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

- Before 2000
  - Single core systems
  - CPUs
    - IBM Cyclops64
    - Cell BE
    - IBM Power 6
    - SGI RASC

- 2010
  - Multicore systems
    - IBM Power 7
    - Xtreme DATA
    - Tilera

- 2017 and moving forward
  - Heterogeneous systems
  - Nvidia Kepler
  - Nvidia Pascal
  - Nvidia Volta
  - TI’s ARM + DSP
  - Stacked DRAM
  - Virtex 7
  - Virtex Ultrascale
  - IBM Power 8
  - IBM Power 9
  - Intel’s Knights Corner
  - Intel’s Knights Landing

schandra@udel.edu
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

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

And more!!!
HPC platforms for NGS Sequencers

- BWA
- NextGenMap
- POSIX
- OpenCL
- Multi-core CPU
- AMD GPU
- NVIDIA GPU
- BarraCUDA
- CUDA
NGS Sequence Aligner Workflow

- Genome Database
- Indexer
- Meta Files FASTA
- Aligner
- Mapping Positions SAM or BAM files
- Query file (FASTQ)
NGS Sequence Aligner Principles

Aligner = Gap + Mismatch Policy + Exact String Matching Algorithm
NGS Sequence Aligner Principles

- Gap + Mismatch Policy
- Exact String Matching Algorithm
- BWA Heuristic for Mismatch + Gap
- FM-index

Integrated

InEX.RECUR(W, i, z, k, l)
if z < D(i) then
  return Ø
if i < 0 then
  return [(k, l)]
I ← Ø
I ← I ∪ InEX.RECUR(W, i - 1, z - 1, k, l)
for each b ∈ [A, C, G, T] do
  k ← C(b) + O(b, k - 1) + 1
  I ← I ∪ InEX.RECUR(W, i, z, k, l)
if k ≤ l then
  I ← I ∪ InEX.RECUR(W, i, z, k, l)
if b = W[i] then
  I ← I ∪ InEX.RECUR(W, i - 1, z, k, l)
else
  I ← I ∪ InEX.RECUR(W, i - 1, z - 1, k, l)
return I

Heuristic

Mismatch & Gap Policy

FM-index

schandra@udel.edu
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)
Three Ways to Accelerate Applications on GPUs

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

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

```c
#pragma acc parallel loop
for( i = 0; i < n; ++i )
a[i] = b[i] + c[i];
```
Potential Cross-platform NGS-HPC Solution

- On-going AccSeq
- Algorithm A
- Algorithm B

OpenACC

- GPUsv
- Traditional X86, OpenPOWER
- KNL (?)

schandra@udel.edu
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

<table>
<thead>
<tr>
<th>percent of time</th>
<th>self seconds</th>
<th>number of calls</th>
<th>self seconds per call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>93.67</strong></td>
<td>144.18</td>
<td>15728640</td>
<td>0</td>
<td>backward_search</td>
</tr>
<tr>
<td>5.59</td>
<td>8.61</td>
<td>15728641</td>
<td>32.29</td>
<td>kseq_read</td>
</tr>
<tr>
<td>0.34</td>
<td>0.52</td>
<td>15728640</td>
<td>0.00</td>
<td>ks.getuntil</td>
</tr>
<tr>
<td>0.16</td>
<td>0.24</td>
<td>1</td>
<td>0.24</td>
<td>load_genes</td>
</tr>
<tr>
<td>0.10</td>
<td>0.15</td>
<td>1</td>
<td>0.15</td>
<td>concat_queries</td>
</tr>
<tr>
<td>0.05</td>
<td>0.08</td>
<td>15728640</td>
<td>0.00</td>
<td>init_query</td>
</tr>
<tr>
<td>0.03</td>
<td>0.04</td>
<td>1</td>
<td>0.04</td>
<td>search_substr</td>
</tr>
</tbody>
</table>
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

schandra@udel.edu
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) {
   range r = backward_search(qs + offs[i], lens[i], count, a1,
   a2, a3, (uint32_t) db_size);
   res[i] = r;
}

schandra@udel.edu
OpenACC Sequencer results contd

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

<table>
<thead>
<tr>
<th>Query size</th>
<th>Sequential</th>
<th>OpenACC-GPU</th>
<th>OpenACC-Multicore</th>
</tr>
</thead>
<tbody>
<tr>
<td>1GB/5million</td>
<td>59.82s</td>
<td>1.87s</td>
<td>2.69s</td>
</tr>
<tr>
<td>2GB/10million</td>
<td>100.48s</td>
<td>2.42s</td>
<td>5.24s</td>
</tr>
<tr>
<td>3GB/15million</td>
<td>181.52s</td>
<td>2.97s</td>
<td>7.72s</td>
</tr>
</tbody>
</table>

Table 3: Performance Evaluation for Computation Process

Table 3 is showing the comparison of total time consuming of our project. The I/O operation has occupied a lot of time and is not consistent all the time. The time could change drastically with different hardware configurations.

However, we could still find the superior of our parallel algorithm.

<table>
<thead>
<tr>
<th>Query size</th>
<th>Sequential</th>
<th>OpenACC-GPU</th>
<th>OpenACC-Multicore</th>
</tr>
</thead>
<tbody>
<tr>
<td>1GB/5million</td>
<td>111.09s</td>
<td>50.58s</td>
<td>47.58s</td>
</tr>
<tr>
<td>2GB/10million</td>
<td>145.13s</td>
<td>58.26s</td>
<td>59.05s</td>
</tr>
<tr>
<td>3GB/15million</td>
<td>235.08s</td>
<td>63.78s</td>
<td>73.98s</td>
</tr>
</tbody>
</table>

Table 4: Performance Evaluation for Total Process Time

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

Total Process time

Conclusion and Future Work
In this report, we have created a parallel FM-index algorithm that has a good speed up compared to sequential version of the algorithm. The next steps will be working towards our aim: the cross-platform NGS tool. Since we have done the FM-index with OpenACC and ran it on different platforms, our aim seems to be promising. We will keep doing research on FM-index to reduce total memory consuming, design policies to handle gaps and mismatches, adding features such as allowing SAM and BAM format output.

References
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