Propagate to Z Summary

S. LANTZ, T. REID, D. RILEY, P. WITTICH [CORNELL]

A. HALL, G. CERATI [FNAL]

B.GRAVELLE, B. NORRIS [U OREGON]

S. LEE [ORNL]

6/12/20

Purpose and Goals

Propagate to Z

- Simplified toy code that implements the first part of the Kalman filter algorithm (prediction step and error propagation) to propagate a given track to the next layer in Z using a single provided hit.
- This toy code is parallelized both by multithreading over the events and bundle of tracks in each event and by vectorization using SIMD operations over a single bundle of tracks
- Various methods of parallelization are compared to find the most viable tools in terms of both performance and usability.
  - CPU: OpenMP, TBB, Eigen(omp), Alpaka(omp), Kokkos, MKL
  - GPU: CUDA, OpenAcc, Eigen(CUDA), Alpaka(CUDA) , Kokkos(CUDA)

Goals

- mkFit milestone: “GPU implementation of Matriplex”
- Submit a computer science paper detailing results in a CS conference
  - Conference to be chosen when the paper is finished.

GitHub repository of the work done can be found here

Pseudocode

Setup

- Input track (6 parameters, symmetric 6x6 covariance matrix) and hit (3 parameters, symmetric 3x3 covariance matrix)
- Randomly smear track and hit parameters to create a collection of tracks for multiple events
- Default settings: 100 iterations, 100 events/iteration, 9600 tracks/event
  - All the tracks per event are split evenly into number of bundles the size of the matriplex
  - 9600 tracks per event comparable to number of tracks in 50PU event.

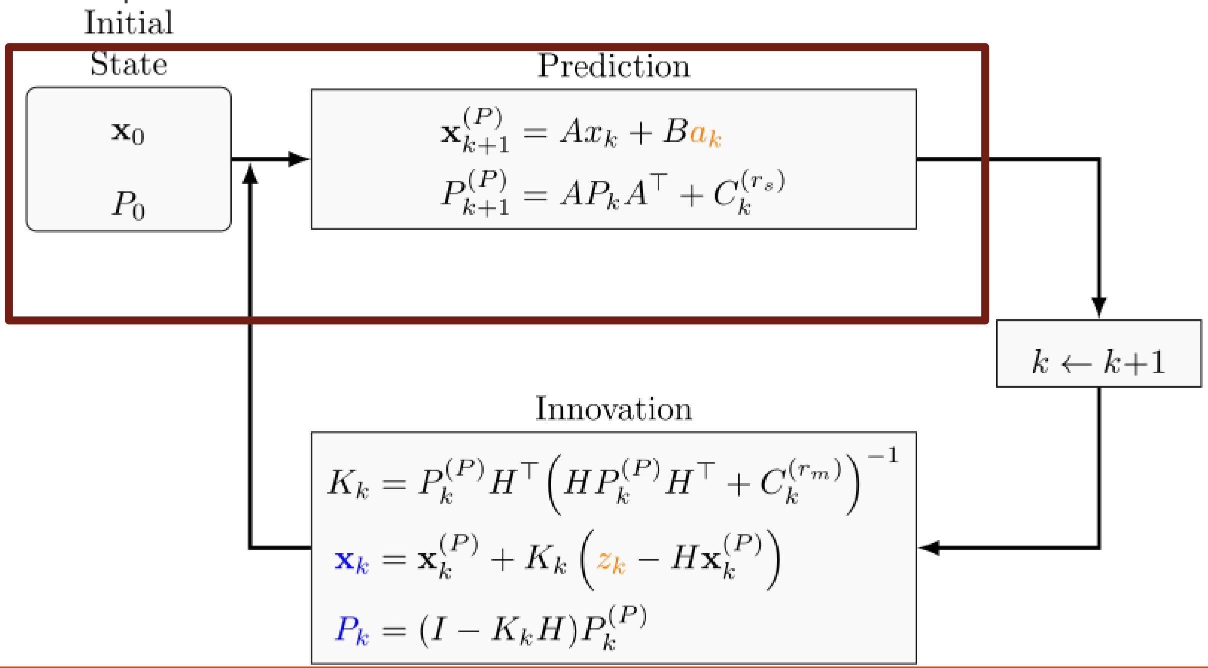
Multi-threading code

- Loop over number of iterations (no parallelization): allows for more reliable timing results
  - Loop over events in total number of events (first level of parallelization: multithreading)
    - Loop over bunches in total number of bunches per event (next level of parallelization)
      - Run propagate to z function

Propagate to Z function

Loop over tracks in the matriplex (SIMD)

- Runs “prediction” step in Kalman filter
- Sets track parameters to new values from helix prediction: (x, y, z, pt, theta, phi)
  - New hit is only used to get the distance to the new layer in z
- Makes error propagation matrix from track parameters
- Propagates covariance using error propagation following Kalman filter
  - Matrix multiplication of the errors are typically done in matriplex fashion.



CPU Implementations

OpenMP
- #pragma omp parallel for over all events
- #pragma omp simd over bunches of tracks and p2z function

Threading Building Blocks (TBB)
- Same as OpenMP but replaces #pragma omp parallel for with tbb:parallel_for(blocked range…)
- Gives slightly better performance than OpenMP due to reduced overhead with icc compiler and control over the block size

Eigen
- An open source C++ Matrix/linear algebra library
- Runs the same as OpenMP but uses Eigen's built in matrix syntax instead of matriplex
- Link to Eigen information

Intel Math Kernel Library (MKL)
- Library of optimized math routines

Alpaka
- A header only C++ library intended to support portability across accelerators
- Uses built in Accelerator backends (serial, Omp4, Threads, Omp2Threads,Omp2Blocks)
- Link to alpaka information

Kokkos
- Provides abstractions for parallelization and data management
- Link to kokkos information

GPU Implementations

CUDA
- Unified memory and explicit memory transfer (v2) versions
- Unified memory makes use of prefetching and memAdvise
- Events are divided among CUDA streams (stream per CPU thread). Blocks run over the events per stream. Threads in y run over bundles of tracks and threads in x run over the propagate function.

OpenACC
- #pragma acc parallel: runs over events and bundles of tracks (collapsed)
- #pragma acc loop vector: runs over the matriplex

Eigen
- CUDA (explicit memory transfer) using eigen matrices and matrix multiplication

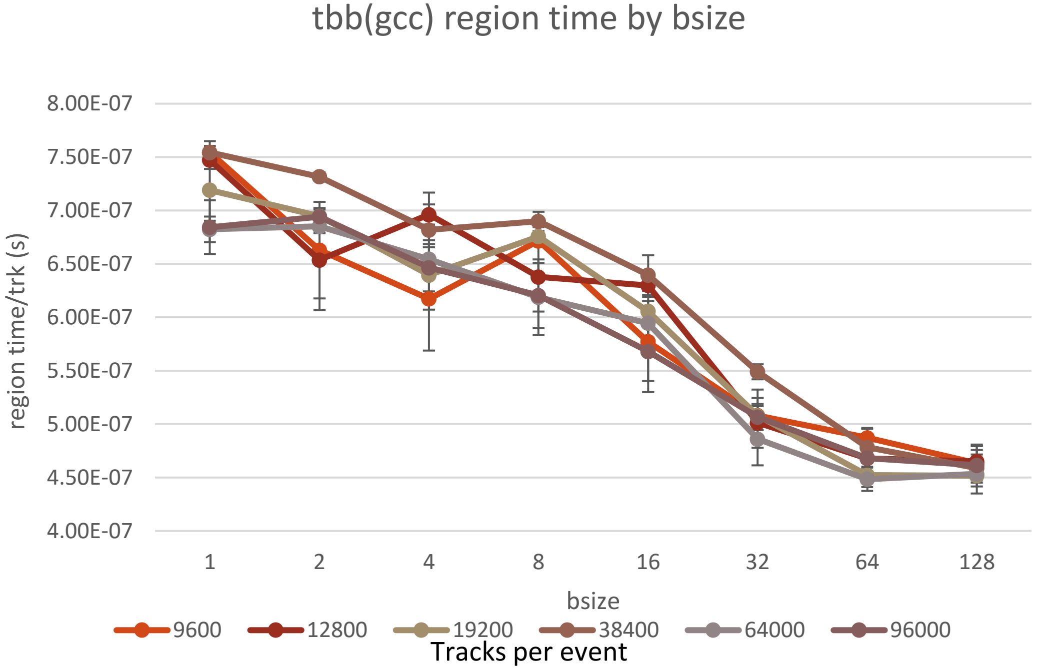
Alpaka
- Accelerator: AccGpuCudaRt
- Not yet working

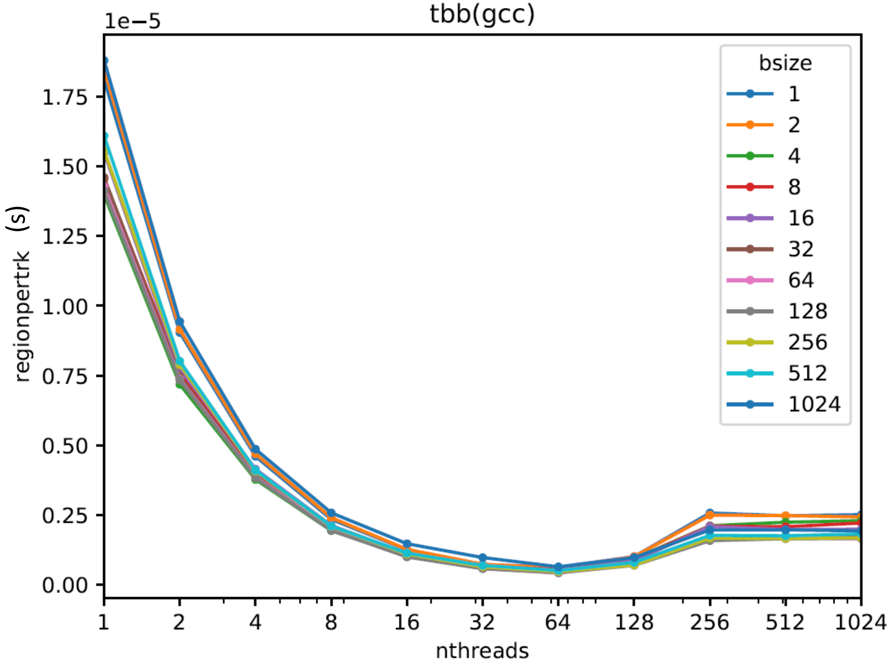
Kokkos
- Cuda as a backend

Source code parameter tuning

A few of the tunable parameters were varied for each implementation to roughly optimize the parameters [preliminary tests]

- Bsize (matriplex size): most cases see speedup with higher values (128 max value checked, left plot). Does not depend on number of threads (colors on right plot)
- Total number of tracks: no noticeable difference on CPU (9600 tracks per event by default, colors on left plot)
- Total number of threads: most cases are fastest around 64 threads (equal to number of cores, right plot)

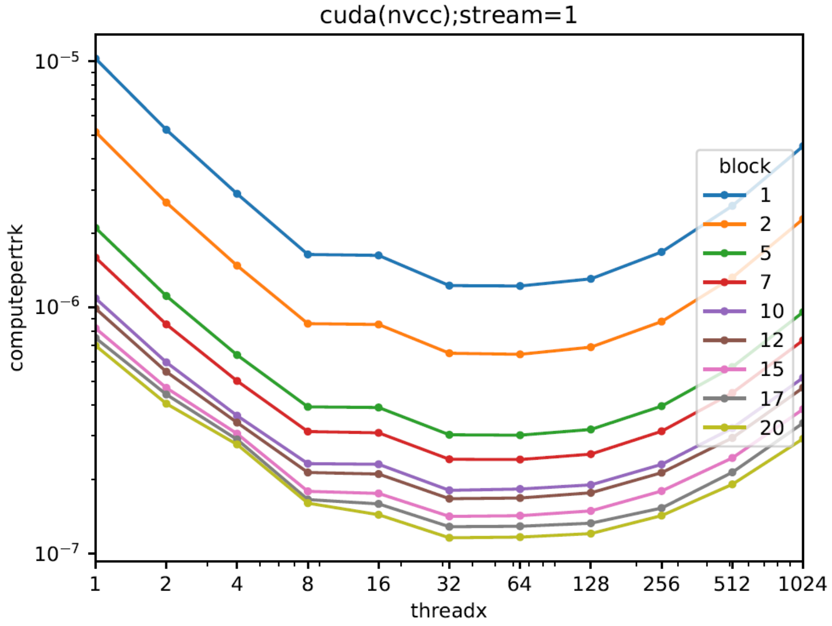


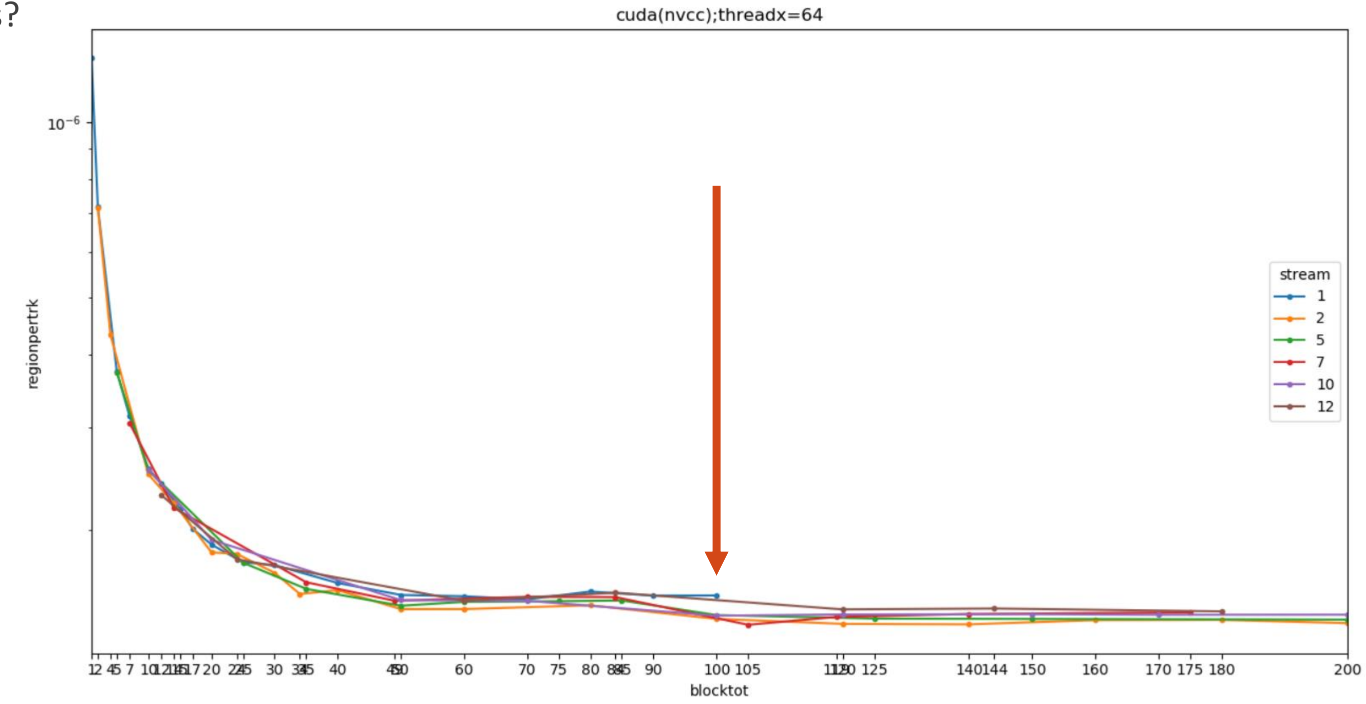


CUDA parameter tuning tests

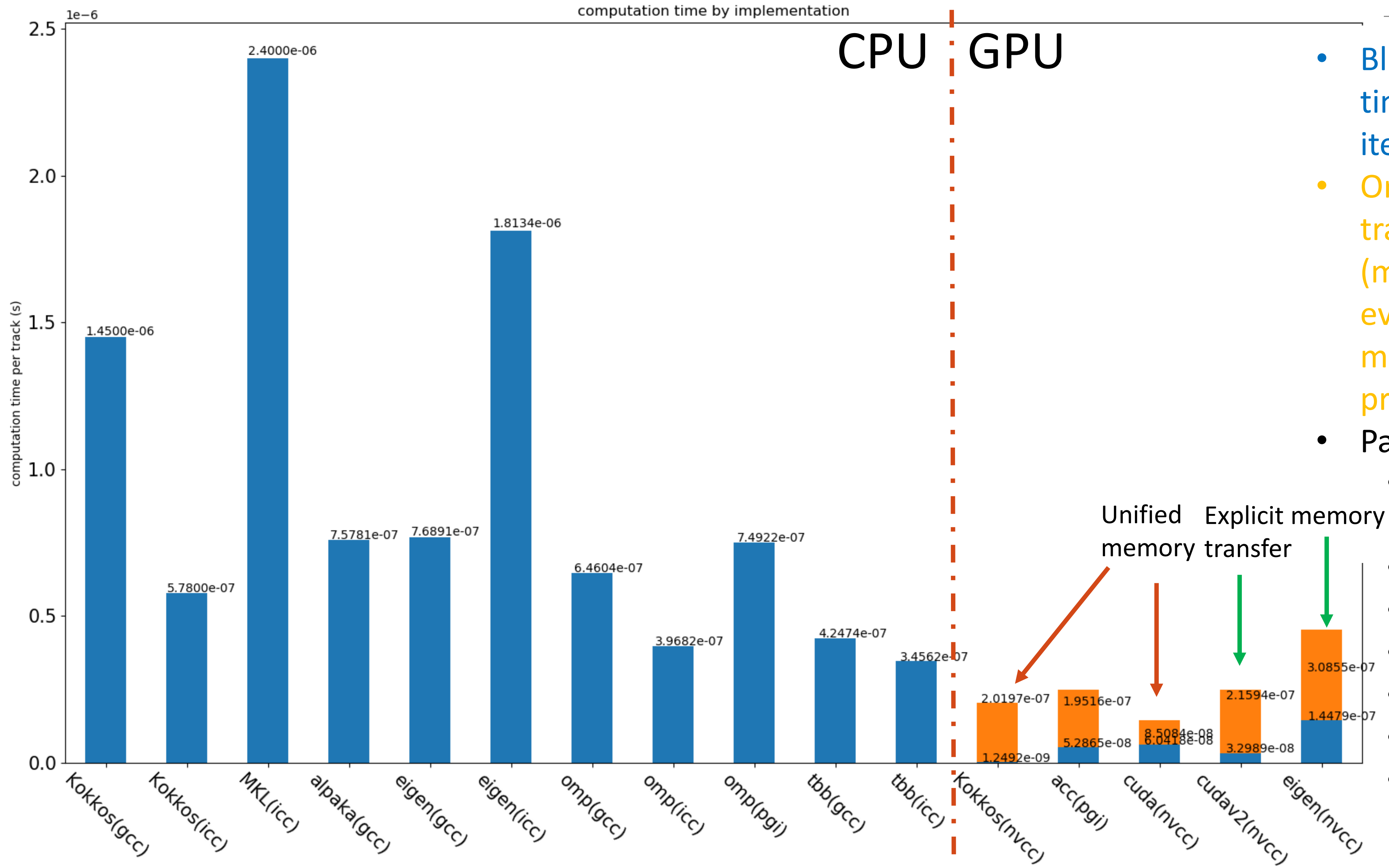
More tuning tests were run for the CUDA implementation to find the optimal number of streams, blocks, and threads (in x and y) [preliminary tests].

- Fixing the total number of threads (threadsx*threadsy) to the maximum allowed number for each block (1024) gives the best results (not shown)
- Giving the threads in x a full warp (32 threads) then gives the best results (left plot)
- No more speedup past ~100 total blocks (7 streams, 15 blocks best performing, right plot)
  - Due to 100 events or 80 streaming multiprocessors?





Performance Comparison (20 run average)



- Blue: Computation time (time to run all iterations/ # tracks)
- Orange: memory transfer time (measured using CUDA events over the memory transfer/ prefetching)
- Parameters use
  - Matriplex size: 128
  - Threads: 64
  - Events: 100
  - Tracks: 9600
  - Cuda streams: 7
  - Cuda blocks: 15
  - Cuda threads (32,32)

Conclusions

We have 16 different implementations of the propagate to z toy code

- The CUDA version using unified memory seems to give the best performance
- So far, the GPU versions seem to generally outperform the CPU versions despite the memory transfer time
  - This speedup is expected to decrease as the code becomes more complex
- Satisfies the GPU matriplex implementation NSF milestone

Performance Comparisons

	fullname	events	gpu	compute	computepertrk	i	mem	mempertrk	region	regionpertrk	setup	setuppertrk
0	Kokkos(gcc)	960000	False	1.392000	1.450000e-06	0.0	0.000000	0.000000e+00	1.392000	1.450000e-06	0.000000	0.000000e+00
1	Kokkos(icc)	960000	False	0.554880	5.780000e-07	0.0	0.000000	0.000000e+00	0.554880	5.780000e-07	0.000000	0.000000e+00
3	MKL(icc)	960000	False	2.304000	2.400000e-06	0.0	0.000000	0.000000e+00	2.304000	2.400000e-06	0.000000	0.000000e+00
5	alpaka(gcc)	960000	False	0.727500	7.578125e-07	9.5	0.000000	0.000000e+00	0.727500	7.578125e-07	1.296500	1.350521e-06
8	eigen(gcc)	960000	False	0.738150	7.689063e-07	9.5	0.000000	0.000000e+00	0.738150	7.689062e-07	1.247300	1.299271e-06
9	eigen(icc)	960000	False	1.740850	1.813385e-06	9.5	0.000000	0.000000e+00	1.740850	1.813385e-06	0.923200	9.616667e-07
11	omp(gcc)	960000	False	0.620200	6.460417e-07	9.5	0.000000	0.000000e+00	0.620200	6.460417e-07	1.294700	1.348646e-06
12	omp(icc)	960000	False	0.380950	3.968229e-07	9.5	0.000000	0.000000e+00	0.380950	3.968229e-07	0.882150	9.189063e-07
13	omp(pgi)	960000	False	0.719250	7.492187e-07	9.5	0.000000	0.000000e+00	0.719250	7.492188e-07	0.999000	1.040625e-06
14	tbb(gcc)	960000	False	0.407750	4.247396e-07	9.5	0.000000	0.000000e+00	0.407750	4.247396e-07	1.299900	1.354062e-06
15	tbb(icc)	960000	False	0.331800	3.456250e-07	9.5	0.000000	0.000000e+00	0.331800	3.456250e-07	0.897600	9.350000e-07
2	Kokkos(nvcc)	960000	True	0.001199	1.249154e-09	0.0	0.193890	2.019683e-07	0.195089	2.032175e-07	14.024739	1.460910e-05
4	acc(pgi)	960000	True	0.050750	5.286458e-08	9.5	0.187350	1.951563e-07	0.238100	2.480208e-07	3.281800	3.418542e-06
6	cuda(nvcc)	960000	True	0.058001	6.041773e-08	9.5	0.081680	8.508375e-08	0.139682	1.455017e-07	1.603200	1.670000e-06
7	cudav2(nvcc)	960000	True	0.031670	3.298924e-08	9.5	0.207305	2.159423e-07	0.238974	2.489317e-07	1.539650	1.603802e-06
10	eigen(nvcc)	960000	True	0.138995	1.447866e-07	9.5	0.296204	3.085458e-07	0.435199	4.533324e-07	1.569850	1.635260e-06

Future Plans

Short term:

- Finish GPU alpaka version
- Redo scaling tests
  - Number of threads/streams
  - Matriplex size
  - Number of events
  - Number of tracks
- Set optimal parameters for each implementation
- Profile each implementation
- Write a computer science paper
  - Submit to a CS conference

Long term:

- Additional implementations
  - OpenCL
  - HIP
  - FPGA
  - OpenMP offloading
  - Auto generation of code.
- Test other architectures
  - Nvidia P100, T4
- Investigate writing the full mkFit algorithm for chosen tools
  - GPU matriplex library?

Backup

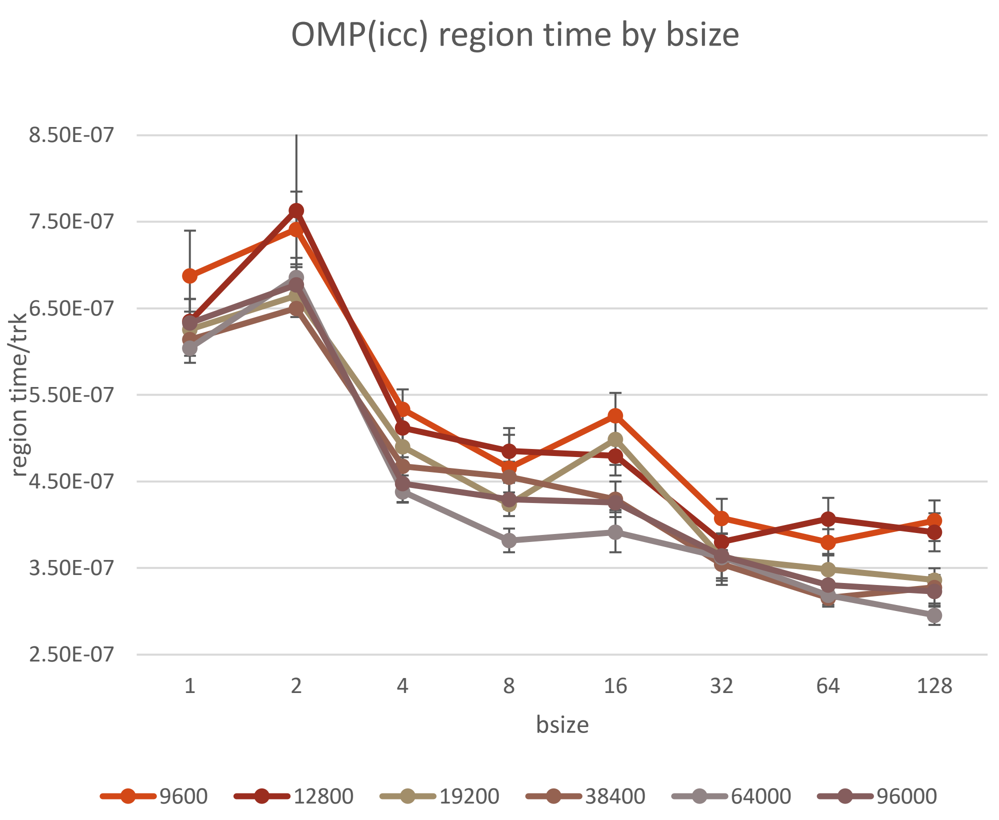
SCALING TESTS

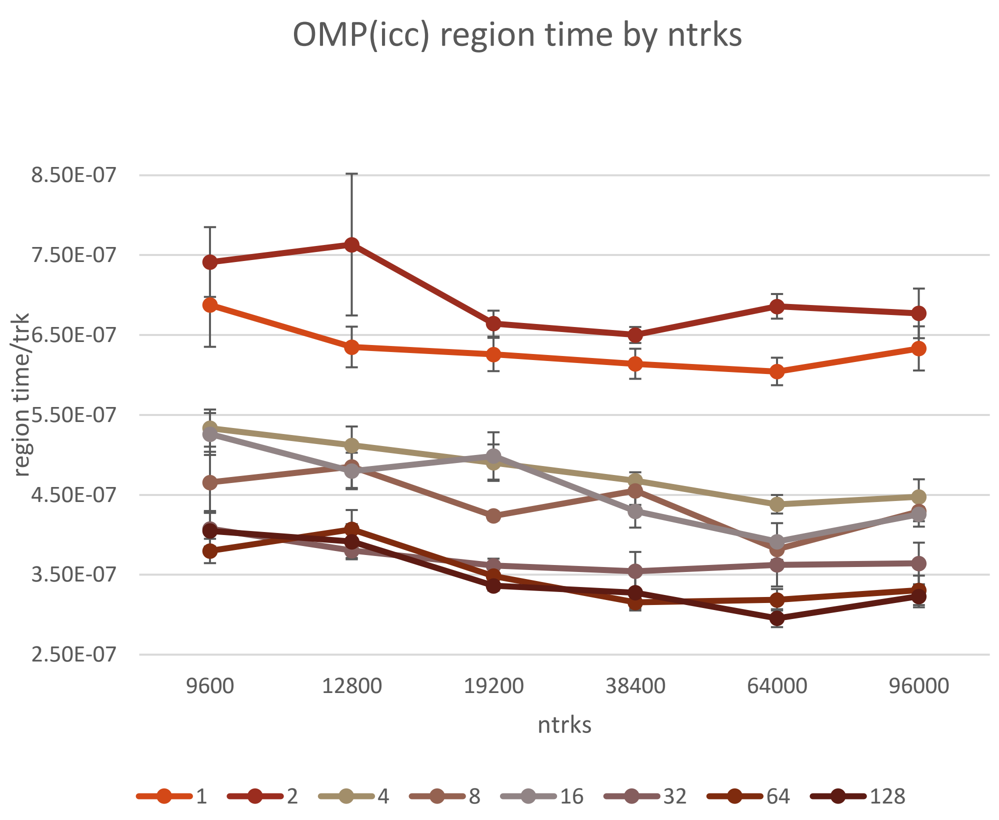
Implementation overview

	OpenMP	TBB	OpenACC	CUDA	Eigen	Alpaka	Kokkos	MKL
Compilers	GCC,ICC,PGI	ICC,GCC	PGI	NVCC	GCC,ICC,NVCC	GCC, NVCC	GCC,ICC,NVCC	Icc
CPU multi-threading	omp	tbb	x	x	omp	omp	omp	omp
GPU multi-threading	x	x	acc	cuda	cuda	cuda	cuda	x
Matriplex	yes	yes	yes	yes	no	yes	no	No

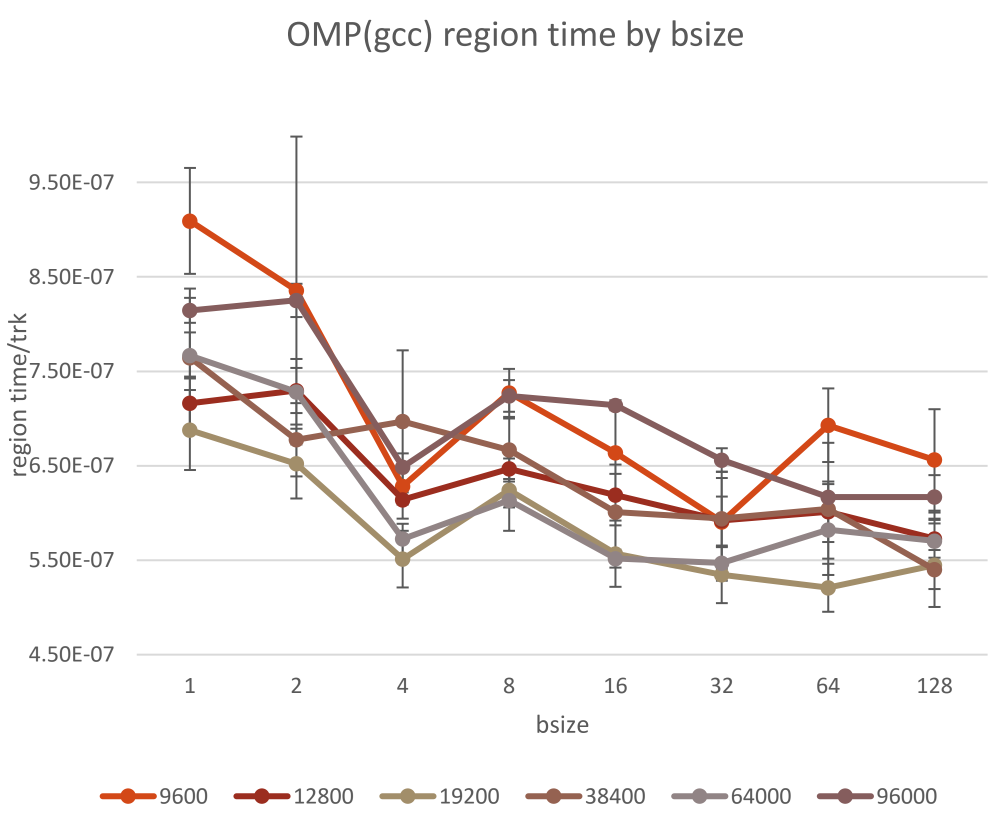
- Architectures
  - Intel Skylake Gold 6142
    - 64 cores
  - Tesla V100
- Compiler versions
  - G++ 8.3.1
  - Icc 19.1.1.217
  - Pgc++ 19.4-0
  - Nvcc 10.1.168
- Parameters use
  - Matriplex size: 128
  - Threads: 64
  - Events: 100
  - Tracks: 9600
- CUDA (GPU) parameters
  - Streams: 7 (10)
  - Blocks: 15 (10)
  - Threads (32,32)

OpenMP icc

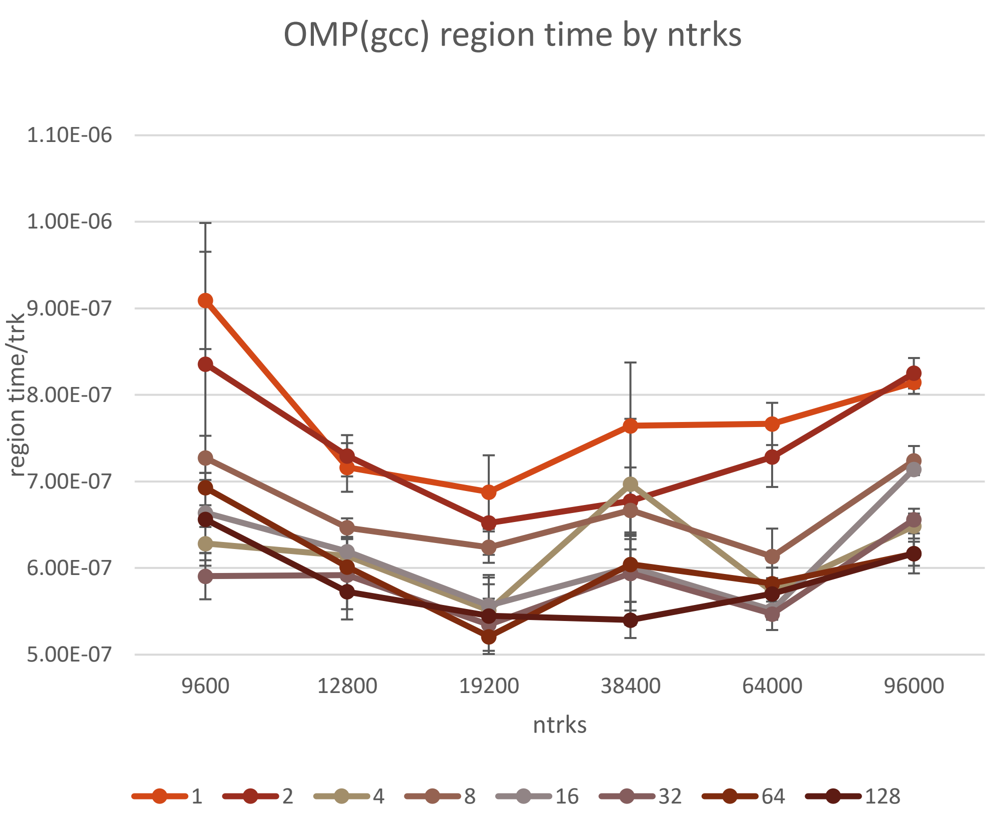

OMP(icc) region time by bsize
8.50E-07
7.50E-07
6.50E-07
5.50E-07
4.50E-07
3.50E-07
2.50E-07
region time/trk
1
2
4
8
16
32
64
128
bsize
9600
12800
19200
38400
64000
96000


OMP(icc) region time by ntrks
8.50E-07
7.50E-07
6.50E-07
5.50E-07
4.50E-07
3.50E-07
2.50E-07
region time/trk
9600
12800
19200
38400
64000
96000
ntrks
1
2
4
8
16
32
64
128

OpenMP gcc

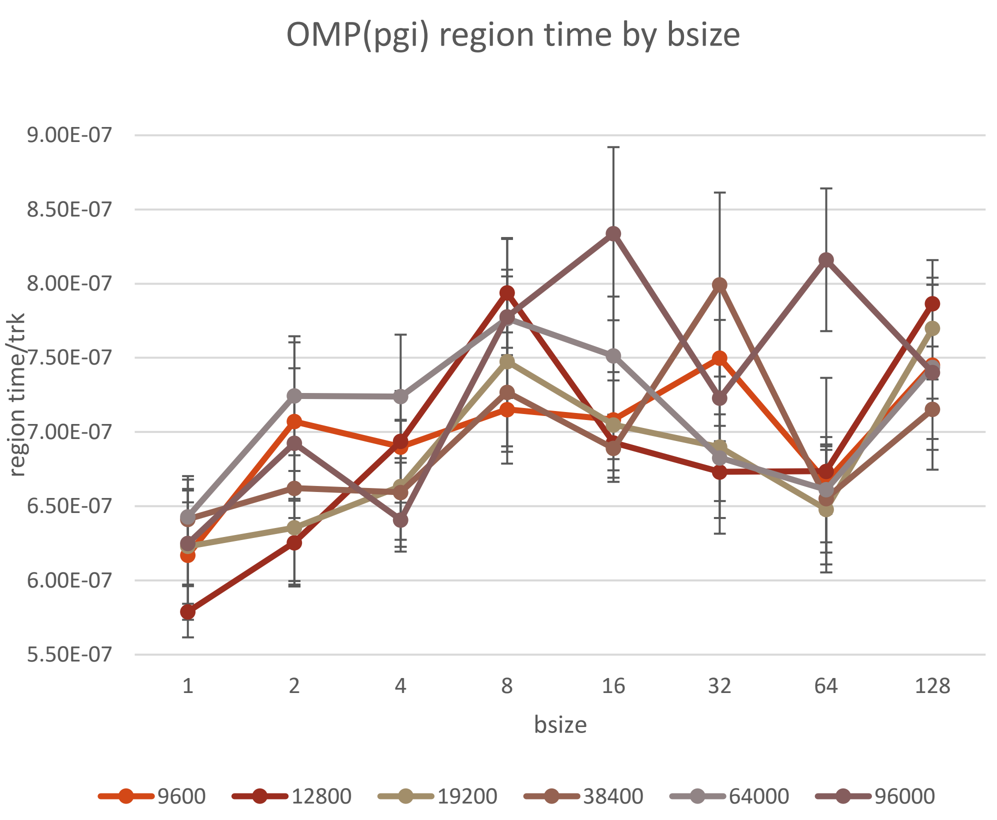


OMP(gcc) region time by bsize
9.50E-07
8.50E-07
7.50E-07
6.50E-07
5.50E-07
4.50E-07
region time/trk
1
2
4
8
16
32
64
128
bsize
9600
12800
19200
38400
64000
96000

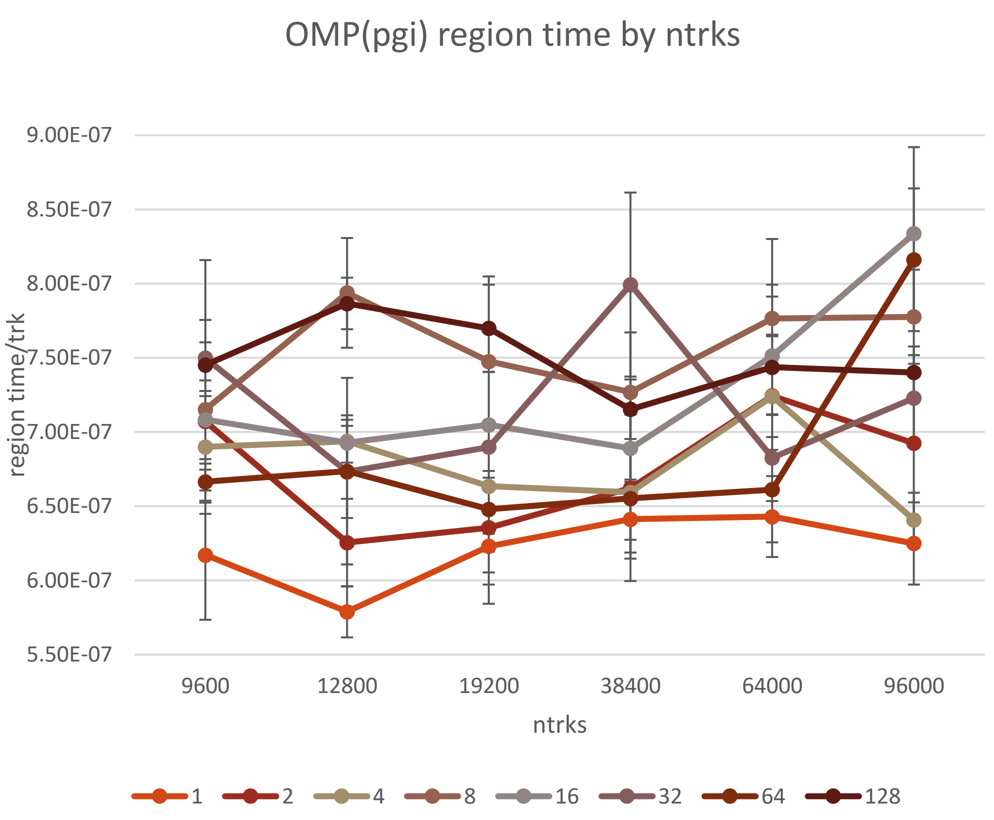


OMP(gcc) region time by ntrks
1.10E-06
1.00E-06
9.00E-07
8.00E-07
7.00E-07
6.00E-07
5.00E-07
region time/trk
9600
12800
19200
38400
64000
96000
ntrks
1
2
4
8
16
32
64
128

OpenMP pgi

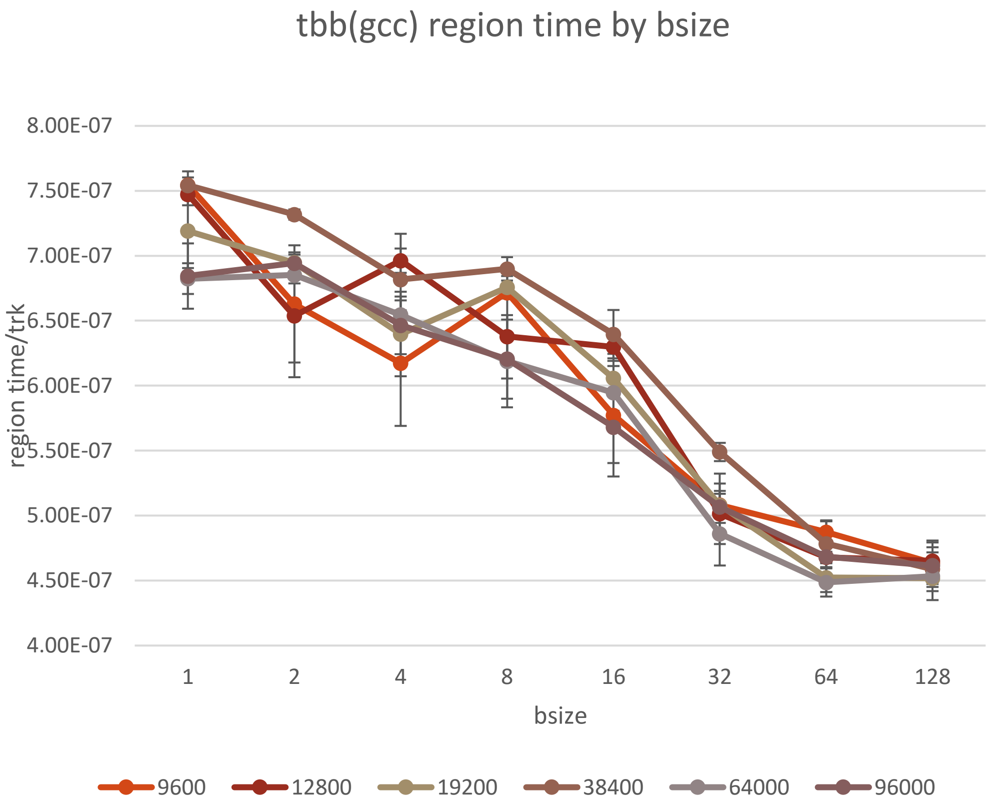


OMP(pgi) region time by bsize
9.00E-07
8.50E-07
8.00E-07
7.50E-07
7.00E-07
6.50E-07
6.00E-07
5.50E-07
region time/trk
1
2
4
8
16
32
64
128
bsize
9600
12800
19200
38400
64000
96000

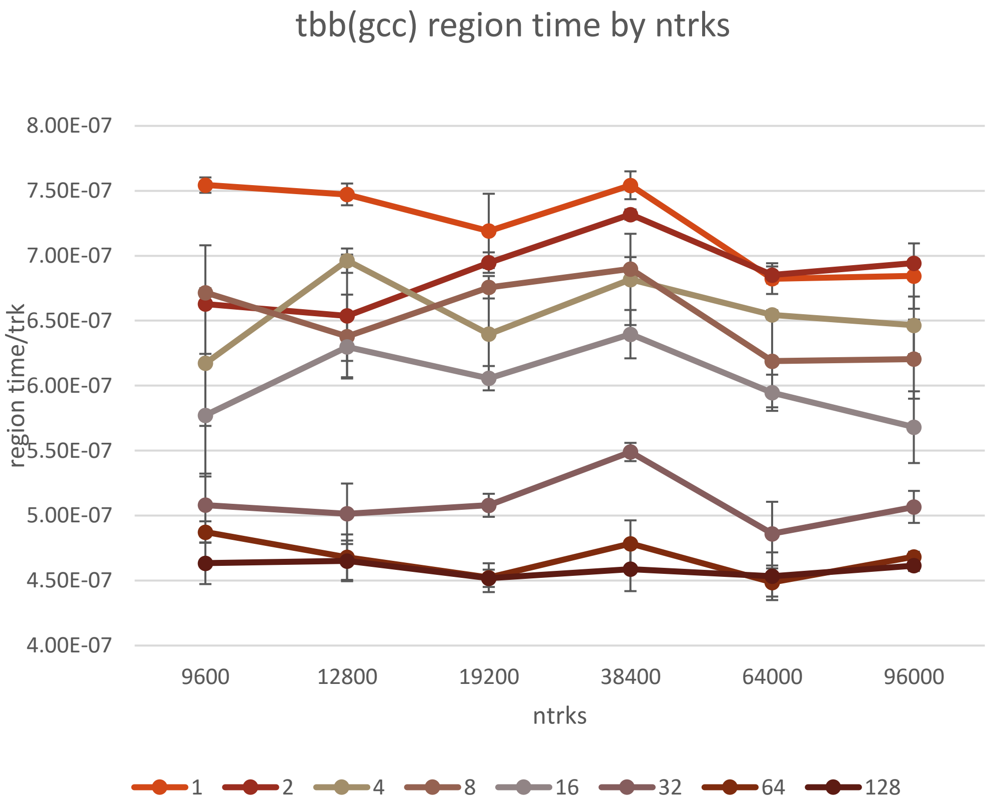


OMP(pgi) region time by ntrks
9.00E-07
8.50E-07
8.00E-07
7.50E-07
7.00E-07
6.50E-07
6.00E-07
5.50E-07
region time/trk
9600
12800
19200
38400
64000
96000
ntrks
1
2
4
8
16
32
64
128

TBB gcc

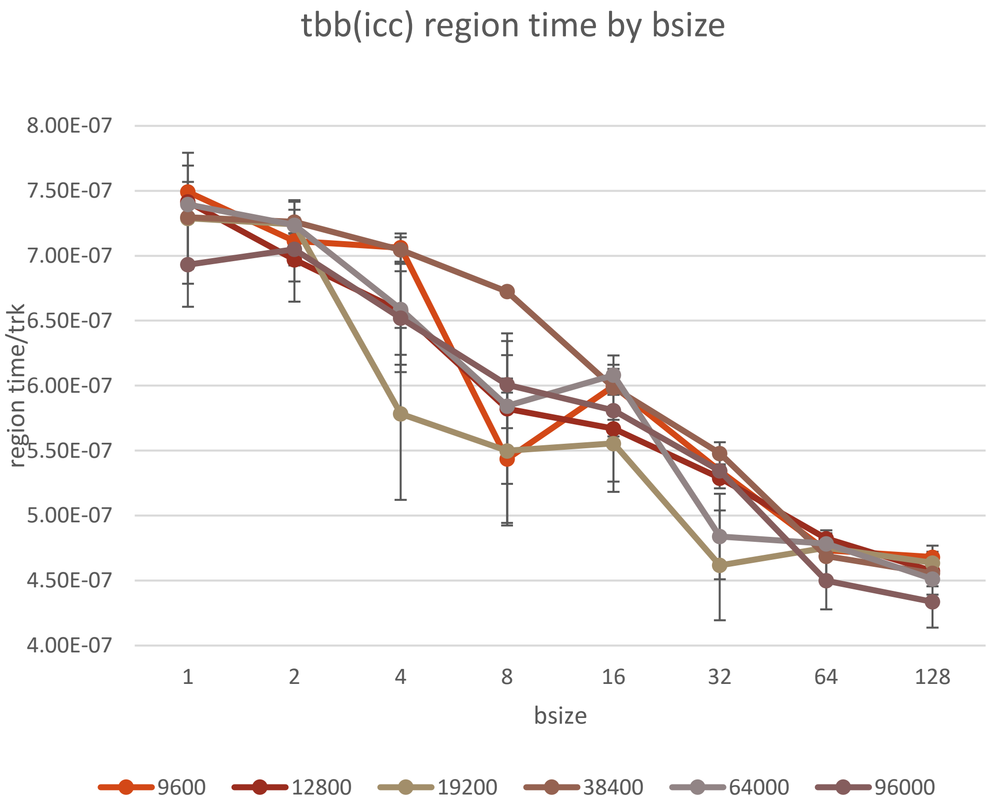


tbb(gcc) region time by bsize
8.00E-07
7.50E-07
7.00E-07
6.50E-07
6.00E-07
5.50E-07
5.00E-07
4.50E-07
4.00E-07
region time/trk
1
2
4
8
16
32
64
128
bsize
9600
12800
19200
38400
64000
96000

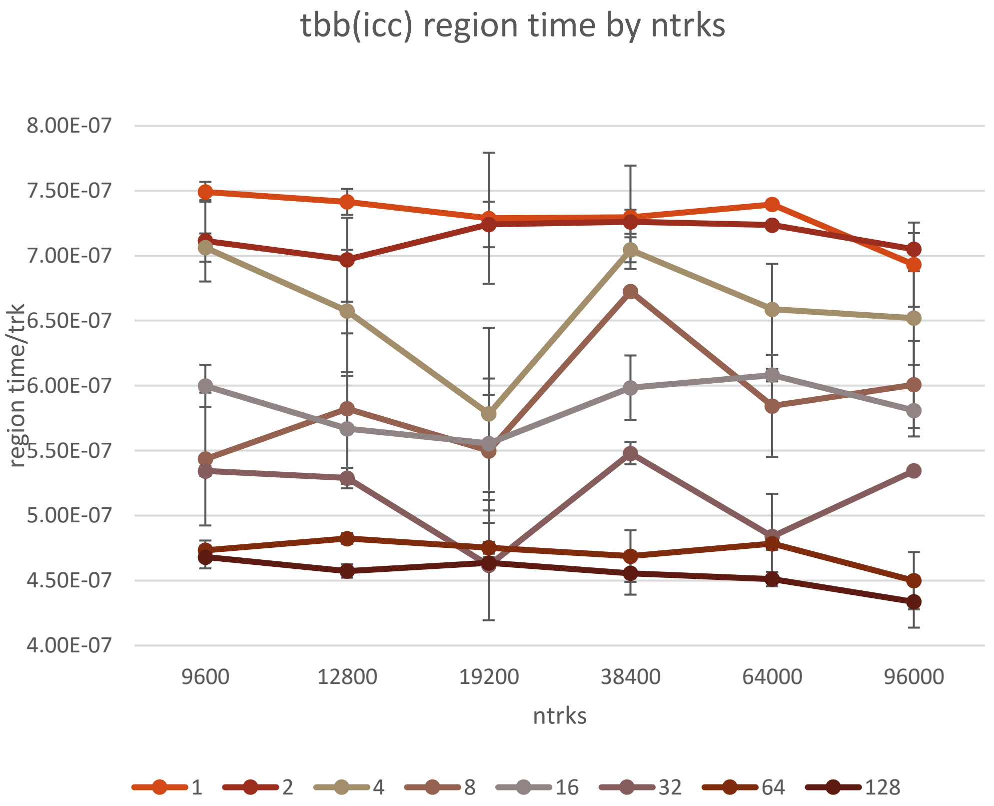


tbb(gcc) region time by ntrks
8.00E-07
7.50E-07
7.00E-07
6.50E-07
6.00E-07
5.50E-07
5.00E-07
4.50E-07
4.00E-07
region time/trk
9600
12800
19200
38400
64000
96000
ntrks
1
2
4
8
16
32
64
128

TBB icc

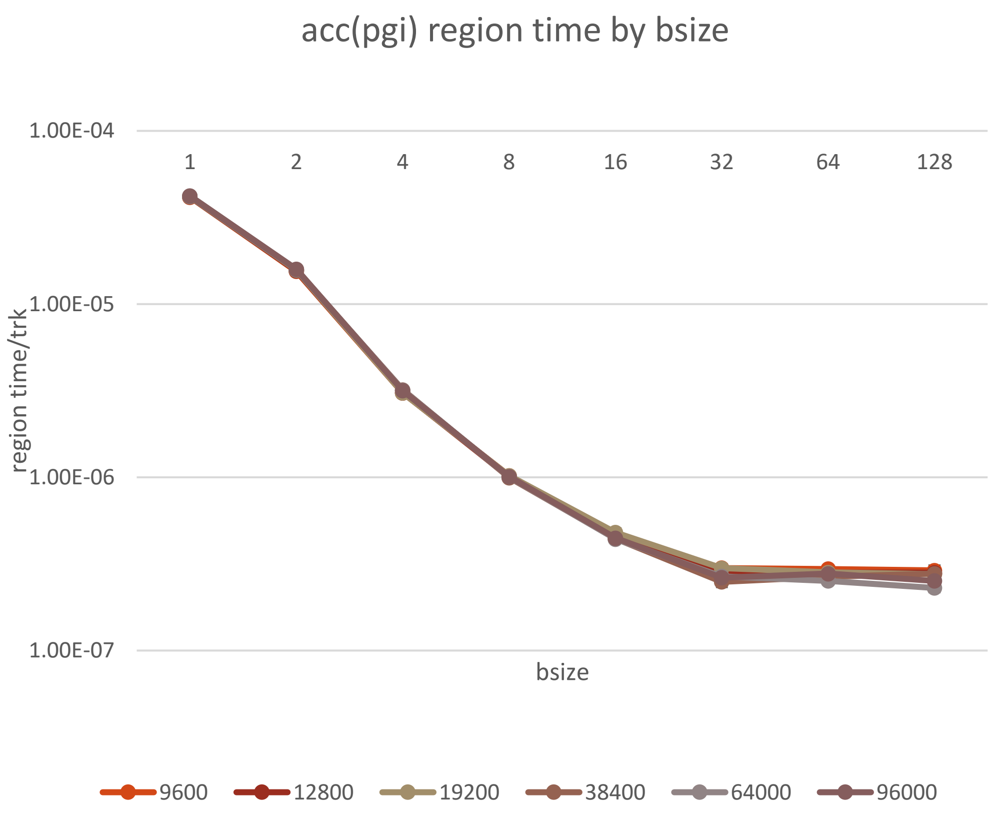


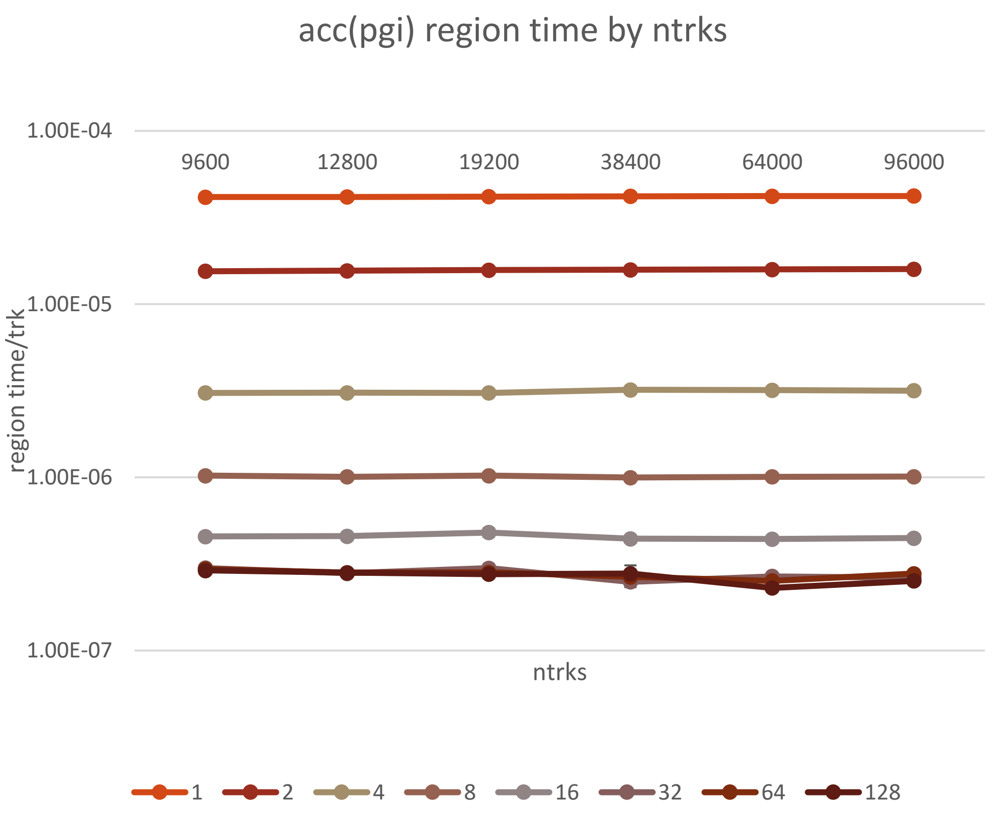
tbb(icc) region time by bsize
8.00E-07
7.50E-07
7.00E-07
6.50E-07
6.00E-07
5.50E-07
5.00E-07
4.50E-07
4.00E-07
region time/trk
1
2
4
8
16
32
64
128
bsize
9600
12800
19200
38400
64000
96000



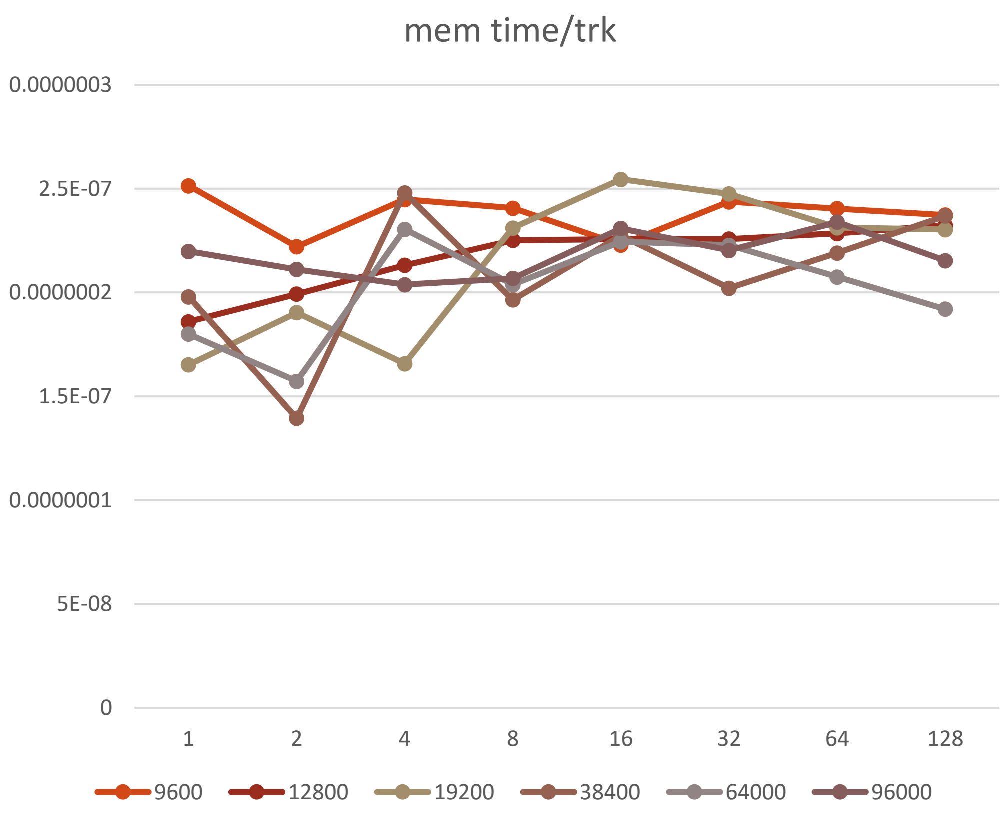
tbb(icc) region time by ntrks
8.00E-07
7.50E-07
7.00E-07
6.50E-07
6.00E-07
5.50E-07
5.00E-07
4.50E-07
4.00E-07
region time/trk
9600
12800
19200
38400
64000
96000
ntrks
1
2
4
8
16
32
64
128

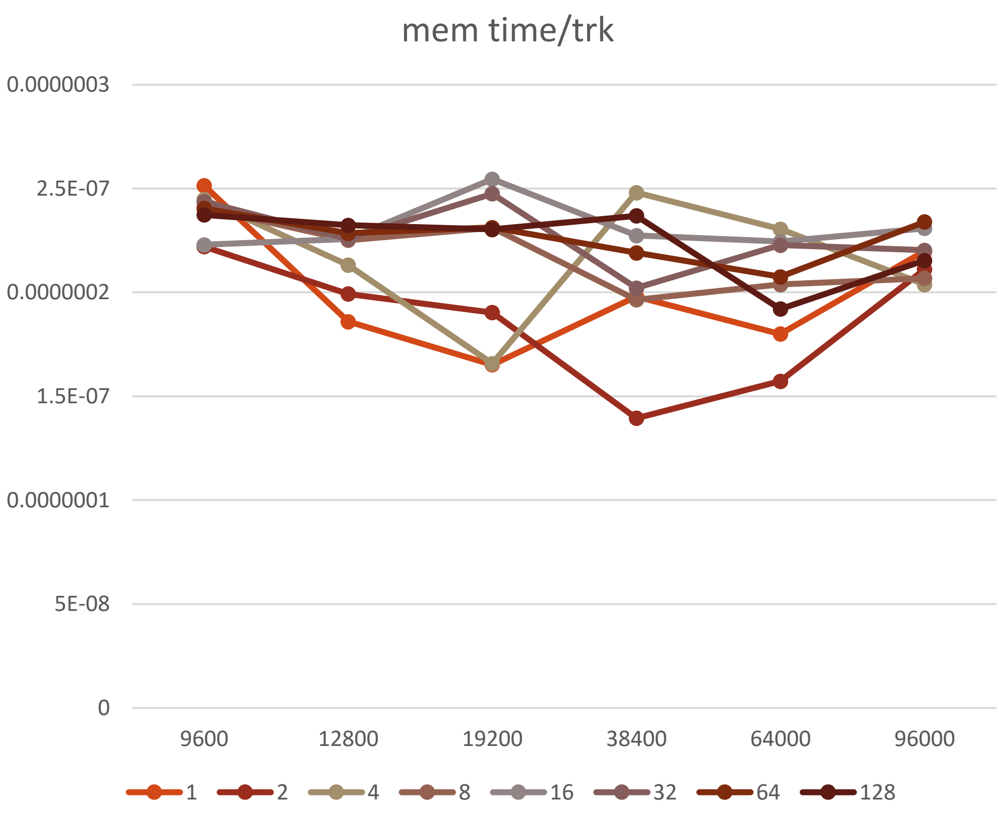
OpenACC pgi


acc(pgi) region time by bsize
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1 2 4 8 16 32 64 128
region time/trk
bsize
9600 12800 19200 38400 64000 96000

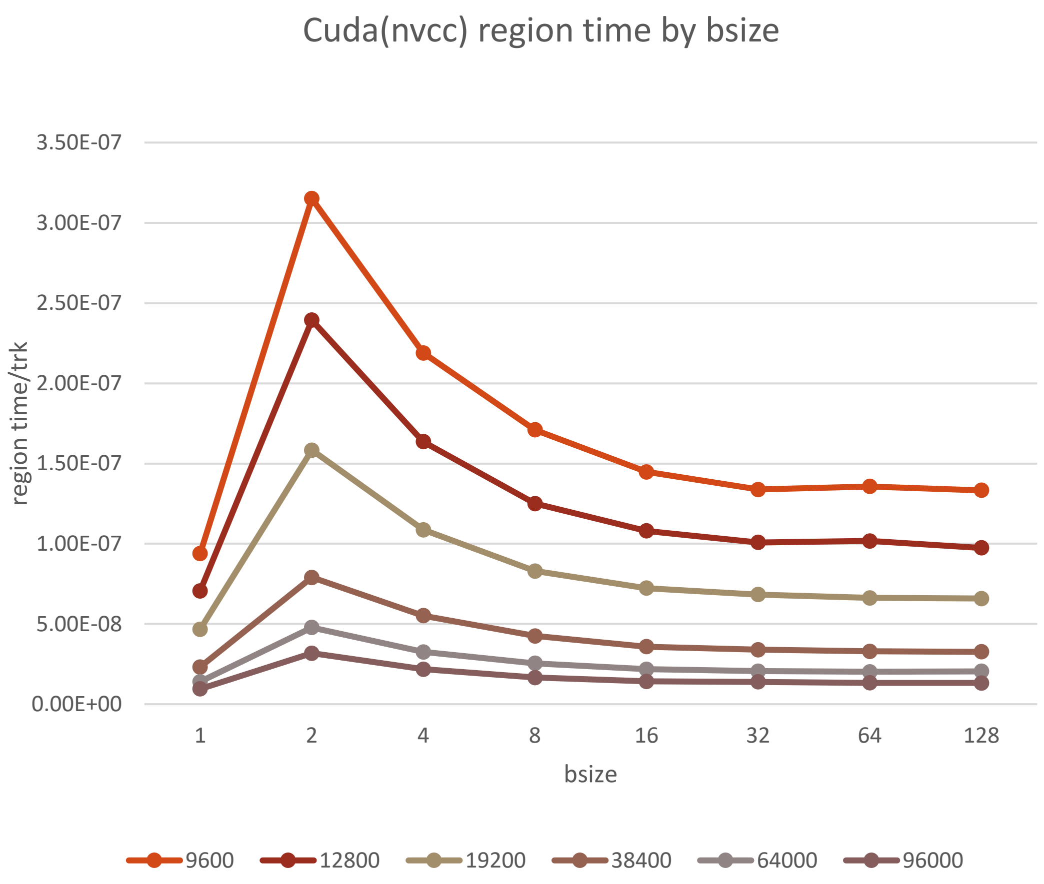

acc(pgi) region time by ntrks
1.00E-04
1.00E-05
1.00E-06
1.00E-07
9600 12800 19200 38400 64000 96000
region time/trk
ntrks
1 2 4 8 16 32 64 128

OpenACC memory transfer

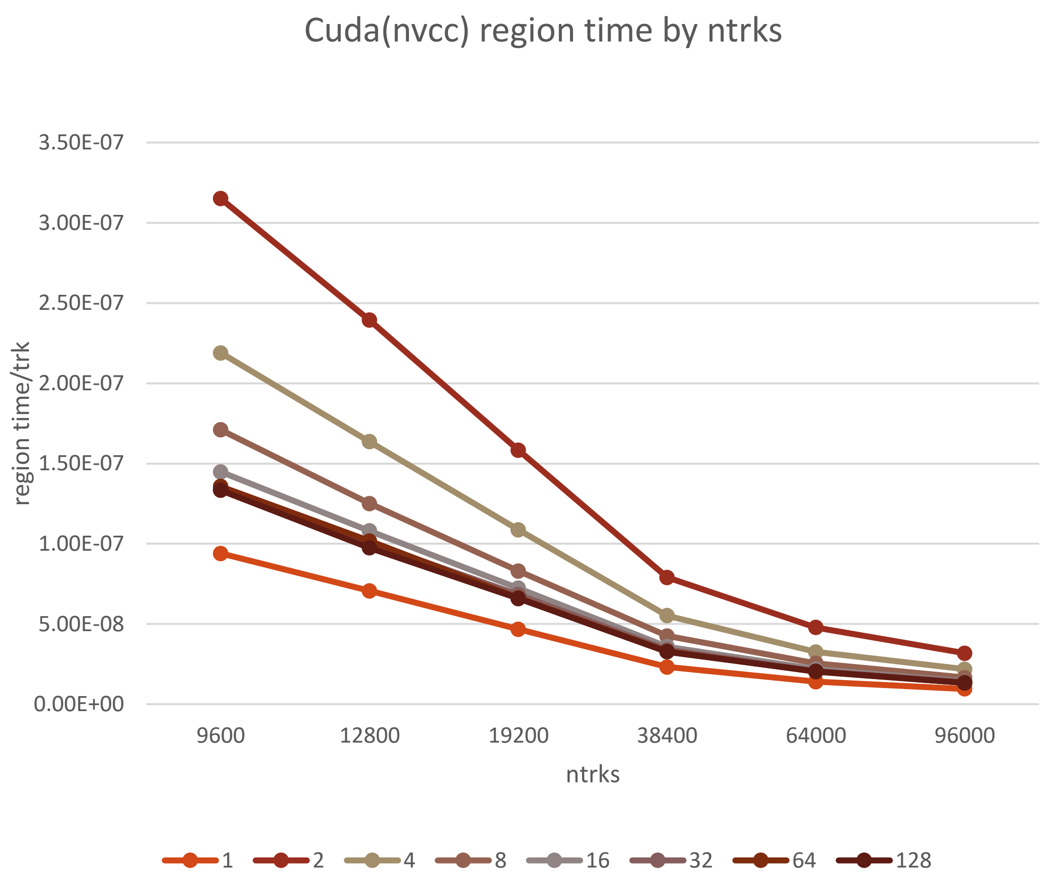




Cuda nvcc

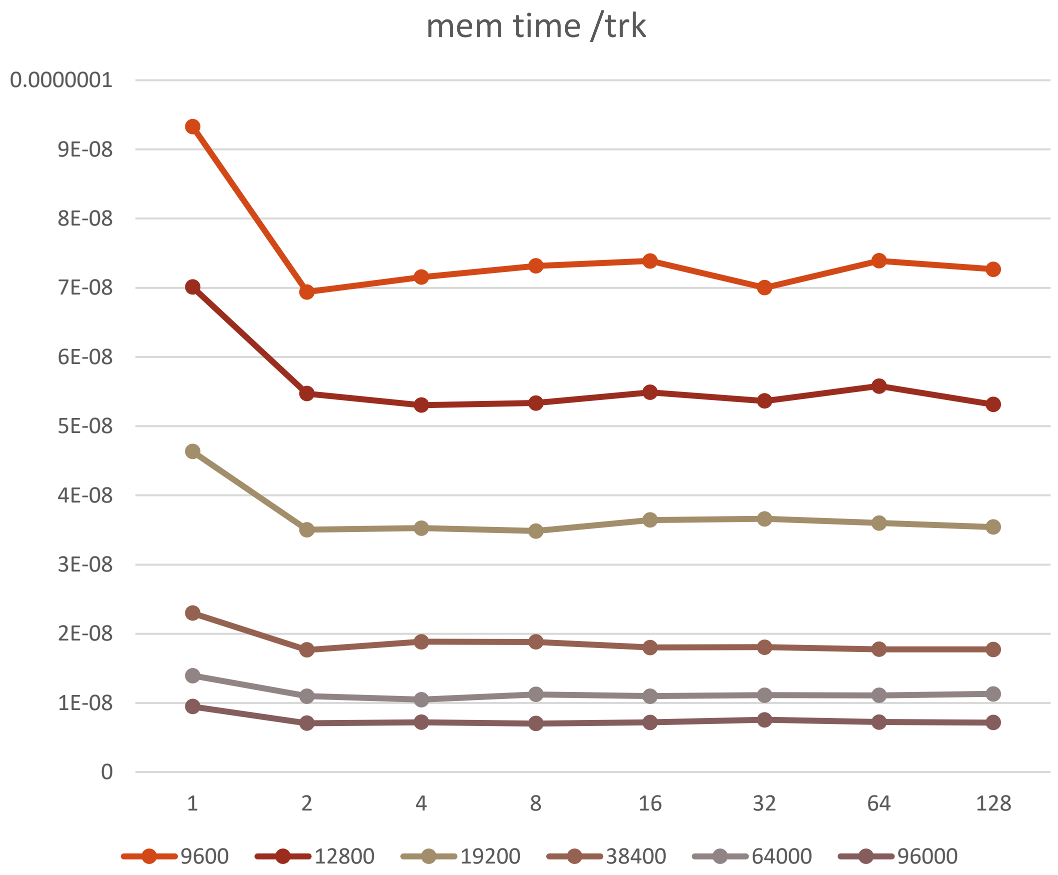


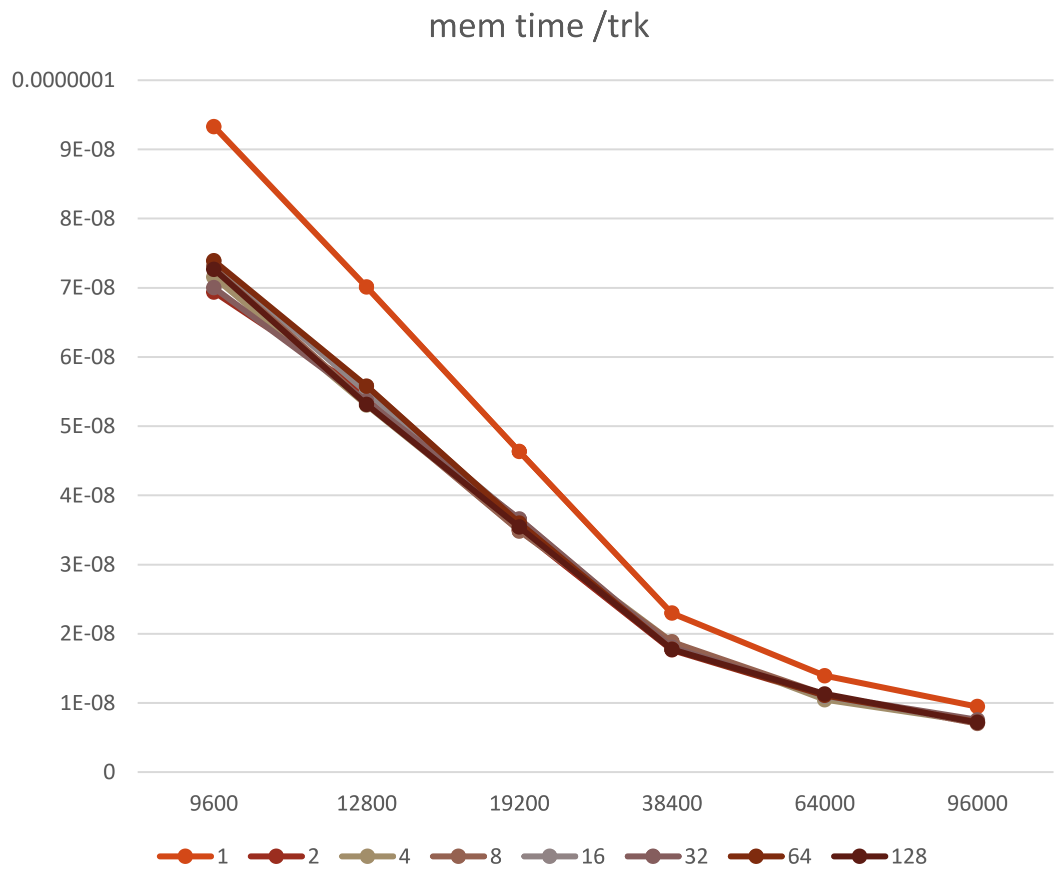
Cuda(nvcc) region time by bsize
3.50E-07
3.00E-07
2.50E-07
2.00E-07
1.50E-07
1.00E-07
5.00E-08
0.00E+00
region time/trk
1
2
4
8
16
32
64
128
bsize
9600
12800
19200
38400
64000
96000



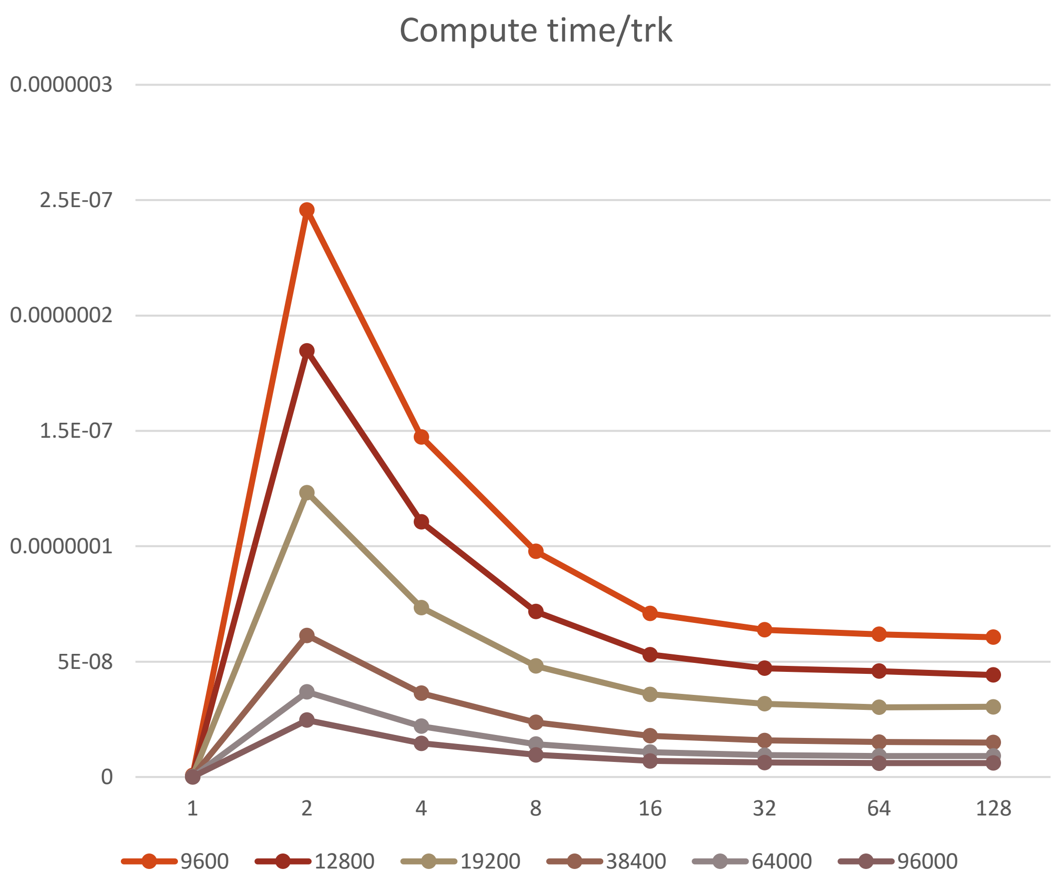
Cuda(nvcc) region time by ntrks
3.50E-07
3.00E-07
2.50E-07
2.00E-07
1.50E-07
1.00E-07
5.00E-08
0.00E+00
region time/trk
9600
12800
19200
38400
64000
96000
ntrks
1
2
4
8
16
32
64
128

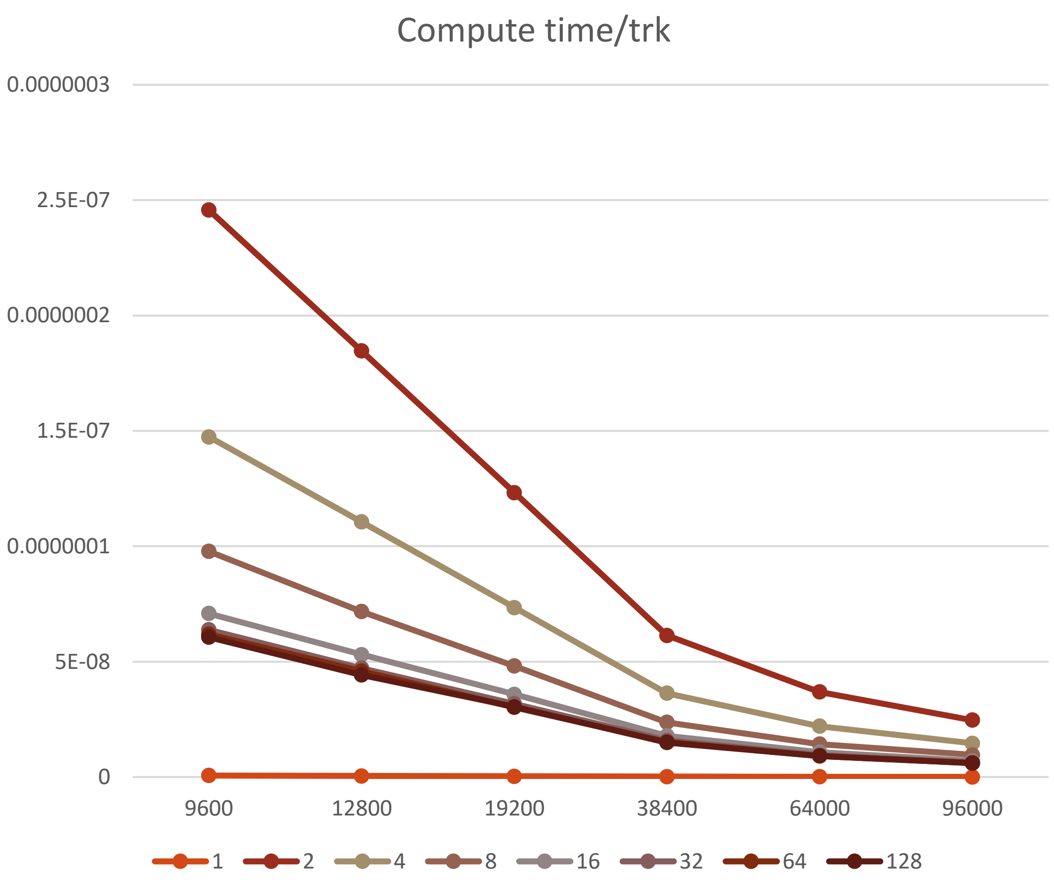
Cuda mem transfer time



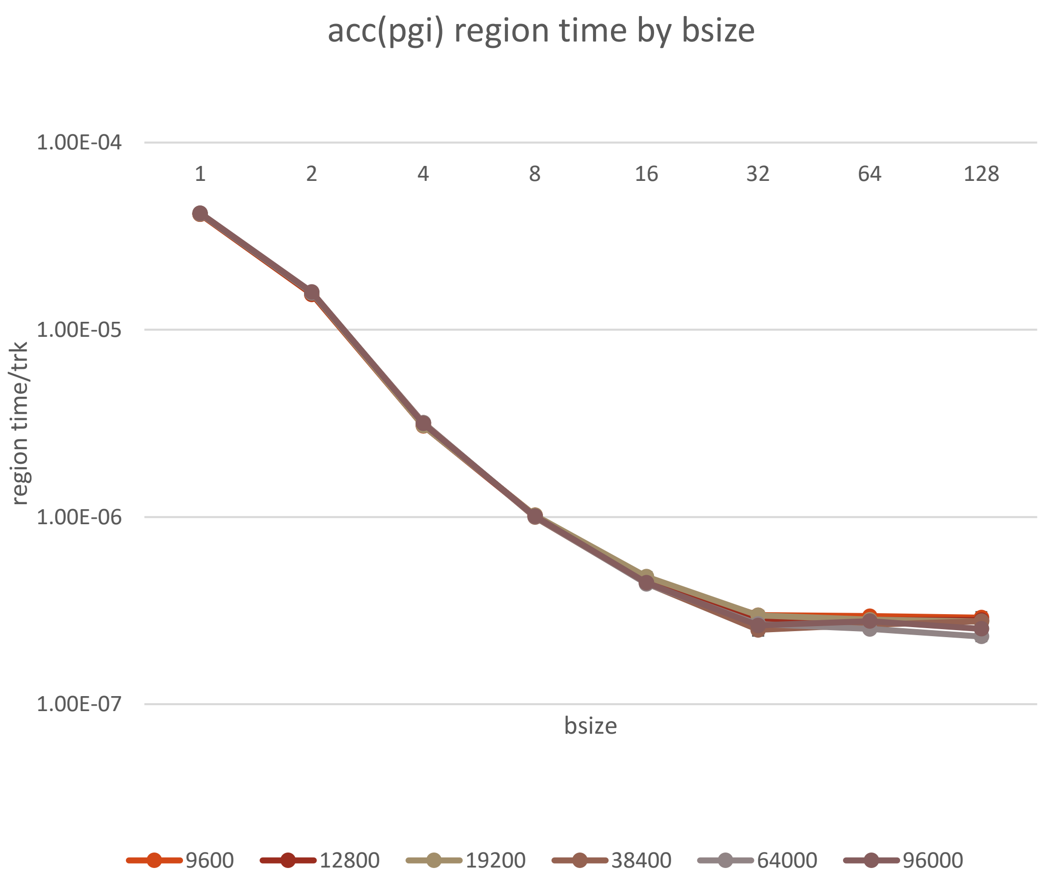


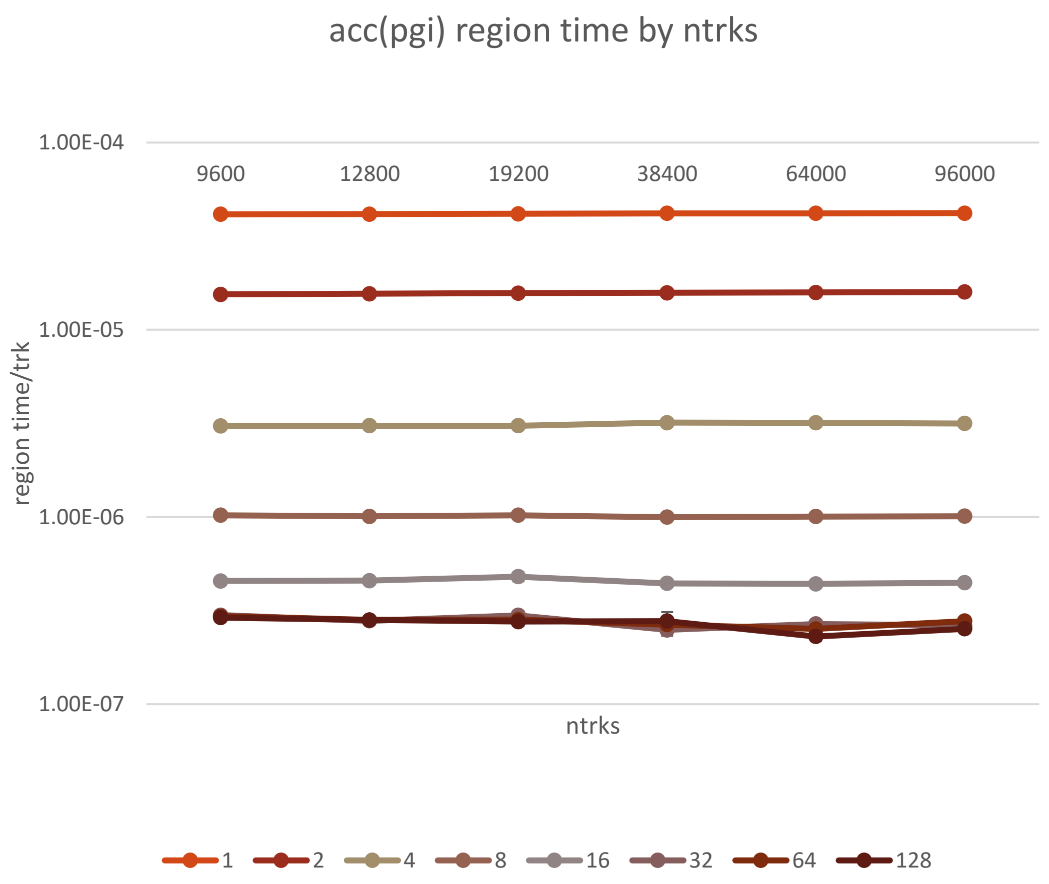
Cuda computation time



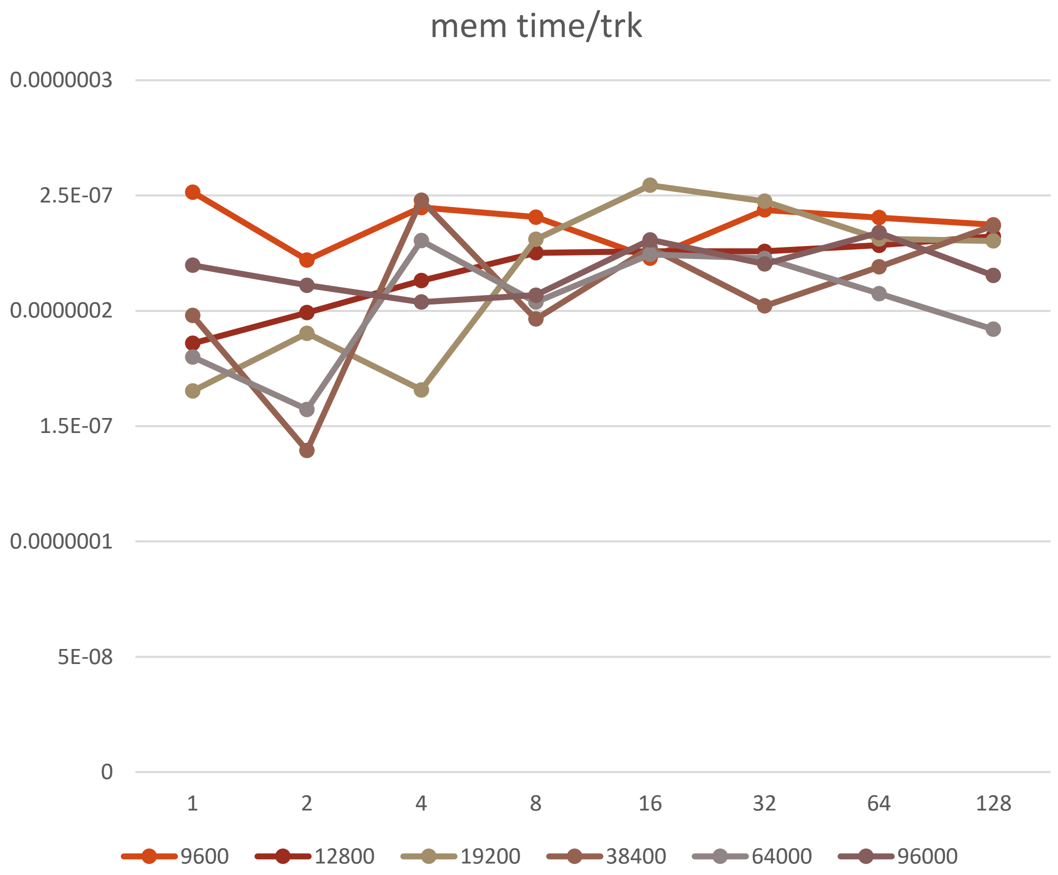


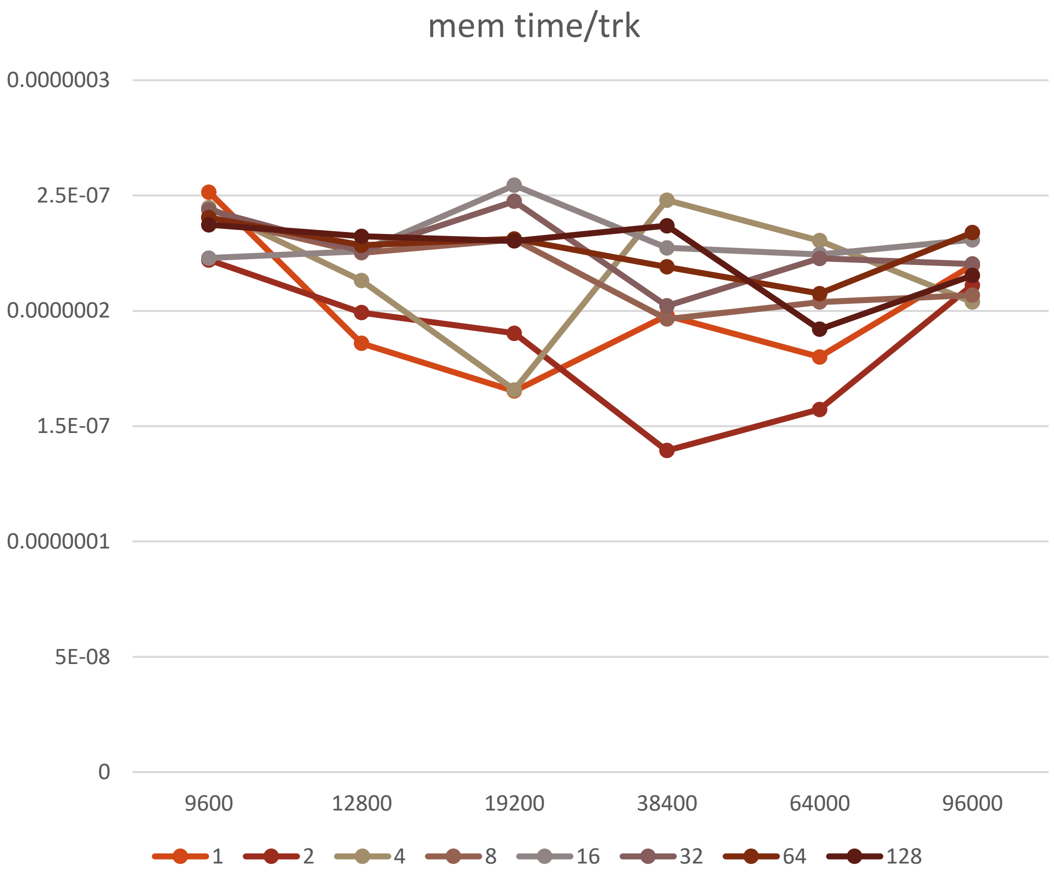
Acc pgi


acc(pgi) region time by bsize
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1 2 4 8 16 32 64 128
region time/trk
bsize
9600 12800 19200 38400 64000 96000

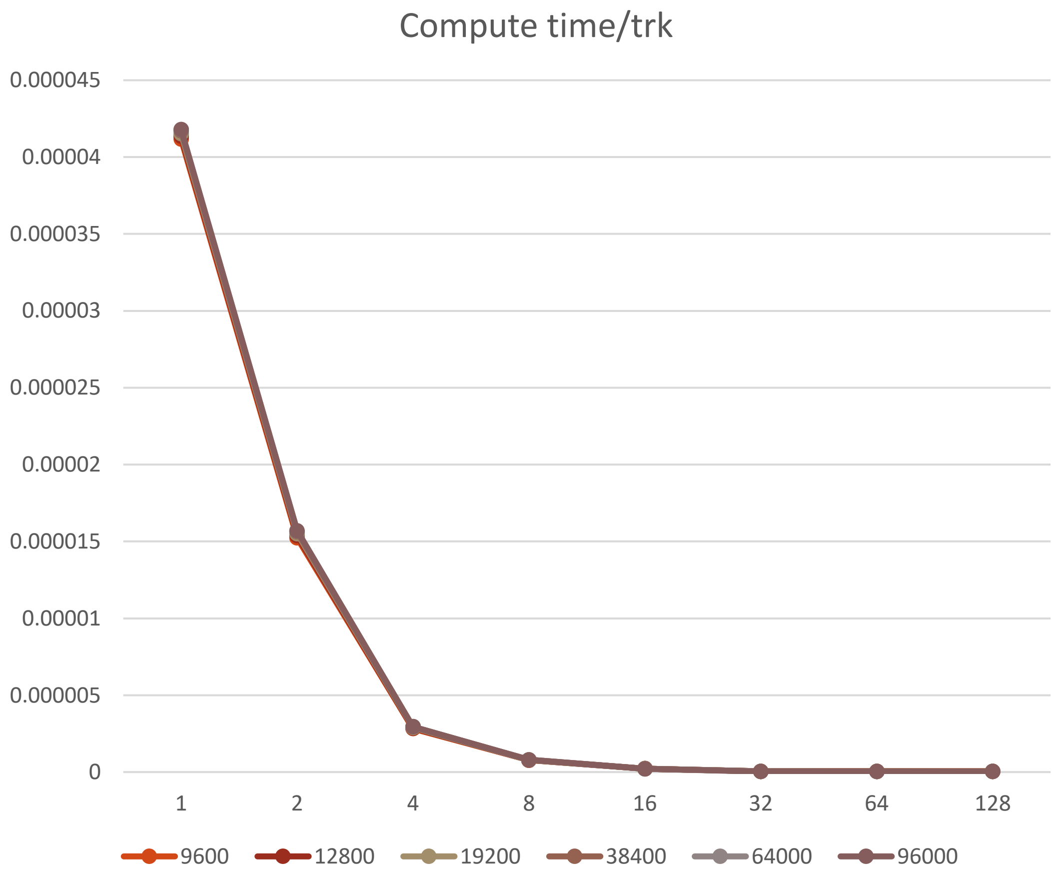

acc(pgi) region time by ntrks
1.00E-04
1.00E-05
1.00E-06
1.00E-07
9600 12800 19200 38400 64000 96000
region time/trk
ntrks
1 2 4 8 16 32 64 128

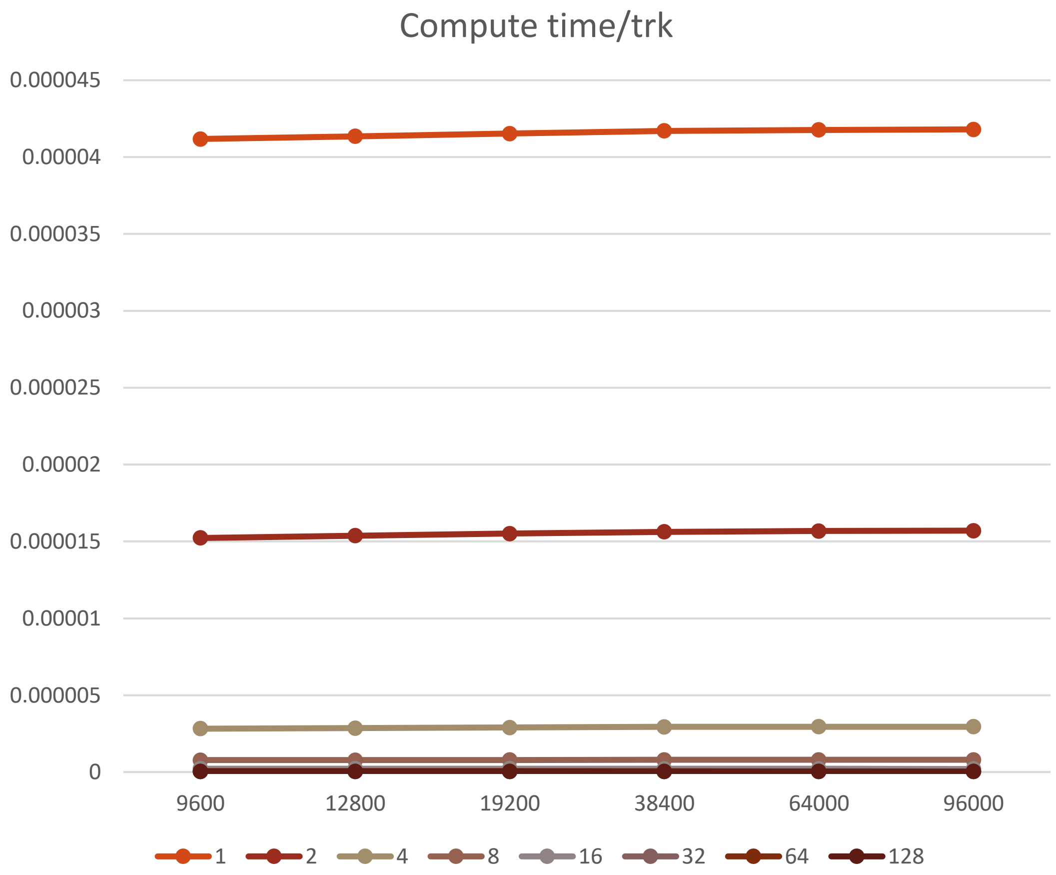
Acc mem transfer time



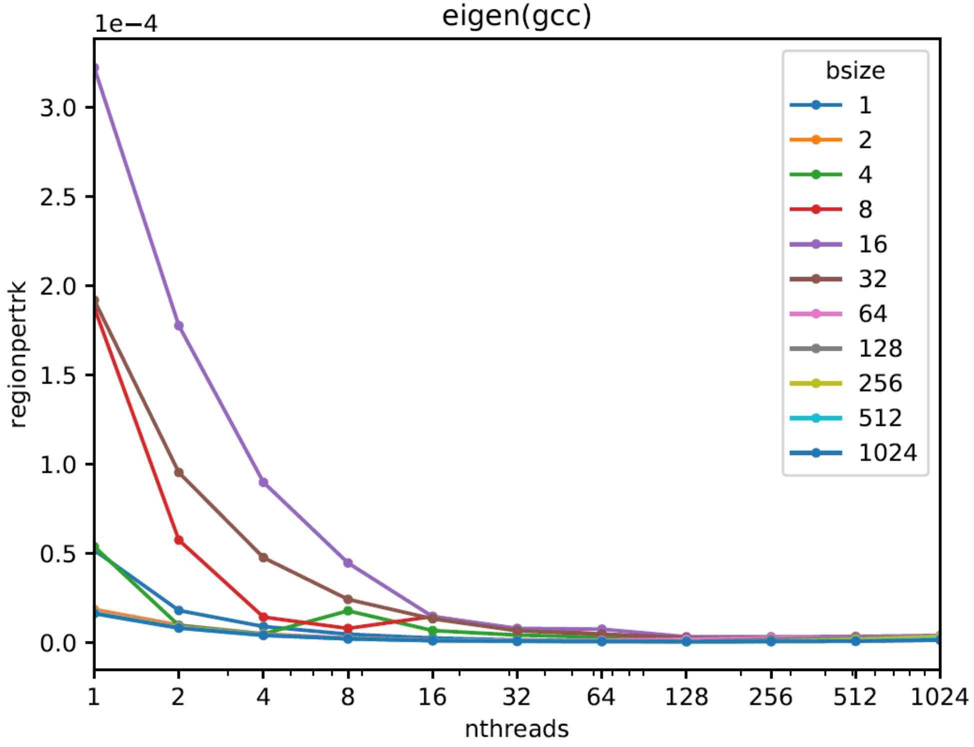


Acc computation time

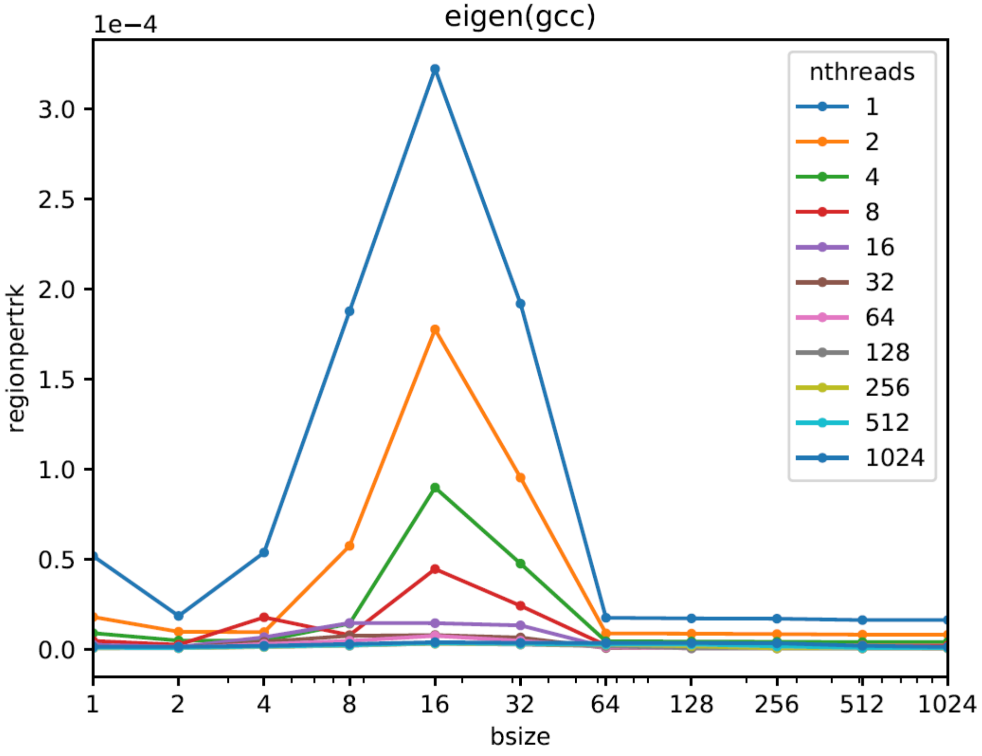




Eigen(gcc)

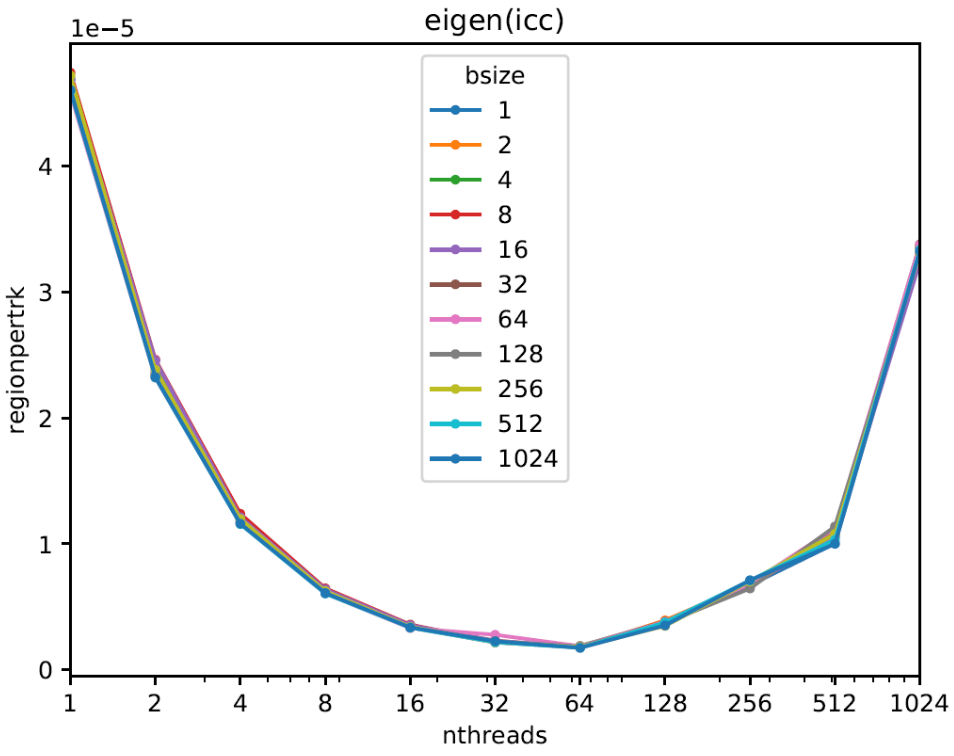


eigen(gcc)
1e−4
regionpertrk
nthreads
bsize
1
2
4
8
16
32
64
128
256
512
1024

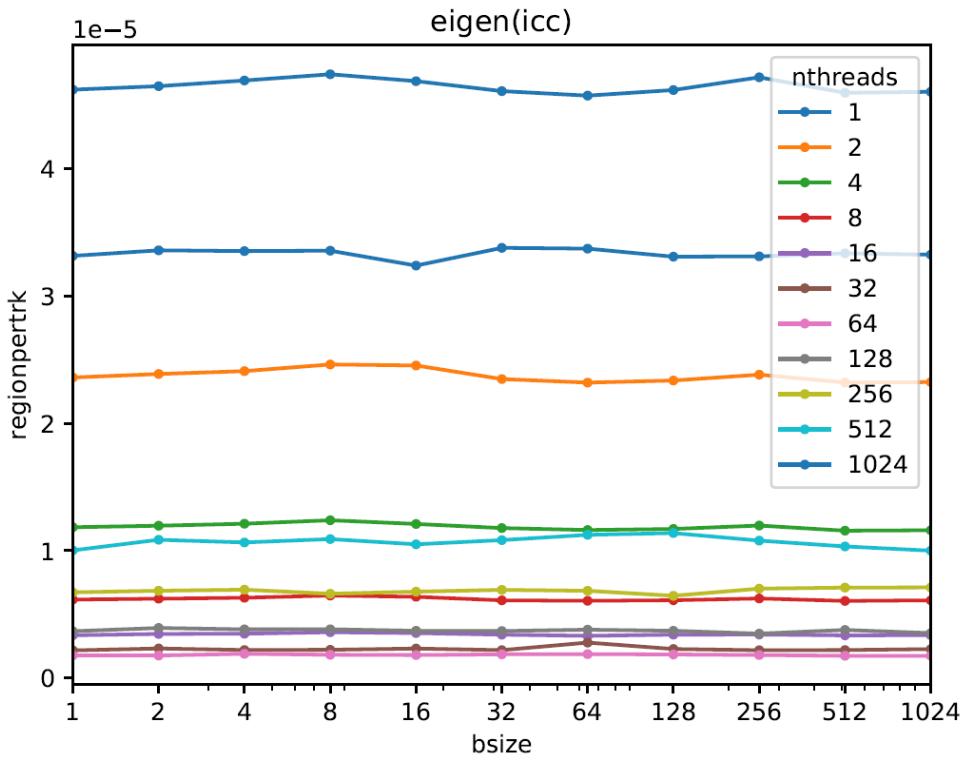


eigen(gcc)
1e−4
regionpertrk
bsize
nthreads
1
2
4
8
16
32
64
128
256
512
1024

Eigen(icc)

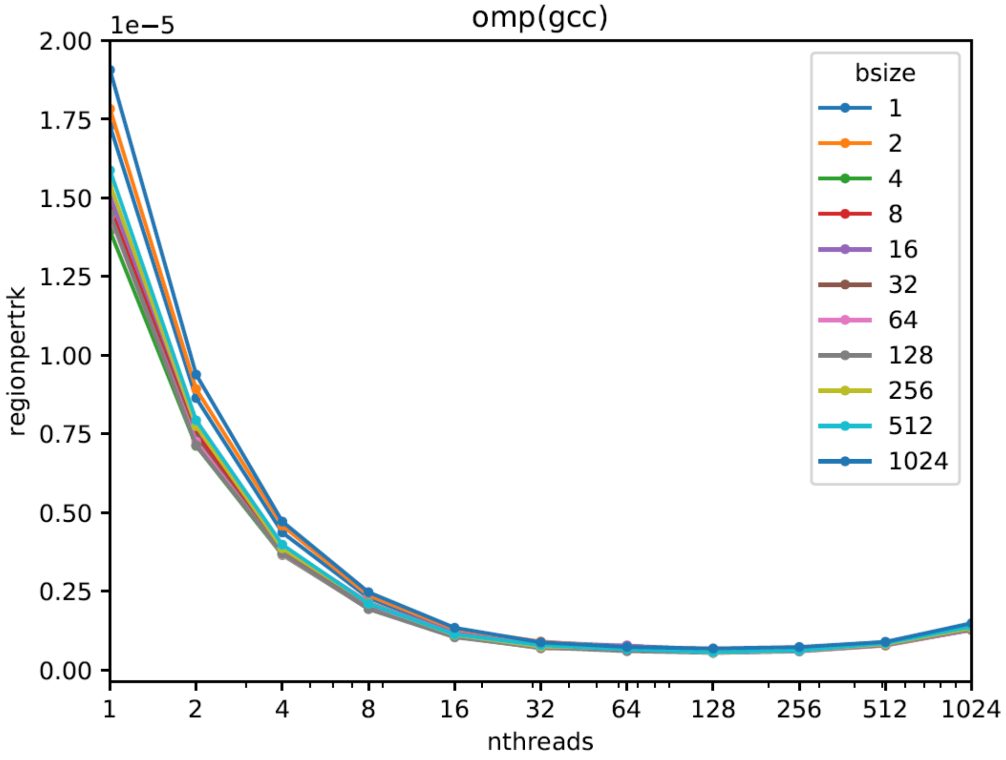


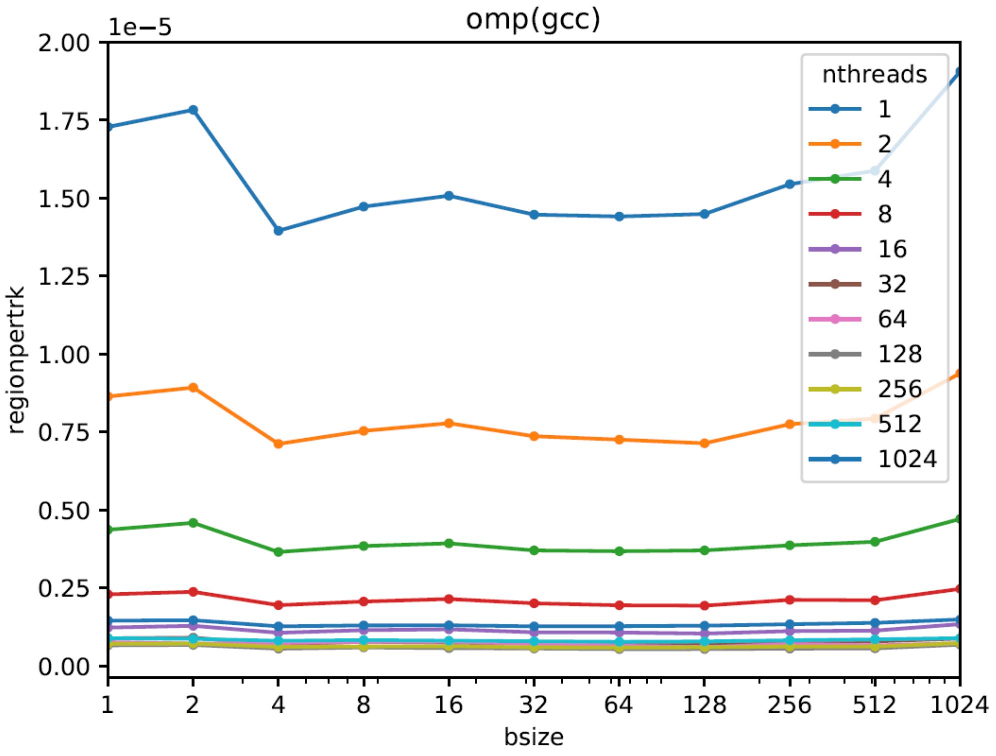
eigen(icc)
1e−5
bsize
1
2
4
8
16
32
64
128
256
512
1024
regionpertrk
0
1
2
3
4
1
2
4
8
16
32
64
128
256
512
1024
nthreads



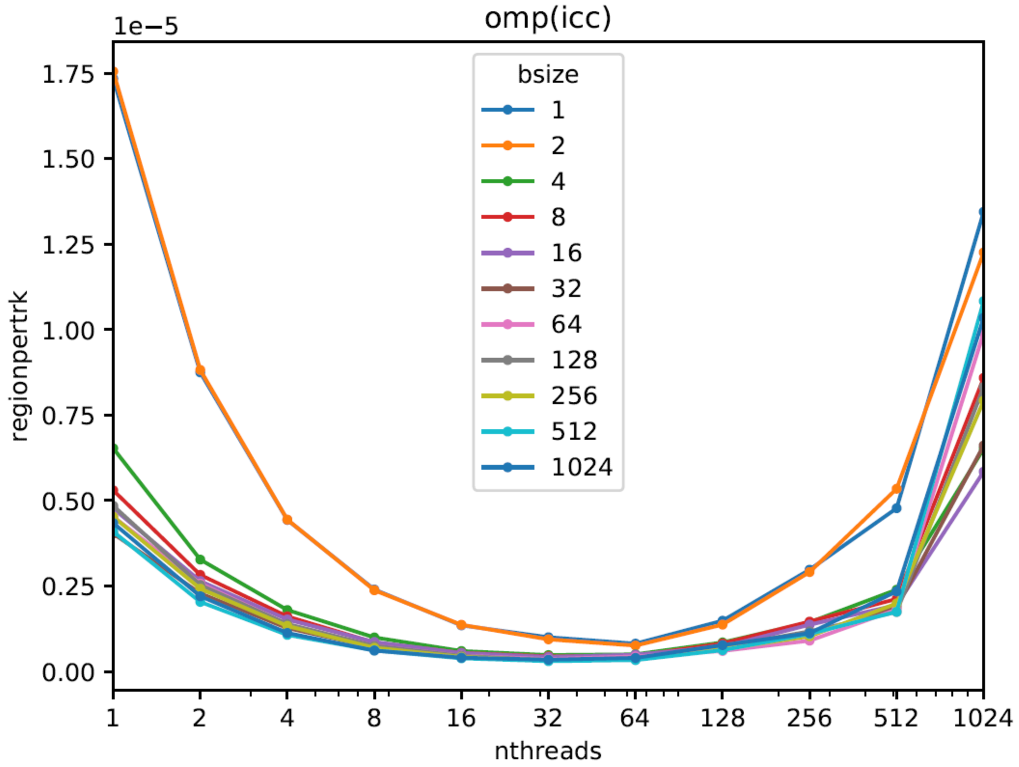
eigen(icc)
1e−5
nthreads
1
2
4
8
16
32
64
128
256
512
1024
regionpertrk
0
1
2
3
4
1
2
4
8
16
32
64
128
256
512
1024
bsize

OMP(gcc)

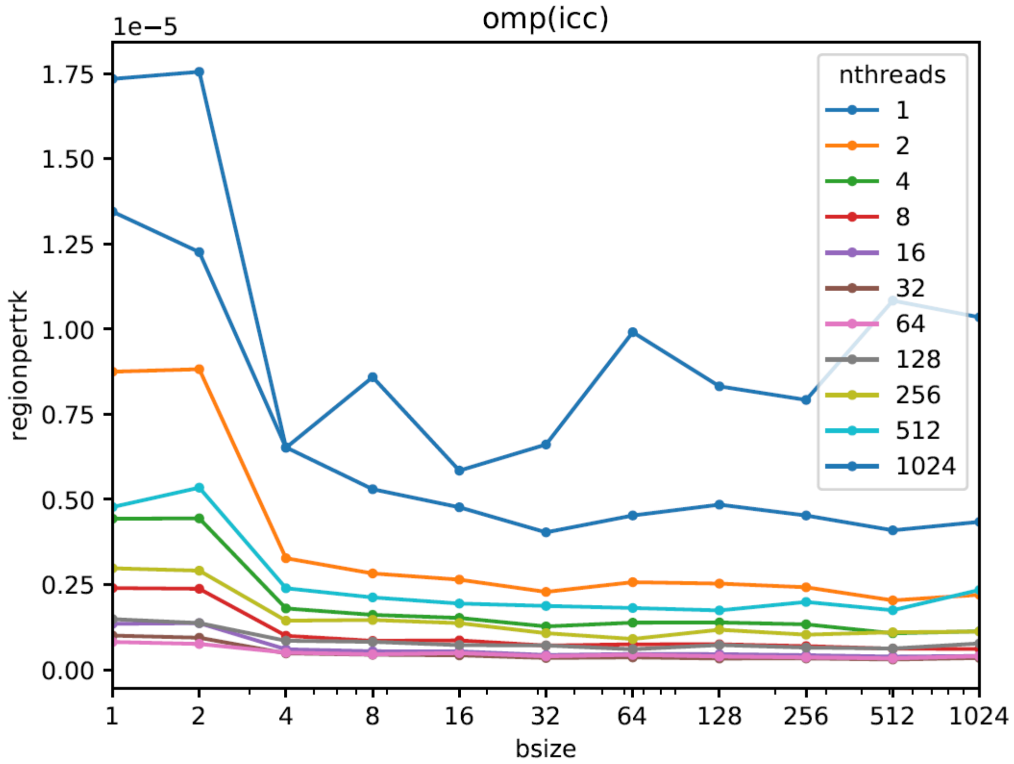

omp(gcc)
1e−5
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
regionpertrk
1
2
4
8
16
32
64
128
256
512
1024
nthreads
bsize
1
2
4
8
16
32
64
128
256
512
1024


omp(gcc)
1e−5
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
regionpertrk
1
2
4
8
16
32
64
128
256
512
1024
bsize
nthreads
1
2
4
8
16
32
64
128
256
512
1024

OMP(icc)

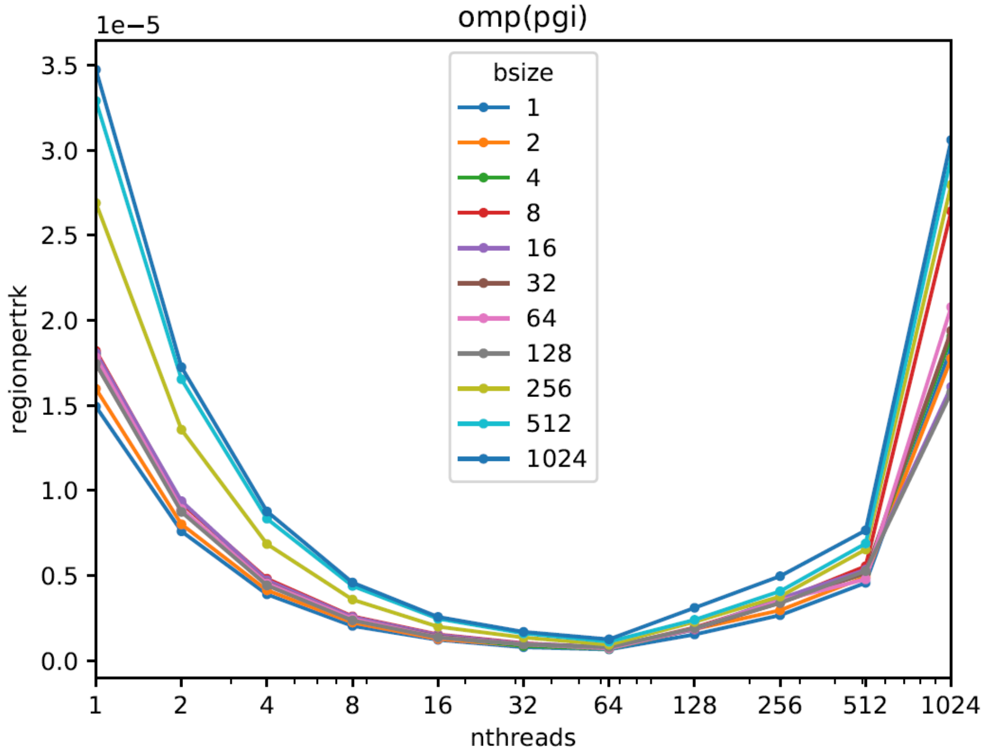


omp(icc)
1e−5
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
regionpertrk
bsize
1
2
4
8
16
32
64
128
256
512
1024
nthreads

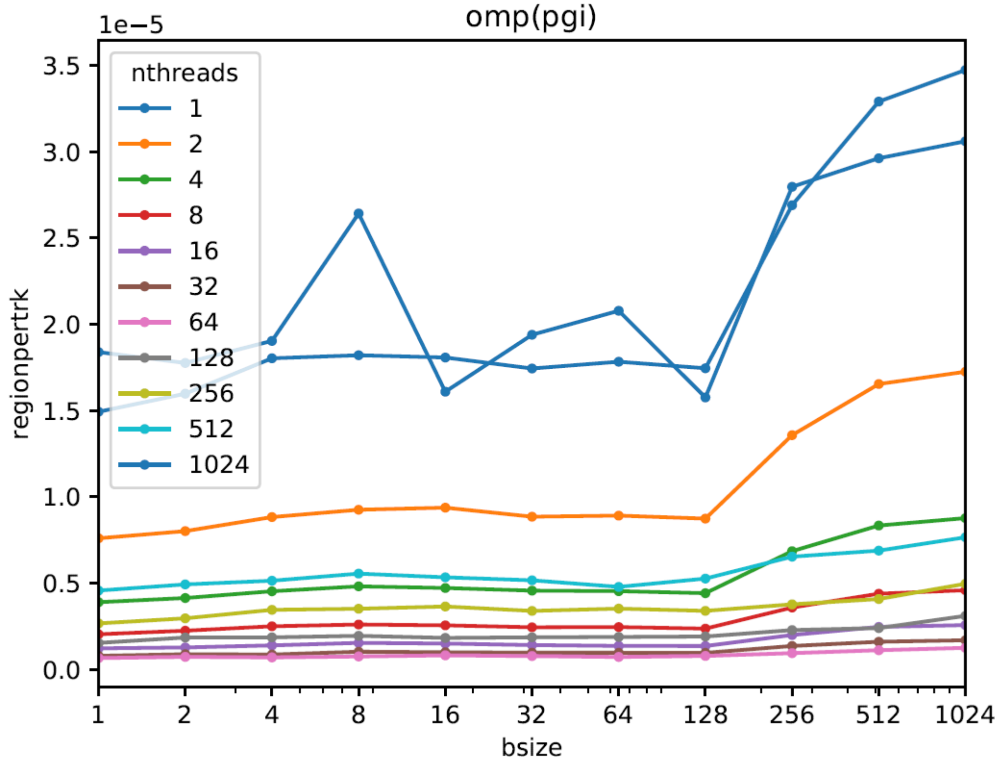


omp(icc)
1e−5
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
regionpertrk
nthreads
1
2
4
8
16
32
64
128
256
512
1024
bsize

OMP(pgi)

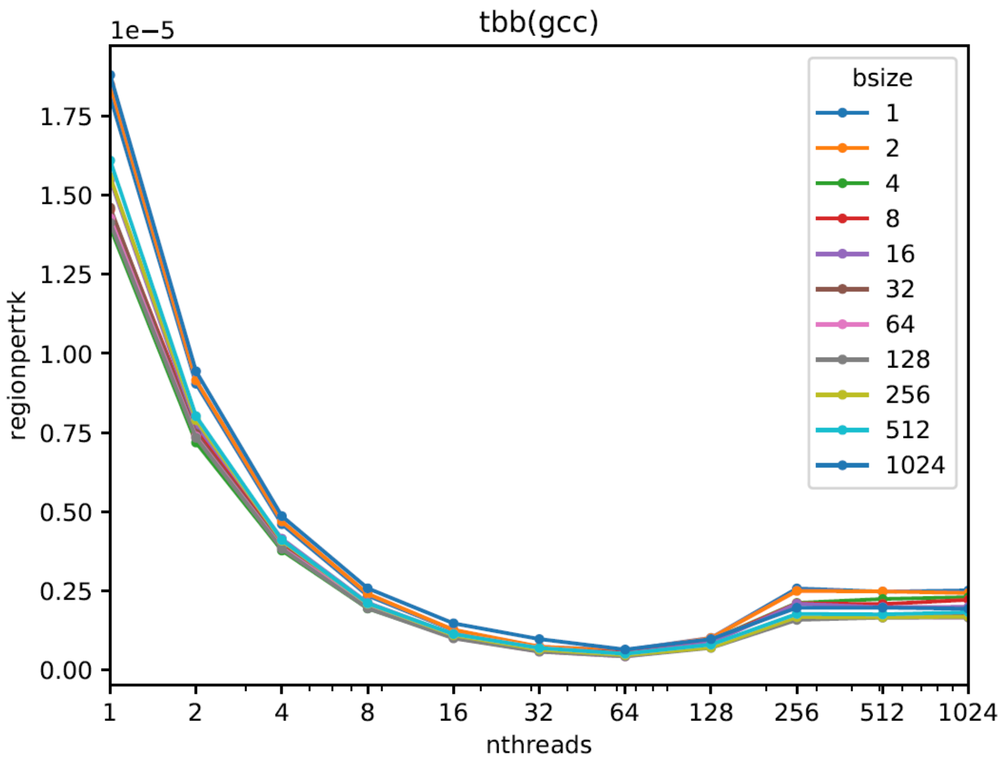


omp(pgi)
1e−5
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
regionpertrk
bsize
1
2
4
8
16
32
64
128
256
512
1024
1
2
4
8
16
32
64
128
256
512
1024
nthreads

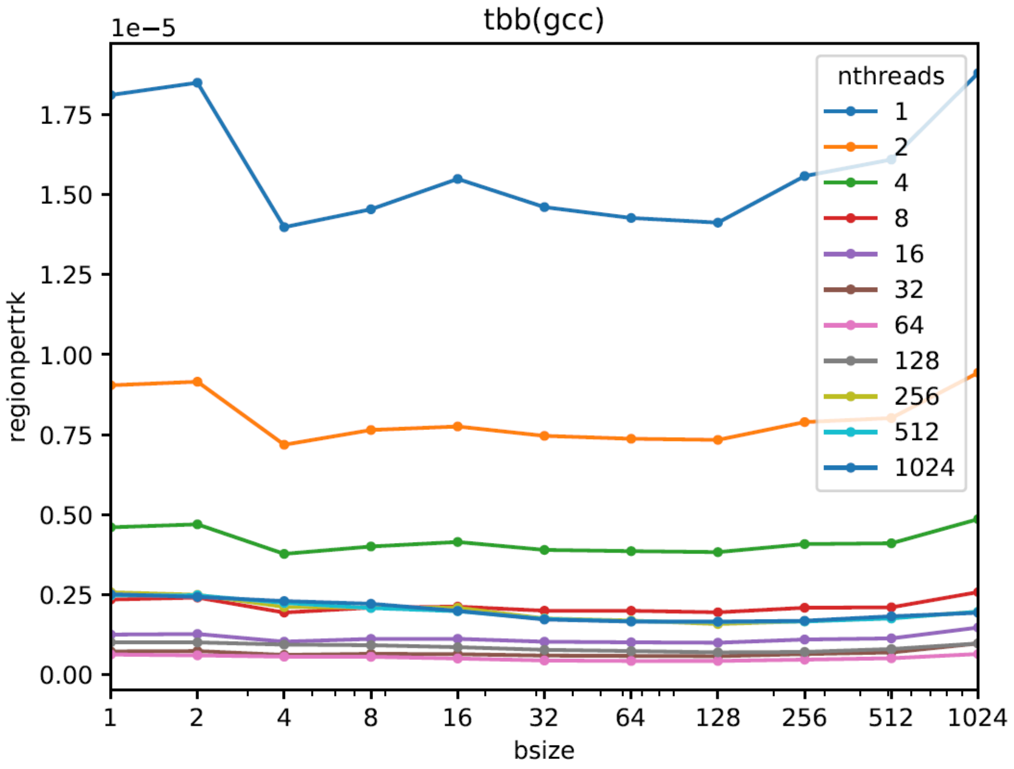


omp(pgi)
1e−5
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
regionpertrk
nthreads
1
2
4
8
16
32
64
128
256
512
1024
1
2
4
8
16
32
64
128
256
512
1024
bsize

TBB(gcc)

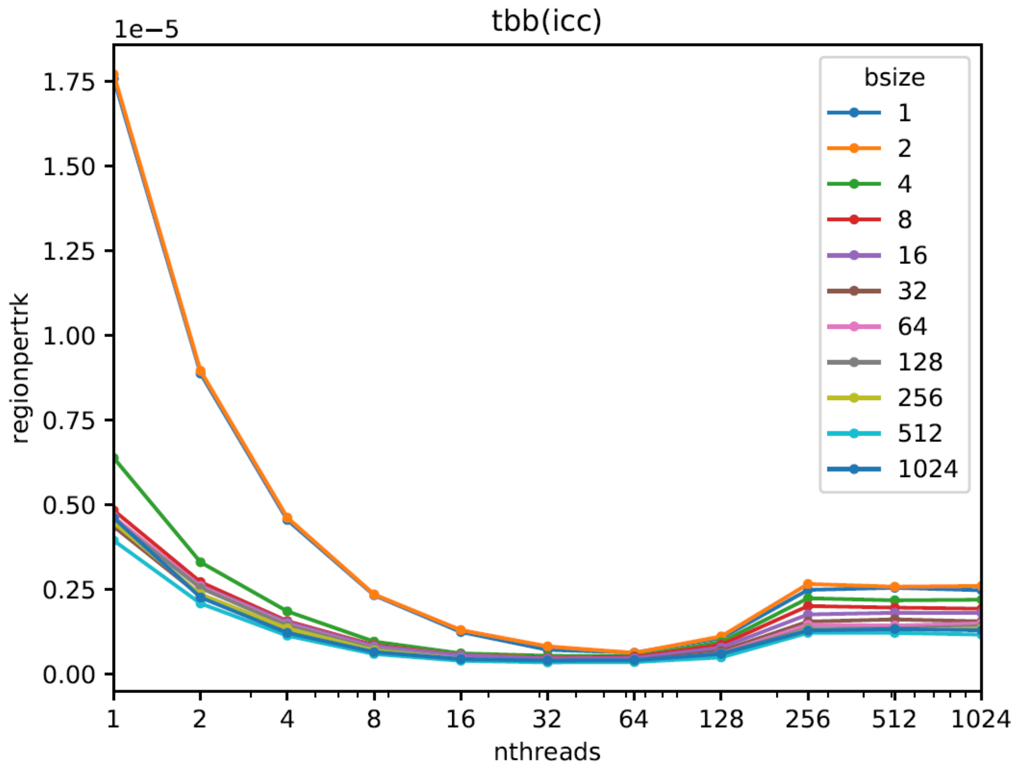


tbb(gcc)
1e−5
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
regionpertrk
1 2 4 8 16 32 64 128 256 512 1024
nthreads
bsize
1
2
4
8
16
32
64
128
256
512
1024

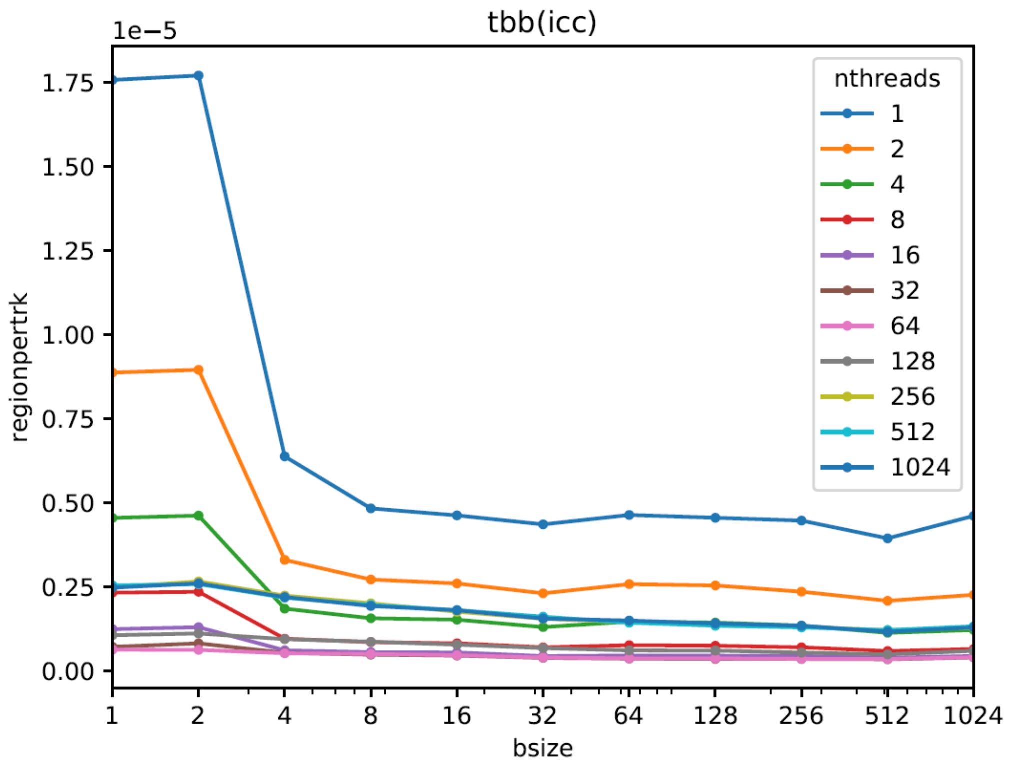


tbb(gcc)
1e−5
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
regionpertrk
1 2 4 8 16 32 64 128 256 512 1024
bsize
nthreads
1
2
4
8
16
32
64
128
256
512
1024

TBB(icc)

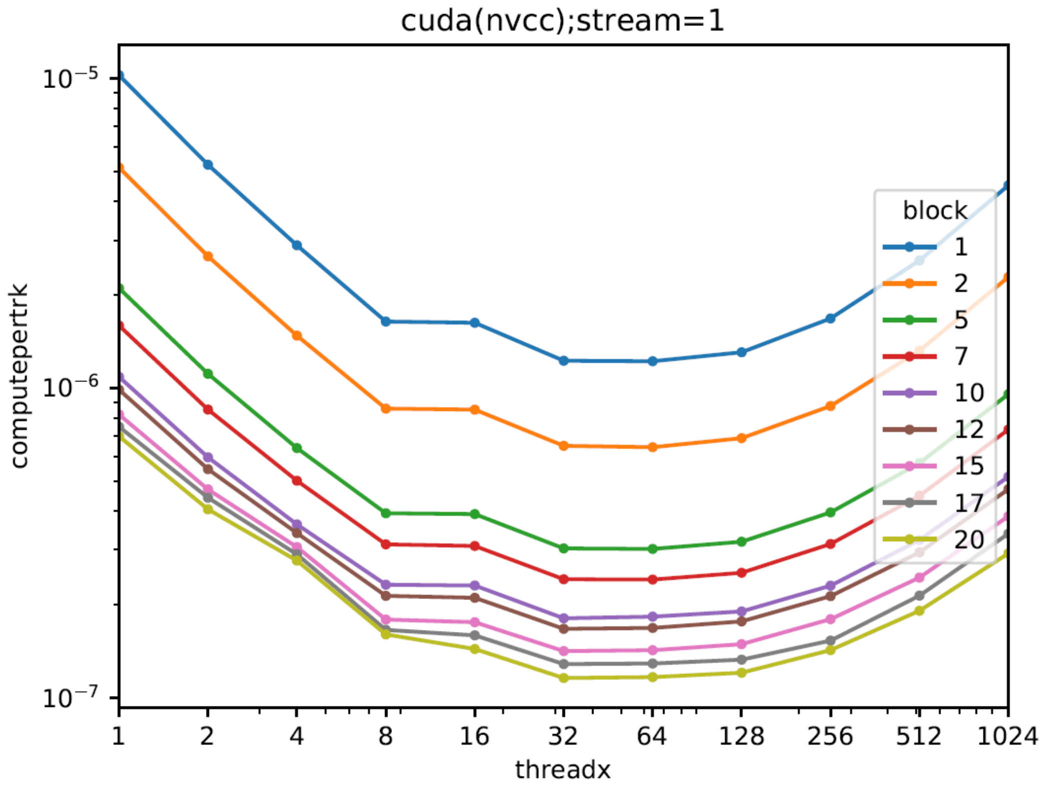


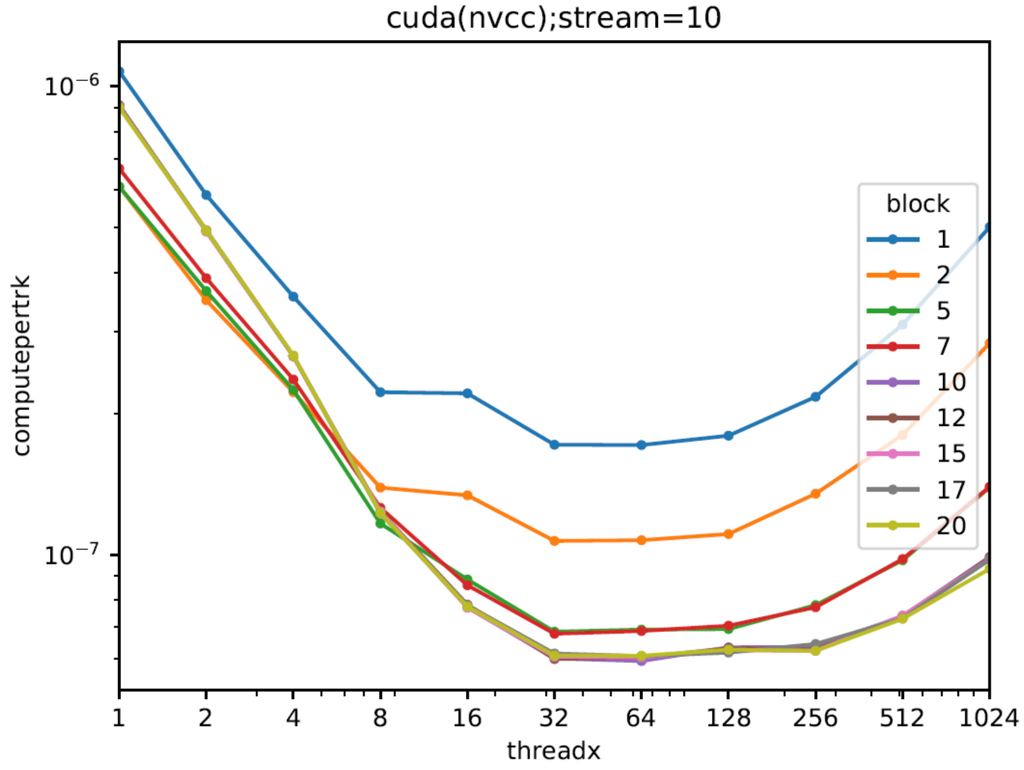
tbb(icc)
1e−5
regionpertrk
nthreads
bsize
1
2
4
8
16
32
64
128
256
512
1024



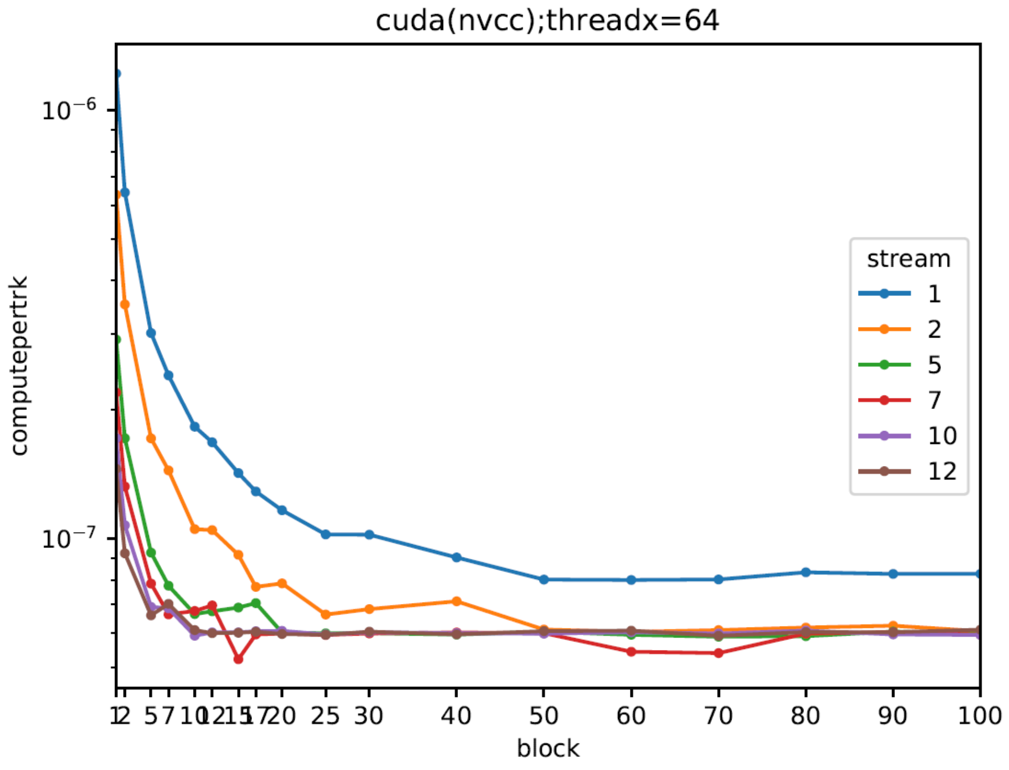
tbb(icc)
1e−5
regionpertrk
bsize
nthreads
1
2
4
8
16
32
64
128
256
512
1024

CUDA Fixed total threads to 1024/block

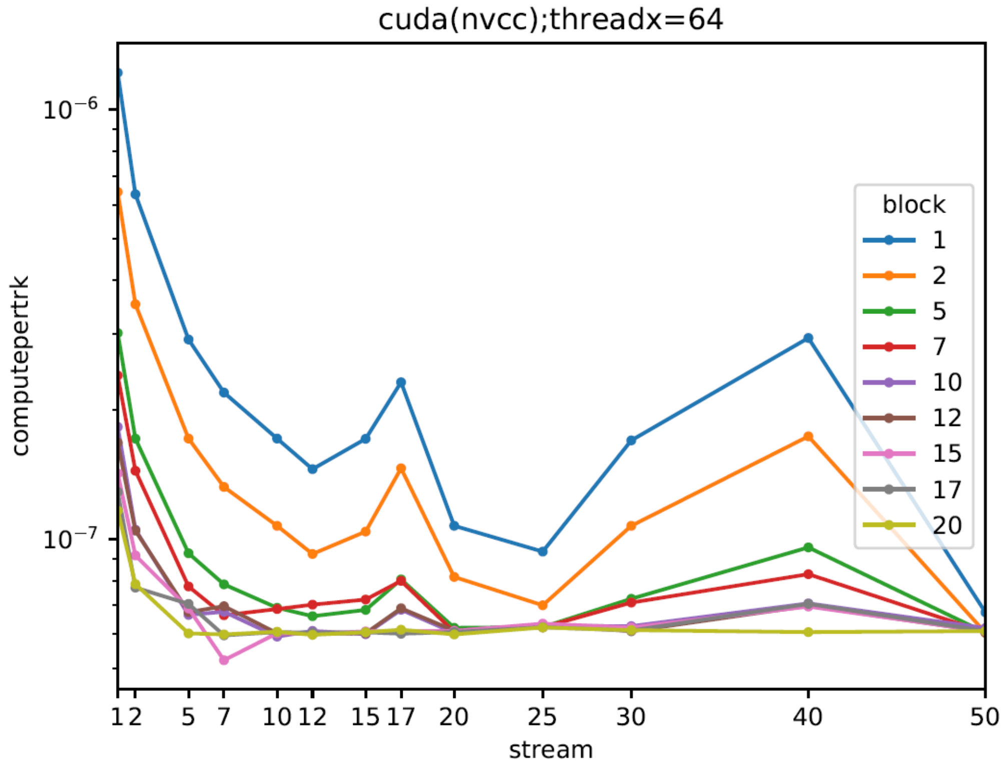




CUDA



cuda(nvcc);threadx=64
computepertrk
block
stream
1
2
5
7
10
12



cuda(nvcc);threadx=64
computepertrk
stream
block
1
2
5
7
10
12
15
17
20