GPU accelerated Number Theoretic Transformation

Baleegh Ahmad*[1] and Animesh Basak Chowdhury[2]

[1]**Net ID: ba1283**
[2]**Net ID: abc586**

ABSTRACT

This is final report of parallel computing architecture project (ECE 9413) where we accelerate the number theoretic transformation using GPU. We transformed the baseline code in CUDA with certain optimizations and additionally performed two lightweight optimization: 1) Pre-computing Twiddle factors 2) Input batching. We achieve upto 10x performance speedup on NYU Greene cluster. We have open sourced our implementation at `https://gitlab.com/animeshbchowdhury/NTT_CUDA_Project`

Keywords: Number theoretic transformation (NTT), CUDA implementation

BASELINE IMPLEMENTATION

We ported the baseline implementation code to CUDA. We used `cudaLaunchCooperativeKernel` to achieve grid synchronization for every iteration of outermost loop in *NTT*. The function `inPlaceNTT_DIT()` calls the cooperative kernel. We used *nvprof* to profile our implementation. We report the following timing information for each stage of optimization:

- Application Time (*AT*) - Timing profiled in main(). This is the sum of GPU activities and API calls.
- GPU Time (*GT*) - Time taken by GPU activities.
- Kernel Time (*KT*) - Time utilized by kernel

The baseline source codes are: *baseline_ba1283* and *baseline_abc586*. We report two versions of baseline implementation:

- **Naive (baseline_ba1283)**: Here, we simply use the CPU centric code in our kernel. We used *threads per block (tpb)* = 256 and *blocks per grid* = $4096/256 = 16$. *The timing profile are:* KT= *2.2655ms,* GT*: 2.272ms and* AT*: 196.6ms.*
- ***Optimized (baseline_abc586)****: We achieve optimization by removing all inner loops. We used threads per block (tpb)* = 128 *and blocks per grid* = $4096/128 = 32$ *as it provided us better timing results. The timing profile are:* KT= *392.73us,* GT*: 398.9us and* AT*: 222.1ms. GPU Performance speedup* $\sim 6x$.

We attach relevant code snippets and *nvprof* screenshots in our detailed report.

REMOVAL OF BIT REVERSAL

We removed the bit reversal assuming it is handled at a higher level. We measured the timing profile and observed dip in timing measurements: *KT:* 376.61 us, *GT*: 382.9 us, and *AT*: 189 ms.

IMPLEMENTATION OF TWIDDLE FACTOR

Here, we identified repetitive usage of twiddle factor values inside GPU kernel, and hence we pre-compute the values and store it in an array. Array dimension: $log(n) * (n/2)$. We achieved around $\sim 7x$ performance improvement in *KT* compared to optimized baseline implementation. The timing profiles are: *KT*= 51.71us, *GT*= 76.53us and *AT*= 206.9ms.

IMPLEMENTATION OF BATCHING

Here, we encounter a specific problem with `cudaLaunchCooperativeKernel` which hits the bottleneck on total number of threads that can be run in parallel and synchronized together across blocks. We briefly describe our challenges and the approach we adopted to mitigate them and achieve high throughput.

1. We use row-major vector 'vec' for given batch size. For eg. $batchsize = 2$ means array length of 4096*2, and vectors are concatenated.

*Sorted by last name

2. We use `cudaDeviceProperty` to identify number of threads that can be run in parallel and synchronized together. Therefore, we set `tpb`=128 and `bpg`=256 (on greene, NOT hardcoded).
3. We define mini_batch_size = # of vectors can be processed in parallel. We divided entire batch into factor of mini_batch_size, and each mini-batch is processed sequentially in loop inside kernel.

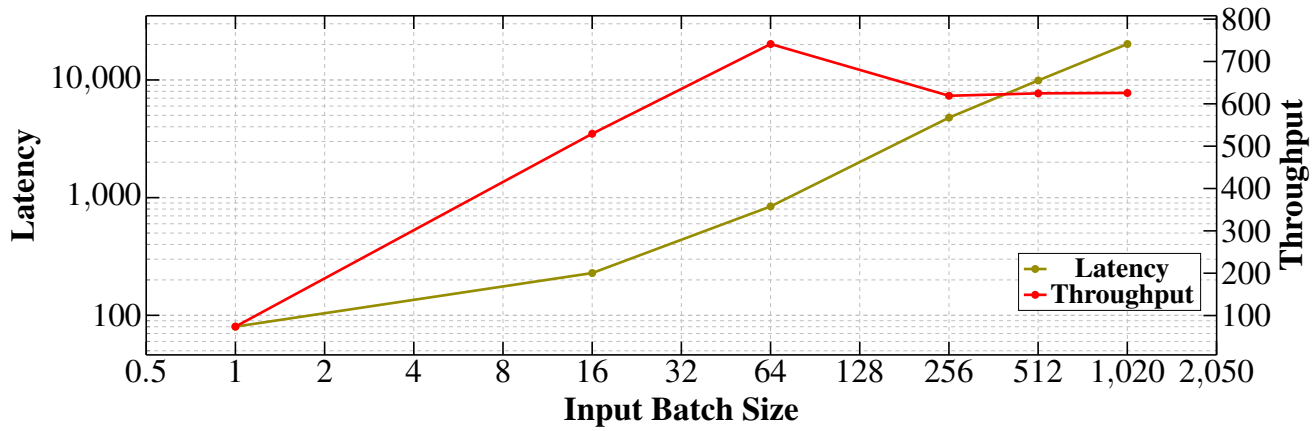
We define two metrics for measuring the performance benefits:

1. **Throughput**: $(\text{Batchsize} * \text{vector size})/KT$
2. **Latency**: It is defined as total time taken by the GPU activities (GT) to process an entire batch.

In table 1, we have reported results for various performance optimizations and speed-up obtained. The improvement column reports speed-up of each optimization compared to baseline CUDA implementation. For input batching, the improvement column show throughput speed-up with respect to input batch-size= 1.

Table 1. Performance report for CUDA implementation. Vector size (n=4096) denotes 4096 elements are processed

Configuration	BatchSize	KT (uS)	GT (uS)	Throughput (# element/uS)	Improvement
Baseline (naive)	-	2265.5	2272.0	1.81	-
Baseline (optimized)	-	392.73	398.9	10.43	5.77x
Baseline(optimized)+ NoBitReversal	-	376.61	382.9	10.88	1.04x
Baseline(optimized)+ NoBitReversal+ Twiddle factor	-	51.71	76.53	79.21	7.28x
Baseline(optimized)+ NoBitReversal+ Twiddle factor + Input Batching	1	55.5	80.16	73.8	-
	16	123.8	228.7	529.37	7.17x
	64	353.66	841.88	741.23	10.05x
	256	1693.9	4783.7	619.03	8.38x
	512	3356.9	9899.2	624.73	8.46x
	1024	6703.5	20190.4	625.69	8.55x



(a) Throughput

Figure 1. Latency and throughput obtained for various input batch sizes

WORK DIVISION

- Baseline(naive), Removal of bit reversal, Implementation of Batching and perf, Detailed report: ba1283
- Baseline(optimized) and Makefile, Implementation of Twiddle factor, Short (1-2 page) report: abc586
- Consistent exchange of ideas throughout the project