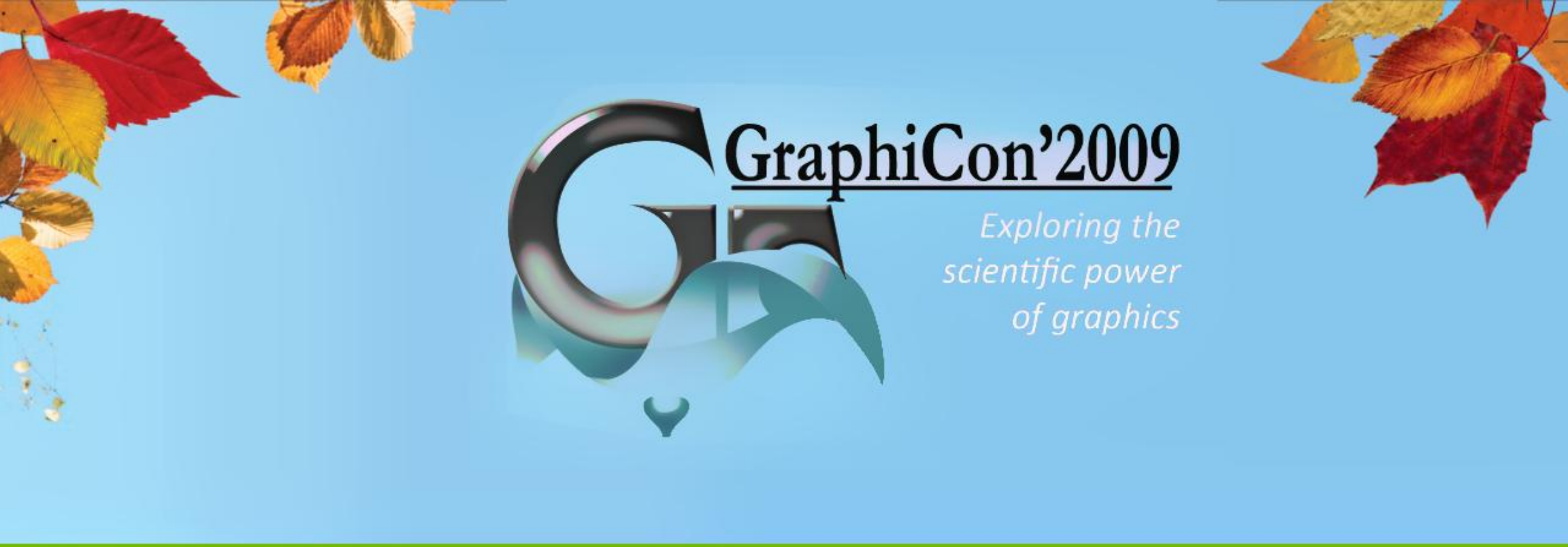


Performance Primitives for Multimedia Processing and Coding

Fairmont Hotel, San Jose | 10.01.2009 | Frank Jargstorff, Anton Obukhov

NVIDIA.

Что такое NPP?

- Библиотека функций (**примитивов**) на языке C, созданных для работы на CUDA
- API соответствует IPP (Intel Integrated Performance Primitives)
  - Подмножество функциональности IPP
  - Фокус на обработке изображений и видео
- Быстрее IPP до 32x раз
- Свободно распространяется
  - Бинарные файлы скомпилированные под Windows, Linux (32- и 64 бит) и Mac OS X
- Release 1.0: Available at
  - http://www.nvidia.com/npp

Взгляд сверху

- Цели и задачи NPP
- Как использовать NPP?
  - Примеры
- Что внутри NPP?
  - Типы данных
  - Функции
- Производительность
  - Масштабирование
  - Сравнение с IPP

Цели и задачи NPP

- Простота использования
  - не трубует знаний об архитектуре GPU
  - легко интегрируется в проекты
    - легко добавляется в существующие проекты
    - работает в связке с другими библиотеками
- Выполнение на GPU с поддержкой CUDA
- Высокая производительность
  - освобождает разработчика от оптимизационного ада
- Инкапсуляция в алгоритмические блоки (Примитивы)
  - Решение задач путем построения цепей из примитивов

Простота использования

- Идентично реализует Intel IPP API
  - IPP широко используется в программах с большим запросом к производительности
  - удобно спроектированный API
- Использует CUDA “Runtime API”
  - указатели NPP API являются указателями в пространстве глобальной памяти GPU
  - управление памятью предоставлено программисту
- Основана на указательной арифметике; API без оверхеда
  - взаимодействие на уровне указателей с кодом (C for CUDA) и библиотеками (cuFFT, cuBLAS, etc.)
  - максимально простой «фреймворк», минимум структур и объектов
  - прозрачность и отсутствие неявных операций

Пример кода

```
        // allocate source image
int sp;
Ipp8u * pSI = ippiMalloc_8u_C1(w, h, &sp);
        // fill with some image content
testPattern_8u_C1(pSI, sp, w, h);









        // allocated destination image
int dp;
Ipp8u * pDI = ippiMalloc_8u_C1(w, h, &dp);
        // Filter mask and achor
IppiSize  mask   = {5, 5};
IppiPoint anchor = {0, 0};
IppiSize  ROI    = {w - mask.width  + 1,
                    h - mask.height + 1};
        // run box filter
ippiFilterBox_8u_C1R(pSI, sp, pDI, dp,
                     ROI, mask, anchor);
```

```
        // allocate host source image
int hp;
Ipp8u * pHI = ippiMalloc_8u_C1(w, h, &hp);
        // fill with some image content
testPattern_8u_C1(pHI, hp, w, h);
        // allocated device source image
int sp;
Npp8u * pSI = nppiMalloc_8u_C1(w, h, &sp);
        // copy test image up to device
cudaMemcpy2D(pSI, sp, pHI, hp, w, h,
             cudaMemcpyHostToDevice);
        // allocate device result image
int dp;
Npp8u * pDI = nppiMalloc_8u_C1(w, h, &dp);
        // Filter mask and achor
NppiSize  mask   = {5, 5};
NppiPoint anchor = {0, 0};
NppiSize  ROI    = {w - mask.width  + 1,
                    h - mask.height + 1};
        // run box filter
nppiFilterBox_8u_C1R(pSI, sp, pDI, dp,
                     ROI, mask, anchor);
```

Что такое NPP?

- Функции обработки изображений
  - подмножество группы “IPPI”
  - ~300 функций
- Ограниченный набор поддерживаемых типов
  - 8u_C1, 8u_C4, 8u_AC4
  - 32s_C1, 32f_C1
- Полный набор функций конвертации форматов

Что такое NPP?

- Data exchange & initialization
  - Set, Convert, CopyConstBorder, Copy, Transpose, SwapChannels
- Arithmetic & Logical Ops
  - Add, Sub, Mul, Div, AbsDiff
- Threshold & Compare Ops
  - Threshold, Compare
- Color Conversion
  - RGB To YCbCr (& vice versa), ColorTwist, LUT_Linear
- JPEG
  - DCTQuantInv/Fwd, QuantizationTable
- Filter Functions
  - FilterBox, Row, Column, Max, Min, Dilate, Erode, SumWindowColumn/Row
- Geometry Transforms
  - Resize , Mirror, WarpAffine/Back/Quad, WarpPerspective/Back/Quad
- Statistics
  - Mean, StdDev, NormDiff, MinMax, Histogram, SqrIntegral, RectStdDev
- Computer Vision
  - Canny Edge Detector

Производительность

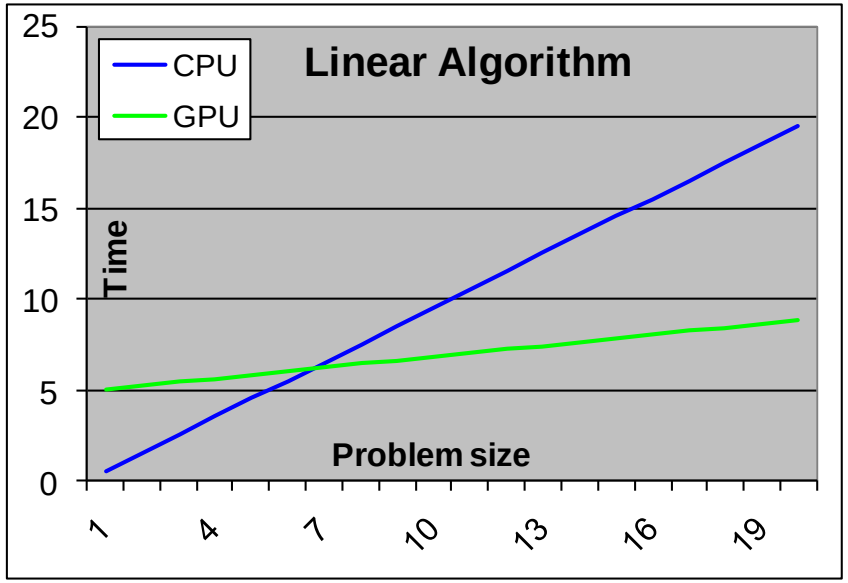
- Сравнение с IPP
  - Методология проведения измерений
- Масштабирование
  - С ростом размера задачи
  - С ростом числа процессоров
- Результаты тестирования производительности

Методология проведения измерений

- На каждый примитив приходится ряд тестов (функциональных и производительных):
  - запуск каждого теста не менее 10 раз
  - каждый запуск делается с одинаковыми данными и параметрами
  - время пересылки данных не учитывается
- Производительность замеряется единой бенчмаркой
  - ~2500 тестов производительности
  - большинство тестов производительности повторяют функциональный тест
    - комбинации невыровненных указателей и необычных ROI
    - комбинации различных входных и конфигурационных параметров
    - размер входного изображения 720p и 2000x2000

Масштабирование с ростом размера задачи (1)

- Примитив: ippi/nppiAbsDiff_32f_C1R
  - вычисляет абсолютную разность двух одноканальных float фреймов попиксельно и записывает результат в третий фрейм
  - сложность проблемы растет линейно
- Ожидаемый результат
  - линейный со сдвигами O_{CPU} & O_{GPU} и наклонными S_{CPU} & S_{GPU}
  - $O_{CPU} < O_{GPU}$, $S_{CPU} > S_{GPU}$
- Где находится точка пересечения?

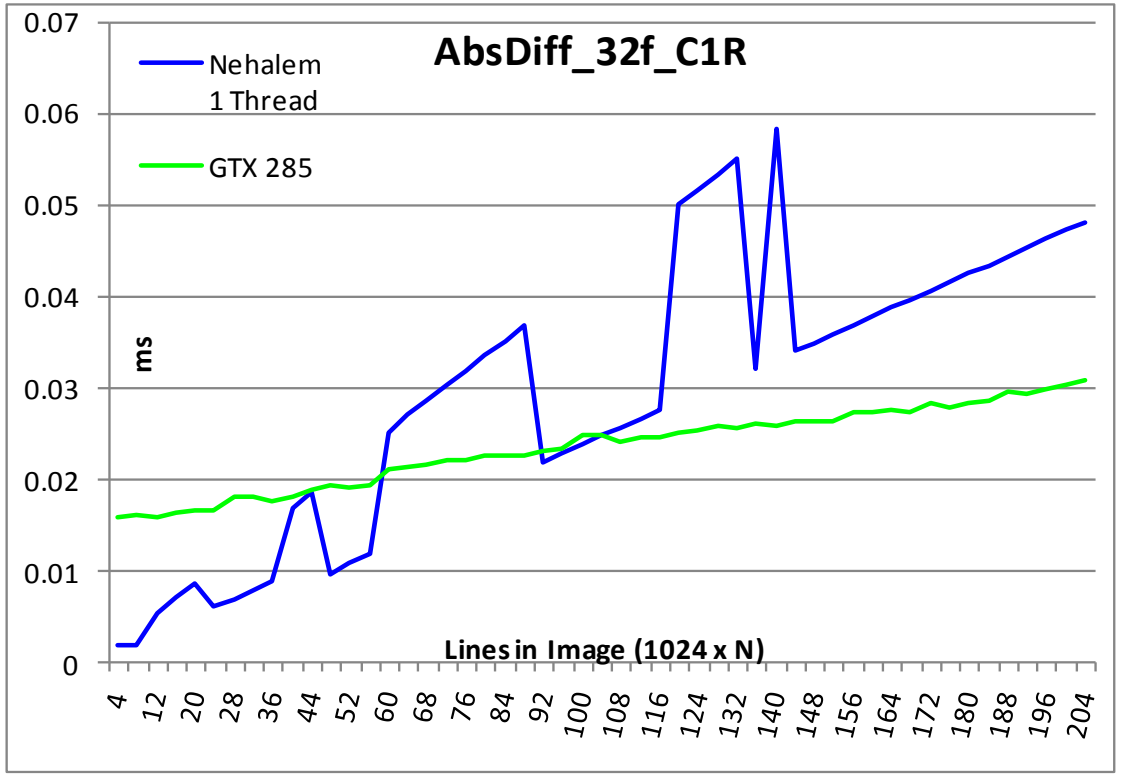


NVIDIA.

Масштабирование с ростом размера задачи (2)

- Начальный размер 1024x4 (16kB)
- Конечный размер 1024x204 (~800kB)
- Пересечение:
  - CPU slow: 48 lines = 48kPixel (4Byte) = 192kB
  - CPU fast: 108 line = 108kPixel (4Byte) = 432kB
  - 720p: 720 x 1280 = 900kPixel

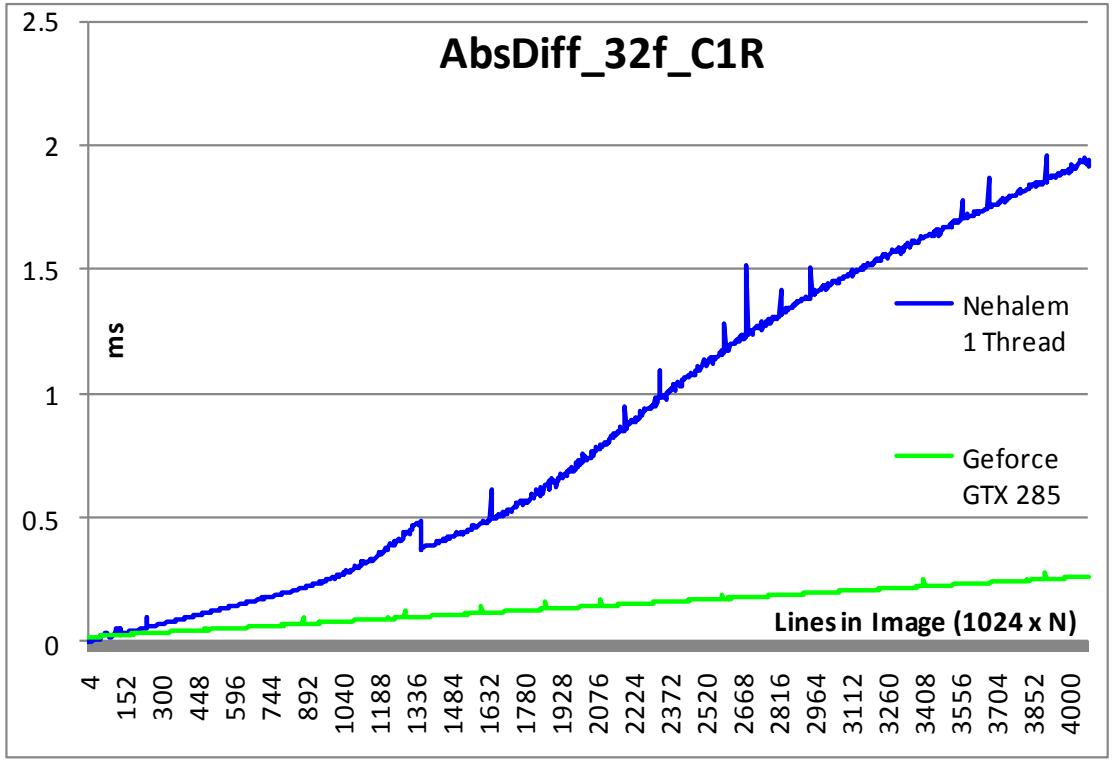
Intel Core i7 Extreme Edition i7-965
3.2 GHz, 4 (8) Core, 8MB Level 3 Cache



NVIDIA.

Масштабирование с ростом размера задачи (3)

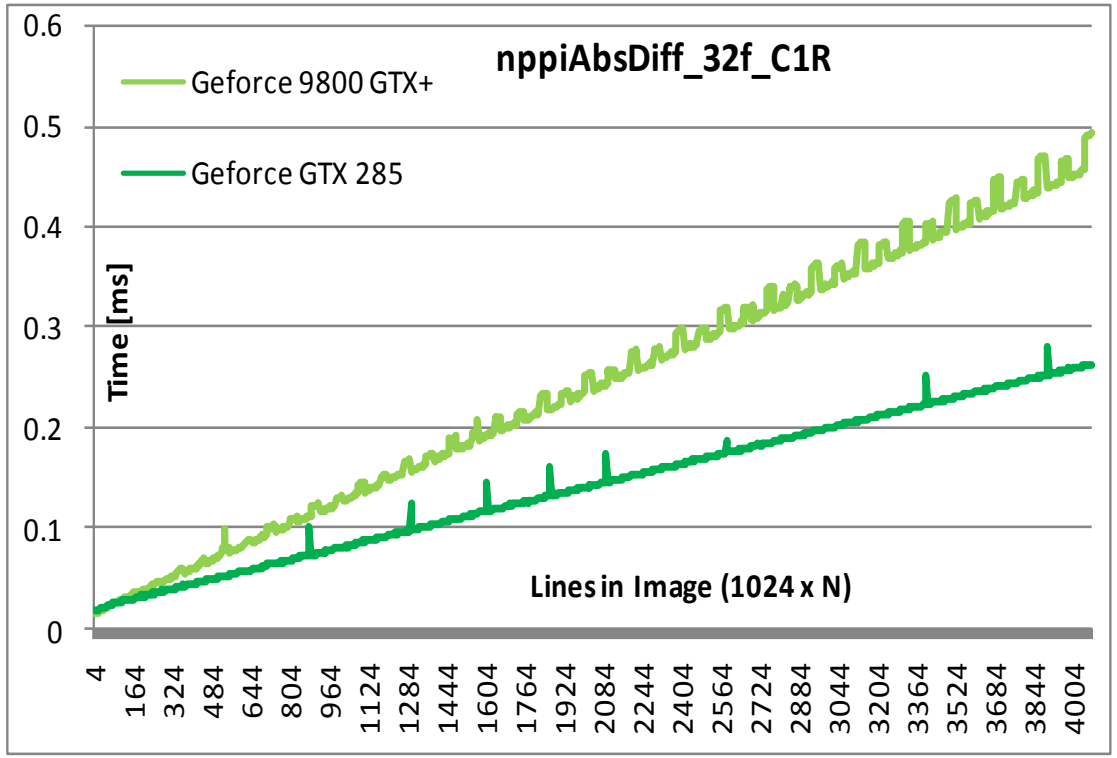
- Увеличиваем размер задачи до 1024 x 4096 пикселей
- Сегмент по оси линий X: [1000, 3000]
  - 1000 lines -> 4MByte image
  - 3 images -> 12MByte working set
  - 8MB level 3 cache size
- GPU масштабируется линейно
- Ассимптотически GPU ~7.5x быстрее



NVIDIA.

Масштабирование с ростом числа процессоров (1)

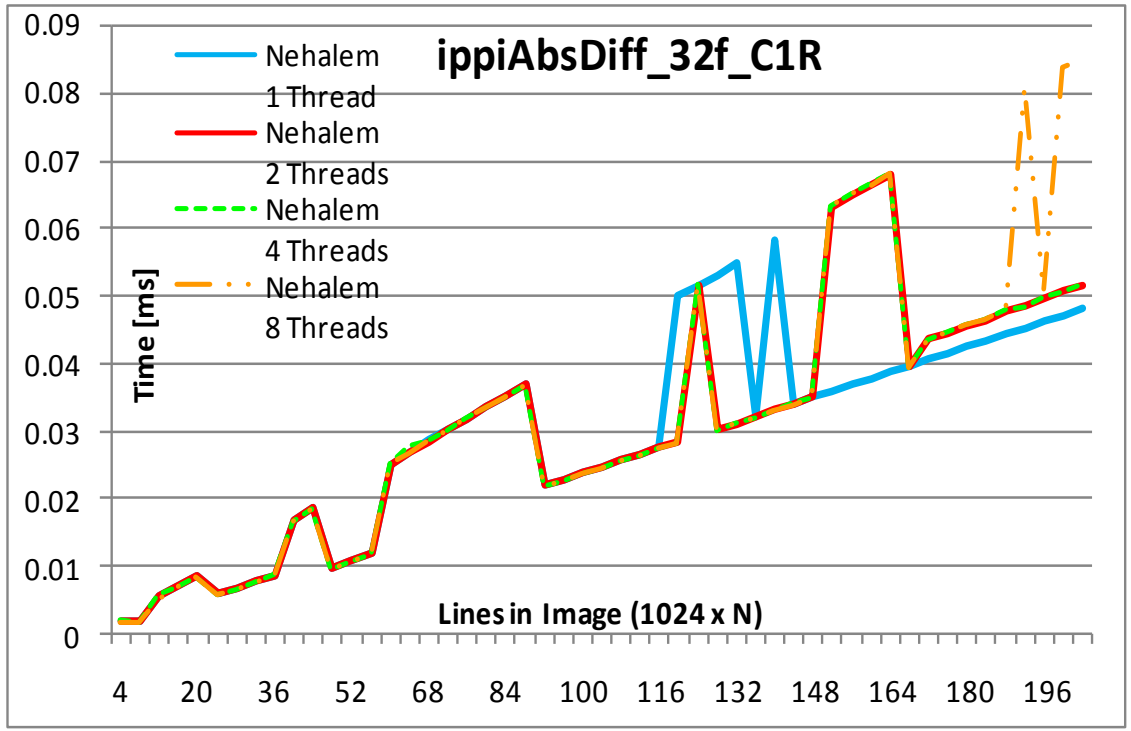
- Контролировать занятость ядер GPU напрямую проблематично
- Сравним на 2-х разных GPU:
  - Geforce 9800 GTX+: 16 SMs, 738MHz => 11808
  - Geforce GTX 285: 30 SMs, 648MHz => 19440
  - (16 * 738) / (30 * 648) = 11808/19440 = 0.6
- Соотношение на макс. размере:
  - 9800 GTX: 4.8ms
  - GTX 285: 2.6ms
  - 2.6/4.8 = 0.54
- **GPU соответствует теоретическим ожиданиям по масштабированию на всем диапазоне размеров задачи**



NVIDIA.

Масштабирование с ростом числа процессоров (2)

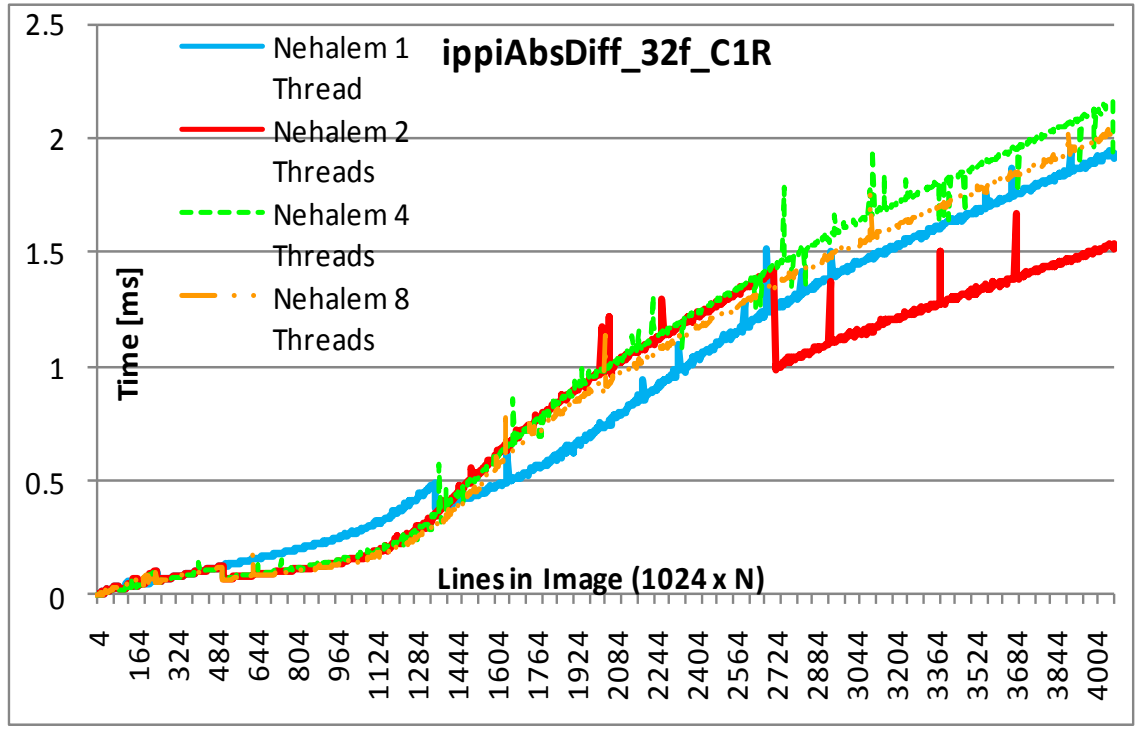
- `ippSetNumThreads(n)` для управления числом ядер
- Для больших размеров задач память становится основным узким местом на CPU
- Ожидали 4 линии на графике
- **На практике примитив AbsDiff не масштабируется с ростом числа ядер даже для маленьких размеров задачи**



NVIDIA.

Масштабирование с ростом числа процессоров (3)

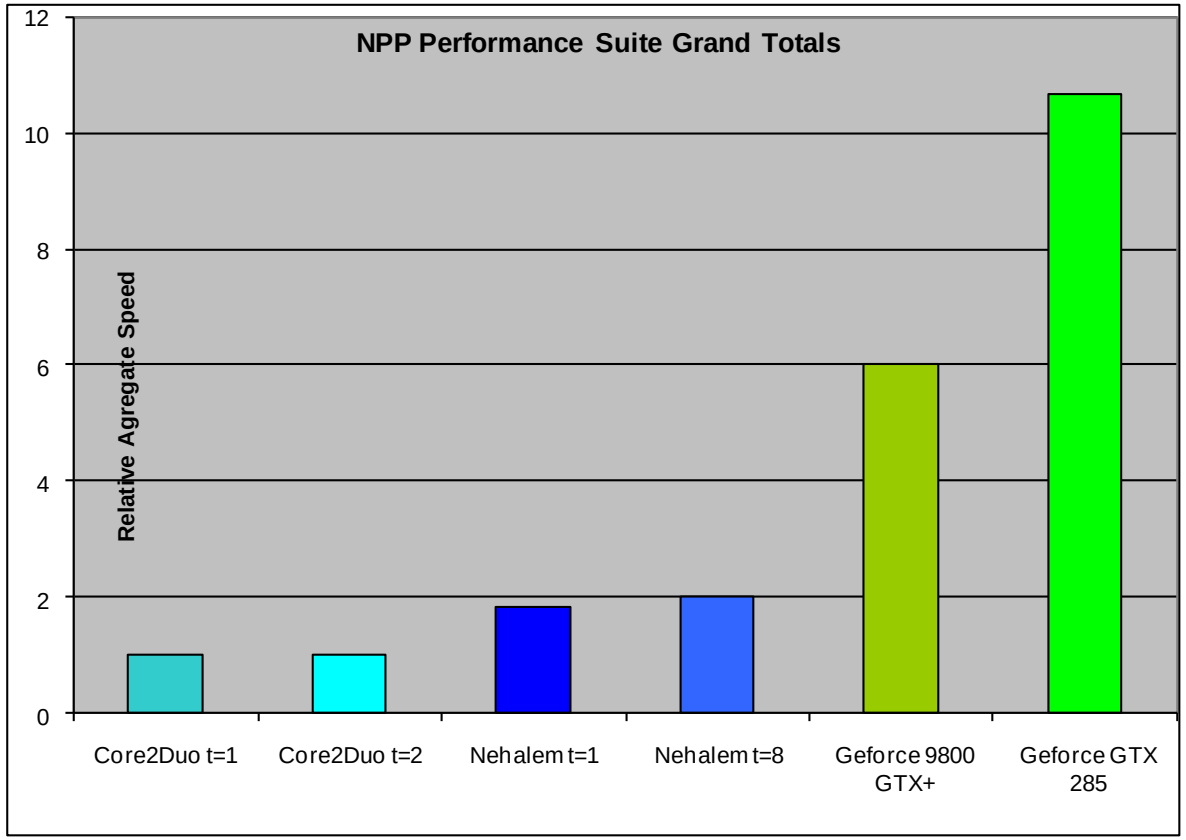
- Полный спектр размеров изображений на CPU
  - Неясно как выбирать число нитей для достижения максимальной производительности
- **Не масштабируется с ростом числа ядер**



NVIDIA.

Результаты (1)

- Результаты усреднены по 2800 NPP тестам производительности
  - каждый тест замеряет npp и ipp
  - размеры изображений 720p и 2k x 2k
- NPP оптимизации:
  - нет оптимизации под процессор
  - нет вставок на ассемблере и т.п.



NVIDIA.

Результаты (2)

Коэффициент ускорения сильно колеблется в зависимости от примитива

- После устранения 0.5% выбросов на границах интервала
- Для средних размеров входных данных:
  - Минимальное ускорение: 0.35x (т.е. CPU в 2.8x быстрее)
  - Максимальное ускорение: 27.1x
- Для крупных размеров входных данных:
  - Минимальное ускорение : 0.26x (т.е. CPU в 3.8x быстрее)
  - Максимальное ускорение : 31.9x

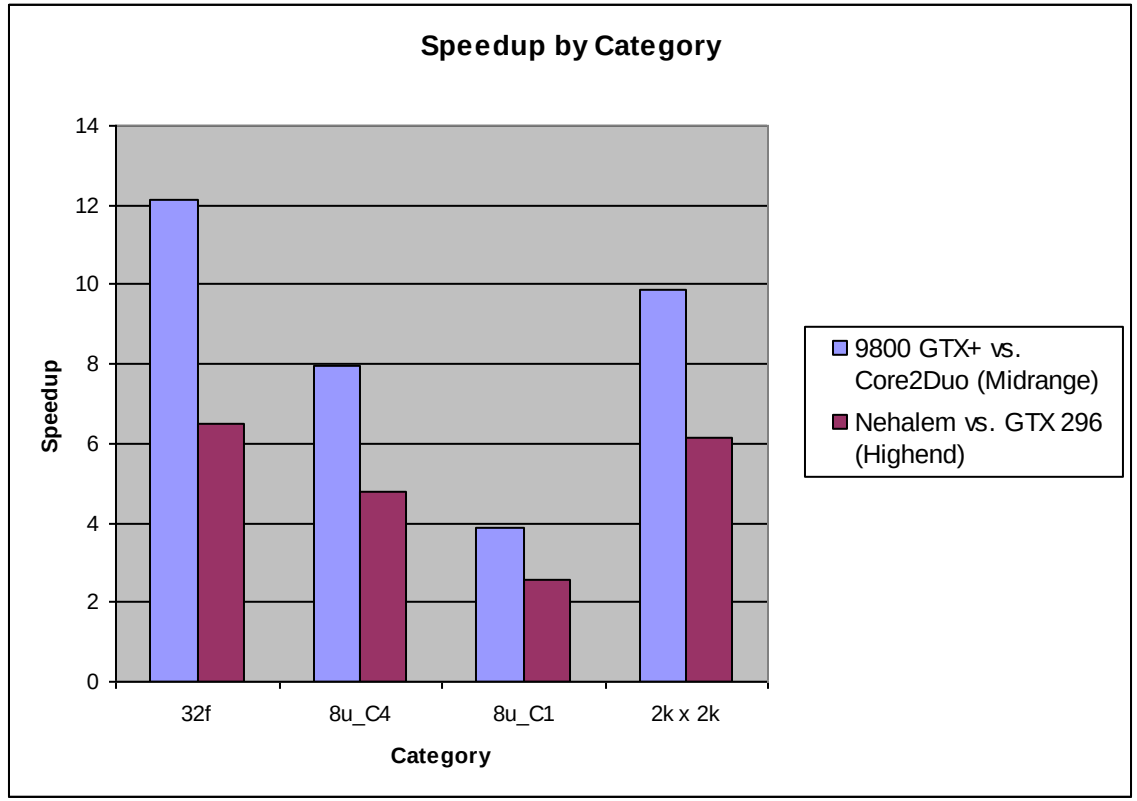
Результаты (3)

- Интересные факты:
  - NPP is 1.0 release
  - был разработан за 6 месяцев
  - без процессорно-специфической оптимизации*
    - all code compiled for compute 1.0 or 1.1
  - оптимизация большинства функций затрагивает только работу с глобальной памятью
- Intel Core i7 vs. GTX 285
  - немного разные поколения (GTX 285 – архитектура 1.5-летней выдержки)
- Значит, есть пространство для оптимизации

* Исключение: некоторые статистические функции используют атомарные операции архитектуры 1.1

Результаты по категориям

- Mid vs. High
  - в каждой категории прирост для high end карт незначителен
- 8u types
  - трудны для GPU
- 8u_C1
  - затруднены объединенные запрсы к памяти, больше вычислений в кернеле
  - малые типы удобны IPP (L2)



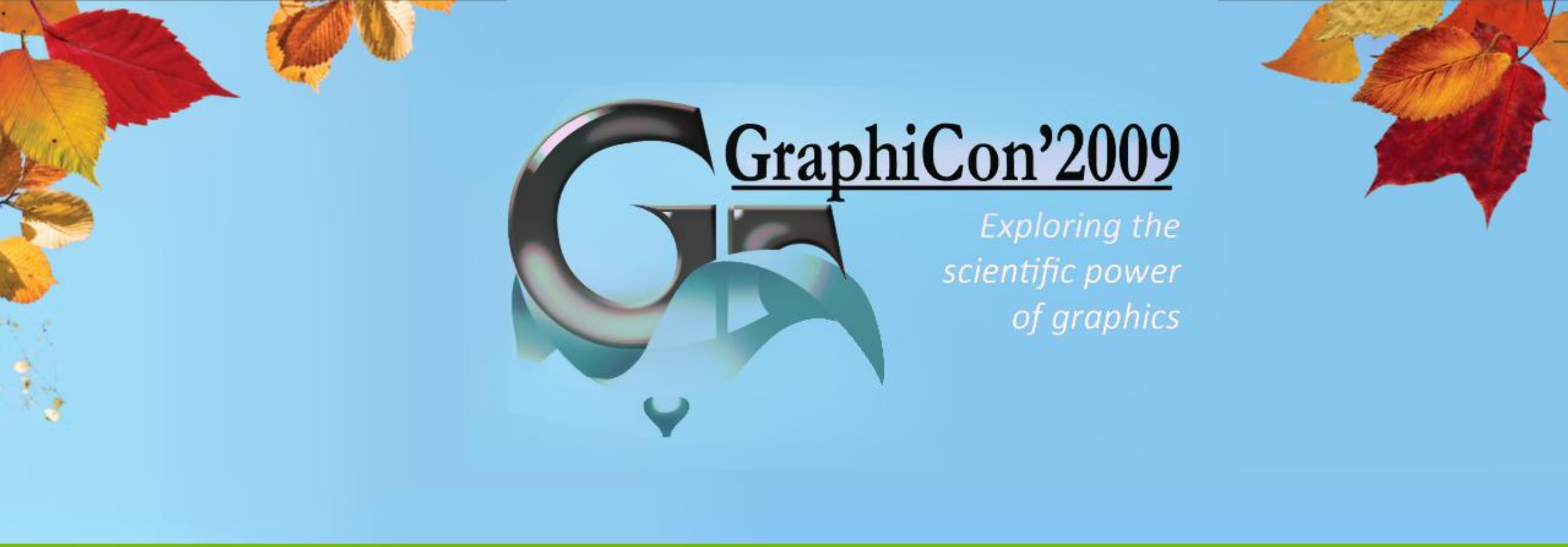
NVIDIA.

Итог

- NPP
  - легок в интеграции
  - предоставляет существенный прирост производительности по сравнению с x86
  - 300 функций
- Производительность GPU/NPP
  - легкость масштабирования по размеру задач и числу процессоров
  - неэффективен для работы с малыми изображениями
- Ограничения
  - нужны больше функций, типов данных и оптимизаций на разных уровнях

Вопросы?

- Рассмотрите использование NPP в своем приложении!
- Предоставьте отзыв:
  - чего не хватает в NPP
  - что в NPP работает недостаточно быстро
  - другие пожелания
- По любым вопросам, связанным с NPP обращайтесь:
  - Frank Jargstorff (fjargsto@nvidia.com)
  - Anton Obukhov (aobukhov@nvidia.com)
  - Ryan Prescott (rprescott@nvidia.com)



Video Codecs on GPU

Fairmont Hotel, San Jose | 10.01.2009 | Anton Obukhov

NVIDIA

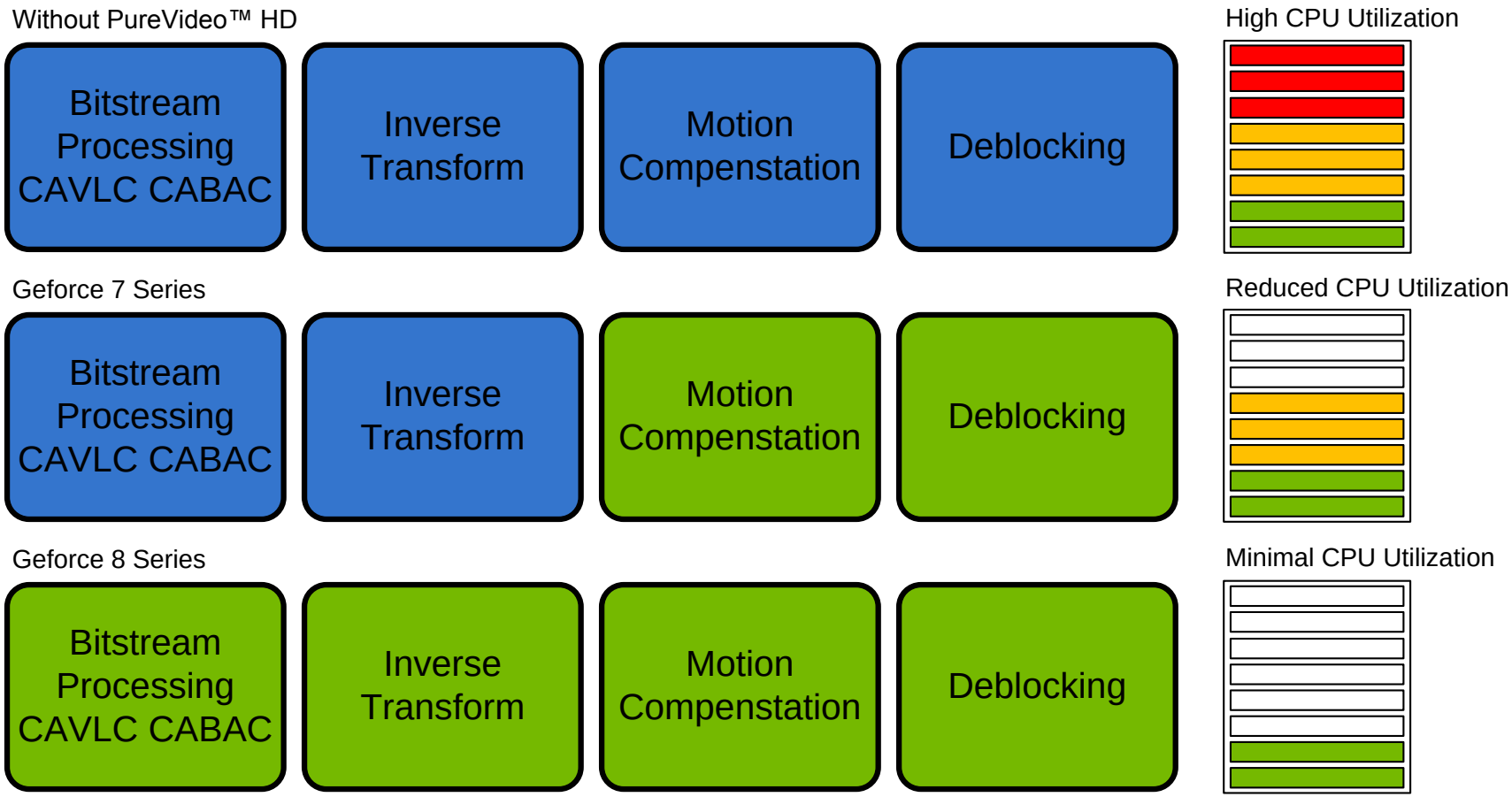
Предпосылки

Задача транскодирования видео нуждается в ускорении как никогда:

Кодирование hi-res видео занимает десятки часов на современных десктопах

Мобильные и портативные устройства обладают нераскрытой вычислительной мощью

Эволюция аппаратного ускорения



NVIDIA.

Кодирование видео на NVIDIA GPU

Средства:

- SW H.264 кодек спроектированный для платформы CUDA
  - Baseline profile
  - Main profile
  - High profile

Интерфейсы:

- C library (NVCUVENC)
- Direct Show API
- Win7 MFT



Декодирование видео на NVIDIA GPU

Средства:

- HW аппаратное ускорение
  - H.264
  - VC1
  - MPEG2
- SW MPEG2 декодер спроектированный для платформы CUDA

Интерфейсы:

- C library (NVCUVID), HW & SW 
- DXVA and Win7 MFT, HW only 
- VDPAU library, HW only 

NVIDIA.

Обработка видео на NVIDIA GPU

Средства:

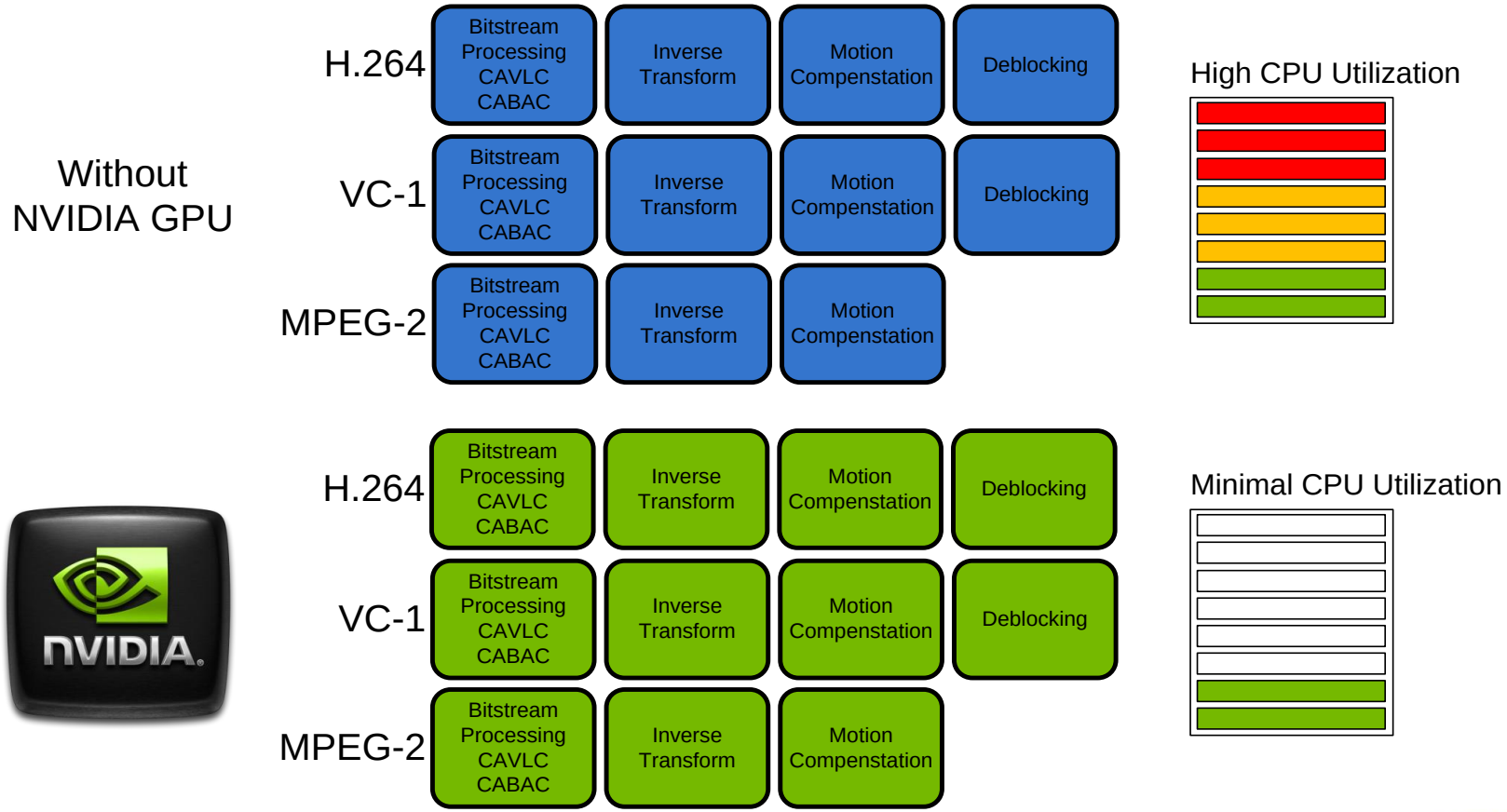
- SW библиотека пре- и пост-обработки (designed for CUDA)
  - Noise Reduction
  - Deinterlacing 
  - Polyphase Scaling
  - Color Processing
  - Deblocking
  - Detail enhance

Интерфейсы:

- VMR/EVR API 

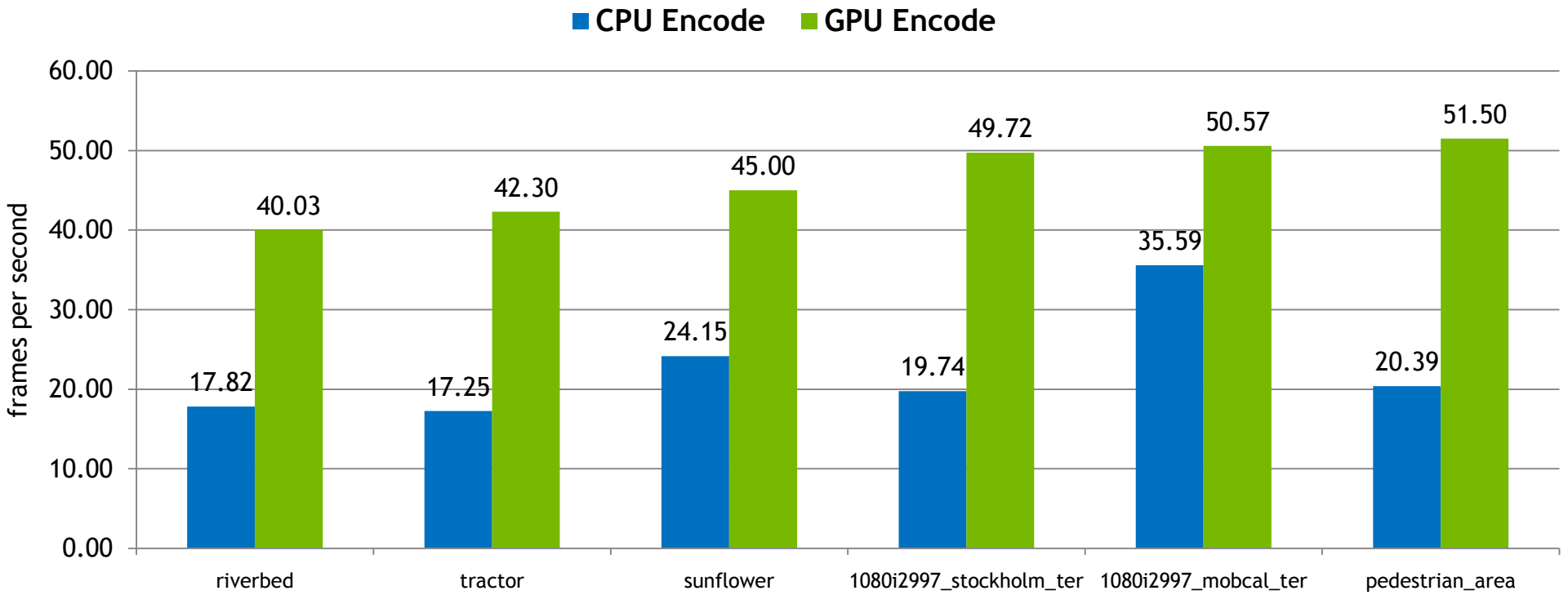
Польза от декодирования на GPU

~100% разгрузка при декодировании 3 основных стандартов



NVIDIA.

Производительность кодирования



Frame size: 1080p
Platform: 3.2 GHz quad core Nehalem, GeForce GTX 280 (128 core) GPU
CPU encoder is MainConcept
GPU encoder is NVIDIA H.264 CUDA encoder.

NVIDIA.

Использование в жизни

Некоторые коммерческие приложения для транскодирования видео при помощи CUDA

- Badaboom
- Nero Move it
- CyberLink PowerDirector
- Loilo SuperLoiloscope
- *Тысячи их!*

Мысли вслух

- Как насчет  и  ?
- Что происходит на системах multi-GPU?

Мысли вслух

- Как насчет  и  ?
  - Linux: только декодирование с VDPAU
  - Mac OSX: QuickTime API

Мысли вслух

- Что происходит на системах multi-GPU?
  - NVIDIA H.264 encoder поддерживает системы dual-GPU

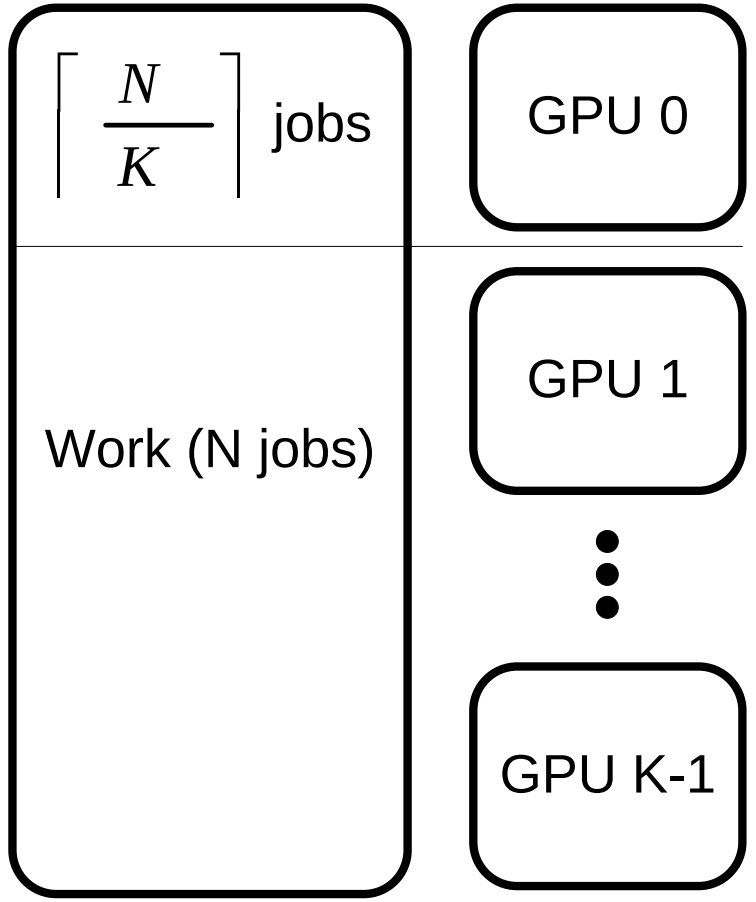
Мысли вслух

- Multi-GPU – обыденность
- Программирование Multi-GPU систем само по себе нетривиально

Мысли вслух

CUDA предоставляет доступ к каждому из установленных GPU

Как заставить их работать над одной задачей одновременно?



Мысли вслух

Создание программных средств для аппаратного ускорения транскодирования видео является приоритетной задачей

Webinar 10/28/2009 9:00 AM - 11:00 AM PDT

- Программирование Multi-GPU
- Приложения к кодированию видео

https://www2.gotomeeting.com/register/628549827

Вопросы?

?

E-mail: aobukhov@nvidia.com

“Introducing a new Multi-GPU framework” webinar, 10/28/2009 9:00 AM - 11:00 AM PDT

https://www2.gotomeeting.com/register/628549827