Radio astronomy beam forming on GPUs

Alessio Sclocco

Vrije Universiteit Amsterdam

December 13, 2010

ToC

Introduction

- Two recent changes in radio astronomy
  - Radio telescopes are mostly implemented in software
  - Arrays of small non-directional antennas replaced "*big dishes*"
- An important algorithm is the **beam forming**
  - Received signals are combined to give the telescope directionality
- Modern developments in radio astronomy will require
  - To run computations: exa-scale computers
  - To power the supercomputers: huge amount of electricity

General-Purpose computation on Graphics Processing Units

- GPUs can be used as accelerators
  - More computational power than CPUs
  - Better power efficiency

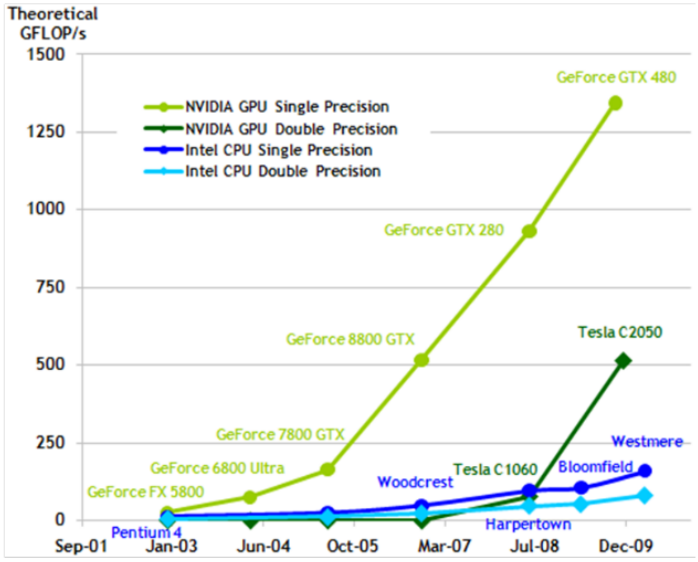

Figure: Comparison between Intel CPUs and NVIDIA GPUs in term of GFLOP/s, courtesy of NVIDIA.

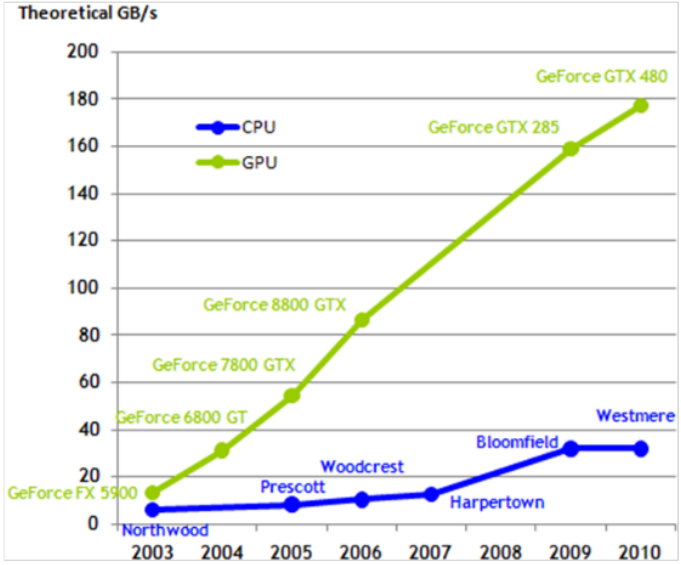

Figure: Comparison between Intel CPUs and NVIDIA GPUs in term of GB/s, courtesy of NVIDIA.

Research question

Is it possible to parallelize the beam forming algorithm efficiently on GPUs ?

ToC

CUDA

- CUDA is a general purpose parallel computing architecture developed by NVIDIA
- Provides programmers with a GPGPU framework for NVIDIA GPUs
- Defines language extensions to C, C++ and Fortran
- Computational hierarchy: thread, block, grid
- Memory hierarchy: local, shared, global, constant[1], texture[1]

[1] Read only

Parallel algorithms (1/2)

- Six developed versions
  - Data parallelism
  - Versions differ in how the computation is structured
- Some versions have different implementations
  - Implementations differ in minor aspects
  - Examples are memory allocation or compile time parameters

Parallel algorithms (2/2)

Kernel	**Operational intensity**	**Registers**
1.0.2	0,65	29
1.0.2	[0,31, 1,18]	25
1.1 / 1.1.1	0,3	17
1.1_2x2 / 1.1.1_2x2	0,41	23
1.2	[0,51, 0,66]	[25, 36]
1.3	[0,56, 0,99]	26
1.4	[0,87, 2,98]	18
1.5	[0,64, 1,22]	[29, 63]

Table: Operational intensity and registers used by each kernel

Experimental setup

- Experiment:
  - Single execution
  - Two varying parameters: receivers $[2^1, 2^8]$ and beams $[2^1, 2^9]$
  - Three constant parameters: 256 channels, 768 time intervals, and 2 polarizations

Experimental setup

- Experiment:
  - Single execution
  - Two varying parameters: receivers $[2^1, 2^8]$ and beams $[2^1, 2^9]$
  - Three constant parameters: 256 channels, 768 time intervals, and 2 polarizations
- Measurements:
  - Total execution time, in seconds
  - Single precision FLOPs, in GFLOP/s
  - Memory bandwidth, in GB/s

Experimental setup

- Experiment:
  - Single execution
  - Two varying parameters: receivers $[2^1, 2^8]$ and beams $[2^1, 2^9]$
  - Three constant parameters: 256 channels, 768 time intervals, and 2 polarizations
- Measurements:
  - Total execution time, in seconds
  - Single precision FLOPs, in GFLOP/s
  - Memory bandwidth, in GB/s
- Questions:
  - How the different algorithms scale ?
  - Which performance is possible to achieve ?
  - Which are the best parallelization strategies ?

Results (1/2)

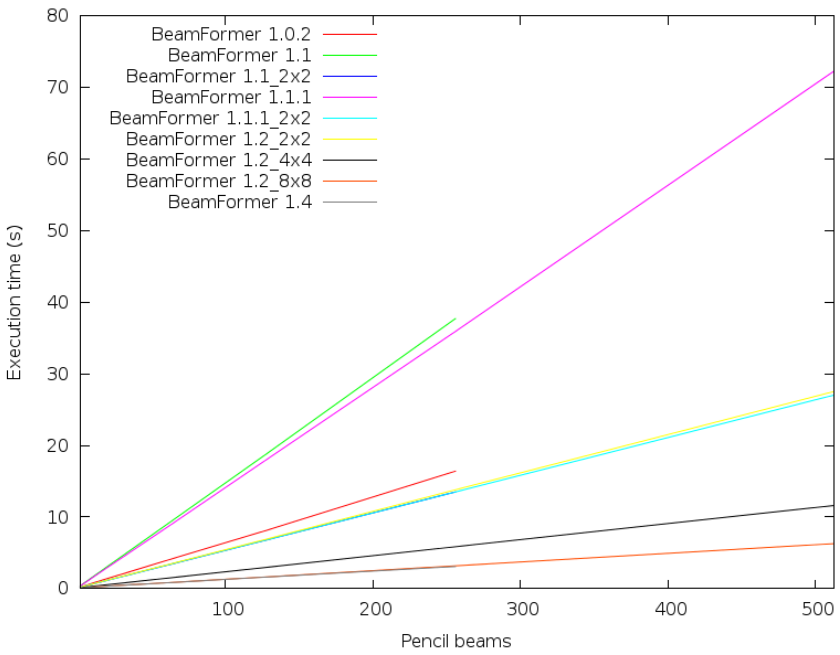

Figure: Execution time in seconds of various BeamFormer versions merging 64 receivers.

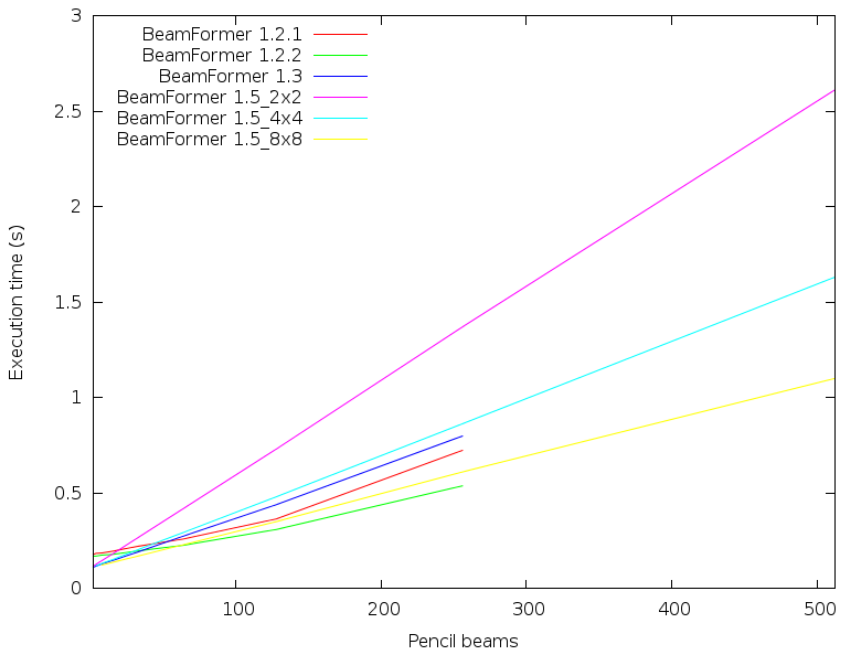

Figure: Execution time in seconds of various BeamFormer versions merging 64 receivers.

Results (2/2)

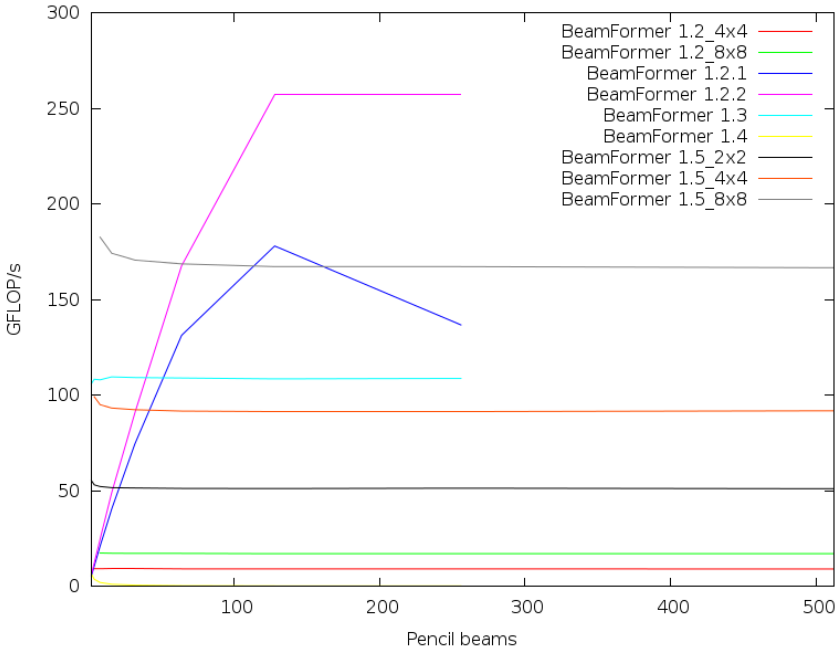

Figure: GFLOP/s of various BeamFormer versions merging 64 receivers.

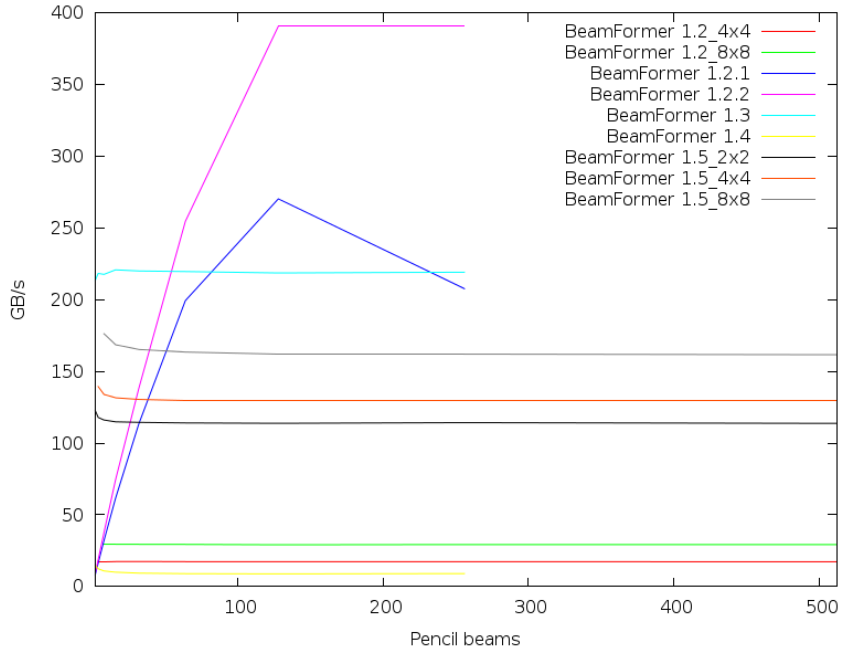

Figure: GB/s of various BeamFormer versions merging 64 receivers.

ToC

Open Computing Language

- Open and royalty-free standard for general purpose parallel programming on heterogeneous architectures
- Allow developers to write portable code
- Defines a C library and a special kernel language
- Platform hierarchy: host, computing devices, compute units, processing elements
- Computational hierarchy: work-items, work-groups, NDRange
- Memory hierarchy: private, local, global, constant[2]

[2] Read only

Results (1/2)

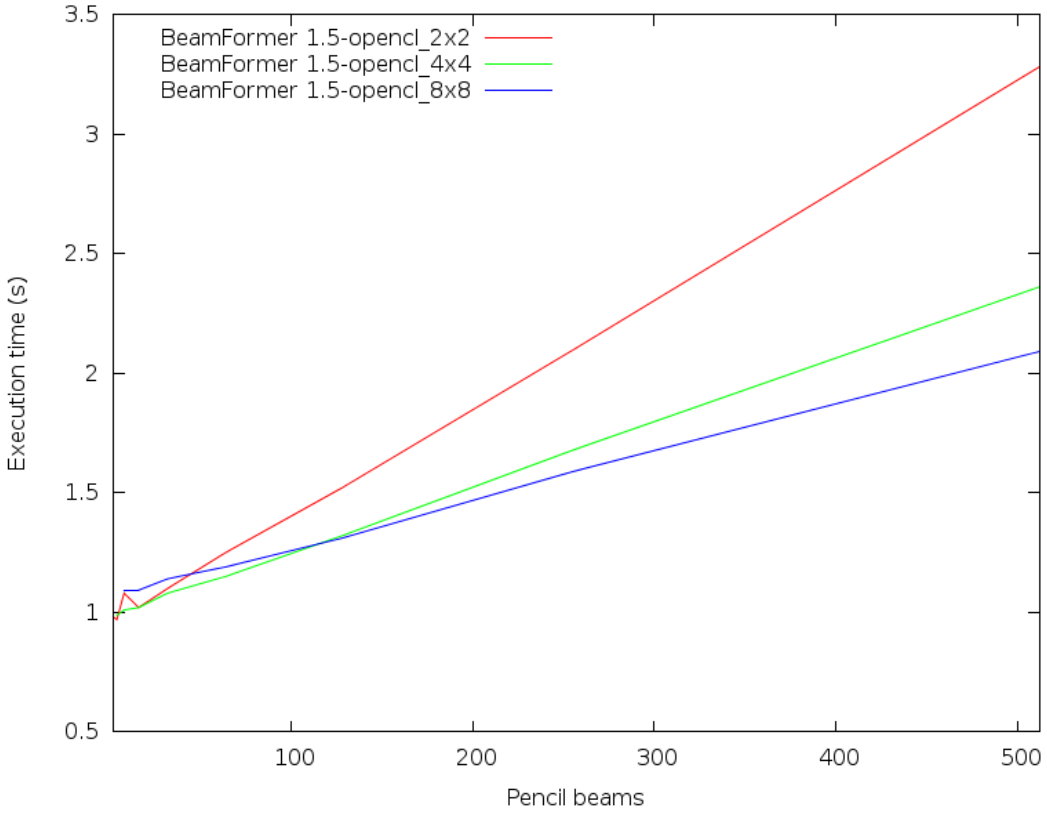

Figure: Execution time in seconds of various OpenCL implementations merging 64 receivers.

Results (2/2)

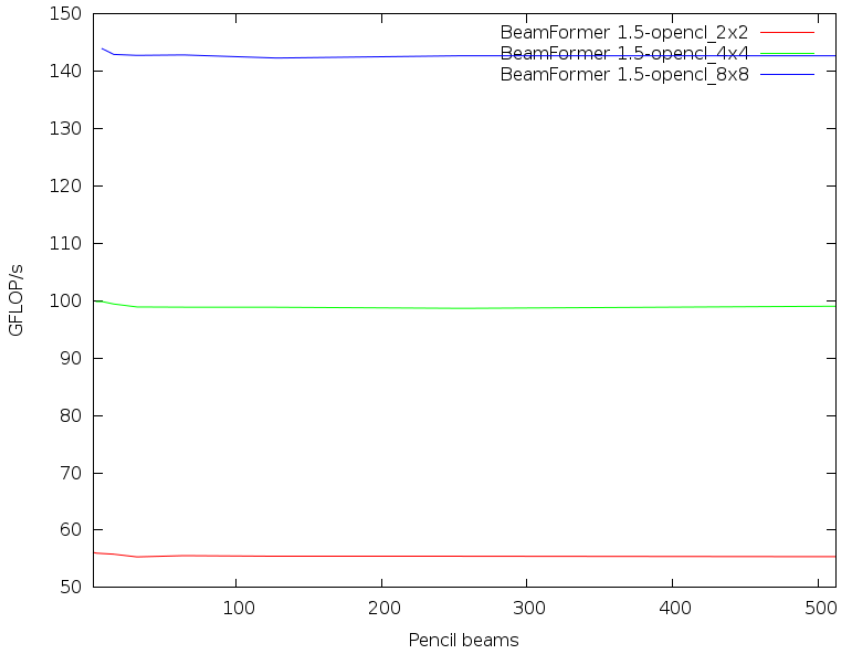

Figure: GFLOP/s of various OpenCL implementations merging 64 receivers.

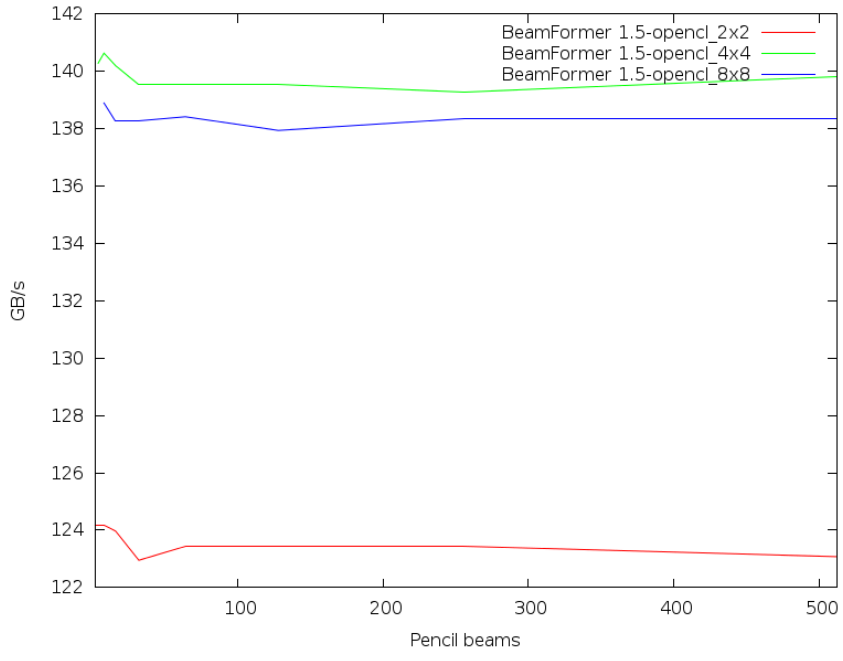

Figure: GB/s of various OpenCL implementations merging 64 receivers.

Comparison CUDA vs OpenCL (1/2)

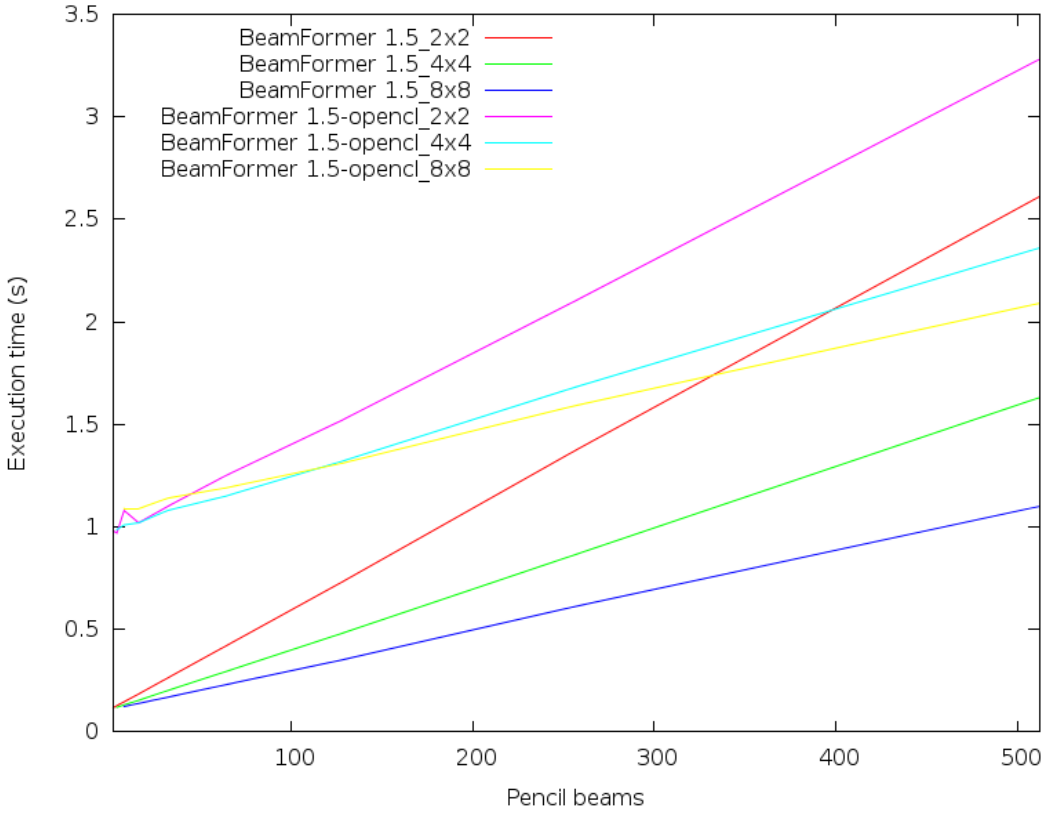

Figure: Execution time in seconds of BeamFormer 1.5 implemented with CUDA and OpenCL merging 64 receivers.

Comparison CUDA vs OpenCL (2/2)

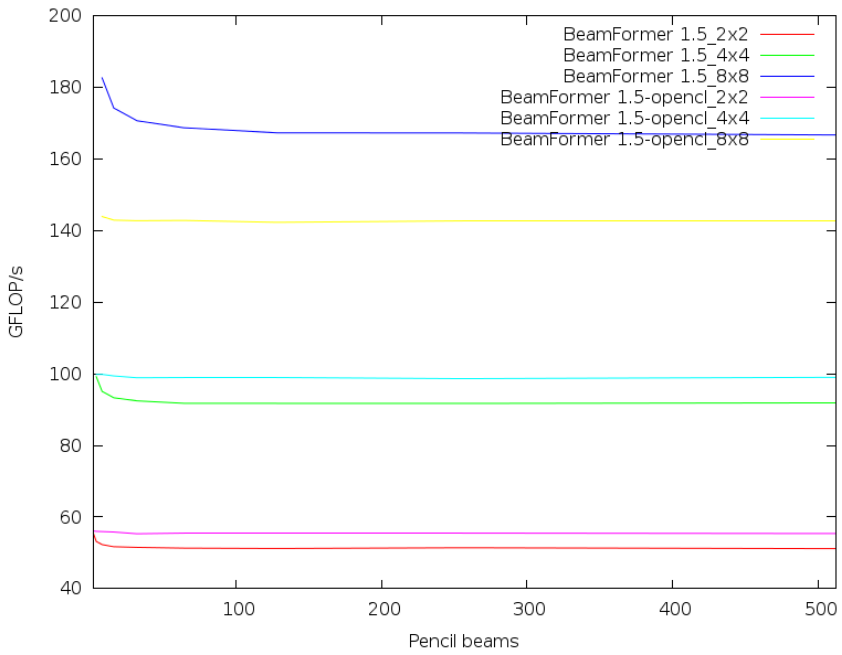

Figure: GFLOP/s of BeamFormer 1.5 implemented with CUDA and OpenCL merging 64 receivers.

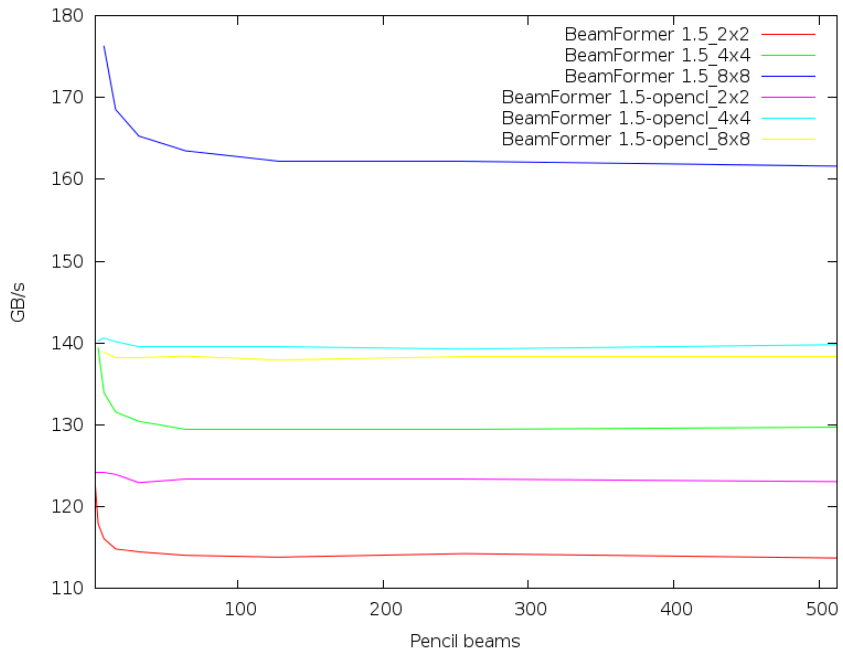

Figure: GB/s of BeamFormer 1.5 implemented with CUDA and OpenCL merging 64 receivers.

ToC

BeamFormer 1.5 best block

- A parameter of the BeamFormer 1.5 is the "*receiver-beam*" block
- The parameter controls
  - How many receivers are merged in a single iteration
  - How many beams are formed in a single iteration
- Question: how the parameter affects the algorithm's performance ?

Experimental setup

- 32 implementations of the BeamFormer 1.5
  - 16 using CUDA and 16 using OpenCL

Experimental setup

- 32 implementations of the BeamFormer 1.5
  - 16 using CUDA and 16 using OpenCL
- Experiment:
  - Single execution
  - Two block dimension intervals
    - From 1×1 to 16×16
    - From 2×1 to 256×16

Experimental setup

- 32 implementations of the BeamFormer 1.5
  - 16 using CUDA and 16 using OpenCL
- Experiment:
  - Single execution
  - Two block dimension intervals
    - From 1×1 to 16×16
    - From 2×1 to 256×16
- Measurements:
  - Single precision FLOPs, in GFLOP/s

Experimental setup

- 32 implementations of the BeamFormer 1.5
  - 16 using CUDA and 16 using OpenCL
- Experiment:
  - Single execution
  - Two block dimension intervals
    - From 1×1 to 16×16
    - From 2×1 to 256×16
- Measurements:
  - Single precision FLOPs, in GFLOP/s
- Questions:
  - Which receiver-beam block provides the best performance ?
  - In which way this parameter affects the algorithm's performance ?

Results (1/2)

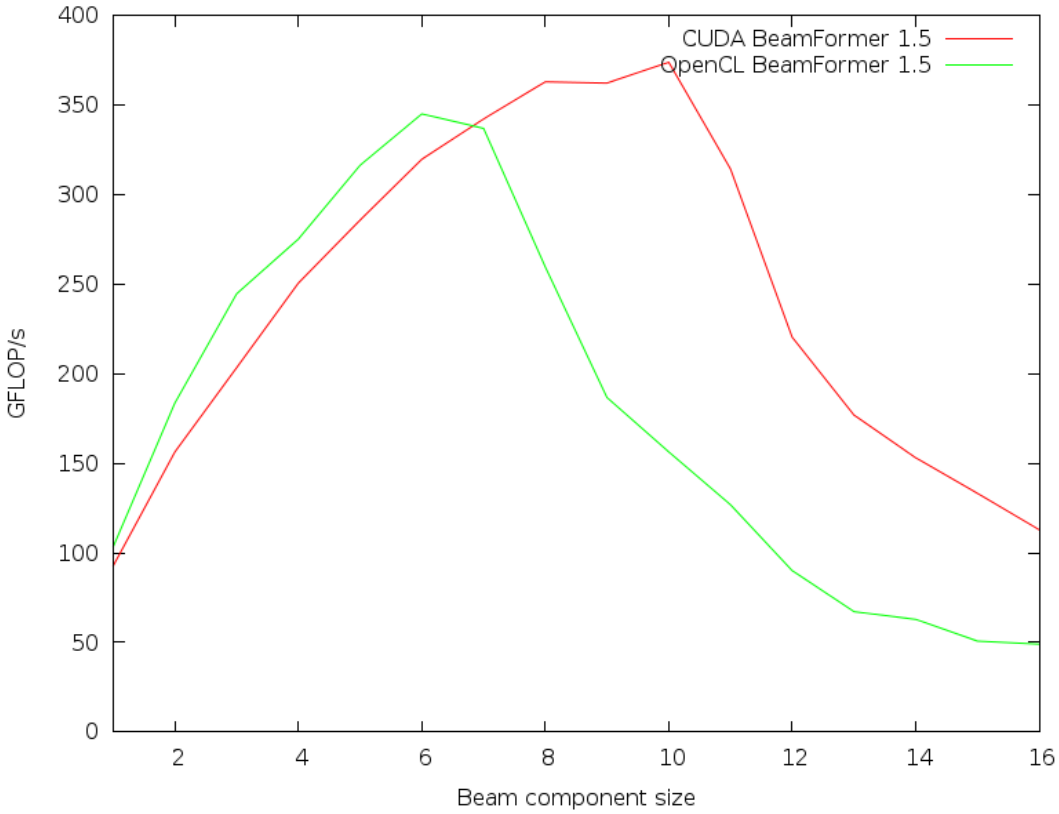

Figure: Comparison of CUDA and OpenCL BeamFormers: block sizes from 64x1 to 64x16.

Results (2/2)

- CUDA peak
  - Block of 256×10
  - 427,07 GFLOP/s
  - 31% of the GPU's theoretical performance peak
- OpenCL peak
  - Block of 256×6
  - 392,16 GFLOP/s
  - 29% of the GPU's theoretical performance peak

ToC

Conclusions (1/4)

Question: "*Is it possible to parallelize the beam forming algorithm efficiently on GPUs ?*"

-	**IBM Blue Gene**	**NVIDIA GTX 480**
% of the theoretical GFLOP/s	80%	30%
Chip's GFLOP/s	10,8	427,07
Power efficiency (GFLOPs/W)	0,456	1,708

Conclusions (1/4)

Question: "*Is it possible to parallelize the beam forming algorithm efficiently on GPUs ?*"

-	**IBM Blue Gene**	**NVIDIA GTX 480**
% of the theoretical GFLOP/s	80%	30%
Chip's GFLOP/s	10,8	427,07
Power efficiency (GFLOPs/W)	0,456	1,708

Answer: It is possible to have an efficient beam forming algorithm on GPUs.

Conclusions (2/4)

- The developed beam forming algorithm scales linearly
- In order to achieve good performance is important to

Conclusions (2/4)

- The developed beam forming algorithm scales linearly
- In order to achieve good performance is important to
  - Have a high number of independent and not idle threads
  - Have a computation structure that permits coalesced access to device memory
  - Reuse data between the kernels of a same block
  - Keep the kernels as simple as possible

Conclusions (3/4)

- A correct setup of the receiver-beam block may improve performance
- The improvement is independent from the implementation framework
- The BeamFormer 1.5 achieves the 30% of the theoretical GPU's GFLOP/s

Conclusions (4/4)

- Good performance are also possible with OpenCL
- But there are two issues
  - Performance reduced by a high register spilling
  - Execution time higher due to the run-time environment

Questions

Questions ?