PHILIPS

sense and simplicity

Applying GPUs for massive calculations

Determination of suitable applications

September 2010
by Dr. A.P. Kostelijk

Personal introduction

Experience within Philips (focused on technical part)
- Professional systems (mostly PC based)
- Real-time embedded consumer systems
- IC-design SW, NP-complete algorithms

Teaching:
- Execution Architecture course
- SW reliability
- 3TU / OOTI – embedded computing platform

Leading the Apptech Performance team

Contents

Introduction

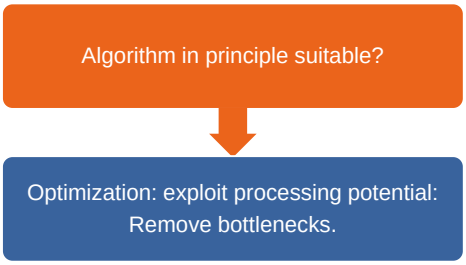
- Main GPU experience: nVidia, G80, Tesla, CUDA.

  → Advanced GPU optimization course for performance team

- This talk is 25 minutes: select and focus
  – GPU, GPGPU: General Purpose is an overstatement, so
  – Determination of suitable software → suitable algorithm
  – A suitable algorithm is fast and energy efficient.

- Skip a lot of GPU introduction, given previous speakers presentations.

Introduction - overview



Suitable software

'kernel'

- A GPU is a parallel co-processor for a CPU, doing one task at a time.
- A GPU consists of ~32 multi-processors (MP)
  →Single Tasking - Multiprocessing
- A multi-processor is a SIMD processor
  – 8x in HW, 32x fault in SW (4-stage pipeline).
- 1 algorithm applied in parallel to multi-data sets
  – Multi-threading: all threads execute the same code.
  – Vector processor is therefore a better description.
- Data-dependent if-then-else : MP serializes then + else (unless ….)
- Sorting / filtering based on ITE less suitable (alike control SW)
  →Number crunching

PHILIPS

Suitable parallelism

Estimation of degree of suitable number of data sets
- HW-Theoretical: 32 x 32 ≈ 1000
- Overhead sources are relatively large (but undocumented), e.g.,
  - GPU data loading
  - GPU code loading (kernel) and kernel invocations
  - PC driver cost (~ 10 - 30 us = 30 MFLOP on Tesla)

- Practical required degree of parallelism: >> 1.000, e.g. 30.000+
  - Depends on typical gpu-job time

→Massive parallel greedy number crunching jobs

PHILIPS

Global Modeling of Algorithms: intro and example

Convention: device = graphics card, host = PC cpu.

GPU (Tesla C1060) hardware characteristics
- 1 TFLOPS max (~87 GFLOPS for double precision)
- Data rate ~100 GB/s max (device transfers)
- Data rate ~3 GB/s max (device-host transfers)

20 DVDs per second

Simple example: C = A + B (large n x n matrices, massive parallel)
3 device-host word transfers, 3 word device transfers, 1 FLOP per word.
Device – host transfer utilization: 1, i.e., 0.75 G C_{ij} per second.
Device transfer utilization: 0.033
FLOPS utilization: 0.75 $x10^{-6}$

PHILIPS

Global Modeling of Algorithms – simple example

- Is a GPU suitable to calculate matrix C = A+B ?

Yes No

- Exploiting 10^{-6} of the FLOP potential only.

PHILIPS

Global Modeling – single precision

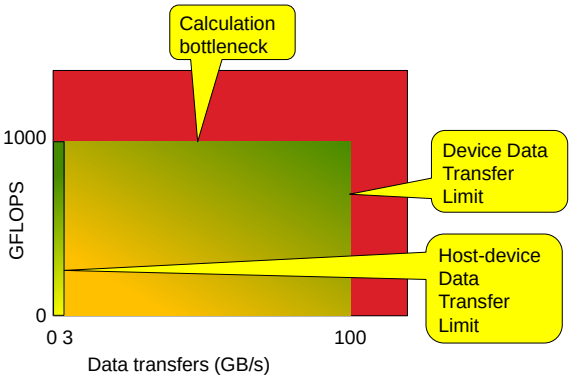
GPU characteristic for single precision:
- ~1 TFLOPS max
- Data rate ~25 G words/s max (device transfers)
- Data rate ~0.75 G words/s max (device-host transfers)

Key is the number of times a single data element is used in calculations
- When on device: ~40 calculations per data element required,
- When on host: ~1300 calculations per data element required ..

before the calculation capacity becomes the bottleneck.

PHILIPS

Global Modeling: boundaries



PHILIPS

Global modeling – steps and target

Analyze essence of the algorithm	Target
• Derive essential parallelism	>> 1000
• Calculations	~TFLOPS
• Data reference patterns:	
– data element refs	~40 dev, ~1300 host

PHILIPS

Global modeling – steps – how to

- Derive essential parallelism
  - Outer loops may be parallelized
- Computations (limited focus in this phase)
- Data reference patterns
  - optimize dataref reuse

PHILIPS

Global modeling – data reference optimization

Data reference pattern optimization:

- Reuse coefficients
- Reuse power calculations
- Reuse cross-terms

..

This is the reason why per processor thousands registers are available, next to the 'shared memory'.

PHILIPS

Data referencing - examples

Algorithm:	host-dev	device
• Matrix addition, subtraction, transposition:	no advantage	
• Matrix multiplication $O(n^3) \rightarrow n^2$ per element	n > ~ 20	n > ~ 5
• Solve linear equations $O(n^3) \rightarrow n^2$ per element	n > ~ 20	n > ~ 5
• 2D FFT $O(n^2 \log n) \rightarrow 5\ n \log n$ per element	n > ~ 32	n > ~ 4

…

The overhead per invocation is so large, that the first part of the range only applies when multiple matrices are handled per invocation.

Complex number calculations are more suitable: mult/div uses data twice.

PHILIPS

Practice: Sequence of algorithms

- Ineffective GPU alg's can be useful to prevent dev-host communication.
- Despite multiple kernel invocations
- Real iterating algorithms enhance greediness and limit dev-host communication.

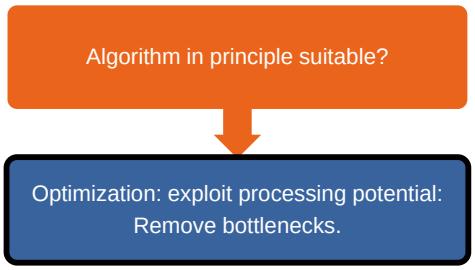
PHILIPS

Summary of candidate SW

- Massive Parallel (>> 1000, e.g. 30.000)
- Computations (~ Teraflops)
- Data transfer is typically the major bottleneck, suitable algorithms have
- many calculations per data element (~1000 per i/o, ~30 per device elt)
- GPU Jobsize should be substantial w.r.t. invocation overhead.
- Set of algorithms of which data resides on the device increases suitability

PHILIPS

Next phase: optimization



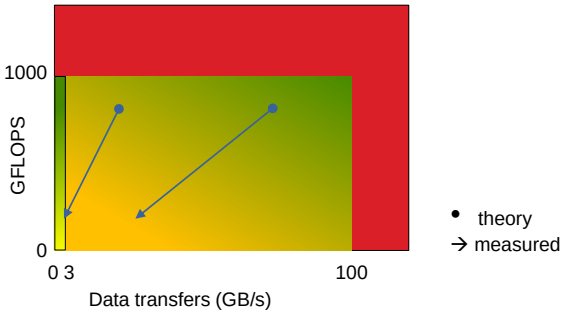
PHILIPS

Optimization

- This part requires explaining the 10 bottlenecks of a GPU.
- How to bypass them.
- A bit too much for now. It is in the optimization course.
- How to recognize a bottleneck is relatively simple, after the suitability of the algorithm has been investigated.

PHILIPS

Optimizations: how to recognize a bottleneck



PHILIPS

Optimization – some remarks

- Direct C to Cuda translation is easy, but is non-optimized.
  - Marketing of how easy and generic CUDA is, shoots in the foot.
  - Available numerical libraries quality is work-in-progress.
  - Positive impression of the support by nVidia.
- Potentially suitable algorithm typically required a complete reordering / rewriting, resulting in a speedup of factors ~100 w.r.t. first version, both for single- and double precision.

PHILIPS

Conclusions

- GPU is suitable for running extremely greedy calculations that run massively parallel, with limited cpu to gpu memory transfers, and lots of calculations per data element.
- Global modeling of an algorithm gives potential suitability, without knowledge of GPU internals.
- Though CUDA hardly contains hardware specifics, real performance requires in-depth understanding how CUDA runs in hardware, what are the bottlenecks.
- It's easy to notice if one suffers from a bottleneck.