

Summer Research Internship
Beginning: June 15
Concluding: August 15
Location: University of California @ Davis
Faculty Advisor/Project Manager: John Owens
Project Team: Andrew Davidson, Mark Harris, John Owens

Summer Research at UCDavis: GPU Computing using CUDA for biological sequence alignment.

Andrew Davidson

Computer Engineering and
Center for Computation and Technology
Louisiana State University
Fred C. Frey Computing Center
Baton Rouge, LA, 70803

Faculty Advisor: Gabrielle Allen

1.1 Project Description

My research at UCDavis involved developing code for a GPU computing library with the appellation CUDPP: CUDA Data Parallel Primitives Library. CUDA is a new computing architecture designed to enable the GPU to solve computational rather than visual problems (discussed in section 2.1). Although powerful in raw performance and capabilities, there are very few libraries available to the public for CUDA programming. Those interested in CUDA and scientific computing currently must build their programs from scratch, making it very inconvenient and tedious. Our goal with this library, is to allow for reusable libraries that include native GPU primitives and adequate examples to demonstrate their efficacy. One such primitive which was used in this biological sequence alignment algorithm is the scan primitive (discussed in section 2.2). This primitive is well suited for parallel algorithms, however my work required the primitive be modified from the addition-operator to the maximum-operator. We expanded this change to allow for easily templated scan operations, therefore my work (with the advice of Mark Harris [NVIDIA Researcher]) was to expand the number of binary operators supported (maximum, minimum, addition, multiplication) with little code modification and no significant performance change. With this change, the biological sequence alignment algorithm (discussed in section 2.3) may use the scan primitive thereby improving performance.

2.1 CUDA

NVIDIA's Compute Unified Device Architecture (CUDA) technology allows for complex computational problems to be solved on the GPU¹. Recently introduced to developers, it has immediately become a matter of great interest to computational scientists with experience working on parallel algorithms. Prior to the CUDA technology, General Purpose programming on GPUs (GPGPU) required extensive knowledge in fragment shader languages, lacked basic debugging tools, and lacked many of the capabilities CUDA provides. For features, an available CUDA SDK/Toolkit, and a few available libraries visit: <http://developer.nvidia.com/object/cuda.html> .

2.2 The Scan Primitive

The CUDPP project has as one of its main goals the implementation of GPU primitives for easy use. One of these primitives is the scan primitive. Scan takes as an input an array of n -elements, and returns an array of the same size. Each element of the output is a result of all binary operations of all previous elements (i.e. if the input is $[a_0, a_1, a_2, a_3, \dots]$, an exclusive scan produces the output $[i, a_0, a_0 \times a_1, a_0 \times a_1 \times a_2, \dots]$, while an inclusive scan produces the output $[a_0, a_0 \times a_1, a_0 \times a_1 \times a_2, a_0 \times a_1 \times a_2 \times a_3, \dots]$ where i is the identity, \times is the binary operator requiring an identity, associative properties and commutative properties³. The concept of the scan primitive was first implemented by Blelloch as a primitive on an early parallel machine². Since then various scan-like algorithms have been created for use in various parallel algorithms (quicksort, sparse-matrix multiplication, convex hull, matrix solvers, etc.). However, none of these algorithms had performed better than $O(n \log n)$. Using CUDA, Sengupta et al introduced an algorithm for scan that performed much better at $O(n)$.

For a more comprehensive investigation of scan refer to *Scan Primitives for GPU Computing*, Shubhabrata Sengupta, Mark Harris, Yao Zhang, John D. Owens. *Graphics Hardware Proceedings*, 2007.

For more information on the CUDPP library which is slated for an upcoming release visit:
<http://www.gpgpu.org/developer/cudpp/>.

2.3 Biological Sequence Algorithm on the GPU

My major goal this summer was to implement a biological sequence algorithm in parallel on the GPU. Previous CPU algorithms have been developed by Futamura and Aluru, though there are no corresponding GPU implementations. The goal of a sequence alignment match is to find the optimal match between two similar DNA sequences of size m and n . For a brief example, take the word 'aberrant' and the misspelled word 'abarrent' our algorithm would attempt to build a scorecard on how well these two words match. However, our DNA sequences are much longer with a goal of at least a length of 10,000 for both sequences m and n . The algorithm we implemented reflects those developed by Futamura and Aluru⁴, however we implemented the templated scan primitive in order to cut down on the GPU compute time.

3. Results

We were successful in implementing the GPU algorithm, however in its current form it has limitations. It is time efficient at $O(mn)$, however the space required is in the order of $O(mn)$ as well. This means that the large sequences we wish to employ will not fit on current GPU cards (we can fit a max result $mn = 36$ million on an NVIDIA 8800GT). Therefore we wish to adapt our algorithm to a 'space-saving' implementation which retains an $O(mn)$ time efficiency but reduces the space size to $O(m+n)$.

4. References

- [1] *NVIDIA CUDA Programming Guide 1.0*. NVIDIA Corporation. Available: <http://developer.nvidia.com/CUDA>.
- [2] Blelloch G. *Vector Models for Data-Parallel Computing*. MIT Press, 1990.
- [3] Sengupta S., Harris M., Zhang Y., Owens J. *Scan Primitive for GPU Computing*. *Graphics Hardware Proceedings*, 2007.
- [4] Aluru S., Futamura N. Mehrotra K. *Parallel biological sequence comparison using prefix computations*. *Journal of Parallel and Distributed Computing*, 2003.