Using OpenACC for NGS Techniques to Create a Portable and Easy-to-Use Code Base

Sanhu Li (Ph.D. student)
Sunita Chandrasekaran (schandra@udel.edu)
Assistant Professor, University of Delaware, DE, USA

May 9, GTC 2017

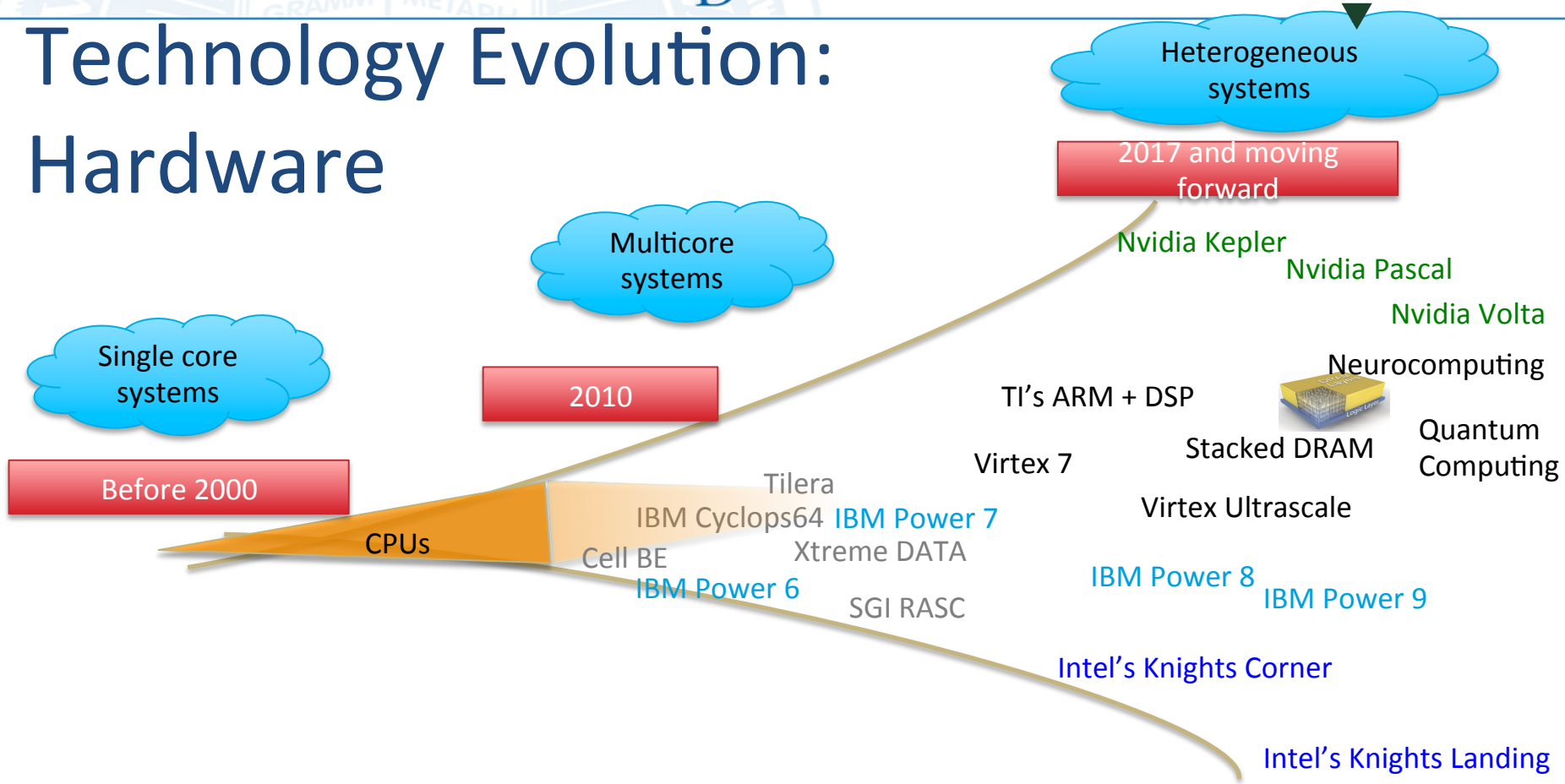
Room 210C

Genome data is evolving

- Next-Generation Sequencing (NGS)
  - Massively parallel sequencing methods
  - Sequencing millions to billions of DNA fragments in parallel
  - High throughput, More cost effective

- Newer and sophisticated sequencing instruments generate increasing amount of un-sequenced data
  - Takes long computation time
  - Generates high demand for data processing and analysis
  - Creates newer algorithms to meet with newer science

Technology Evolution: Hardware



Technology Evolution: Software

- Hardware evolves too rapidly
- Programming complexity rises dramatically
- We need newer parallel algorithms with increasing capacity in a single node
- Future architectures will have 100K cores/node
  - Offers dramatic optimization effort
- Migrating legacy code to future platforms – a real challenge

Software and toolsets

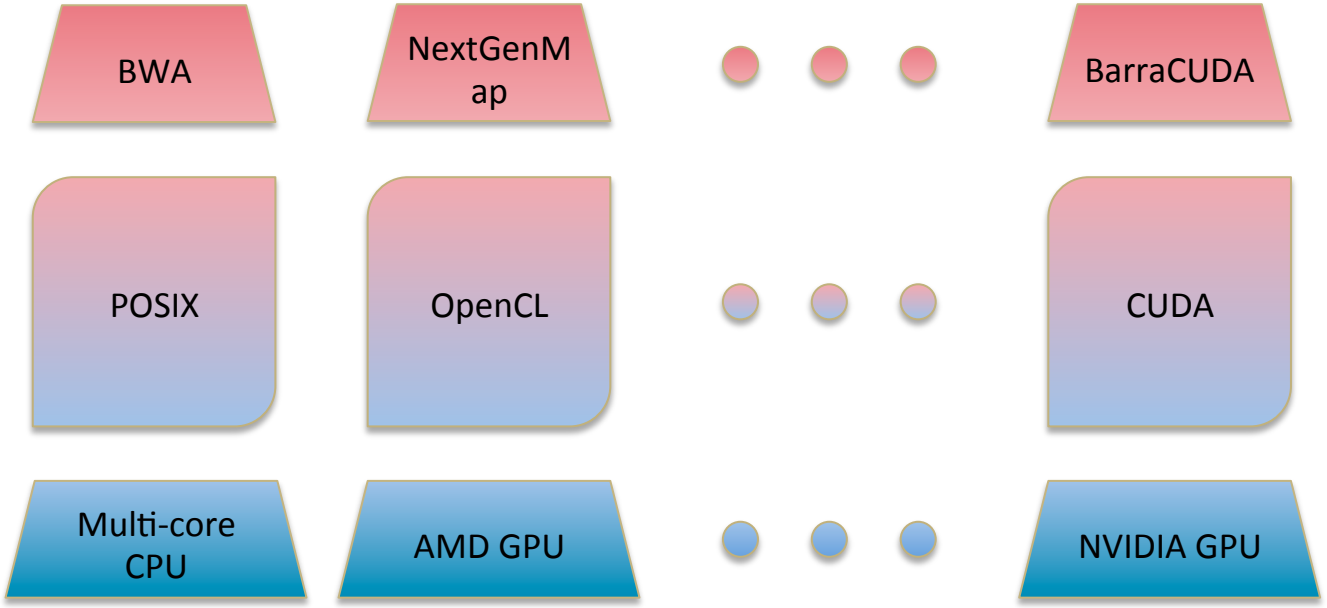
- With growing dataset and evolving hardware:
  - Software that incurs less programming effort
    - less debugging effort
  - Allow programmers to incrementally improve code
  - Software that is easily maintainable
  - Create once and reuse many times
  - Need tools that can facilitate better software

HPC platforms for NGS Sequencers

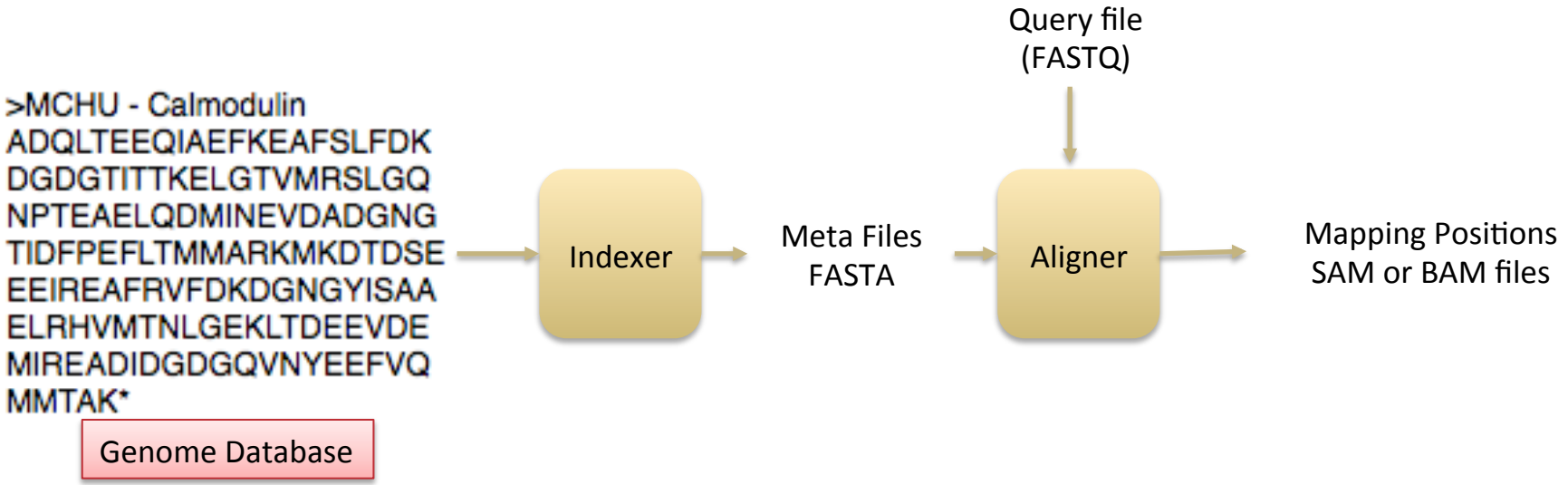
Sequence Alignment Tool	HPC Platform	Year
Bowtie, nvbowtie	POSIX Threads, GPU	2009, >2014
BWA, BWA-PSSM	Multi-core CPU systems	2009, 2014
BarraCUDA, SOAP3, CUSHAW, MUMerGPU, CUDASW++…	CUDA and POSIX Threads	~ 2012 onwards
NextGenMap	CUDA/OpenCL/POSIX Threads	2013
FHAST (bowtie), Shepard	FPGA	2015, 2012
SparkBWA, DistMap, Seal	MapReduce	2016, 2013, 2011
Subread	POSIX Threads	2016

And more !!!

HPC platforms for NGS Sequencers



NGS Sequence Aligner Workflow



NGS Sequence Aligner Principles

NGS Sequence Aligner Principles

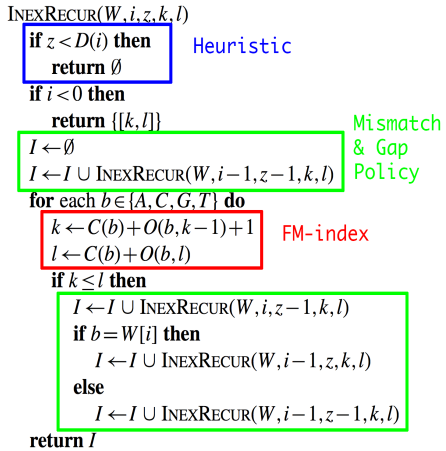
Gap + Mismatch Policy

BWA Heuristic for Mismatch + Gap

Exact String Matching Algorithm

FM-index

Integrated



State-of-the-art Sequence Mapping Tools

- BWA, BarraCUDA, bowtie etc.
  - Uses brute force search method using heuristics to generate search space
  - Uses an FM-index algorithm for alignment
    - Fast text indexing using limited memory resources unlike Suffix Array
- Subread
  - Uses hash-based algorithm to do alignment w/o errors
    - Unfortunately this uses more memory and there is no accelerator-based implementation (only uses POSIX threads)
  - High accuracy and fast alignment speed (due to special gap and mismatch policy – seed and vote)

[1] Slide based on a talk from Will Ramey of NVIDIA, https://developer.nvidia.com

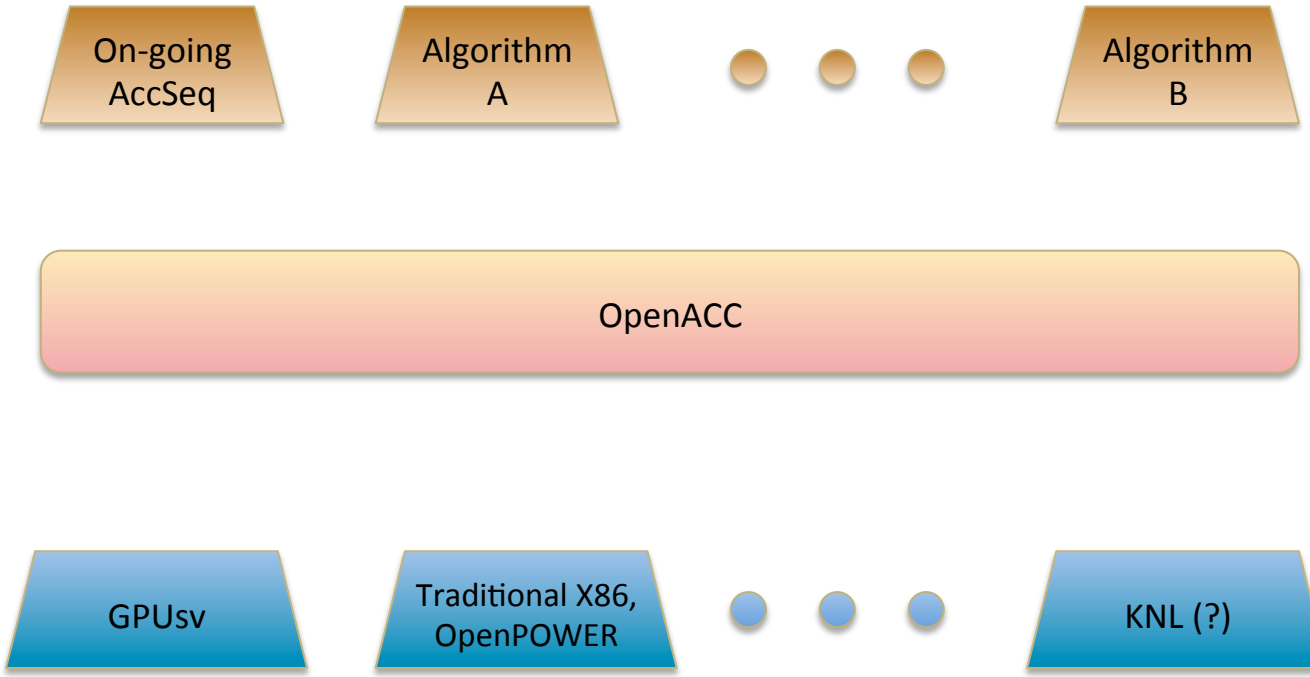
OpenACC – Parallel Programming Model

- Large user base: MD, weather, particle physics, CFD, seismic
  - Directive-based, high level, allows programmers to provide hints to the compiler to parallelize a given code
- OpenACC code is portable across a variety of platforms and evolving
  - Ratified in 2011
  - Supports X86, OpenPOWER, GPUs. Development efforts on KNL and ARM have been reported publicly
  - Mainstream compilers for Fortran, C and C++
  - Compiler support available in PGI, Cray, GCC and in research compilers OpenUH, OpenARC, Omni Compiler

```
#pragma acc kernel
    {
        for( i = 0; i < n; ++i )
         a[i] = b[i] + c[i];
    }
```

```
#pragma acc parallel loop
        for( i = 0; i < n; ++i )
        a[i] = b[i] + c[i];
```

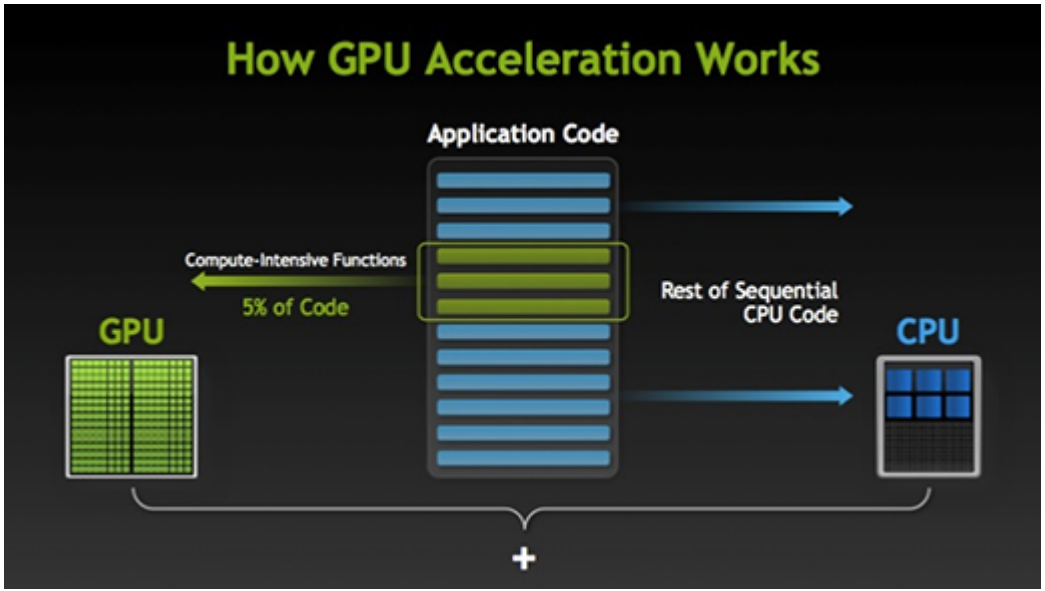
Potential Cross-platform NGS-HPC Solution



What do we plan to do?

- Build a high-level directive-based solution using OpenACC
  - Create a portable codebase
  - Incurs no steep learning curve
  - Maintain a single code base easily
  - Target multiple platforms such as CPUs, CPUs+GPUs, OpenPOWER systems (IBM Power Processor + GPUs – a pre-exacale platform)
- Create a FM-index based algorithm and Subread for exact string matching
  - To use less memory and maintain high accuracy
  - Create an accelerator-friendly solution

GPU Accelerated Computing



http://www.nvidia.com/object/what-is-gpu-computing.html

Profiling results

- On a serial code, the backward search stage in FM-index takes 94%
- Functions reading FASTA and FASTQ consumes the rest of the time

percent of time	self seconds	number of calls	self seconds per call	name
93.67	144.18	15728640	0	backward_search
5.59	8.61	15728641	32.29	kseq_read
0.34	0.52	15728640	0.00	ks_getuntil
0.16	0.24	1	0.24	load_genes
0.10	0.15	1	0.15	concat_queries
0.05	0.08	15728640	0.00	init_query
0.03	0.04	1	0.04	search_substr

Experimental Setup

- Version 1 and 2
  - UDEL Farber Community Cluster
  - Intel(R) Xeon(R) CPU E5-2660
  - Kepler K80
- Version 3
  - NVIDIA PSG Cluster
  - Single node has 32 Intel Xeon E5-2698 and 4 NVIDIA P100 GPUs at runtime
  - Sequential code runs on a single core
  - OpenACC GPU runs on a single GPU (P100)
  - OpenACC multicore uses 12 -13 cores
  - PGI 17.4

Most relevant OpenACC features used

- OpenACC features
  - Kernels
  - Loop
  - Copyin Copyout
  - Loop independent
  - Routines

OpenACC Sequencer preliminary results

- Created a preliminary version of OpenACC version for
  - FM-index + BWA policy (using DFS)
- Issues in V1
  - Too much memory consumption (only 290MB query could be considered)
  - Did not get good performance
- Issues in V2
  - Improved memory consumption (can take > 3GB queries as input) *PRO*
  - Performance worse than V1 ☹ *CON*

OpenACC Sequencer code snippet

```
1  const char *qs = concat_queries(queries, lens, offs, total);
2  #pragma acc kernels loop independent copyin(qs[:total],
       lens[:num_q], offs[:num_q], a1[:((db_size + 1) / 12 + 1) * 4],
       a2[:((db_size + 1) / 1 + 1) * 4], a3[:(db_size + 1) * 4])
3    for (size_t i = 0; i < num_q; ++i) {
4      range r = backward_search(qs + offs[i], lens[i], count, a1,
           a2, a3, (uint32_t) db_size);
5      res[i] = r;
6    }
```

OpenACC Sequencer results contd

- Version 3 (work in progress)
  - Parallelized FM-index

Query size	Sequential	OpenACC-GPU	OpenACC-Multicore
1GB/5million	59.82s	1.87s	2.69s
2GB/10million	100.48s	2.42s	5.24s
3GB/15million	181.52s	2.97s	7.72s

Computation Process
~19x -22x on multicore
~30x – 60x on GPU

Query size	Sequential	OpenACC-GPU	OpenACC-Multicore
1GB/5million	111.09	50.58s	47.58s
2GB/10million	145.13s	58.26s	59.05s
3GB/15million	235.08s	63.78s	73.98s

Total Process time

Summary and Next Steps

- Parallelized an important step in alignment using OpenACC
  - Code can be further improved as it is based on directives
  - Making algorithmic changes shouldn't be too complicated.
- Further improvements
  - Parallelize sub-read, plug-in with FM-index, and use real data to analyze

Contact

- Sunita Chandrasekaran (schandra@udel.edu)
- Sanhu Li (lisanhu@udel.edu)

Thanks to: Mat Colgrove, NVIDIA