S7367

GPU-accelerated similarity searching in a database of short DNA sequences

Richard Wilton
Department of Physics and Astronomy
Johns Hopkins University

S7367: GPU-accelerated similarity searching in a database of short DNA sequences

GPU vs Database

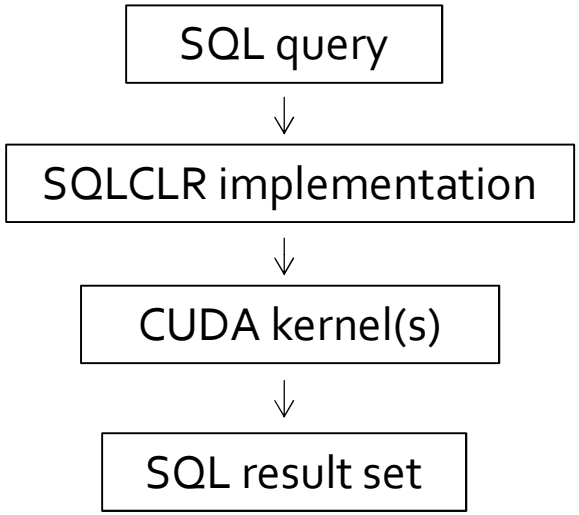
- What kinds of database queries are amenable to GPU acceleration?
  - Compute intensive (compute > I/O)
  - Long-running (amortize the overhead of the CPU-GPU-CPU roundtrip)
- How to design the code
  - GPU code is not native to the SQL database
    - Execute GPU code
    - Export data to the GPU
    - Import data to the database
  - Serialize (or otherwise coordinate) calls to GPU code from database threads

Programming tools

- Tools we used
  - Windows
  - SQL Server
  - NVidia GPU and CUDA toolkit
  - Microsoft Visual Studio (C# and C++ debuggers)
- Programming environments
  - SQL: SSDS (“SQL Server Development Studio”)
  - C# / SQLCLR (“SQL Common Language Runtime”): Visual Studio (C#)
  - CUDA: Visual Studio (C++ compiler, Nvidia Nsight debugger)

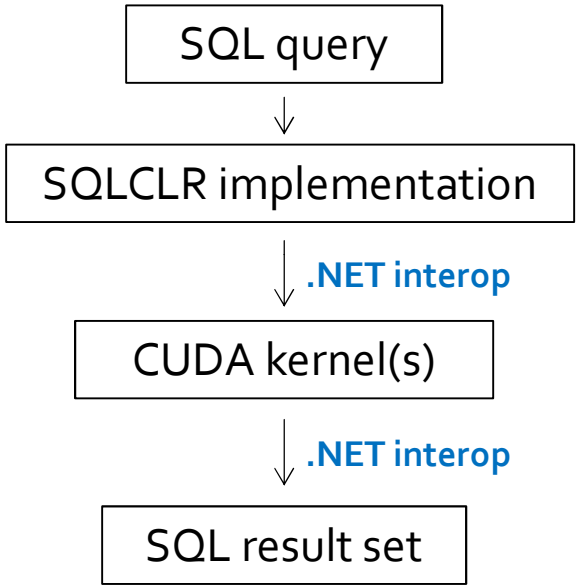
S7367: GPU-accelerated similarity searching in a database of short DNA sequences

Calling CUDA code from a SQL query



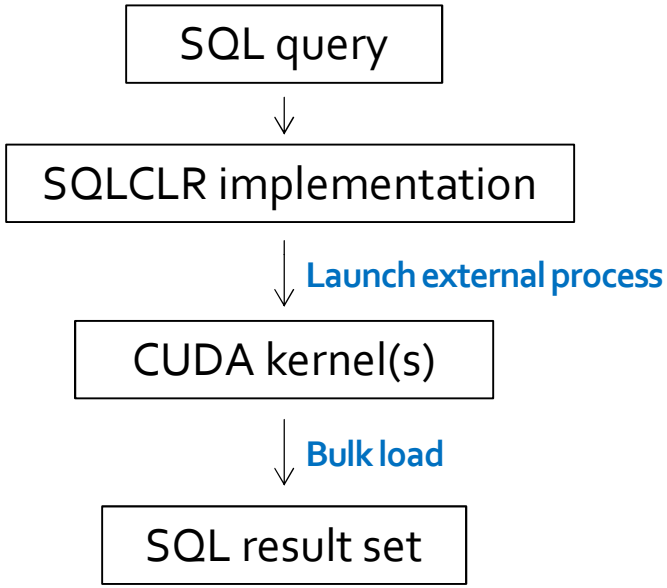

```
create table #tbl
( sqId bigint   not null,
  V    smallint not null )

exec _UDF.GetASHMapping '#tbl', @Q, @Jt, @Vt
```

Calling a CUDA kernel: .NET interop?



- The way it "ought to work"
  - Use .NET interop to call from a SQL-callable C# function into the CUDA C++ implementation
  - Data is marshalled between the C# and C++ implementations
- But…
  - We have to deal with contention for available host resources (memory, CPU threads)
  - Permissions are difficult to configure correctly
  - For non-trivial amounts of data, data-transfer speed is suboptimal

Calling a CUDA kernel: launching an external process



- The way we made it work
  - Use .NET interop to launch an external process
  - The external process is a compiled CUDA C++ application
  - Data moves between the SQLCLR C# caller and the external process using
    - Command-line parameters
    - File system
- Why do it this way?
  - The CUDA application does not require special permissions to execute
  - The OS manages memory and threads
  - Bulk loading of data into a SQL table from a file is faster than interop marshalling

S7367: GPU-accelerated similarity searching in a database of short DNA sequences

The Terabase Search Engine: searching for similar DNA sequences

- The Terabase Search Engine (TSE)
  - Relational database of short DNA sequences from 271 publicly-available human genomes
  - Sequences indexed according to where they map to the human reference genome
    - 354,818,438,126 sequences (length 94-102)
      - 340,366,087,112 (95.9%) mapped
      - 14,452,351,014 (4.1%) unmapped (6,570,283,262 distinct)
- The problem: how do we query those 6.5 billion unmapped sequences?
  - Query by similarity to a given sequence
  - Queries should run in "interactive time" (no more than about 30 seconds)
  - Brute-force comparison is too slow
  - A simple hash table (one hash per sequence) would work only for exact matches

How to compute similarity

- Build a hash table using short, overlapping subsequences from each sequence
  - Query for sequences associated the same (or a similar) set of substrings
  - The problem is then: how to identify similar but not identical sequences
- Jaccard index
  - Query for sequences associated with the same (or a similar) set of substrings
  - Given two sets of values, the Jaccard index is the ratio of the size of the intersection of the two sets to the size of the union of the two sets:

$$J = \frac{|A \cap B|}{|A \cup B|}$$

S7367: GPU-accelerated similarity searching in a database of short DNA sequences

MinHash (locality sensitive hashing)

- The idea is to compute a hash value or "signature" for each sequence that can be used to compute similarity (i.e., similar signatures mean similar sequences)
- How to build an integer "signature" for a sequence:
  - Hash each subsequence
  - Sort the hash values
  - Sample the bits in each of the first N values in the sorted list; use the value of the sampled bits to identify a bit whose value is set to 1 in the signature value
- Compute the Jaccard index for two sequences using the signature bit-patterns of the sequences

S7367: GPU-accelerated similarity searching in a database of short DNA sequences

MinHash example

AGCCGTCTTAGAGCAGCTCGAACGTGTACGAA…

1

AGCCGTCT
GCCGTCTT
CCGTCTTA
CGTCTTAG
⋮

2

0x1534D637
0x09D678F9
0xC0F23A8B
0x80845A6E
⋮

3

0x09D678F9
0x1534D637
0x80845A6E
0xC0F23A8B

4

0x39 = 57
0x37 = 55
0x2E = 46
0x0B = 11

5

57 55 46 11
0000000101000000001000000000000000000000000000000000001000000000000

1. Extract subsequences
2. Compute hash values
3. Sort the list of hash values
4. Extract 6 bits from the first N hash values in the sorted list
5. Use each 6-bit value to identify a bit to set to 1 in the 64-bit signature value

Computing similarity

We want to compute a Jaccard index for two 64-bit signatures A and B:

$$J = \frac{|A \cap B|}{|A \cup B|}$$

When A and B are represented as bits in an integer, the computation is reduced to four logical bit operations and a divide:

```
J = popc( A & B ) / popc( A | B )
```

Running a CUDA-accelerated query

- The CUDA application is parameterized with…
  - A query sequence
  - A threshold Jaccard index value
  - A threshold Smith-Waterman alignment score
- The application computes a 64-bit signature for the query sequence
- The application …
  - Executes a CUDA kernel that computes Jaccard indexes with all of the sequences in the database
  - Computes a Smith-Waterman alignment for sequences with above-threshold Jaccard indexes

S7367: GPU-accelerated similarity searching in a database of short DNA sequences

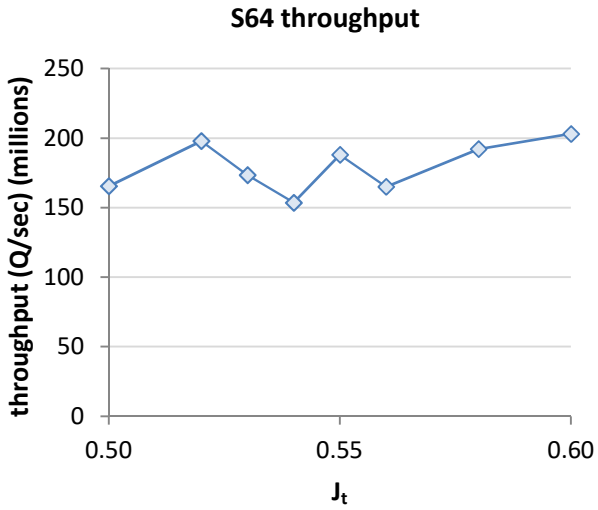
Computing a Jaccard index in CUDA

```
static __global__ void tuScanS64_10_Kernel(       UINT32* const              pC,         // out: candidates for SW alignment
                                           const UINT64* const __restrict__ pS64buf,    // in: pointer to S64 (sketch bits)
                                           const UINT32                     celS64buf, // in: size of S64 buffer
                                           const UINT64                     s64q,      // in: target S64 (sketch bits) value
                                           const double                     Jt         // in: Jaccard similarity threshold
                                          )
{
    // compute the 0-based index of the CUDA thread
    const UINT32 tid = (((blockIdx.x * gridDim.x) + blockIdx.y) * blockDim.x) + threadIdx.x;
    if( tid >= celS64buf )
        return;

    // compute the Jaccard index
    const UINT64 s64 = pS64buf[tid];
    double J = static_cast<double>(__popc64(s64&s64q)) / __popc64(s64|s64q);

    /* If the Jaccard index is at or above the specified threshold, save the offset of the S64 value.
       Otherwise, save a null value (all bits set). */
    if( J >= Jt )
        pC[tid] = tid;
}
```

S7367: GPU-accelerated similarity searching in a database of short DNA sequences

How fast is it?

	J_t 0.50	0.52	0.53	0.54	0.55	0.56	0.58	0.60
S64 (sec)	8.259	6.903	7.878	8.899	7.266	8.280	7.110	6.729
S64 Q/sec	165433435	197930572	173434214	153535761	188042216	165013857	192168037	203048706
$J \geq J_t$	14295564	3233701	3214319	3206316	556256	552785	550410	71411



- 1,366,314,740 Jaccard index computations
- GTX 750ti
- Average ≈ 178 million/sec

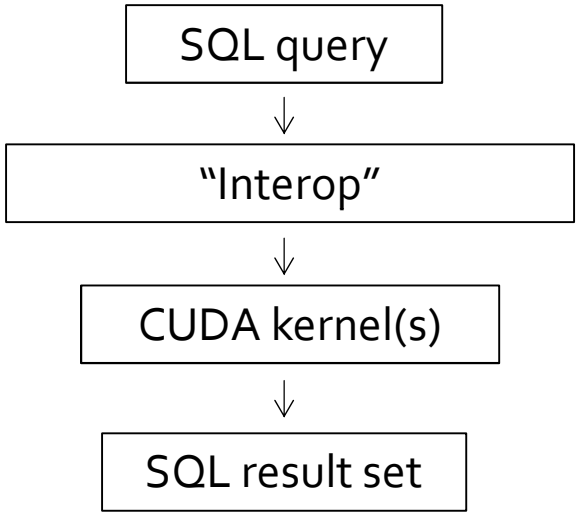
S7367: GPU-accelerated similarity searching in a database of short DNA sequences

Problems and solutions

- Serializing access to the GPU
  - One SQL query can keep the GPU busy for several tens of seconds
  - A global synchronization object serializes access to the GPU
  - If multi-user concurrent access ever becomes a problem… we'll worry about it then
- Our GPU-accelerated queries tend to be limited by disk read speed
  - Other system processes (particularly the SQL database server) contend for disk bandwidth
  - SSDs and buffering (for repeated queries) help but do not entirely fix the problem

S7367: GPU-accelerated similarity searching in a database of short DNA sequences

A strategy for GPU acceleration of a SQL query



- Choose a query that is well suited to GPU acceleration
- Minimize the overhead of data transfers between the host and the GPU
- Attend to the system-level details
  - Permissions
  - Shared resources (CPU threads, memory, disk)
  - Synchronization
- Evaluate performance

S7367
GPU-accelerated similarity searching in a database of short DNA sequences

Questions / Comments