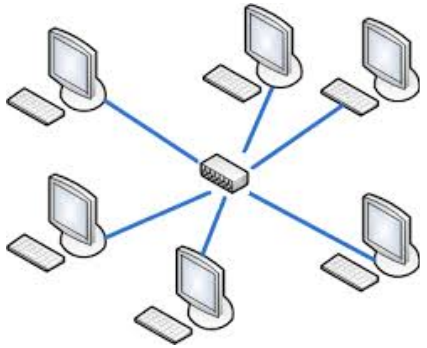
Пути увеличения эффективности реализации алгоритмов машинного обучения

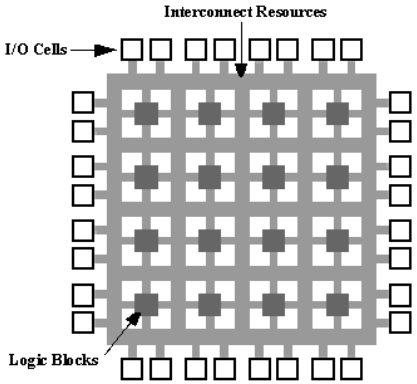
Бабий А.С. - apratster@gmail.com

Для чего необходимо задумываться о скорости вычислений для ML?

1. Сокращаем время обучения алгоритмов.
2. Достигаем приемлемого времени перенастройки «на лету».
3. Больше факторов.
4. Большие объемы данных для обучения.
5. Возможность масштабирования.
6. Экономия средств.

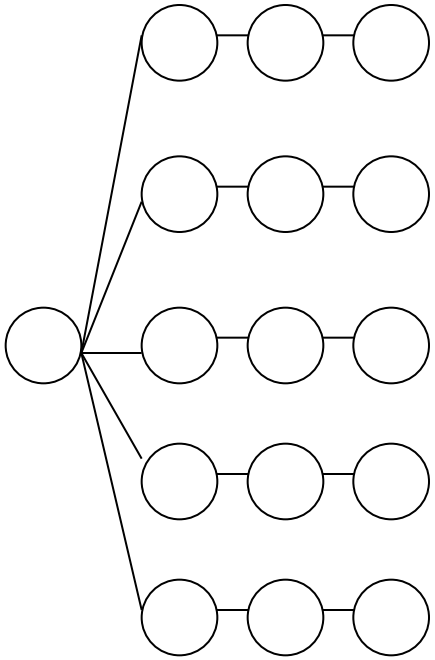
Пути увеличения производительности



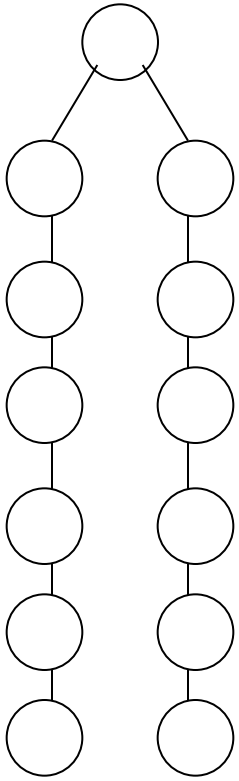


Подходы к увеличению производительности ML распараллеливанием

Широкая - специфичная для GPU



Узкая - специфична для CPU



SIMD vs SIMT vs SMT

Оптимизация одного потока вычисления

Когда используется:

- Возможности распараллеливания ограничены аппаратно либо алгоритмом
- Код исполняется параллельно, каждый из потоков необходимо оптимизировать

Возможности:

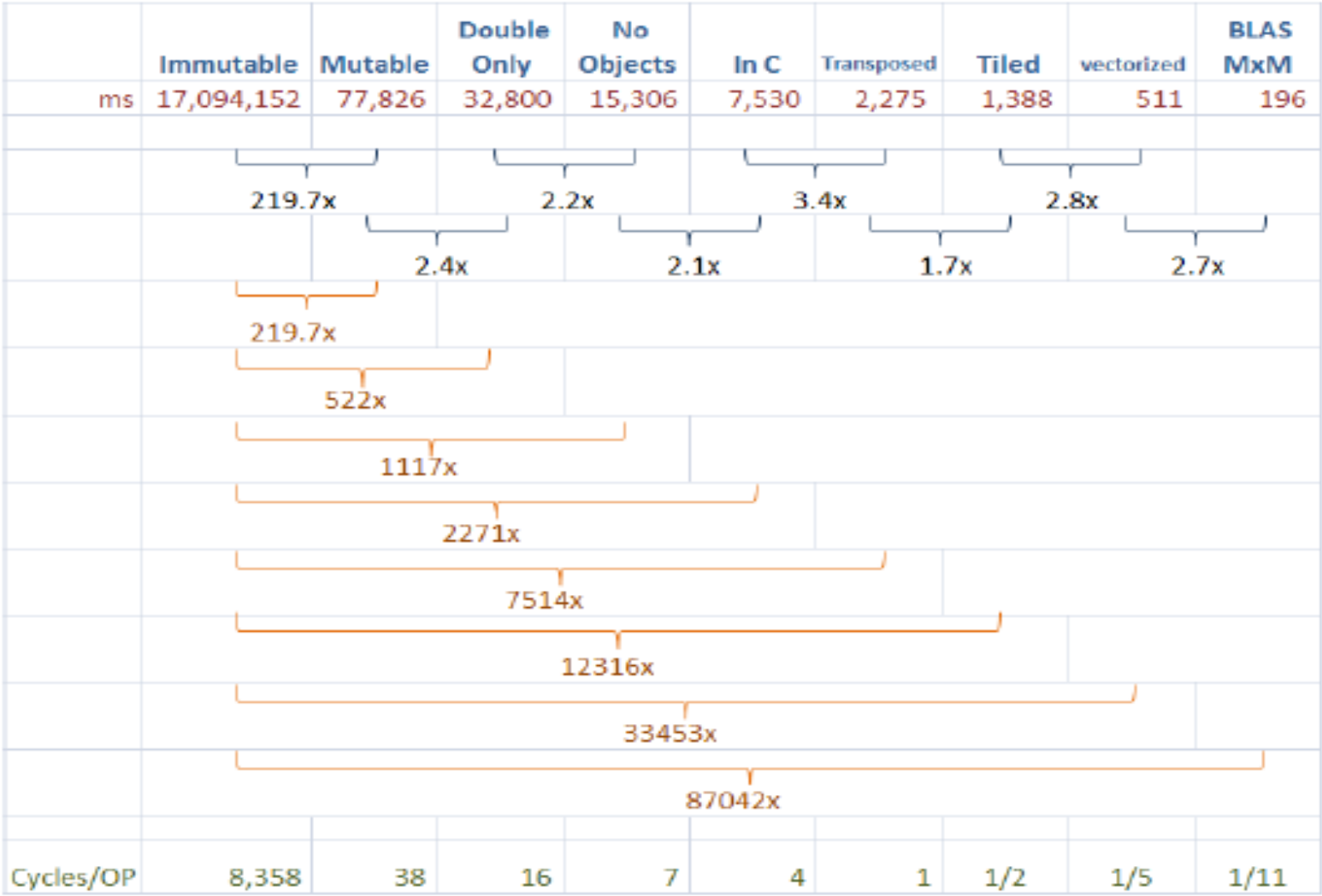
В удачных случаях (например умножение матриц) – до 16 раз быстрее.
Исходная и оптимизированная программа на C

Например переход с оптимизированной Java версии на C дает
2x кратное ускорение работы программы.

Ограничения:

- Необходимо понимать архитектуру CPU который используется
- Скорость разработки замедляется и усложняется

Сравнительная таблица оптимизаций для умножения матриц[1]



1. Saman Amarasinghe, Matrix Multiply, a case study – 2008.

Прежде чем начать оптимизировать….

1. Используйте профилировщик (gprof, valgrind, …)

2. Не использует ли приложение уже оптимизированный код BLAS

3. Если не использует, то перейти к матричной и векторной форме работы с данными, и использовать BLAS

4. SIMD – иногда дает максимальный прирост

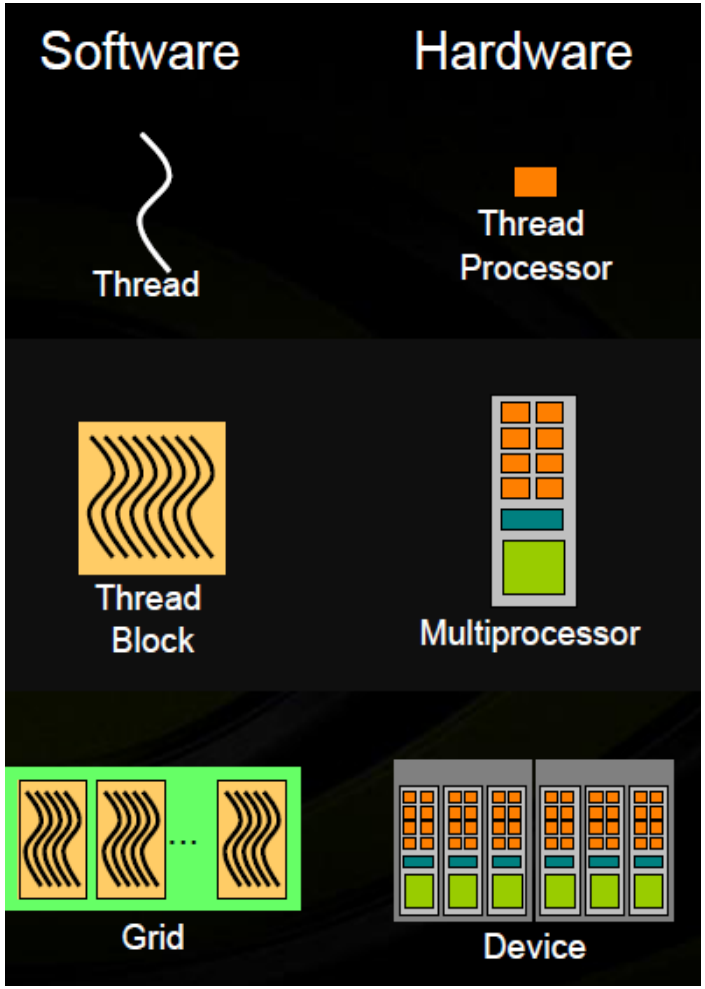
Распараллеливание.

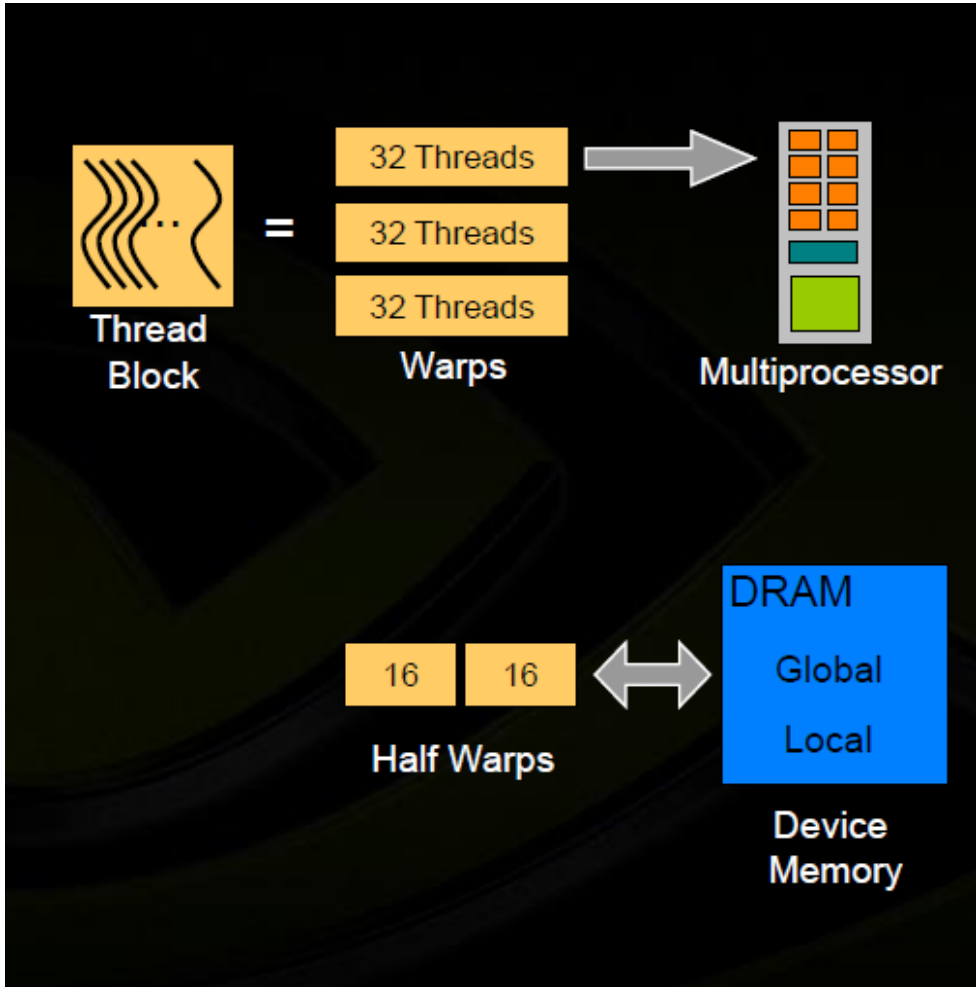
1. Часто ограничены аппаратным обеспечением и архитектурой.
2. KML, PBLAS, ATLAS
3. Когда CPU Multicore эффективнее GPU ?
4. NVIDIA CUDA.
5. OpenCL – переносимость и ее цена.

Основные идеи:

1. Функция которая распараллеливается называется «kernel-function».
2. Потоки исполняющие ядерные функции группируются в блоки.
3. Потоки и блоки нумеруются.
4. Только одна ядерная функция может исполняться на устройстве в один момент времени.
5. Блоки потоков исполняются на мультипроцессорах, потоки на потоковых процессорах.
6. Блоки состоят из WARP, которые исполняются в SIMD режиме

Архитектура CUDA



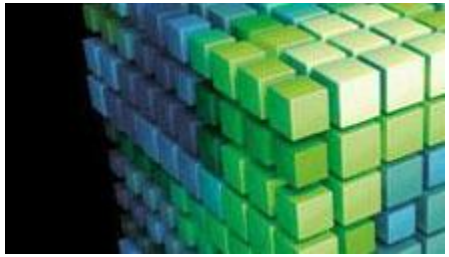


Виды памяти

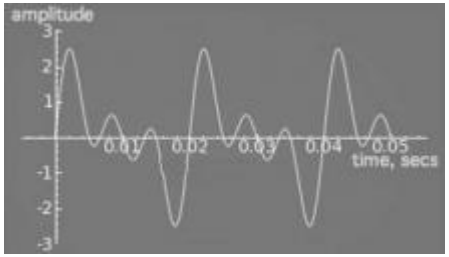
Memory	Location	Cached	Access	Scope	Lifetime
Register	On-chip	N/A	R/W	One thread	Thread
Local	Off-chip	No	R/W	One thread	Thread
Shared	On-chip	N/A	R/W	All threads in a block	Block
Global	Off-chip	No	R/W	All threads + host	Application
Constant	Off-chip	Yes	R	All threads + host	Application
Texture	Off-chip	Yes	R	All threads + host	Application

Доступные средства для быстрой разработки с GPU

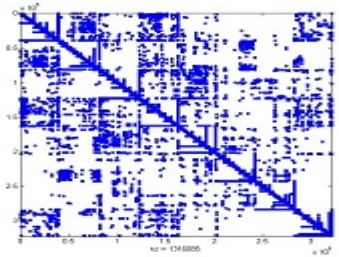
Область параллельных вычислений развивается очень динамично, достаточно большое количество библиотек для вычислений с GPU уже существует.



cuBLAS

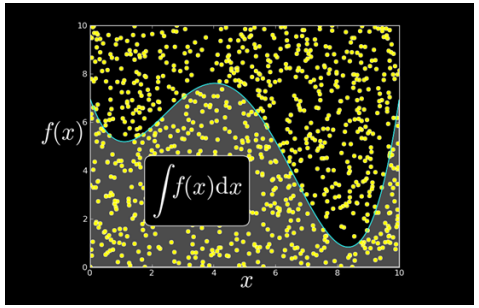


cuFFT



cuSparse

Thrust



cuRand

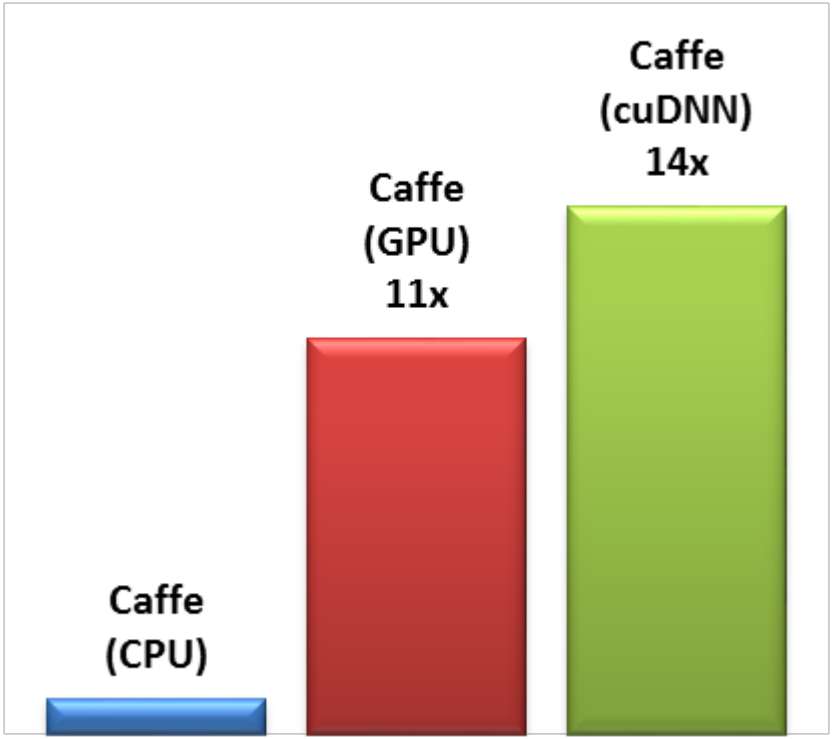
[HiPLAR]

HiPLAR

Набор примитивов для сетей Deep Learning

cuDNN

1. Сверточный слой
2. Слой фильтрации
3. Обобщающий слой



Интеграция с Caffe

24-core Intel E5-2679v2 CPU @ 2.4GHz vs K40, NVIDIA

Применяем cuDNN в своих проектах!

Требует:
cuDNN supports NVIDIA GPUs of compute capability 3.0 and higher
NVIDIA Driver compatible with CUDA Toolkit 6.5.

Предоставляет:
- Прямой и обратный слой конволюции, включая кросс-корелляцию
- Слой обобщения прямое и обратное
- Слой Софтмакс прямой и обратный

Функции активации
- Rectified linear (ReLU)
- Sigmoid
- Hyperbolic tangent (TANH)
- Tensor transformation functions

Библиотека Thrust

Доступные параллельные алгоритмы:

Трансформация (1\2 операнда, +, -,abs, * …)
Редукция (sum, max, min,)
Префикс-сумма (inclusive scan, exclusive scan – для сортировки напр.)
Сортировка
Перестановка (copy_if, remove_if, unique …)

Основная идея – сделать параллельные вычисления доступными для понимания, как и шаблоны C++

За счет абстракции может несколько теряться производительность, но повышается скорость разработки

Пример (Код J.Hintona для RBM)

```
    data = batchdata(:,:,batch);
    poshidprobs = 1./(1 + exp(-data*vishid - repmat(hidbiases,numcases,
  1)));
    batchposhidprobs(:,:,batch)=poshidprobs;
    posprods    = data' * poshidprobs;
    poshidact   = sum(poshidprobs);
    posvisact = sum(data);
……
 vishidinc = momentum*vishidinc + ...
                epsilonw*( (posprods-negprods)/numcases - weightcost*vishid);
    visbiasinc = momentum*visbiasinc + (epsilonvb/numcases)*(posvisact-
negvisact);
    hidbiasinc = momentum*hidbiasinc + (epsilonhb/numcases)*(poshidact-
neghidact);

    vishid = vishid + vishidinc;
    visbiases = visbiases + visbiasinc;
    hidbiases = hidbiases + hidbiasinc;
```

Библиотеки и ссылки

Python - Theano, Pylearn2, cudamat, mPoT

C++ - Cuda-Convnet, Eblearn, CXXNET

Matlab – DeepLearnToolbox, Deep Belief Networks, deepmat

http://deeplearning.net/software_links/

Выводы

1. Для редких, экспериментальных, не реализованных в библиотеках Алгоритмов иногда достаточно оптимизировать код C для последовательной версии.

2. Параллелизация для CPU, как правило диктуется сложностью в «широком» распараллеливании алгоритма, либо аппаратными ограничениями

3. Использовать готовые оптимизированные пакеты функций BLAS, cuBlas, Thrust

4. cuDNN + Caffe дает возможности быстрого прототипирования DNN

Спасибо за внимание !