SOFTWARE ARTICLE   **Open Access**

Accelerating calculations of RNA secondary structure partition functions using GPUs

Harry A Stern[1] and David H Mathews[2]*

Abstract

Background: RNA performs many diverse functions in the cell in addition to its role as a messenger of genetic information. These functions depend on its ability to fold to a unique three-dimensional structure determined by the sequence. The conformation of RNA is in part determined by its secondary structure, or the particular set of contacts between pairs of complementary bases. Prediction of the secondary structure of RNA from its sequence is therefore of great interest, but can be computationally expensive. In this work we accelerate computations of base-pair probabilities using parallel graphics processing units (GPUs).

Results: Calculation of the probabilities of base pairs in RNA secondary structures using nearest-neighbor standard free energy change parameters has been implemented using CUDA to run on hardware with multiprocessor GPUs. A modified set of recursions was introduced, which reduces memory usage by about 25%. GPUs are fastest in single precision, and for some hardware, restricted to single precision. This may introduce significant roundoff error. However, deviations in base-pair probabilities calculated using single precision were found to be negligible compared to those resulting from shifting the nearest-neighbor parameters by a random amount of magnitude similar to their experimental uncertainties. For large sequences running on our particular hardware, the GPU implementation reduces execution time by a factor of close to 60 compared with an optimized serial implementation, and by a factor of 116 compared with the original code.

Conclusions: Using GPUs can greatly accelerate computation of RNA secondary structure partition functions, allowing calculation of base-pair probabilities for large sequences in a reasonable amount of time, with a negligible compromise in accuracy due to working in single precision. The source code is integrated into the RNAstructure software package and available for download at http://rna.urmc.rochester.edu.

Keywords: RNA, Secondary structure, Partition function, Graphical processing unit, CUDA

Background

RNA performs many diverse functions in the cell in addition to its role as a messenger of genetic information. It can form enzymes, for example for cleavage of itself or of other RNA, or to create peptide bonds as a fundamental constituent of the ribosome [1]. It can act as a signalling molecule for regulation of gene expression, for protein export, or for guiding post-translational modifications [2-5].

As for proteins, RNA function depends on its folding to a well-defined three-dimensional shape. In contrast to proteins, the folding of RNA is hierarchical [6]. Secondary structure, or the particular set of contacts between pairs of complementary bases mediated by hydrogen bonding and stacking of bases, provides a significant amount of information. This can be helpful in predicting function or accessibility to ligands [7-10]. Computational prediction of the secondary structure of RNA from its sequence is therefore of great interest. The most widely-used automated prediction methods attempt to estimate the thermodynamic stability of RNA, using empirical parameters determined from experiments on oligonucleotides [11,12].

*Correspondence: david_mathews@urmc.rochester.edu
[2]Department of Biochemistry & Biophysics and Center for RNA Biology, University of Rochester Medical Center, 601, Elmwood Avenue Box 712, Rochester, NY, 14642, USA
Full list of author information is available at the end of the article

CUDA is a programming interface developed by the company NVIDIA to facilitate general-purpose, parallel high-performance computing on multiprocessor graphics processing units (GPUs) [13]. In recent years, many scientific computing applications have been implemented on GPUs using CUDA, in many cases yielding speed-ups of several orders of magnitude [14,15]. However, to our knowledge, only a handful of publications have appeared describing GPU implementations of codes for RNA secondary structure prediction. Rizk and Lavenier described a CUDA implementation of structure prediction by free energy minimization [16]. Their work was limited to relatively short sequences (up to 120 bases) and a simplified energy model that neglects coaxial stacking. The GPU implementation was faster than the serial implementation by a factor of 10 and 17 depending on the particular hardware. More recently, Lei et al. [17] also reported a parallelized implementation of free energy minimization using CUDA. They used only a coarse-grained parallelization scheme where the minimum-free-energy structures for subsequences of a given length are calculated in parallel, but the search over structures for each subsequence is done serially. Their work was also limited to relatively short sequences (up to 221 bases). They reported speedups of up to a factor of 16. It should be noted that these parallelized implementations neither use the latest thermodynamic parameters for loops [18], nor include coaxial stacking interactions [19].

In addition, recent work demonstrates that calculations of base-pairing probabilities calculated with partition functions can provide additional useful information [20]. Structures composed of highly probable pairs are more accurate than lowest free energy structures [20,21]. The base pair probabilities provide confidence intervals for prediction of individual pairs. Base pairing probabilities can also be used to predict pseudoknots [22]. We have additionally extended this work to predictions using multiple, homologous sequences, where the same principles hold true [23-26].

In this paper, we present the calculation of base-pair probabilities for 44 sequences containing up to 10,000 bases, using an optimized and parallelized version of the "partition" code in the RNAstructure package [27]. Our test set contained both random sequences of varying lengths (up to 10,000 bases) and actual sequences of biological importance (see Table 1), the longest being the HIV-1 NL43 genome (GenBank: AF324493) containing 9709 bases.

We employed a sophisticated and accurate energy calculation that includes coaxial stacking [19]. Before attempting a parallel implementation, we first wrote an optimized serial version of the original code (the "partition" program in RNAstructure), implementing only a subset of its functionality, while improving efficiency and reducing

Table 1 Biological RNA sequences examined*

Sequence	# bases	Reference
tRNA RQ2640	75	[28]
tRNA RD0500	76	[28]
tRNA RA7680	76	[28]
tRNA RD0260	77	[28]
tRNA RR1664	77	[28]
Candida albicans 5S rRNA	114	[29]
Escherichia coli 5S rRNA	120	[30]
P546 folding domain of *Tetrahymena thermophilia* group I intron	155	[31]
Bacillus stearothermophilus SRP RNA	268	[32]
3′ UTR of *Bombyx mori* R2 element with flanking vector sequence	300	[33]
Tetrahymena thermophilia group I intron	433	[30]
Saccharomyces cerevisiae A5 group II intron	632	[34]
Escherichia coli small subunit rRNA	1542	[30]
Escherichia coli large subunit rRNA	2904	[29]
human ICAM-1 mRNA	2986	[35]
HIV-1 NL43 genome (GenBank: AF324493)	9709	[36]

*28 random sequences of length 10–10,000 were also examined.

memory usage. Subsequently, we parallelized the optimized code for GPU hardware, using CUDA. Here we made use of a fine-grained parallelization scheme, in which the calculation of the restricted partition function for each subsequence of a given length is parallelized, as well as the calculation of the restricted partition function for each subsequence. We found that this fine-grained parallelization resulted in greater speedups than a simpler coarse-grained-only parallelization (up to factors of ∼60 compared to the optimized serial version and ∼116 compared to the original code).

Implementation
Calculating base pair probabilities
The probability $p_{i,j}$ of a canonical pair between bases i and j related to the standard free energy change $\Delta G_{i,j}^0$ of the restricted ensemble of structures containing the pair, and the standard free energy change ΔG^0 of the unrestricted ensemble of all possible structures, both relative to the state in which all bases are unpaired:

$$p_{i,j} = \frac{w\left(\Delta G_{i,j}^0\right)}{w\left(\Delta G^0\right)} \tag{1}$$

where

$$w(g) \equiv \exp\left(-g/RT\right) \tag{2}$$

is the Boltzmann weight corresponding to standard free energy change g at temperature T.

The standard free energy changes $\Delta G^0_{i,j}$ and ΔG^0 are estimated using the Turner nearest-neighbor rules [11,18,37], and pseudoknots are excluded. In that case, $\Delta G^0_{i,j}$ depends on two independent contributions, one for the interior fragment containing the pair i, j and bases in between (but excluding bases from the 5' end to $i - 1$ and from $j + 1$ to the 3' end), and one for the exterior fragment containing the pair and bases from the two ends (but excluding bases from $i + 1$ to $j - 1$). We define $V_{i,j}$ to be the relative standard free energy change for the interior fragment in the case that $i < j$ and for the exterior fragment otherwise, following the convention used in the mfold prediction software [38]. In that case,

$$\Delta G^0_{i,j} = -RT \log \left[\exp \left(-V_{i,j}/RT \right) + \exp \left(-V_{j,i}/RT \right) \right] \tag{3}$$

or more succinctly

$$w \left(\Delta G^0_{i,j} \right) = w \left(V_{i,j} \right) + w \left(V_{j,i} \right) \tag{4}$$

The standard free energy changes $V_{i,j}$ are calculated using the following set of recursions:

$$w \left(V_{i,j} \right) = \begin{cases} Q^{\text{hairpin}}_{i,j} + Q^{\text{stack}}_{i,j} + Q^{\text{internal}}_{i,j} + Q^{\text{multibranch}}_{i,j} & \text{for } i < j \\[2mm] Q^{\text{exterior}}_{i,j} + Q^{\text{stack}}_{i,j} + Q^{\text{internal}}_{i,j} + Q^{\text{multibranch}}_{i,j} & \text{for } i > j \end{cases} \tag{5}$$

where

$$Q^{\text{hairpin}}_{i,j} = w \left(\Delta G^{\text{hairpin}}_{i,j} \right) \tag{6}$$

$$Q^{\text{stack}}_{i,j} = w \left(V_{i+1,j-1} + \Delta G^{\text{stack}}_{i,j,i+1,j-1} \right) \tag{7}$$

$$Q^{\text{internal}}_{i,j} = \sum_{i<i'<j'<j} w \left(V_{i',j'} + \Delta G^{\text{internal}}_{i,j,i',j'} \right) \tag{8}$$

$$\begin{aligned} Q^{\text{multibranch}}_{i,j} = {}& w \left(W^{\text{MB}}_{i+1,j-1} + a + c \right) + w \left(W^{\text{MB}}_{i+2,j-1} + \Delta G^{3'\text{dangle}}_{i,j,i+1} + a + b + c \right) + \\ & w \left(W^{\text{MB}}_{i+1,j-2} + \Delta G^{5'\text{dangle}}_{i,j,j-1} + a + b + c \right) + \\ & w \left(W^{\text{MB}}_{i+2,j-2} + \Delta G^{\text{terminal mismatch}}_{i,j,i+1,j-1} + a + 2b + c \right) + \\ & \sum_{i<k<j} w \left(V_{i+1,k} + Y_{k+1,j-1} + \Delta G^{\text{coaxial flush}}_{j,i,i+1,k} + a + 2c \right) + \\ & \sum_{i<k<j} w \left(V_{i+2,k} + Y_{k+2,j-1} + \Delta G^{\text{coaxial mismatch(2)}}_{j,i,i+2,k} + a + 2b + 2c \right) + \\ & \sum_{i<k<j} w \left(V_{i+2,k} + Y_{k+1,j-2} + \Delta G^{\text{coaxial mismatch(1)}}_{j,i,i+2,k} + a + 2b + 2c \right) + \\ & \sum_{i<k<j} w \left(V_{k,j-1} + Y_{i+1,k-1} + \Delta G^{\text{coaxial flush}}_{k,j-1,j,i} + a + 2c \right) + \\ & \sum_{i<k<j} w \left(V_{k,j-2} + Y_{i+1,k-2} + \Delta G^{\text{coaxial mismatch(1)}}_{k,j-2,j,i} + a + 2b + 2c \right) + \\ & \sum_{i<k<j} w \left(V_{k,j-2} + Y_{i+2,k-1} + \Delta G^{\text{coaxial mismatch(2)}}_{k,j-2,j,i} + a + 2b + 2c \right) \end{aligned} \tag{9}$$

$$Q_{i,j}^{\text{exterior}} = w\left(W_{i+1}^{3'} + W_{j-1}^{5'}\right) + w\left(W_{i+2}^{3'} + W_{j-1}^{5'} + \Delta G_{i,j,i+1}^{3'\text{dangle}} +\right) +$$

$$w\left(W_{i+1}^{3'} + W_{j-2}^{5'} + \Delta G_{i,j,j-1}^{5'\text{dangle}} +\right) + w\left(W_{i+2}^{3'} + W_{j-2}^{5'} + \Delta G_{i,j,i+1,j-1}^{\text{terminal mismatch}}\right) +$$

$$\sum_{i<k<j} w\left(V_{i+1,k} + W_{k+1}^{3'} + W_{j-1}^{5'} + \Delta G_{j,i,i+1,k}^{\text{coaxial flush}}\right) +$$

$$\sum_{i<k<j} w\left(V_{i+2,k} + W_{k+2}^{3'} + W_{j-1}^{5'} + \Delta G_{j,i,i+2,k}^{\text{coaxial mismatch}(2)}\right) +$$

$$\sum_{i<k<j} w\left(V_{i+2,k} + W_{k+1}^{3'} + W_{j-2}^{5'} + \Delta G_{j,i,i+2,k}^{\text{coaxial mismatch}(1)}\right) +$$

$$\sum_{i<k<j} w\left(V_{k,j-1} + W_{i+1}^{3'} + W_{k-1}^{5'} + \Delta G_{k,j-1,j,i}^{\text{coaxial flush}}\right) +$$

$$\sum_{i<k<j} w\left(V_{k,j-2} + W_{i+1}^{3'} + W_{k-2}^{5'} + \Delta G_{k,j-2,j,i}^{\text{coaxial mismatch}(1)}\right) +$$

$$\sum_{i<k<j} w\left(V_{k,j-2} + W_{i+2}^{3'} + W_{k-1}^{5'} + \Delta G_{k,j-2,j,i}^{\text{coaxial mismatch}(2)}\right) \tag{10}$$

$$w\left(W_{i,j}^{\text{L}}\right) = w\left(W_{i+1,j}^{\text{L}} + b\right) + w\left(V_{i,j} + c\right) + w\left(V_{i,j-1} + \Delta G_{j-1,i,j}^{3'\text{dangle}} + b + c\right) +$$

$$w\left(V_{i+1,j} + \Delta G_{j,i+1,i}^{5'\text{dangle}} + b + c\right) + w\left(V_{i+1,j-1} + \Delta G_{j-1,i+1,j,i}^{\text{terminal mismatch}} + 2b + c\right) \tag{11}$$

$$w\left(W_{i,j}^{\text{Q}}\right) = w\left(V_{i,j}\right) + w\left(V_{i,j-1} + \Delta G_{j-1,i,j}^{3'\text{dangle}}\right) + w\left(V_{i+1,j} + \Delta G_{j,i+1,i}^{5'\text{dangle}}\right) +$$

$$w\left(V_{i+1,j-1} + \Delta G_{j-1,i+1,j,i}^{\text{terminal mismatch}}\right) \tag{12}$$

$$w\left(W_{i,j}\right) = w\left(W_{i,j-1} + b\right) + w\left(W_{i,j}^{\text{L}}\right) \tag{13}$$

$$w\left(W_{i,j}^{\text{coax}}\right) = \sum_{i<k<j} w\left(V_{i,k} + V_{k+1,j} + \Delta G_{i,k,k+1,j}^{\text{coaxial flush}} + 2c\right) +$$

$$\sum_{i<k<j} w\left(V_{i+1,k} + V_{k+2,j} + \Delta G_{i+1,k,kj+2,j}^{\text{coaxial mismatch}(1)} + 2b + 2c\right) +$$

$$\sum_{i<k<j} w\left(V_{i,k} + V_{k+2,j-1} + \Delta G_{i,k,k+2,j-1}^{\text{coaxial mismatch}(2)} + 2b + 2c\right) \tag{14}$$

$$w\left(Z_{i,j}\right) = w\left(W_{i,j}^{\text{coax}}\right) + w\left(V_{i,j} + c\right) + w\left(V_{i,j-1} + \Delta G_{j-1,i,j}^{3'\text{dangle}} + b + c\right) +$$

$$w\left(V_{i+1,j} + \Delta G_{j,i+1,i}^{5'\text{dangle}} + b + c\right) + w\left(V_{i+1,j-1} + \Delta G_{j-1,i+1,j,i}^{\text{terminal mismatch}} + 2b + c\right) \tag{15}$$

$$w\left(W_{i,j}^{\text{MBL}}\right) = w\left(W_{i+1,j}^{\text{MBL}}\right) + w\left(W_{i,j}^{\text{coax}}\right) + \sum_{i<k<j} w\left(Z_{i,k} + Y_{k+1,j}^{\text{L}}\right) \tag{16}$$

$$w\left(W_{i,j}^{\text{MB}}\right) = w\left(W_{i,j-1}^{\text{MB}} + b\right) + w\left(W_{i,j}^{\text{MBL}}\right) \tag{17}$$

$$w\left(Y_{i,j}\right) = w\left(W_{i,j}\right) + w\left(W_{i,j}^{\text{MB}}\right) \tag{18}$$

$$w\left(Y_{i,j}^{\text{L}}\right) = w\left(W_{i,j}^{\text{L}}\right) + w\left(W_{i,j}^{\text{MBL}}\right) \tag{19}$$

$$w\left(W_{i}^{5'}\right) = w\left(W_{i-1}^{5'}\right) + \sum_{j<i} w\left(W_{j-1}^{5'} + W_{j,i}^{\text{Q}}\right) \tag{20}$$

$$w\left(W_{i}^{3'}\right) = w\left(W_{i+1}^{3'}\right) + \sum_{j>i} w\left(W_{j+1}^{3'} + W_{i,j}^{\text{Q}}\right) \tag{21}$$

These recursions are slightly different from—but equivalent to—those presented in reference [20] and used in the previous code. It should be noted that there was an error in equation 15 of reference [20]: in the second line, $\mathrm{WMBL}(k+1,j)$ should be replaced by $[\mathrm{WMBL}(k+1,j) + \mathrm{WL}(k+1,j)]$.

The quantities V, W, W^{MB}, W^{L}, W^{coax}, $W^{5'}$, and $W^{3'}$ are simply $-RT$ times the logarithms of the quantities in reference [20]. In addition, four new arrays are introduced:

1. W^{Q} is the standard free energy change corresponding to the sum of terms on the right hand side of equation 11 of reference [20] not including the scaling by $\mathrm{W5}(k)$.
2. Elements of Y are $-RT$ times the logarithm of the sum of the Boltzmann weights of corresponding elements of W and W^{MB}.
3. Elements of Y^{L} are $-RT$ times the logarithm of the sum of the Boltzmann weights of corresponding elements of W^{L} and W^{MBL}.
4. Elements of Z are $-RT$ times the logarithm of the sum of the Boltzmann weights of corresponding elements of W^{L} (except for the term depending on the next-smallest fragment) and W^{coax}.

Reorganizing the recursions in this way might appear to use more memory because of the additional arrays. In fact, the modified version requires less memory, because several of the arrays do not need to be stored in their entirety. Specifically, using the modified recursions, storage is only required for two diagonals of W, W^{L}, and W^{MBL}; for five diagonals of W^{MB}; and for a half-triangle of W^{Q}. Reducing memory usage is important as the size of the full arrays scales as $O(N^2)$ and the available GPU memory on our hardware was limited to ∼2.5 GB. The modified recursions use four full $N \times N$ arrays and one half-triangle, rather than the six full $N \times N$ arrays used in the original recursions, and therefore reduce memory usage by about 25%. In addition, the calculation of $W^{5'}$ and $W^{3'}$ is simplified (compare equations 20 and 21 above with equation 11 of reference [20]).

As in the previous work, the various ΔG parameters above are from the Turner nearest-neighbor rules [11], while a, b, and c are from the following estimate of the standard free energy change for multibranch loop initiation:

$$\Delta G^0 = a + bn + ch \qquad (22)$$

Here n is the number of unpaired nucleotides and h is the number of branching helices [21]. By convention, the size of internal loops is limited to thirty unpaired nucleotides, so the number of terms in equation 8 and the overall computational expense scales as $O(N^3)$ where N is the size of the sequence.

The largest difference in the new implementation is that logarithms of probabilities and partition functions (i.e., standard free energy changes) are used rather than probabilities themselves, which is convenient when working in single precision in order to avoid overflow or underflow errors. This requires that exponentials and logarithms are calculated at each step of the calculation where sums are performed. This approach is a departure from the previous implementation, which used scaling factors. Having to compute logarithms and exponentials does entail some additional computational expense, but this does not appear to be exhorbitant on the GPU, because optimized intrinsic mathematical functions are used. (i.e., the code was compiled with the NVIDIA compiler using the `-use_fast_math` option). The function required for the sum of two free energies a and b (expressed in units such that $RT = 1$) is

$$f(a, b) = -\log\left(e^{-a} + e^{-b}\right) \qquad (23)$$

We calculated this in the following way:

$$f(a, b) = \begin{cases} a - \log\left(1 + e^{a-b}\right) & a < b \\ b - \log\left(1 + e^{b-a}\right) & a > b \\ a - \log(2) & a = b \end{cases} \qquad (24)$$

This was done for two reasons: it requires at most a single call to `exp`, rather than two; and it can make use of the `log1p` function from the standard math library, which calculates $\log(1 + x)$ accurately even for small x. This is important because often e^{-a} and e^{-b} will differ by several orders of magnitude, and simply adding them and then taking the logarithm can lead to significant roundoff error.

In order to determine how much additional computational overhead was imposed by the calculation of `exp` and `log1p` we performed a comparison with an artificial reference calculation, which was identical except that calls to these functions were omitted. We found that for a 1,000-mer, the actual GPU calculation is only ∼20% more expensive than this reference calculation.

For a serial calculation on the CPU, there is a larger performance hit; the actual calculation is about a factor of two more expensive than the reference without `exp` or `log1p`. However, it should be noted that this is not the entire story, because overall, the new optimized serial code, which uses logarithms, is still faster than the original code, which does not. Running the calculation in log space results in simplifications such as not requiring checking for overflow and not having to multiply by scaling factors, which reduces computational expense.

Parallelization of the partition function calculation using CUDA

In the CUDA programming model, overall execution of the program is still governed by the CPU. Compute-intensive portions of the program are then delegated to subroutines executed by the GPU, or kernels. In general, the GPU has its own memory, so data must be copied to and from the GPU before and after kernel execution. Many copies of a kernel, or threads, run in parallel, each of which belongs to a block. During kernel execution, threads belonging to the same block can share data and synchronize, whereas threads belonging to different blocks cannot. A program can contain many kernels, which can execute either serially or in parallel [13].

The algorithm for calculating partition functions is recursive: partition functions for larger fragments depend on those for smaller fragments. As such, the overall calculation proceeds serially, in order of fragment size. We used two levels of parallelization, a block level and a thread level. Calculations for all fragments of a given size may be done in parallel, with no communication. This was implemented in CUDA at the level of blocks of threads. The partition function for a given fragment depends on sums, with the number of terms on the order of the fragment size (e.g., equation 9). These sums were parallelized at the level of threads within a block, since calculating a sum in parallel relies on communication between the threads. In our experience, a greater speedup was obtained from this "inner loop" parallelization, even though it requires more communication between threads. Most likely, this is because optimal efficiency on GPU hardware is obtained when identical mathematical operations are performed in lockstep on different data [13]. We stress that these two different levels of parallelization are not mutually exclusive and optimal performance was obtained from including both. A separate block of threads was run for each fragment, while 256 threads were run within a block. The number of threads per block was chosen by trial and error and was optimal for our hardware (the simple sum reduction scheme we chose requires it to be a power of two). In our code, this value is set at compile time (but this is not required by CUDA—it could be set at run time if desired).

Results and discussion

Accuracy

Peak floating point performance for NVIDIA Tesla GPUs are faster by a factor of two when working in single compared with working in double precision (www.nvidia.com), but single precision introduces greater roundoff error. In order to examine accuracy, we calculated base-pair probabilities for the same sequences using both the parallel CUDA/GPU implementation in single precision, and the serial implementation in double precision. We also calculated probabilities in double precision using a set of nearest-neighbor parameters slightly modified by adding a random variate chosen from a Gaussian distribution with mean 0 and standard deviation 0.01 kcal/mol, which is comparable to or smaller than their experimental uncertainty (0.1 kcal/mol for parameters describing helical stacking and 0.5 kcal/mol for parameters describing loops [18,37]).

Figure 1 shows the root-mean-square deviation (RMSD) between base-pair probabilities calculated using double precision and either single precision, or the modified parameters. The RMSD of the calculation using modified parameters decreases as the size of the sequences increases (because the fraction of pairs with very small probabilities increases). In contrast, the roundoff error due to working in single precision increases as the cube of the number of bases in the sequence (i.e., the number of operations involved in the calculation). However, it remains relatively small for sequences of up to 10,000 bases, and negligible compared with the differences resulting from the modified parameters. Our conclusion is that working in single precision does not introduce unacceptable roundoff error for RNA secondary structure prediction for sequences of this size and most likely substantially larger.

We also used the calculated base pair probabilities to determine a single consensus structure for each sequences, using the ProbKnot algorithm [22]. In this
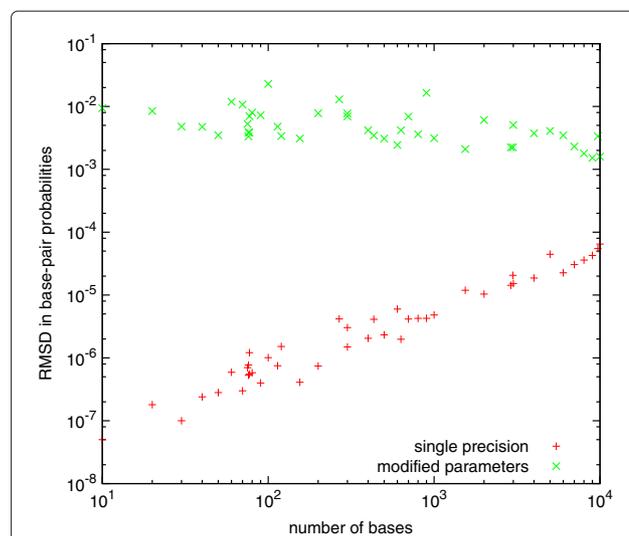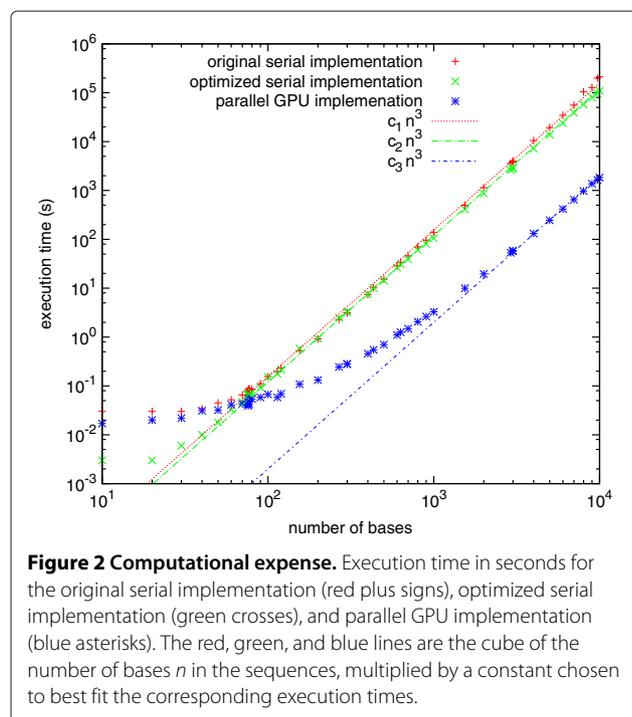

Figure 1 Accuracy of base-pair probabilities. Root-mean-square deviation in base-pair probabilities calculated using the CUDA single-precision implementation (red plus signs) and the serial double-precision implementation with slightly modified parameters (green crosses), compared with the serial double-precision implementation using the original parameters.

case, working in single precision led to differences with double precision only for one sequence (a random sequence of 6000 bases) out of the 44 we examined, and these were very small (only three bases out of the 6000 were matched with a different partner). Working with the modified parameters led to larger discrepancies although these were still fairly small (for sequences containing more than 100 bases, at most 6% of bases were matched with different partners). This is consistent with a previous report that predictions using base pair probabilities are significantly less sensitive to errors in thermodynamic parameters than using only lowest free energy structures [39].

Computational expense

We compared overall execution time for the original serial code developed in our laboratory, the optimized serial code, and the parallel CUDA/GPU implementation. Calculations were performed for sequences containing from 10 to 10,000 bases, on compute nodes containing dual hex-core Intel Xeon 2.67 GHz CPUs and dual NVIDIA Tesla M2050 GPUs, each of which contains 448 multiprocessor cores (only a single GPU was used). Figure 2 shows the execution time (from the user time reported by the UNIX "time" command) of the original and optimized serial codes as well as the parallel CUDA/GPU implementation. The execution time for all codes scales as the cube of the sequence length for large sequences. The CUDA version was able to calculate base-pair probabilities for

Figure 2 Computational expense. Execution time in seconds for the original serial implementation (red plus signs), optimized serial implementation (green crosses), and parallel GPU implementation (blue asterisks). The red, green, and blue lines are the cube of the number of bases n in the sequences, multiplied by a constant chosen to best fit the corresponding execution times.

the sequence for the full HIV-1 NL43 genome (9709 nucleotides) in 27 minutes. We note that there is a small overhead (a few hundredths of a second) involved in running calculations with either the original serial code or the CUDA code, which is not present for the optimized serial code. This overhead has different origins: for the original code, it is probably due to reading the parameter files from disk, while for the CUDA code, it is most likely due to copying parameters and other data between the GPU and CPU.

Conclusions

In this work, we introduced a modified set of recursions for calculating RNA secondary structure partition functions and base-pairing probabilities using a dynamic programming algorithm, and implemented these in parallel using the CUDA framework for multiprocessor GPUs. For large sequences, the GPU implementation reduces execution time by a factor of close to 60 compared with an optimized serial implementation, and by a factor of 116 compared with the original code. It is clear from our work that using GPUs can greatly accelerate computation of RNA secondary structure partition functions, allowing calculation of base-pair probabilities for large sequences in a reasonable amount of time, with a negligible compromise in accuracy due to working in single precision. It is expected that parallelization using CUDA should be applicable to other implementations of dynamic programming algorithms [12] besides ours, and result in similar speedups.

Two levels of parallelization were implemented. Calculations for all fragments of a given size were done in parallel, with no communication between threads. This was implemented in CUDA at the level of blocks of threads. In addition, the sums contributing to the partition function for a given fragment were calculated in parallel, with communication required between threads. These sums were parallelized at the level of thread within a block. We found that this "inner loop" parallelization resulted in a significantly greater speedup than the "outer loop" parallelization alone.

Availability and requirements

- Project name: partition-cuda; part of RNAstructure, version 5.5 and later
- Project home page: http://rna.urmc.rochester.edu/RNAstructure.html
- Operating system(s): Unix
- Programming languages: C and CUDA
- Other requirements: CUDA compiler, available from http://www.nvidia.com/object/cuda_home_new.html
- License: GNU GPL
- Any restrictions to use by non-academics: None.

Author details
[1]Center for Integrated Research Computing, Taylor Hall, University of Rochester, Rochester, NY, 14627, USA. [2]Department of Biochemistry & Biophysics and Center for RNA Biology, University of Rochester Medical Center, 601, Elmwood Avenue Box 712, Rochester, NY, 14642, USA.

References
1. Doudna JA, Cech TR: **The chemical repertoire of natural ribozymes.** *Nature* 2002, **418:**222.
2. Bachellerie JP, Cavaille J, Huttenhofer A: **The expanding snoRNA world.** *Biochimie* 2002, **84:**775.
3. Walter P, Blobel G: **Signal recognition particle contains a 7S RNA essential for protein translocation across the endoplasmic reticulum.** *Nature* 1982, **299:**691.
4. Dykxhoorn DM, Novina CD, Sharp PA: **Killing the messenger: Short RNAs that silence gene expression.** *Nat Rev Mol Cell Biol* 2003, **4:**457.
5. Wu L, Belasco JG: **Let me count the ways: Mechanisms of gene regulation by miRNAs and siRNAs.** *Mol Cell* 2008, **29:**1.
6. Tinoco I Jr, Bustamante C: **How RNA folds.** *J Mol Biol* 1999, **293:**271.
7. Nawrocki EP, Kolbe DL, Eddy SR: **Infernal 1.0: Inference of RNA alignments.** *Bioinformatics* 2009, **25:**1335.
8. Li X, Quon G, Lipshitz HD, Morris Q: **Predicting *in vivo* binding sites of RNA-binding proteins using mRNA secondary structure.** *RNA* 2010, **16:**1096.
9. Lu ZJ, Mathews DH: **Efficient siRNA selectrion using hybridization thermodynamics.** *Nucleic Acids Res* 2007, **36:**640.
10. Tafer H, Ameres SL, Obernosterer G, Gebeshuber CA, Schroeder R, Martinez J, Hofacker IL: **The impact of target site accessibility on the deseign of effective siRNAs.** *Nat Biotechnol* 2008, **26:**578.
11. Turner DH, Mathews DH: **NNDB: The nearest neighbor parameter database for predicting stability of nucleic acid secondary structure.** *Nucleic Acids Res* 2010, **38:**D280.
12. Lorenz R, Bernhart SH, zu Siederdissen CH, Tafer H, Flamm C, Stadler PF, Hofacker IL: **ViennaRNA package 2.0.** *Algorithms Mol Biol* 2011, **6:**26.
13. Sanders J, Kandrot E: *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Boston: Addison-Wesley; 2011.
14. Farber RM: **Topical perspective on massive threading and parallelism.** *J Mol Graph Model* 2011, **30:**82.
15. Gotz AW, Williamson MJ, Xu D, Poole D, Le Grand S, Walker RC: **Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. Generalized Born.** *J Chem Theory Comput* 2012, **8:**1542.
16. Rizk G, Lavenier D: **GPU accelerated RNA folding algorithm.** *Lect Notes Comput Sci* 2009, **5544:**1004.
17. Lei G, Dou Y, Wan W, Xia F, Li R, Ma M, Zou D: **CPU-GPU hybrid accelerating the Zuker algorithm for RNA secondary structure prediction applications.** *BMC Genomics* 2012, **13**(Suppl 1):S14.
18. Mathews DH, Disney MD, Childs JL, Schroeder SJ, Zuker M, Turner DH: **Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure.** *Proc Natl Acad Sci USA* 2004, **101:**7287.
19. Kim J, Walter AE, Turner DH: **Thermodynamics of coaxially stacked helices with GA and CC mismatches.** *Biochemistry* 1996, **35:**13753.
20. Mathews DH: **Using an RNA secondary structure partition function to determine confidence in base pairs predicted by free energy minimization.** *RNA* 2004, **10:**1178.
21. Lu ZJ, Gloor JW, Mathews DH: **Improved RNA secondary structure prediction by maximizing expected pair accuracy.** *RNA* 2009, **15:**1805.
22. Bellaousov S, Mathews DH: **ProbKnot: Fast prediction of RNA secondary structure including pseudoknots.** *RNA* 2010, **16:**1870.
23. Harmanci AO, Sharma G, Mathews DH: **PARTS: Probabilistic alignment for RNA joint secondary structure predcition.** *Nucleic Acids Res* 2008, **36:**2406.
24. Harmanci AO, Sharma G, Mathews DH: **Stochastic sampling of the RNA structural alignment space.** *Nucleic Acids Res* 2009, **37:**4063.
25. Harmanci AO, Sharma G, Mathews DH: **Efficient pairwise RNA structure prediction using probabilistic alignment constraints in Dynalign.** *BMC Bioinformatics* 2011, **27:**626.
26. Seetin MG, Mathews DH: **TurboKnot: Rapid prediction of conserved RNA secondary structures including pseudoknots.** *Bioinformatics* 2012, **28:**792.
27. Reuter JS, Mathews DH: **RNAstructure: software for RNA secondary structure prediction and analysis.** *BMC Bioinformatics* 2010, **11:**129.
28. Sprinzl M, Vassilenko KS: **Compilation of tRNA sequences and sequences of tRNA genes.** *Nucleic Acids Res* 2005, **33:**D139.
29. Szymanski M, Barciszewska MZ, Barciszewski J, Erdmann VA: **5S ribosomal RNA database Y2K.** *Nucleic Acids Res* 2000, **28:**166.
30. Cannone JJ, Subramanian S, Schnare MN, Collett JR, D'Souza LM, Du Y, Feng B, Lin N, Madabusi LV, Muller KM, Pande N, Shang Z, Yu N, Gutell RR: **The comparaitive RNA web (CRW) site: An online databae of comparative sequence and structure information for ribosomal, intron, and other RNAs.** *BMC Bioinformatics* 2002, **3:**2.
31. Cate JH, Gooding AR, Podell E, Zhou K, Golden BL, Kundrot CE, Cech TR, Doudna JA: **Crystal structure of a group I ribozyme domain: Principles of RNA packing.** *Science* 1996, **273:**1678.
32. Larsen N, Samuelsson T, Zwieb C: **The signal recognition particle database (SRPDB).** *Nucleic Acids Res* 1998, **26:**177.
33. Mathews DH, Banerjee A R Luan, D D, Eickbush TH, Turner DH: **Secondary structure model of the RNA recognized by the reverse transcriptase from the R2 retrotransposable element.** *RNA* 1997, **3:**1.
34. Michel F, Umesono K, Ozeki H: **Comparative and functional anatomy of group II catalytic introns—a review.** *Gene* 1989, **82:**5.
35. Staunton DE, Marlin SD, Stratowa C, Dustin ML, Springer TA: **Primary structure of ICAM-1 demonstrates interaction between members of the immunoglobulin and integrin supergene families.** *Cell* 1988, **52:**925.
36. Adachi A, Gendelman HE, Koenig S, Folks T, Willey R, Rabson A, Martin MA: **Production of acquired immunodeficiency syndrome-associated retrovirus in human and nonhuman cells transfected with an infectious molecular clone.** *J Virol* 1986, **59:**284.
37. Xia T, Santa Lucia J Jr, Burkhard ME, Kierzek R, Schroeder SJ, Jaio X, Cox C, Turner DH: **Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson-Crick pairs.** *Biochemistry* 1998, **37:**14719.
38. Zuker M: **On finding all suboptimal foldings of an RNA molecule.** *Science* 1989, **244:**48.
39. Layton DM, Bundschuh R: **A statistical analysis of RNA folding algorithms through thermodynamic parameter perturbation.** *Nucleic Acids Res* 2005, **33:**519.