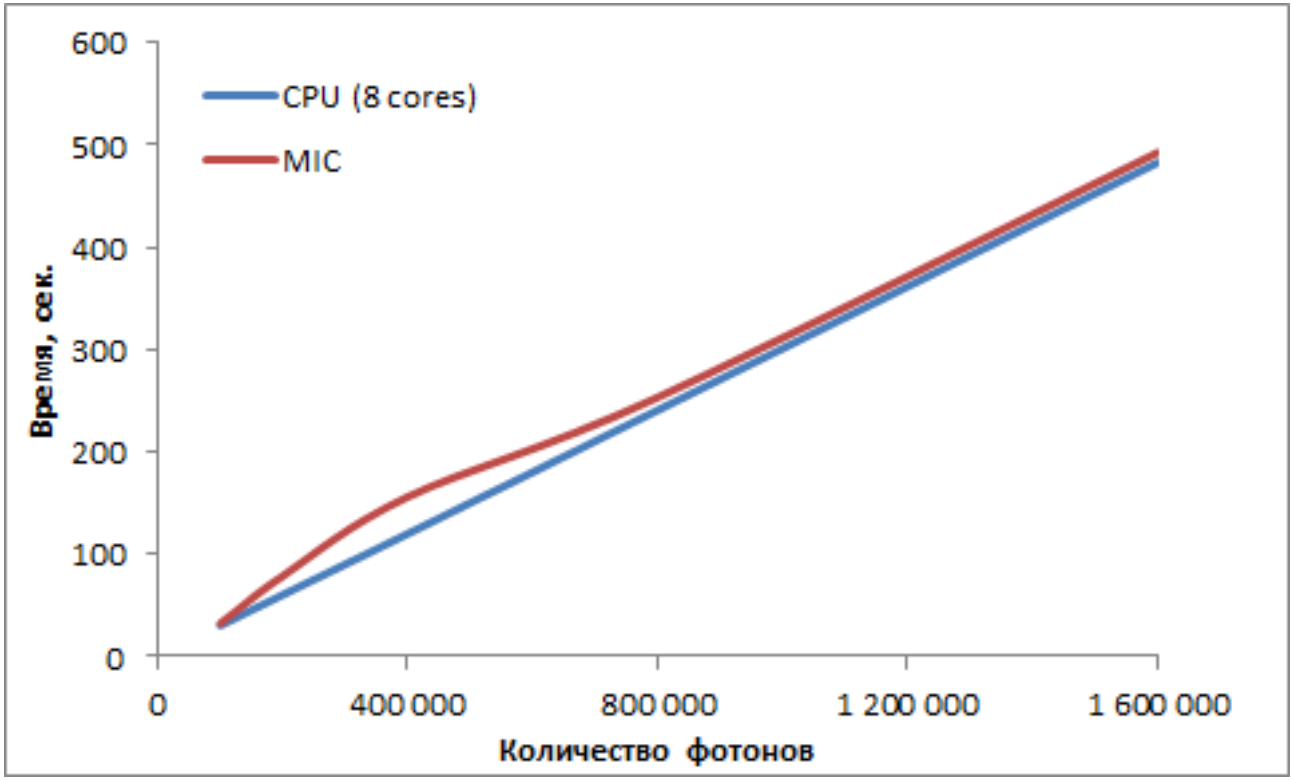
Оптимизация 3: балансировка нагрузки

- Время работы на CPU незначительно уменьшилось.
- Время работы на MIC уменьшилось на 9%.
- Минимальное время на MIC достигается на 120 потоках.



Общие результаты оптимизации

- Удалось повысить производительность программы для Intel Xeon Phi вдвое.
- Время работы программы на сопроцессоре Intel Xeon Phi примерно соответствует времени ее работы на CPU.



Дальнейшая оптимизация

- Для эффективной работы программы на Intel Xeon Phi требуется ее векторизация.
- Необходим анализ существующих алгоритмов поиска пересечений с целью выбора наиболее подходящего для работы на сопроцессоре, и его дальнейшая оптимизация.
- Отдельного обсуждения заслуживает выбор модели программирования на сопроцессоре, которую следует использовать для переноса.
- Не стоит пренебрегать такими стандартными подходами к оптимизации кода на Intel Xeon Phi, как работа с выровненными данными, использование команд программной предвыборки данных и др.

Дополнительные задания

- Реализовать версию алгоритма с дублированием массива сигналов на детекторах – output->weightInDetector (избавиться от синхронизации записи результатов в этот массив). Оценить целесообразность такой оптимизации.
- Реализовать версию алгоритма, работающего в режиме offload. Выносить на сопроцессор имеет смысл только распараллеленную часть кода, все остальное должно делаться на процессоре.
- Применить технику двойной буферизации в режиме offload для обеспечения эффективной передачи результатов моделирования с сопроцессора на CPU. На каждой итерации, начиная со второй, в момент обсчета новой порции фотонов можно организовать передачу результатов трассировки предыдущей порции на CPU.

Литература

- Intel Xeon Phi Coprocessor System Software Developers Guide, revision 2.03, 2012
- Best Known Methods for Using OpenMP on Intel Many Integrated Core (Intel MIC) Architecture, Volume 1a, January 29, 2013
- J. Jeffers, J. Reinders. Intel Xeon Phi Coprocessor High Performance Programming. -Morgan Kaufmann, 2013. -432 p.
- Intel Developer Zone [http://software.intel.com/en-us/mic-developer]

Авторский коллектив

- Горшков Антон Валерьевич,
  ассистент кафедры Математического обеспечения ЭВМ факультета ВМК ННГУ.
  anton.v.gorshkov@gmail.com