LA-UR-13-26483
Approved for public release; distribution is unlimited.

Title: Summer Student Internship working on the GPU

Author(s): Abney, Stephanie T.

Intended for: School purposes

Issued: 2013-08-15

Stephanie Abney

Mentor Joanne Wendelberger

Los Alamos National Laboratory

13 August 2013

Summer Student Internship working on the GPU

This summer internship consisted of learning computer programming skills and languages including Thrust, CUDA, C++, C and Linux. The application of this programming is for use in GPU, graphics processing units, that are processors that perform math intensive calculations rapidly in three dimensions typically in advanced video games ("NVIDIA GeForce…"). Our goal is to apply this technology towards exa-scale computing on the LANL supercomputing infrastructure. My mentors use these machines to do computations. The completed algorithms include KD tree and Top K. In order to access the GPU, CUDA was used. CUDA is a parallel computer architecture for general computing on the GPUs. There are different means by which to access CUDA. The first approach uses libraries, like Thrust, Linear Algebra, and Signal and Image Processing. Another approach parallelizes loops in Fortran or C and in the last approach people develop custom parallel algorithms using familiar language ("What is CUDA"). The algorithms I worked on this summer use the first approach of using the thrust library of parallel algorithms and data structures.

Thrust is a C++ template library making the GPU more accessible to C++ programmers ("Thrust"). Before programmers were specially trained to work on the GPU. With Thrust, programmers with C++ knowledge can use the GPU for any purpose. Examples of Thrust commands include thrust::sequence, thrust::copy, thrust::transform, and thrust::zip_iterator ("QuickStartGuide"). Thrust

increases productivity by having containers, handling memory management, having explicit algorithm selection, and having a large set of algorithms and being flexible (An Introduction…). The containers are the host vector, h_vector, on the CPU and the device vector, d_vector, on the GPU. This allows movement from the CPU storage to the GPU platforms. Thrust also contains generic algorithms that support general functions. The standard algorithms in Thrust are transformations, reductions, prefix sums, reordering and sorting. Some transformations include thrust::replace and thrust::transform. Thrust also contains "fancy" iterators that act like regular iterators. Iterators act like pointers but point to places in an array and have the ability to track memory space in both host and device containers. Pointers identify a location of a variable instead of pointing to the actual variable. They make the program more flexible. Pointers also allow variables to pass from a main function to another function. The "&" gives the address of the pointer and the "*" gives the data at the pointer (QuickStartGuide). Examples of fancy iterators are constant_iterator, and zip_iterator. The zip_iteartor is used in the KD_tree code to help tuple the vectors together. A tuple is a mathematical term that orders a list.

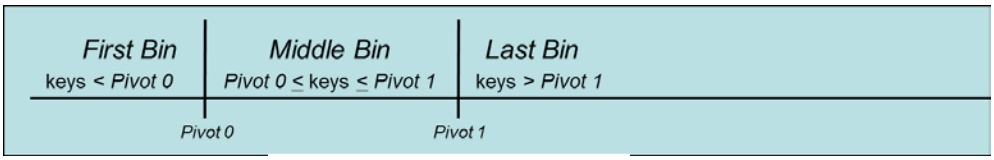
Throughout the summer I learned the basics on how to access the program files. Examples are as follows. Breeze terminal was used to access the algorithms. The README file contains the information needed to set-up the environment correctly to get the program to execute. The environment is the address where the README file and RUN_combo files are located. The RUN_combo.py file executes the algorithm by giving directions to the executables. The .py extension means the file is a python script. An error occurs if in a python script the spacing does not match. Inside this file are contained all the arguments for the Top_K and KD_tree algorithms. The arguments include the different quantiles, probabilities and the length of the list. The length of the list is coded as a power of two, therefore, 17 means 2^{17}. These list lengths are used to develop large lengths and measure the time it takes to complete them. The larger the length the slower the program executes. It is structured in the program as expList=array('i', [12, 13, 14]). The program executes this by first inputting the first

size 2^{12} and calculates everything for that length and then moves to the next size. It continues this process until it reaches the end of the expList.

For both algorithms, specified files belong in specific places. The run files belong together and the program files belong together as well. The make files compile the programs, but those programs have a certain location, just like the executable files are sent to a precise location as well where the RUN_combo file reads from a specified location. The make files compile the files that contain the program. If anything is changed in the program files, the program must be compiled again for it to execute correctly. The make command instead of a make all command will suffice, since the make file will only compile all the files edited. However the RUN_combo file only needs to be saved to run. Executable files are the files that the computer reads. In this program some are contained in /bin/linux. The C++ and the CUDA files create executable files using a make file. Then the executable files are read by the computer and controlled by the RUN_combo files. The commandstring command in the RUN_combo.py script is the instruction that would be written on the console if the program ran manually. To run a program manually use g++ filename arguments structure on the console for C++ programs and gcc for C programs. The program's files that end in .cpp are C++ files, .cu is CUDA files, .txt means a text file and lastly .h means a file that contains the libraries in the program. Each program file includes the .h file instead of rewriting each library. This prevents making a mistake in a single file that would be hard to find. The predicate is also located in the .h file and says which element to grab in the tuple in the zip iterator. If a new file is made, check the permission to the file. In order to run the file it must have the permission to execute it. Using the command chmod on the linux console changes the permission and allows a user to allow others and themselves to read, write and execute a file. In order to pass a function, it must be declared in the main function as well as outside of the main function. It is possible to declare a function by writing the type of function just like declaring a variable. All the arguments of a function declare the type of variable they are.

Probabilistic computing is a method for computing difficult or impossible problems using probabilistic or statistical methods (Monroe et al). The first algorithm Top_K uses the Las Vegas method that is similar to the Monte Carlo method but finds a correct answer. The Monte Carlo method takes random samples of data points and performs deterministic calculation to converge arbitrarily close to a solution. The Monte Carlo method iterates and converges to a close to correct answer while the Las Vegas method iterates and computes a correct answer. The Las Vegas algorithm was used to test graph isomorphism and is used for stochastic optimization (Monroe et al, 2013). The reason to use probabilistic computing is because it is a faster method than long computations.

The Top_K program essentially performs a guess and check method. First the program takes a random sample out of the list and creates a binomial distribution based on the order statistics of the points. After a probability is chosen, the pivots are determined using the binomial distribution to make the three bins where the K^{th} term has the chosen probability of being in the middle bin. The last bin contains the elements on the binomial distribution greater than K, the first bin contains the elements less than K and the middle bin contains elements both greater and smaller than K.

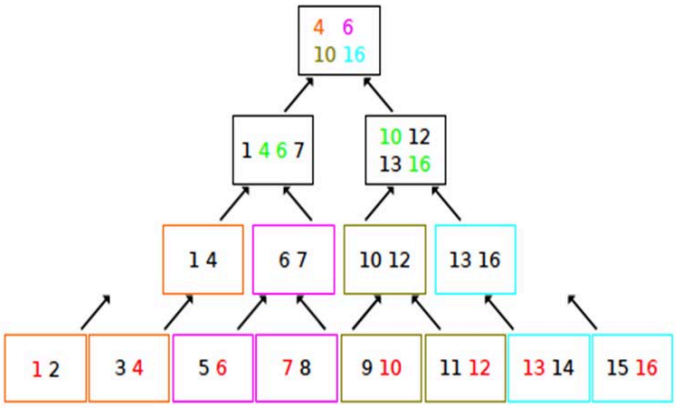


(See Monroe)

K resides in the middle bin if the number in the last bin is smaller than K, and the number in the last bin added to the number in the middle bin is greater than k. The middle bin is then sorted and produces the K^{th} element and therefore the Top K. However if the check is not correct the pivots of the three bins are moved and the program checks again (Monroe et al).

The KD_tree code was tested in a similar method as the Top_K algorithm. The KD tree program is a multidimensional partitioning program. It is used for several applications such as searching a

multidimensional array. It is also used in the nearest neighbor algorithm which is the next algorithm that will be pursued. The KD_tree program reads in a file, and then places the values from the file into three different

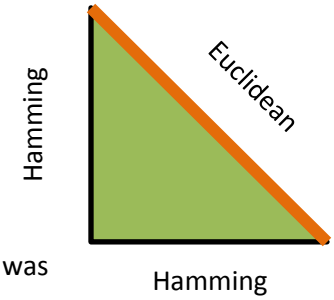


(See Woodring et al,2011)

vectors. The first vector, x, is sorted and partitions the list at the median, while keeping the y vector, and z vector attached. Then from those two partition segments the first segment is sorted by the y vector and partitioned at the y segment's median. The second segment is sorted by the y vector segment, and partitioned at that median. These vectors, x,y, and z, are partitioned three times so far, once for the x vector and then twice for the y vector, and has four partitioned segments. Lastly in the first of four partition segment the z vector is sorted and partitioned at that median and continues down to the end three more times. There are now eight partition segments. A fourth vector, the u vector, was added later to make the program work for four dimensions. In this case each eight partition segments will be partitioned to create sixteen partition segments. This can continue up to ten dimensions, because the zip_iterator which connects the different vectors can only tuple up to 10 elements. In the KD_tree program I figured out how to read the text file into the program. This was accomplished by using the ifstream command, which obtains the file and inserts the data into different vectors. I also added the vector u into the program figuring out how to use the zip_iterator and the predicate that comes with it. The original partition shifted the median using binary. A shift or a >> shifts binary like dividing by 2^shift. To shift 1, divide one by (2^1) or 1/2. This means that it would modulo of the numbers by 2 to calculate the partition, if it is a half, a fourth, an eight, or a sixteenth. However a bug was found and the new

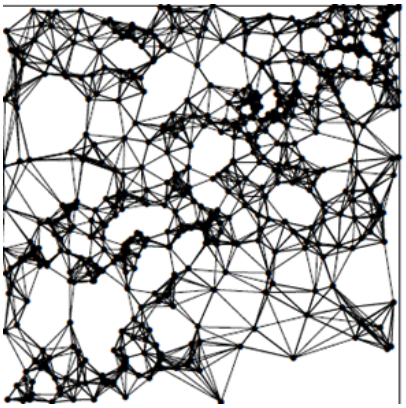
program uses a partition array, which counts the number of elements before the partition, because the median might not divide the list evenly especially if there are duplicates of a number.

The nearest neighbor algorithm has been discussed, including which distance to use. The first distance, called Hamming distance, is the distance of blocks on a grid if the traveler was in a car and the grid was the roads. It is not straight to the destination. The second distance, called Euclidian distance, is the distance measured straight to the destination, as if the traveler was a bird flying directly to a destination. These two types of distances can be illustrated by looking at a right triangle, where the two legs are the Hamming distance and the hypotenuse is the Euclidean distance. My task was to write a program that calculates the time it takes to compute both of the distances and compare them. This determines which distance is faster to compute for the nearest neighbor algorithm. When writing programs, think about flexibility of the program, but make sure it is not so flexible that it is difficult to write or read. I inserted the points into a string and from the string to an array with a pointer. Inserting the data into one array makes the program more flexible in regards to how many vectors there are. The data is inserted with each first element and then the second element in order of the vectors. The distances are calculated in two functions that are then called. Then CPU time is not accurate enough so the CUDA timing on the GPU will be used. The function inserts the list and then will generate a point list that will calculate the distance for each point. Some things that will be looked at will be to determine whether there is a difference associated with the order in which the distances are calculated. If it is faster to calculate a point with every point then move to the next point or another method. The nearest neighbor algorithm will incorporate the Top_K and KD_tree algorithms.



The next step will be to use these algorithms to address real problems. The algorithm Top K has been used in the CLEAN algorithm that chooses the top brightest pixels from an image. It was used in

radio-astronomy to remove noise from images (Monroe et al, 2011). The KD tree and Top K algorithms will be used in the K^{th} nearest neighbor algorithm which is being created now. Top K will be used to calculate the top closest points to a point. The KD tree will be used to find the points easily. The nearest neighbor algorithm is used to find the clusters or voids in data. For example this nearest neighbor plot was part of a particle simulation with large amounts of data that used random samples to "thin" the data. This was used to compare the distributional differences to evaluate the sample algorithm (Figg et al, 2011).



(See Figg et al,2011)

I would like to thank the following people for their contribution in my summer internship. Without them this would not be possible.

Joanne Wendelberger

Laura Monroe

Sarah Michalak

Bill Rust

Work Cited

"An Introduction to the Thrust Parallel Algorithms Library." GPU Technology Conference. NIVIDIA Corporation, n.d. Web. 13 Aug. 2013. http://on-demand.gputechconf.com/gtc/2012/presentations/S0602-GTC2012-Thrust-Parallel-Library.pdf.

Figg, J., Wendelberger, J., Ahrens, J., Woodring, J., and Heitman, K., Comparative Analysis of Particle Data Sets, LA-UR-02794, 2011.

Monroe, L., Wendelberger, J., and Michalak, S., "Probabilistic Computing", LA-UR-13-22565.

Monroe, L., Wendelberger, J., and Michalak, S., Randomized Selection on the GPU, HPG '11, *High Performance Graphics Proceedings*, Association for Computing Machinery, 2011.

Monroe, L., Wendelberger, J., Michalak, S. and Owens, J.. "Probabilistic Algorithms for New Computer Architectures" July 2013.

"QuickStartGuide A brief tutorial for new Thrust developers." Thrust. Google Project Hosting, n.d. Web. 13 Aug. 2013. http://code.google.com/p/thrust/wiki/QuickStartGuide.

"Thrust." CUDA Toolkit Documentation. NVIDIA Developer Zone, 19 July 2013. Web. 13 Aug. 2013 http://docs.nvidia.com/cuda/thrust/index.html.

"What is CUDA." NVIDIA Developer Zone. NVIDIA Corporation, 2013. Web. 13 Aug. 2013. https://developer.nvidia.com/what-cuda.

"NVIDIA GeForce 256 Review GPU Overview." Inno3D. NVIDIA, 21 Oct. 1999. Web. 13 Aug. 2013. <http://www.nvnews.net/reviews/geforce_256/gpu_overview.shtml>.

Woodring, J., Ahrens, J., Figg, J., Wendelberger, J. and Heitman, K., In-situ Sampling of a Large-scale Particle Simulation for Interactive Visualization and Analysis, EuroVis 2011, 30, 3, 2011.