Migrating CUDA to oneAPI: A Smith-Waterman Case Study

Manuel Costanzo
Enzo Rucci
Carlos García-Sánchez
Marcelo Naiouf
Manuel Prieto-Matías

IWBBIO
2022

Agenda

- Motivation and Goal
- oneAPI
- Smith-Waterman
- Experimental Work
- Experimental Results
- Conclusions

Agenda

- **Motivation and Goal**
- oneAPI
- Smith-Waterman
- Experimental Work
- Experimental Results
- Conclusions

Motivation

Heterogeneous computing and massively parallel architectures have proven to be an effective strategy for maximizing the performance and energy efficiency of computing system.

Motivation

Heterogeneous computing and massively parallel architectures have proven to be an effective strategy for maximizing the performance and energy efficiency of computing system.

Because of that, programmers typically rely on a variety of hardware, such as CPUs, GPUs, FPGAs, and other kinds of accelerator

Motivation

Heterogeneous computing and massively parallel architectures have proven to be an effective strategy for maximizing the performance and energy efficiency of computing system.

Because of that, programmers typically rely on a variety of hardware, such as CPUs, GPUs, FPGAs, and other kinds of accelerator

CPUs

Motivation

Heterogeneous computing and massively parallel architectures have proven to be an effective strategy for maximizing the performance and energy efficiency of computing system.

Because of that, programmers typically rely on a variety of hardware, such as CPUs, GPUs, FPGAs, and other kinds of accelerator

CPUs

GPUs

Motivation

Heterogeneous computing and massively parallel architectures have proven to be an effective strategy for maximizing the performance and energy efficiency of computing system.

Because of that, programmers typically rely on a variety of hardware, such as CPUs, GPUs, FPGAs, and other kinds of accelerator

CPUs

GPUs

FPGAs

Motivation

Heterogeneous computing and massively parallel architectures have proven to be an effective strategy for maximizing the performance and energy efficiency of computing system.

Because of that, programmers typically rely on a variety of hardware, such as CPUs, GPUs, FPGAs, and other kinds of accelerator

CPUs

GPUs

FPGAs

VPUs

Motivation - oneAPI

Intel recently introduced the oneAPI programming ecosystem, which provides a unified programming model for a wide range of hardware architectures.

Motivation - oneAPI

Intel recently introduced the oneAPI programming ecosystem, which provides a unified programming model for a wide range of hardware architectures.

Data Parallel C++ (DPC++)

- C++
- SYCL

Motivation - oneAPI

Intel recently introduced the oneAPI programming ecosystem, which provides a unified programming model for a wide ra

Same source code on different architectures

Data Pa
- C++
- SYCL

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

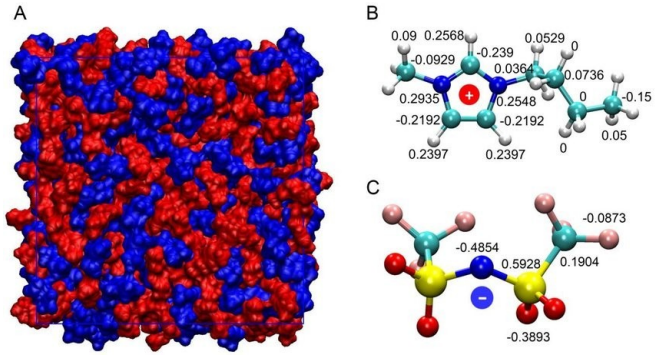
Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Molecular dynamics



Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

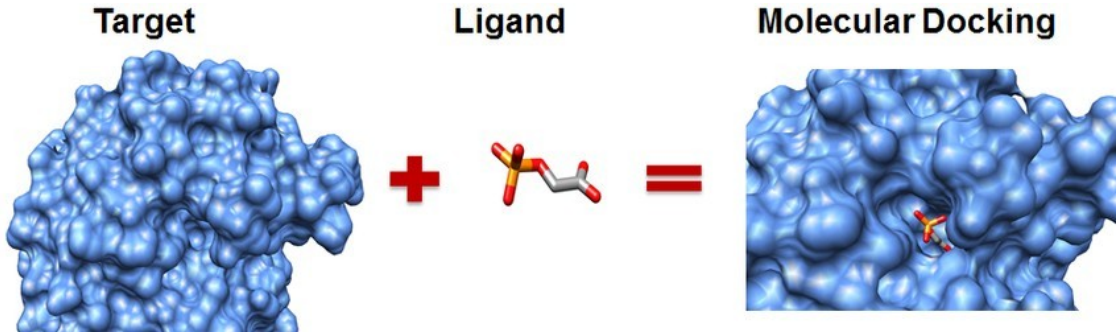
Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Molecular docking



Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

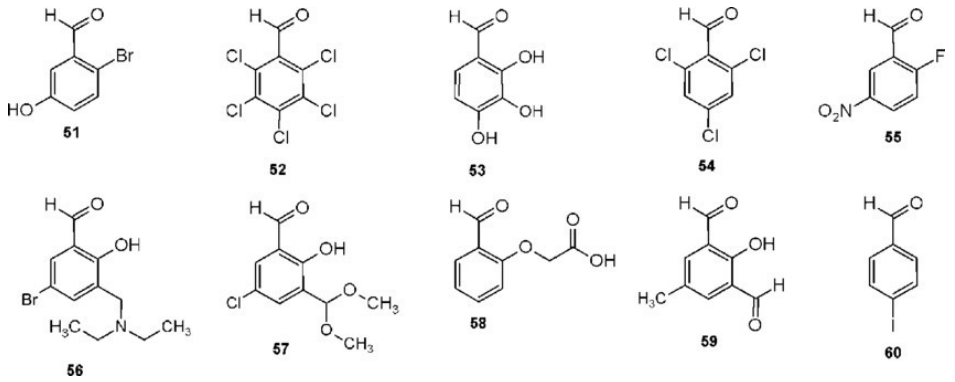
Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Prediction and searching of molecular structures



Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Sequence alignment

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Bioinformatics and Computational Biology have been exploiting GPUs for more than two

One of the most widely used implementations

Sequence alignment

Motivation - Bioinformatics

GPUs can be considered the dominant accelerator, while CUDA is the most popular programming language for them nowadays

Bioinformatics and Computational Biology have been exploiting GPUs for more than two decades.

Motivation - Bioinformatics

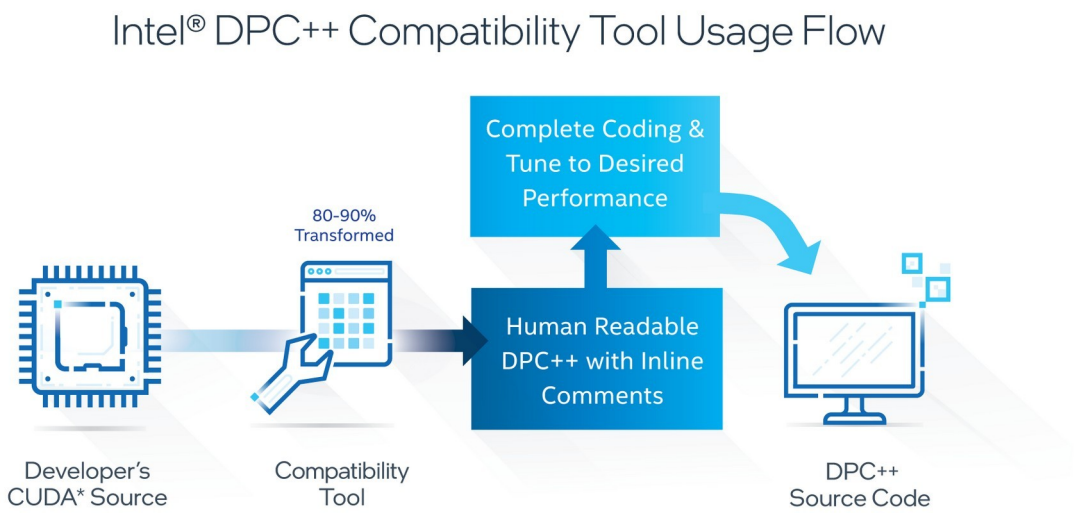
GPUs ca… …hile
CUDA is … …them
nowadays.

Bioinform…
exploiting GPUs for more than two decades.

❌ Difficulty in porting the code to other architectures.

oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.



oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

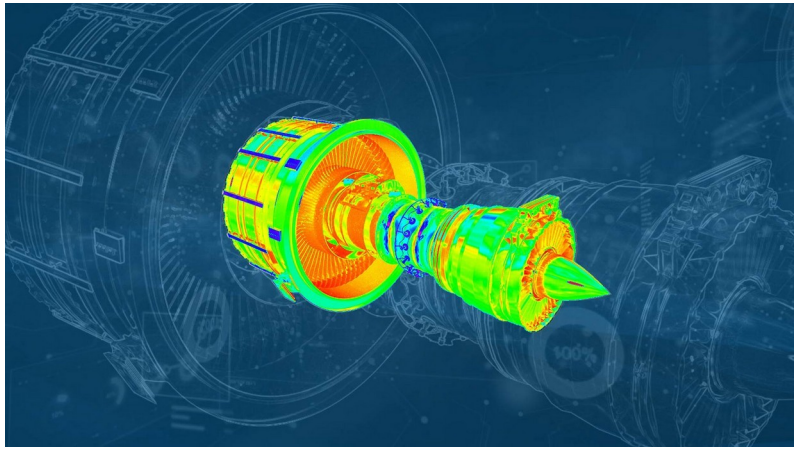
A few preliminary studies assessing the usefulness of dpct can be found

oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

A few preliminary studies assessing the usefulness of dpct can be found

Simulation



oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

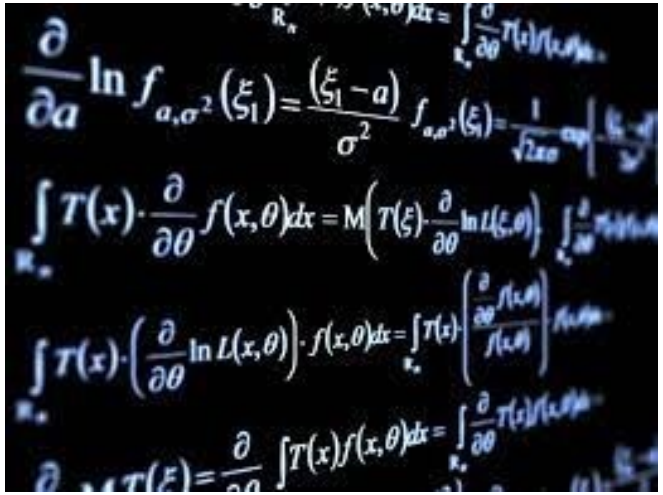
A few preliminary studies assessing the usefulness of dpct can be found

oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

A few preliminary studies assessing the usefulness of dpct can be found

Math



oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

A few preliminary studies assessing the usefulness of dpct can be found

oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

A few preliminary studies assessing the usefulness of dpct can be found

Cryptography

oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

A few preliminary studies assessing the usefulness of dpct can be found

oneAPI - DPCT

oneAPI facilitates the migration to the SYCL-based DPC++ programming language through the DPCT tool.

A few preliminary studies assessing the usefulness of dpct can be found

No study has assessed their utility in Bioinformatics

Goal

To present our experiences porting a biological software tool to DPC++ using dpct

SW#db: *a CUDA-based, memory efficient implementation of the Smith-Waterman algorithm*

Goal

To present our experiences porting a biological software tool to DPC++ using dpct

SW#db: *a CUDA-based, memory efficient implementation of the Smith-Waterman algorithm*

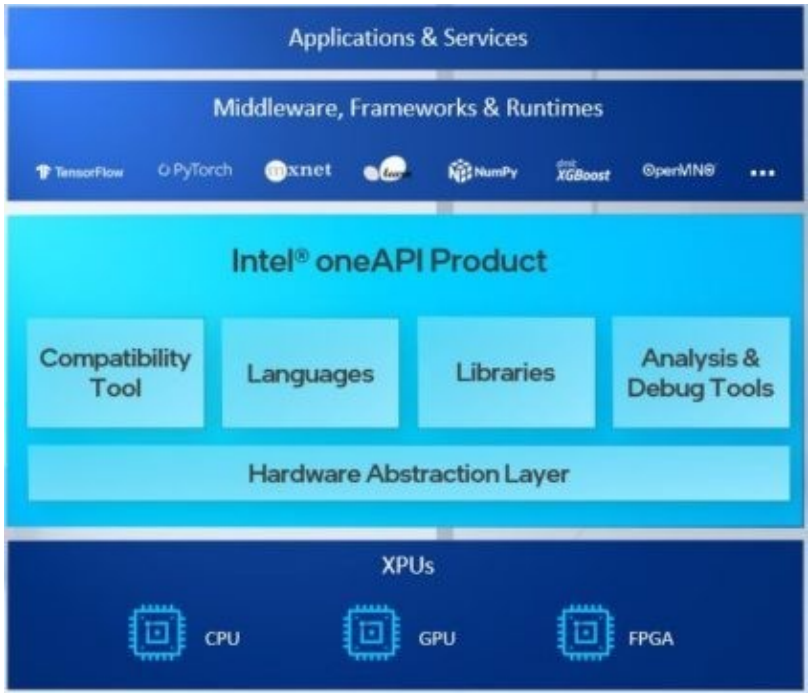
- ➔ An analysis of the dpct effectiveness for CUDA code migration
- ➔ An analysis of the DPC++ code's portability, considering different target platforms and vendors (CPU and GPUs).
- ➔ A comparison of the performance on different hardware architectures (CPU and GPUs).

Agenda

- Motivation and Goal
- **oneAPI**
- Smith-Waterman
- Experimental Work
- Experimental Results
- Conclusions

Technologies - oneAPI

oneAPI is a programming ecosystem, which provides a unified programming model for a wide range of hardware architectures



Technologies - SYCL & DPC++

SYCL allows the programmer to write host code in C++ and make it compatible to run on different architectures.

Technologies - SYCL & DPC++

SYCL allows the programmer to write host code in C++ and make it compatible to run on different architectures.

Technologies - SYCL & DPC++

SYCL allows the programmer to write host code in C++ and make it compatible to run on different architectures.

DPC++ combines the SYCL language with modified C++.

Technologies - SYCL & DPC++

SYCL allows the programmer to write host code in C++ and make it compatible to run on different architectures.

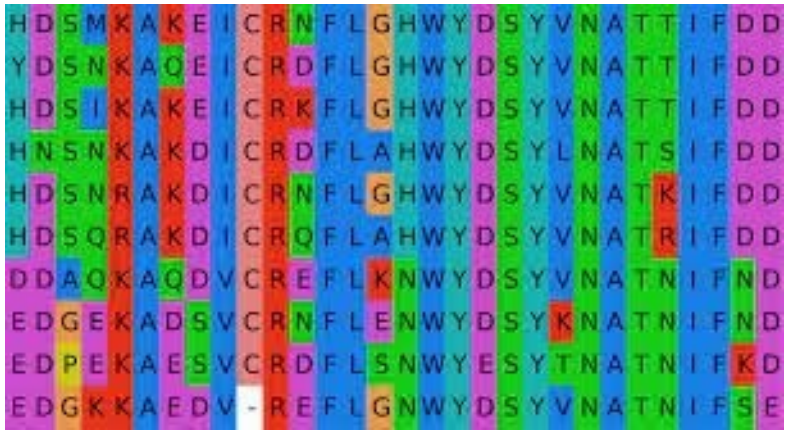
DPC++ combines the SYCL language with modified C++.

Agenda

- Motivation and Goal
- oneAPI
- **Smith-Waterman**
- Experimental Work
- Experimental Results
- Conclusions

Smith-Waterman

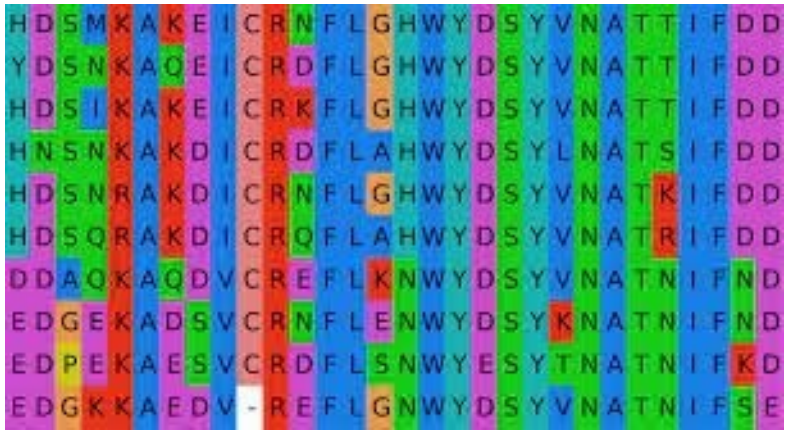
Obtains the optimal local alignment between two biological sequences.



Smith-Waterman

Obtains the optimal local alignment between two biological sequences.

It has been used as the basis for many subsequent algorithms

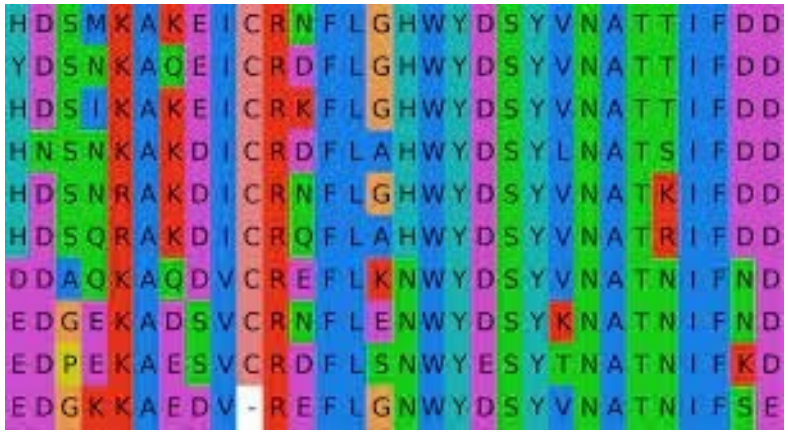


Smith-Waterman

Obtains the optimal local alignment between two biological sequences.

It has been used as the basis for many subsequent algorithms

Is often employed as a benchmark when comparing different alignment techniques



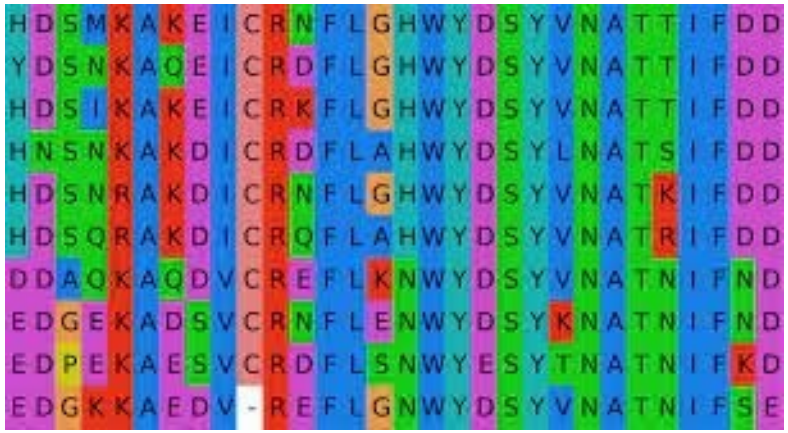
Smith-Waterman

Obtai gical
sequ

It has orithms

SW# is a CUDA tool for computing biological sequence alignments that works with both protein and DNA sequences.

Is often employed as a benchmark when comparing different alignment techniques

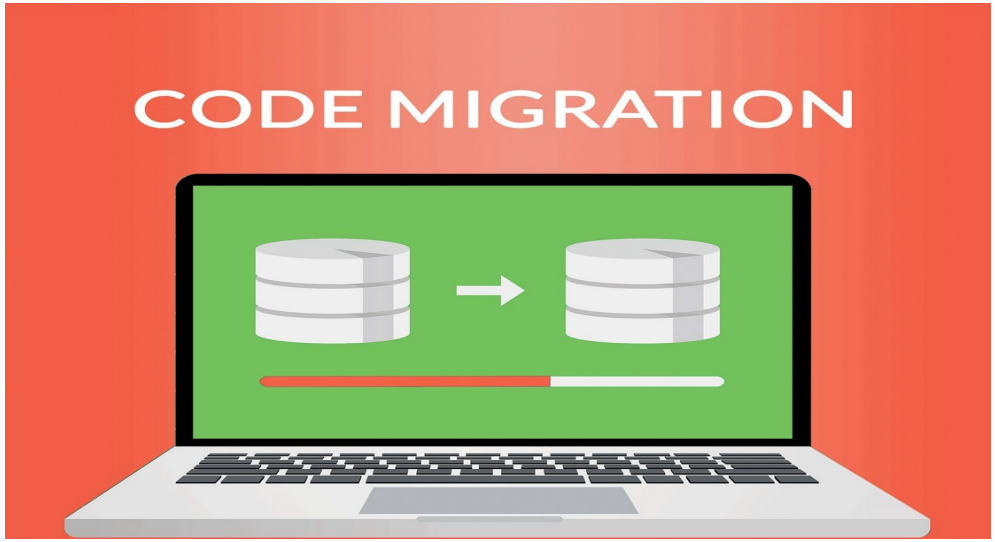


Agenda

- Motivation and Goal
- oneAPI
- Smith-Waterman
- **Experimental Work**
- Experimental Results
- Conclusions

Experimental Work

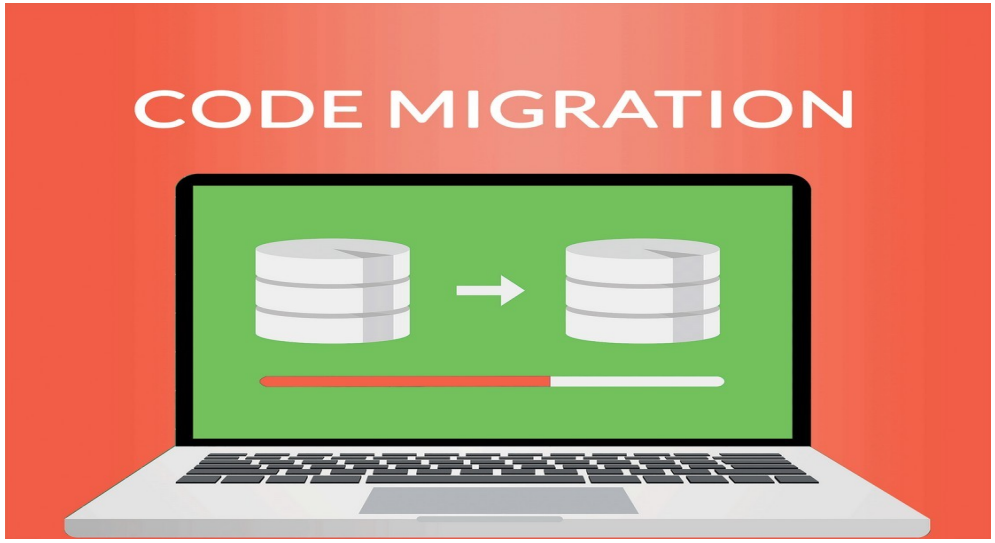
The migration process consists of 4 stages:



Experimental Work

The migration process consists of 4 stages:

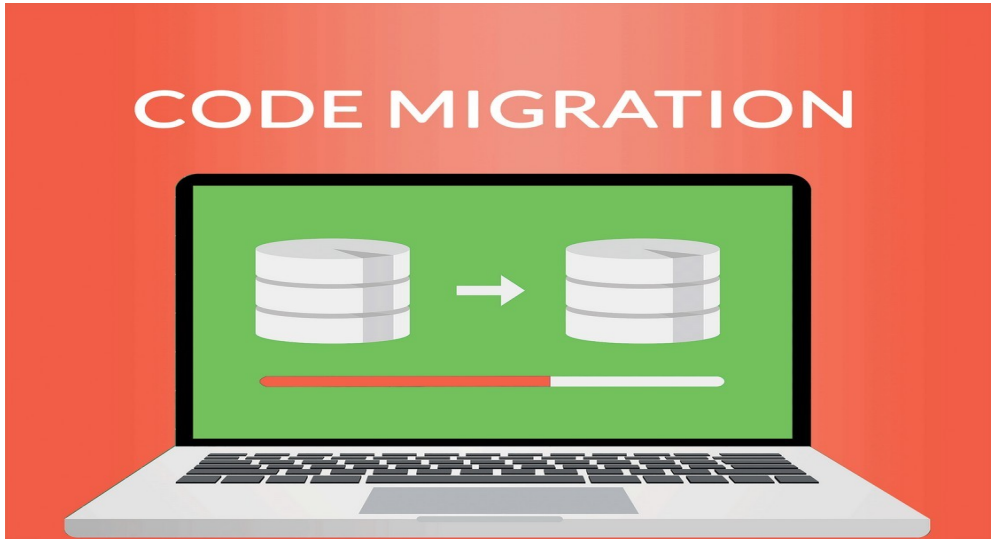
1. Use the DPCT tool for generating a first version (Stage 1).



Experimental Work

The migration process consists of 4 stages:

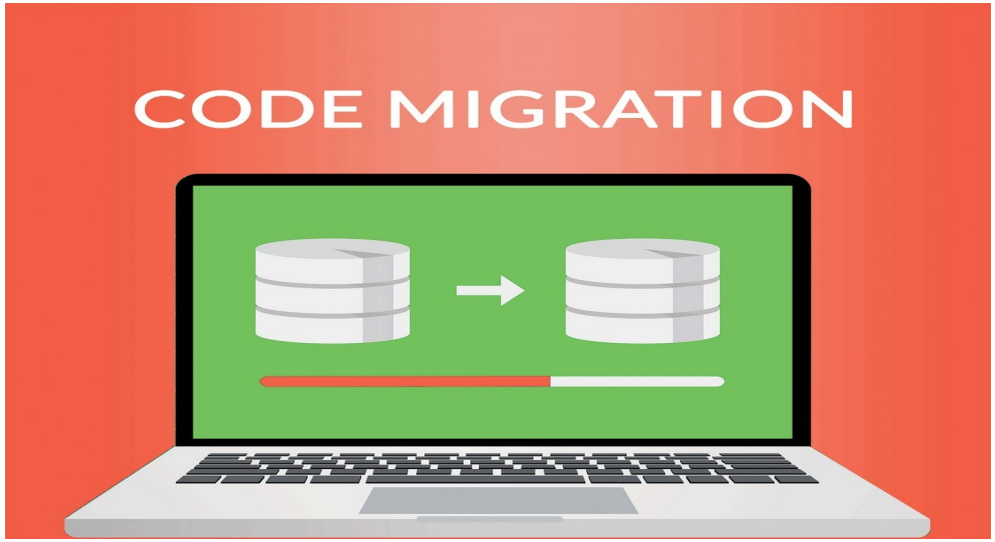
1. Use the DPCT tool for generating a first version (Stage 1).
2. Based on the DPCT alerts, modify the resulting code (Stage 2).



Experimental Work

The migration process consists of 4 stages:

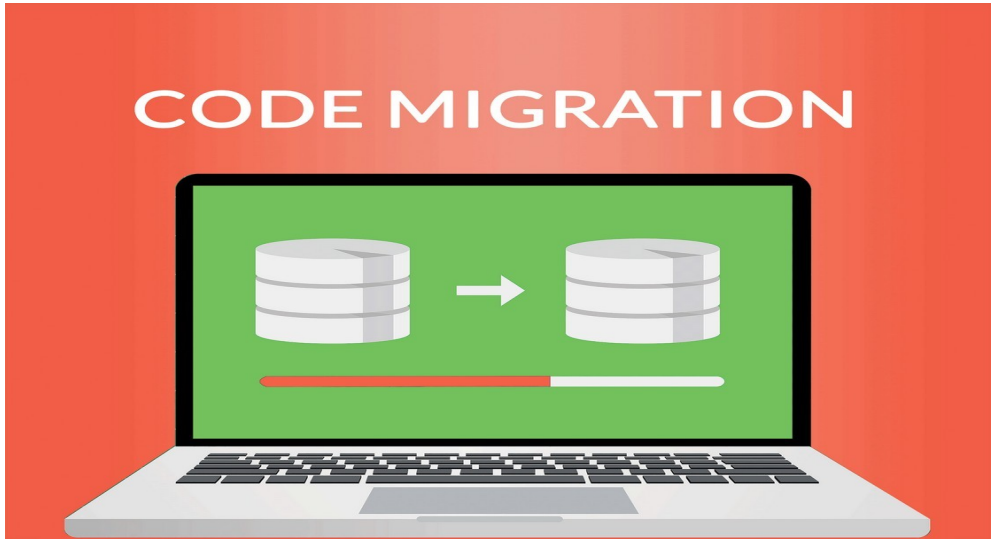
1. Use the DPCT tool for generating a first version (Stage 1).
2. Based on the DPCT alerts, modify the resulting code (Stage 2).
3. Based on runtime errors, modify the code (Stage 3).



Experimental Work

The migration process consists of 4 stages:

1. Use the DPCT tool for generating a first version (Stage 1).
2. Based on the DPCT alerts, modify the resulting code (Stage 2).
3. Based on runtime errors, modify the code (Stage 3).
4. Verify that the results are correct (Stage 4).

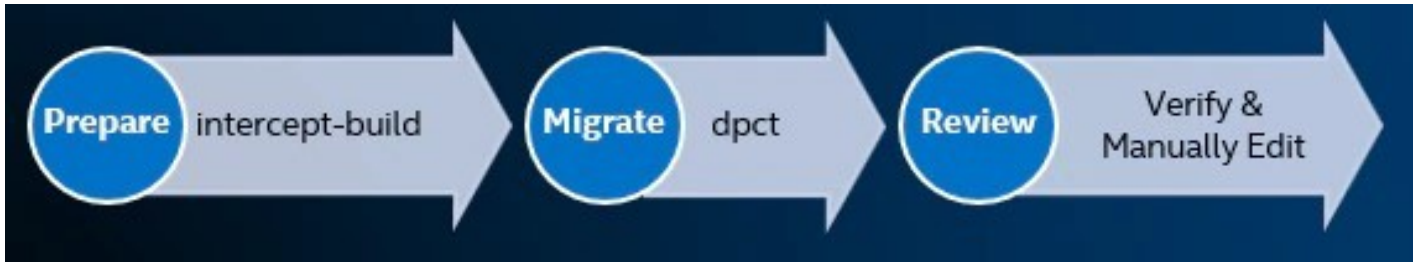


Experimental Work - Stage 1

Generate first version of DPC++ code using DPCT tool

Experimental Work - Stage 1

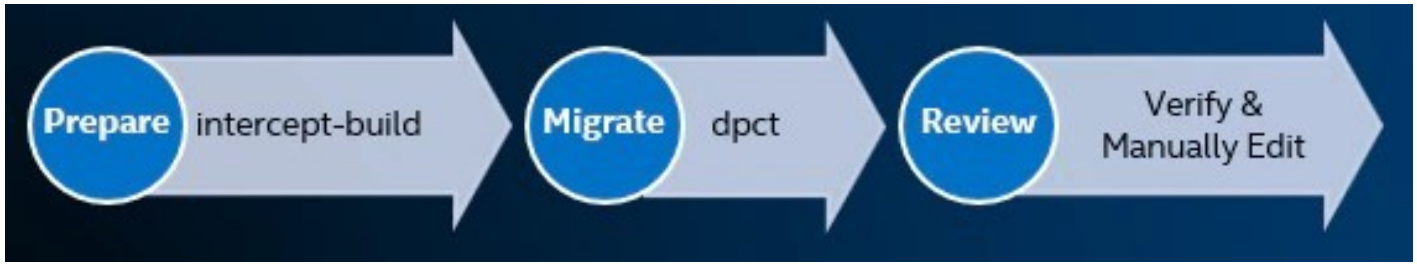
Generate first version of DPC++ code using DPCT tool



Experimental Work - Stage 1

Generate first version of DPC++ code using DPCT tool

1. *intercept-build make*

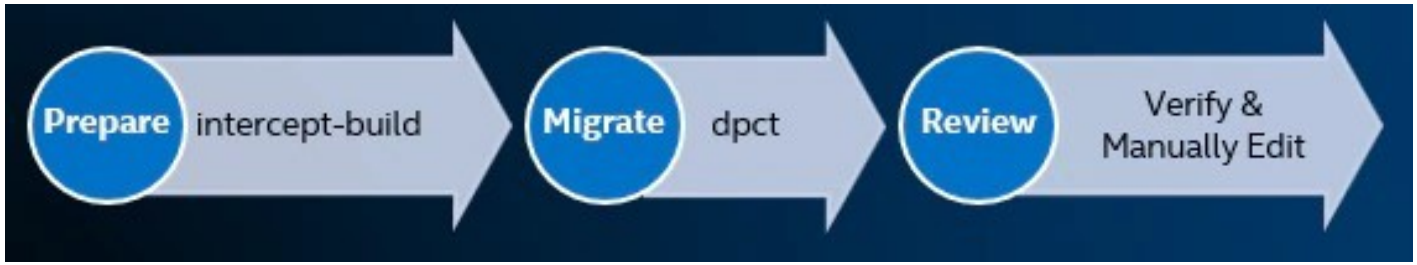


Experimental Work - Stage 1

Generate first version of DPC++ code using DPCT tool

1. *intercept-build make*

2. *dpct -p compile_commands.json -in-root=$PROJ_DIR*



Experimental Work - Stage 2

DPCT1003: Migrated API does not return error code. (, 0) is inserted. You may need to rewrite this code*

DPCT1009: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

Experimental Work - Stage 2

DPCT1003: Migrated API does not return error code. (, 0) is inserted. You may need to rewrite this code*

DPCT1009: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL(
6      cudaMalloc(
7          &valuesGpu, valuesSize
8      )
9  );
```

(a) CUDA

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL((
6   valuesGpu = (double *)sycl::malloc_device(
7      valuesSize, dpct::get_default_queue()),
8  0));
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1003: Migrated API does not return error code. (, 0) is inserted. You may need to rewrite this code*

DPCT1009: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL(
6      cudaMalloc(
7          &valuesGpu, valuesSize
8      )
9  );
```

(a) CUDA

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL((
6   valuesGpu = (double *)sycl::malloc_device(
7       valuesSize, dpct::get_default_queue()),
8   0));
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1003: Migrated API does not return error code. (, 0) is inserted. You may need to rewrite this code*

DPCT1009: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL(
6      cudaMalloc(
7          &valuesGpu, valuesSize
8      )
9  );
```

(a) CUDA

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL((
6   valuesGpu = (double *)sycl::malloc_device(
7      valuesSize, dpct::get_default_queue()),
8  0));
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1003: Migrated API does not return error code. (, 0) is inserted. You may need to rewrite this code*

DPCT1009: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL(
6      cudaMalloc(
7          &valuesGpu, valuesSize
8      )
9  );
```

(a) CUDA

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL((
6   valuesGpu = (double *)sycl::malloc_device(
7      valuesSize, dpct::get_default_queue()),
8  0));
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1003: Migrated API does not return error code. (, 0) is inserted. You may need to rewrite this code*

DPCT1009: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL(
6      cudaMalloc(
7          &valuesGpu, valuesSize
8      )
9  );
```

(a) CUDA

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL((
6   valuesGpu = (double *)sycl::malloc_device(
7      valuesSize, dpct::get_default_queue()),
8  0));
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1003: Migrated API does not return error code. (, 0) is inserted. You may need to rewrite this code*

DPCT1009: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

DPCT uses Unified Shared Memory (USM) model by default

```
1
2
3
4
5
6
7          &valuesGpu, valuesSize
8      )
9  );
```

(a) CUDA

```
6    valuesGpu = (double *)sycl::malloc_device(
7        valuesSize, dpct::get_default_queue()),
8    0));
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1003: Migrated API does not return error code. (, 0) is inserted. You may need to rewrite this code*

DPCT1009: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL(
6      cudaMalloc(
7          &valuesGpu, valuesSize
8      )
9  );
```

(a) CUDA

```
1  size_t valuesSize =
2      databaseLen * sizeof(double);
3  double* valuesGpu;
4
5  CUDA_SAFE_CALL((
6   valuesGpu = (double *)sycl::malloc_device(
7      valuesSize, dpct::get_default_queue()),
8  0));
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1005: The SYCL device version is different from CUDA Compute Compatibility. You may need to rewrite this code.

Experimental Work - Stage 2

DPCT1005: The SYCL device version is different from CUDA Compute Compatibility. You may need to rewrite this code.

```
1   cudaDeviceProp properties;
2   cudaGetDeviceProperties(
3       &properties, card
4   );
5
6   bool major = properties.major < 2;
7   int threads = major ? 64 : 128;
8   int blocks = major ? 360 : 480;
```

(a) CUDA

```
1   dpct::device_info properties;
2
3   dpct::dev_mgr::instance()
4       .get_device(card)
5       .get_device_info(properties);
6
7   bool major = false;
8   int threads = major ? 64 : 128;
9   int blocks = major ? 360 : 480;
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1005: The SYCL device version is different from CUDA Compute Compatibility. You may need to rewrite this code.

```
1  cudaDeviceProp properties;
2  cudaGetDeviceProperties(
3      &properties, card
4  );
5
6  bool major = properties.major < 2;
7  int threads = major ? 64 : 128;
8  int blocks = major ? 360 : 480;
```

(a) CUDA

```
1  dpct::device_info properties;
2
3  dpct::dev_mgr::instance()
4      .get_device(card)
5      .get_device_info(properties);
6
7  bool major = false;
8  int threads = major ? 64 : 128;
9  int blocks = major ? 360 : 480;
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1005: The SYCL device version is different from CUDA Compute Compatibility. You may need to rewrite this code.

```
1   cudaDeviceProp properties;
2   cudaGetDeviceProperties(
3       &properties, card
4   );
5
6   bool major = properties.major < 2;
7   int threads = major ? 64 : 128;
8   int blocks = major ? 360 : 480;
```

(a) CUDA

```
1   dpct::device_info properties;
2
3   dpct::dev_mgr::instance()
4       .get_device(card)
5       .get_device_info(properties);
6
7   bool major = false;
8   int threads = major ? 64 : 128;
9   int blocks = major ? 360 : 480;
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1005: The SYCL device version is different from CUDA Compute Compatibility. You may need to rewrite this code.

```
1   cudaDeviceProp properties;
2   cudaGetDeviceProperties(
3       &properties, card
4   );
5
6   bool major = properties.major < 2;
7   int threads = major ? 64 : 128;
8   int blocks = major ? 360 : 480;
```

(a) CUDA

```
1   dpct::device_info properties;
2
3   dpct::dev_mgr::instance()
4       .get_device(card)
5       .get_device_info(properties);
6
7   bool major = false;
8   int threads = major ? 64 : 128;
9   int blocks = major ? 360 : 480;
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1005: The SYCL device version is different from CUDA Compute Compatibility. You may need to rewrite this code.

```
1   cudaDeviceProp properties;
2   cudaGetDeviceProperties(
3       &properties, card
4   );
5
6   bool major = properties.major < 2;
7   int threads = major ? 64 : 128;
8   int blocks = major ? 360 : 480;
```

(a) CUDA

```
1   dpct::device_info properties;
2
3   dpct::dev_mgr::instance()
4       .get_device(card)
5       .get_device_info(properties);
6
7   bool major = false;
8   int threads = major ? 64 : 128;
9   int blocks = major ? 360 : 480;
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1049: The workgroup size passed to the SYCL kernel may exceed the limit. To get the device limit, query info::device::max_work_group_size. Adjust the workgroup size if needed.

Experimental Work - Stage 2

DPCT1049: The workgroup size passed to the SYCL kernel may exceed the limit. To get the device limit, query info::device::max_work_group_size. Adjust the workgroup size if needed.

```
1   solveShort<<<blocks, threads>>>(...);
```

(a) CUDA

```
1   dpct::get_default_queue()
2       .submit([&](sycl::handler &cgh) {
3    ...
4   cgh.parallel_for(
5     sycl::nd_range<3>
6       (sycl::range<3>(1, 1, blocks) *
7       sycl::range<3>(1, 1, threads),
8       sycl::range<3>(1, 1, threads)),
9     [=](sycl::nd_item<3> item_ct1) {
10      solveShort(...);
11    });
12  });
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1049: The workgroup size passed to the SYCL kernel may exceed the limit. To get the device limit, query info::device::max_work_group_size. Adjust the workgroup size if needed.

```
1  solveShort<<<blocks, threads>>>(...);
```

(a) CUDA

```
1   dpct::get_default_queue()
2       .submit([&](sycl::handler &cgh) {
3     ...
4   cgh.parallel_for(
5     sycl::nd_range<3>
6       (sycl::range<3>(1, 1, blocks) *
7       sycl::range<3>(1, 1, threads),
8       sycl::range<3>(1, 1, threads)),
9     [=](sycl::nd_item<3> item_ct1) {
10      solveShort(...);
11    });
12  });
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1049: The workgroup size passed to the SYCL kernel may exceed the limit. To get the device limit, query info::device::max_work_group_size. Adjust the workgroup size if needed.

```
1   solveShort<<<blocks, threads>>>(...);
```

(a) CUDA

```
1   dpct::get_default_queue()
2       .submit([&](sycl::handler &cgh) {
3    ...
4   cgh.parallel_for(
5     sycl::nd_range<3>
6       (sycl::range<3>(1, 1, blocks) *
7       sycl::range<3>(1, 1, threads),
8       sycl::range<3>(1, 1, threads)),
9     [=](sycl::nd_item<3> item_ct1) {
10      solveShort(...);
11    });
12  });
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1065: Consider replacing sycl::nd_item::barrier() with sycl::nd_item::barrier(sycl::access::fence_space::local_space) for better performance if there is no access to global memory.

Experimental Work - Stage 2

DPCT1065: Consider replacing sycl::nd_item::barrier() with sycl::nd_item::barrier(sycl::access::fence_space::local_space) for better performance if there is no access to global memory.

```
1   ...
2   __syncthreads();
3   ...
```

(a) CUDA

```
1   ...
2   item_ct1.barrier();
3   ...
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1065: Consider replacing sycl::nd_item::barrier() with sycl::nd_item::barrier(sycl::access::fence_space::local_space) for better performance if there is no access to global memory.

```
1  ...
2  __syncthreads();
3  ...
```

(a) CUDA

```
1
2  item_ct1.barrier();
3
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1065: Consider replacing sycl::nd_item::barrier() with sycl::nd_item::barrier(sycl::access::fence_space::local_space) for better performance if there is no access to global memory.

```
1   ...
2   __syncthreads();
3   ...
```

(a) CUDA

```
1   ...
2   item_ct1.barrier();
3   ...
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1065: Consider replacing sycl::nd_item::barrier() with sycl::nd_item::barrier(sycl::access::fence_space::local_space) for better performance if there is no access to global memory.

item_ct1.barrier(sycl::access::fence_space::local_space)

```
1   ...
2   __syncthreads();
3   ...
```

(a) CUDA

```
1   ...
2   item_ct1.barrier();
3   ...
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```cuda
texture<char> colTexture;

int colSize = colsGpu * sizeof(char);
char *colGpu;
cudaMalloc(&colGpu, colSize);
cudaMemcpy(colGpu, colCpu,
    colSize, TO_GPU);
cudaBindTexture(NULL, colTexture,
    colGpu, colSize);

char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```oneapi
//dpct::image_wrapper<char, 1> colTexture;
dpct::image_wrapper<sycl::char4, 1> colTexture;
int colSize = colsGpu * sizeof(char);
char* colGpu;

colGpu = (char *)sycl::malloc_device(
    colSize, dpct::get_default_queue());

dpct::get_default_queue()
    .memcpy(colGpu, colCpu, colSize).wait();

colTexture.attach(colGpu, colSize);

// DIV 4 y MOD 3
char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```cuda
texture<char> colTexture;

int colSize = colsGpu * sizeof(char);
char *colGpu;
cudaMalloc(&colGpu, colSize);
cudaMemcpy(colGpu, colCpu,
    colSize, TO_GPU);
cudaBindTexture(NULL, colTexture,
    colGpu, colSize);

char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```oneapi
//dpct::image_wrapper<char, 1> colTexture;
dpct::image_wrapper<sycl::char4, 1> colTexture;
int colSize = colsGpu * sizeof(char);
char* colGpu;

colGpu = (char *)sycl::malloc_device(
    colSize, dpct::get_default_queue());

dpct::get_default_queue()
    .memcpy(colGpu, colCpu, colSize).wait();

colTexture.attach(colGpu, colSize);

// DIV 4 y MOD 3
char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```cuda
texture<char> colTexture;

int colSize = colsGpu * sizeof(char);
char *colGpu;
cudaMalloc(&colGpu, colSize);
cudaMemcpy(colGpu, colCpu,
    colSize, TO_GPU);
cudaBindTexture(NULL, colTexture,
    colGpu, colSize);

char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```oneapi
//dpct::image_wrapper<char, 1> colTexture;
dpct::image_wrapper<sycl::char4, 1> colTexture;
int colSize = colsGpu * sizeof(char);
char* colGpu;

colGpu = (char *)sycl::malloc_device(
    colSize, dpct::get_default_queue());

dpct::get_default_queue()
    .memcpy(colGpu, colCpu, colSize).wait();

colTexture.attach(colGpu, colSize);

// DIV 4 y MOD 3
char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```
1   texture<char> colTexture;
2
3   int colSize = colsGpu * sizeof(char);
4   char *colGpu;
5   cudaMalloc(&colGpu, colSize);
6   cudaMemcpy(colGpu, colCpu,
7       colSize, TO_GPU);
8   cudaBindTexture(NULL, colTexture,
9       colGpu, colSize);
10
11  char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```
1   //dpct::image_wrapper<char, 1> colTexture;
2   dpct::image_wrapper<sycl::char4, 1> colTexture;
3   int colSize = colsGpu * sizeof(char);
4   char* colGpu;
5
6   colGpu = (char *)sycl::malloc_device(
7       colSize, dpct::get_default_queue());
8
9   dpct::get_default_queue()
10      .memcpy(colGpu, colCpu, colSize).wait();
11
12  colTexture.attach(colGpu, colSize);
13
14  // DIV 4 y MOD 3
15  char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```cuda
texture<char> colTexture;

int colSize = colsGpu * sizeof(char);
char *colGpu;
cudaMalloc(&colGpu, colSize);
cudaMemcpy(colGpu, colCpu,
    colSize, TO_GPU);
cudaBindTexture(NULL, colTexture,
    colGpu, colSize);

char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```oneapi
//dpct::image_wrapper<char, 1> colTexture;
dpct::image_wrapper<sycl::char4, 1> colTexture;
int colSize = colsGpu * sizeof(char);
char* colGpu;

colGpu = (char *)sycl::malloc_device(
    colSize, dpct::get_default_queue());

dpct::get_default_queue()
    .memcpy(colGpu, colCpu, colSize).wait();

colTexture.attach(colGpu, colSize);

// DIV 4 y MOD 3
char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```
1   texture<char> colTexture;
2
3   int colSize = colsGpu * sizeof(char);
4   char *colGpu;
5   cudaMalloc(&colGpu, colSize);
6   cudaMemcpy(colGpu, colCpu,
7       colSize, TO_GPU);
8   cudaBindTexture(NULL, colTexture,
9       colGpu, colSize);
10
11  char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```
1   //dpct::image_wrapper<char, 1> colTexture;
2   dpct::image_wrapper<sycl::char4, 1> colTexture;
3   int colSize = colsGpu * sizeof(char);
4   char* colGpu;
5
6   colGpu = (char *)sycl::malloc_device(
7       colSize, dpct::get_default_queue());
8
9   dpct::get_default_queue()
10      .memcpy(colGpu, colCpu, colSize).wait();
11
12  colTexture.attach(colGpu, colSize);
13
14  // DIV 4 y MOD 3
15  char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```cuda
texture<char> colTexture;

int colSize = colsGpu * sizeof(char);
char *colGpu;
cudaMalloc(&colGpu, colSize);
cudaMemcpy(colGpu, colCpu,
    colSize, TO_GPU);
cudaBindTexture(NULL, colTexture,
    colGpu, colSize);

char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```oneapi
//dpct::image_wrapper<char, 1> colTexture;
dpct::image_wrapper<sycl::char4, 1> colTexture;
int colSize = colsGpu * sizeof(char);
char* colGpu;

colGpu = (char *)sycl::malloc_device(
    colSize, dpct::get_default_queue());

dpct::get_default_queue()
    .memcpy(colGpu, colCpu, colSize).wait();

colTexture.attach(colGpu, colSize);

// DIV 4 y MOD 3
char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```cuda
texture<char> colTexture;

int colSize = colsGpu * sizeof(char);
char *colGpu;
cudaMalloc(&colGpu, colSize);
cudaMemcpy(colGpu, colCpu,
    colSize, TO_GPU);
cudaBindTexture(NULL, colTexture,
    colGpu, colSize);

char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```oneapi
//dpct::image_wrapper<char, 1> colTexture;
dpct::image_wrapper<sycl::char4, 1> colTexture;
int colSize = colsGpu * sizeof(char);
char* colGpu;

colGpu = (char *)sycl::malloc_device(
    colSize, dpct::get_default_queue());

dpct::get_default_queue()
    .memcpy(colGpu, colCpu, colSize).wait();

colTexture.attach(colGpu, colSize);

// DIV 4 y MOD 3
char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 2

DPCT1059: SYCL only supports 4-channel image format. Adjust the code.

```cuda
texture<char> colTexture;

int colSize = colsGpu * sizeof(char);
char *colGpu;
cudaMalloc(&colGpu, colSize);
cudaMemcpy(colGpu, colCpu,
    colSize, TO_GPU);
cudaBindTexture(NULL, colTexture,
    colGpu, colSize);

char v = tex1Dfetch(colTexture, 10);
```

(a) CUDA

```oneapi
//dpct::image_wrapper<char, 1> colTexture;
dpct::image_wrapper<sycl::char4, 1> colTexture;
int colSize = colsGpu * sizeof(char);
char* colGpu;

colGpu = (char *)sycl::malloc_device(
    colSize, dpct::get_default_queue());

dpct::get_default_queue()
    .memcpy(colGpu, colCpu, colSize).wait();

colTexture.attach(colGpu, colSize);

// DIV 4 y MOD 3
char v = colTexture.read(10 >> 2)[10 & 3];
```

(b) oneAPI

Experimental Work - Stage 3

For a 1D/2D image/image array, the width must be a Value >= 1 and <= CL_DEVICE_IMAGE2D_MAX_WIDTH.

Experimental Work - Stage 3

For a 1D/2D image/image array, the width must be a Value >= 1 and <= CL_DEVICE_IMAGE2D_MAX_WIDTH.

```
1  texture<int, 2,
2    cudaReadModeElementType> seqsTexture;
3
4  cudaArray *sequencesGpu;
5  cudaChannelFormatDesc channel =
6    seqsTexture.channelDesc;
7  cudaMallocArray(&sequencesGpu,
8      &channel, sequencesCols, sequencesRows);
9  cudaMemcpyToArray(sequencesGpu, 0, 0,
10     sequences, sequencesSize, TO_GPU);
11 cudaBindTextureToArray(
12     seqsTexture, sequencesGpu);
```

(a) CUDA

```
1  static int *seqsGpu;
2
3  seqsGpu = (int *)sycl::malloc_device(
4   sequencesCols * sequencesRows * sizeof(int),
5   dpct::get_default_queue());
6
7  dpct::get_default_queue()
8   .memcpy(seqsGpu, sequences, sequencesCols *
9         sequencesRows * sizeof(int))
10   .wait();
```

(b) oneAPI

```
1  int columnCodes = tex2D(
2    seqsTexture, colOff, j + rowOff);
```

(a) CUDA

```
1  int columnCodes =
2      seqsGpu[(j + rowOff) * sequencesCols + colOff];
```

(b) oneAPI

Experimental Work - Stage 3

For a 1D/2D image/image array, the width must be a Value >= 1 and <= CL_DEVICE_IMAGE2D_MAX_WIDTH.

```
1  texture<int, 2,
2    cudaReadModeElementType> seqsTexture;
3
4  cudaArray *sequencesGpu;
5  cudaChannelFormatDesc channel =
6    seqsTexture.channelDesc;
7  cudaMallocArray(&sequencesGpu,
8      &channel, sequencesCols, sequencesRows);
9  cudaMemcpyToArray(sequencesGpu, 0, 0,
10     sequences, sequencesSize, TO_GPU);
11 cudaBindTextureToArray(
12     seqsTexture, sequencesGpu);
```

(a) CUDA

```
1  static int *seqsGpu;
2
3  seqsGpu = (int *)sycl::malloc_device(
4   sequencesCols * sequencesRows * sizeof(int),
5   dpct::get_default_queue());
6
7  dpct::get_default_queue()
8   .memcpy(seqsGpu, sequences, sequencesCols *
9         sequencesRows * sizeof(int))
10   .wait();
```

(b) oneAPI

```
1  int columnCodes = tex2D(
2    seqsTexture, colOff, j + rowOff);
```

(a) CUDA

```
1  int columnCodes =
2      seqsGpu[(j + rowOff) * sequencesCols + colOff];
```

(b) oneAPI

Experimental Work - Stage 3

For a 1D/2D image/image array, the width must be a Value >= 1 and <= CL_DEVICE_IMAGE2D_MAX_WIDTH.

```
1  texture<int, 2,
2    cudaReadModeElementType> seqsTexture;
3
4  cudaArray *sequencesGpu;
5  cudaChannelFormatDesc channel =
6    seqsTexture.channelDesc;
7  cudaMallocArray(&sequencesGpu,
8      &channel, sequencesCols, sequencesRows);
9  cudaMemcpyToArray(sequencesGpu, 0, 0,
10     sequences, sequencesSize, TO_GPU);
11 cudaBindTextureToArray(
12     seqsTexture, sequencesGpu);
```

(a) CUDA

```
1  static int *seqsGpu;
2
3  seqsGpu = (int *)sycl::malloc_device(
4   sequencesCols * sequencesRows * sizeof(int),
5   dpct::get_default_queue());
6
7  dpct::get_default_queue()
8   .memcpy(seqsGpu, sequences, sequencesCols *
9         sequencesRows * sizeof(int))
10   .wait();
```

(b) oneAPI

```
1  int columnCodes = tex2D(
2    seqsTexture, colOff, j + rowOff);
```

(a) CUDA

```
1  int columnCodes =
2      seqsGpu[(j + rowOff) * sequencesCols + colOff];
```

(b) oneAPI

Experimental Work - Stage 3

For a 1D/2D image/image array, the width must be a Value >= 1 and <= CL_DEVICE_IMAGE2D_MAX_WIDTH.

```
1   texture<int, 2,
2     cudaReadModeElementType> seqsTexture;
3
4   cudaArray *sequencesGpu;
5   cudaChannelFormatDesc channel =
6     seqsTexture.channelDesc;
7   cudaMallocArray(&sequencesGpu,
8       &channel, sequencesCols, sequencesRows);
9   cudaMemcpyToArray(sequencesGpu, 0, 0,
10      sequences, sequencesSize, TO_GPU);
11  cudaBindTextureToArray(
12      seqsTexture, sequencesGpu);
```

(a) CUDA

```
1   static int *seqsGpu;
2
3   seqsGpu = (int *)sycl::malloc_device(
4    sequencesCols * sequencesRows * sizeof(int),
5    dpct::get_default_queue());
6
7   dpct::get_default_queue()
8    .memcpy(seqsGpu, sequences, sequencesCols *
9           sequencesRows * sizeof(int))
10   .wait();
```

(b) oneAPI

```
1   int columnCodes = tex2D(
2     seqsTexture, colOff, j + rowOff);
```

(a) CUDA

```
1   int columnCodes =
2       seqsGpu[(j + rowOff) * sequencesCols + colOff];
```

(b) oneAPI

Experimental Work - Stage 3

For a 1D/2D image/image array, the width must be a Value >= 1 and <= CL_DEVICE_IMAGE2D_MAX_WIDTH.

```
1  texture<int, 2,
2    cudaReadModeElementType> seqsTexture;
3
4  cudaArray *sequencesGpu;
5  cudaChannelFormatDesc channel =
6    seqsTexture.channelDesc;
7  cudaMallocArray(&sequencesGpu,
8      &channel, sequencesCols, sequencesRows);
9  cudaMemcpyToArray(sequencesGpu, 0, 0,
10     sequences, sequencesSize, TO_GPU);
11 cudaBindTextureToArray(
12     seqsTexture, sequencesGpu);
```

(a) CUDA

```
1  static int *seqsGpu;
2
3  seqsGpu = (int *)sycl::malloc_device(
4   sequencesCols * sequencesRows * sizeof(int),
5   dpct::get_default_queue());
6
7  dpct::get_default_queue()
8   .memcpy(seqsGpu, sequences, sequencesCols *
9         sequencesRows * sizeof(int))
10   .wait();
```

(b) oneAPI

```
1  int columnCodes = tex2D(
2    seqsTexture, colOff, j + rowOff);
```

(a) CUDA

```
1  int columnCodes =
2      seqsGpu[(j + rowOff) * sequencesCols + colOff];
```

(b) oneAPI

Experimental Work - Stage 3

For a 1D/2D image/image array, the width must be a Value >= 1 and <= CL_DEVICE_IMAGE2D_MAX_WIDTH.

```
1  texture<int, 2,
2    cudaReadModeElementType> seqsTexture;
3
4  cudaArray *sequencesGpu;
5  cudaChannelFormatDesc channel =
6    seqsTexture.channelDesc;
7  cudaMallocArray(&sequencesGpu,
8      &channel, sequencesCols, sequencesRows);
9  cudaMemcpyToArray(sequencesGpu, 0, 0,
10     sequences, sequencesSize, TO_GPU);
11 cudaBindTextureToArray(
12     seqsTexture, sequencesGpu);
```

(a) CUDA

```
1  static int *seqsGpu;
2
3  seqsGpu = (int *)sycl::malloc_device(
4   sequencesCols * sequencesRows * sizeof(int),
5   dpct::get_default_queue());
6
7  dpct::get_default_queue()
8   .memcpy(seqsGpu, sequences, sequencesCols *
9         sequencesRows * sizeof(int))
10   .wait();
```

(b) oneAPI

```
1  int columnCodes = tex2D(
2    seqsTexture, colOff, j + rowOff);
```

(a) CUDA

```
1  int columnCodes =
2      seqsGpu[(j + rowOff) * sequencesCols + colOff];
```

(b) oneAPI

Experimental Work - Stage 3

For a 1D/2D image/image array, the width must be a Value >= 1 and <= CL_DEVICE_IMAGE2D_MAX_WIDTH.

```
1  texture<int, 2,
2    cudaReadModeElementType> seqsTexture;
3
4  cudaArray *sequencesGpu;
5  cudaChannelFormatDesc channel =
6    seqsTexture.channelDesc;
7  cudaMallocArray(&sequencesGpu,
8      &channel, sequencesCols, sequencesRows);
9  cudaMemcpyToArray(sequencesGpu, 0, 0,
10     sequences, sequencesSize, TO_GPU);
11 cudaBindTextureToArray(
12     seqsTexture, sequencesGpu);
```

(a) CUDA

```
1  static int *seqsGpu;
2
3  seqsGpu = (int *)sycl::malloc_device(
4   sequencesCols * sequencesRows * sizeof(int),
5   dpct::get_default_queue());
6
7  dpct::get_default_queue()
8   .memcpy(seqsGpu, sequences, sequencesCols *
9         sequencesRows * sizeof(int))
10   .wait();
```

(b) oneAPI

```
1  int columnCodes = tex2D(
2    seqsTexture, colOff, j + rowOff);
```

(a) CUDA

```
1  int columnCodes =
2      seqsGpu[(j + rowOff) * sequencesCols + colOff];
```

(b) oneAPI

Experimental Work - Stage 4

Different tests were run to prove that the DPC++ code generates correct results

Agenda

- Motivation and Goal
- oneAPI
- Smith-Waterman
- Experimental Work
- **Experimental Results**
- Conclusions

Experimental Results - Hardware Platforms

CPU			GPU			
ID	**Processor**	**RAM (Memory)**	**ID**	**Vedor (Type)**	**Model (Architecture)**	**GFLOPS peak (SP)**
Core-i5	Intel Core i5-7400	16 GB	Titan	NVIDIA (Discrete)	Titan X (Pascal)	10970
Core-i3	Intel Core i3-4160	8 GB	RTX	NVIDIA (Discrete)	RTX 2070 (Turing)	7465
Core-i9	Intel Core i9-10920X	32 GB	Iris XE	Intel (Discrete)	Iris Xe MAX Graphics (Gen 12.1)	2534
Xeon	Intel Xeon E-2176G	65 GB	P630	NVIDIA (Integrated)	UHD Graphics P630 (Gen 9.5)	441.6

Experimental Results - Design

1. **UniProtKB/Swiss-Prot** database (release 2021_04) contains **204173280** amino acid residues in **565928** sequences with a maximum length of 35213

Experimental Results - Design

1. **UniProtKB/Swiss-Prot** database (release 2021_04) contains **204173280** amino acid residues in **565928** sequences with a maximum length of 35213
2. **20** queries with length from **144** to **5478** (accession numbers: P02232, P05013, P14942, P07327, P01008, P03435, P42357, P21177, Q38941, P27895, P07756, P04775, P19096, P28167, P0C6B8, P20930, P08519, Q7TMA5, P33450, and Q9UKN1)

Experimental Results - Design

1. **UniProtKB/Swiss-Prot** database (release 2021_04) contains **204173280** amino acid residues in **565928** sequences with a maximum length of 35213
2. **20** queries with length from **144** to **5478** (accession numbers: P02232, P05013, P14942, P07327, P01008, P03435, P42357, P21177, Q38941, P27895, P07756, P04775, P19096, P28167, P0C6B8, P20930, P08519, Q7TMA5, P33450, and Q9UKN1)
3. Scoring matrix: **BLOSUM62**
   Gap insertion: **10**
   Gap extension: **2**

Experimental Results - Configuration & tests

1. Single thread at the CPU level (using flag T=1).

Experimental Results - Configuration & tests

1. Single thread at the CPU level (using flag T=1).

2. Different work group sizes were configured for kernel execution.

Experimental Results - Configuration & tests

1. Single thread at the CPU level (using flag T=1).

2. Different work group sizes were configured for kernel execution.

3. Each test was run twenty times and the performance was calculated as the average.

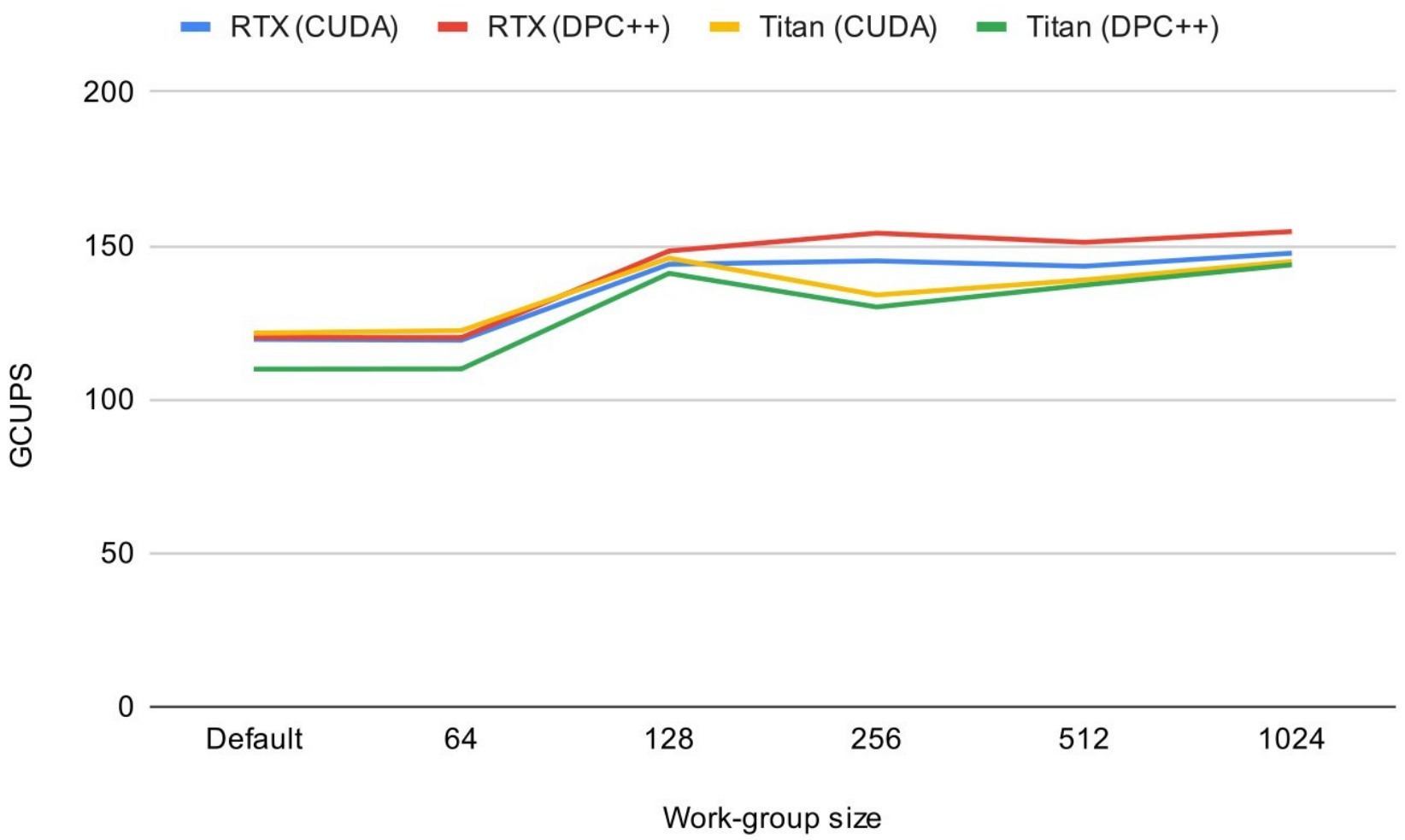
Experimental Results - Configuration & tests

1. Single thread at the CPU level (using flag T=1).
2. Different work group sizes were configured for kernel execution.
3. Each test was run twenty times and the performance was calculated as the average.
4. GCUPS as the performance metric.

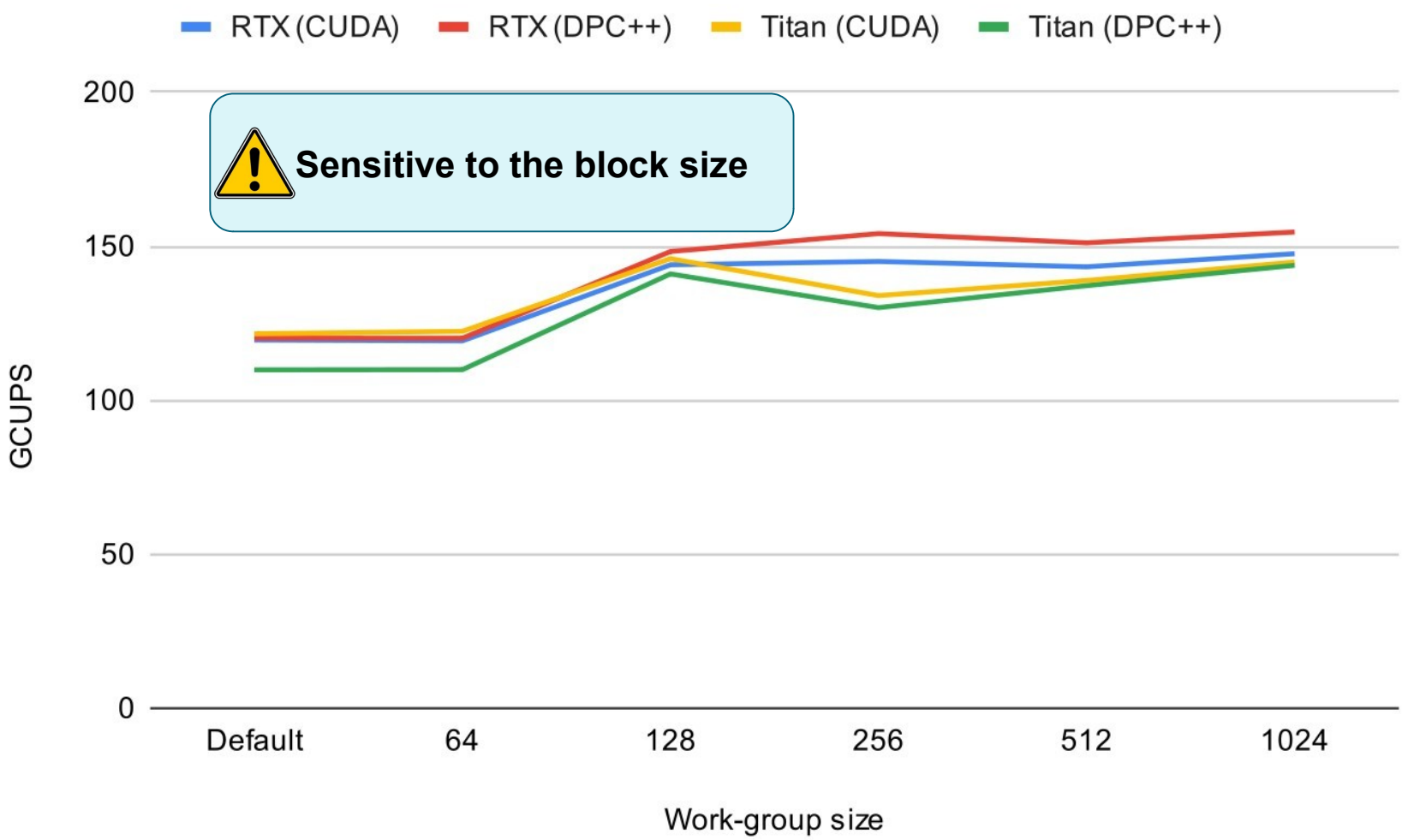
$$\frac{|Q| * |D|}{t * 10^9}$$

Q ⇾ Database length
D ⇾ Sequence length
t ⇾ Execution time in seconds

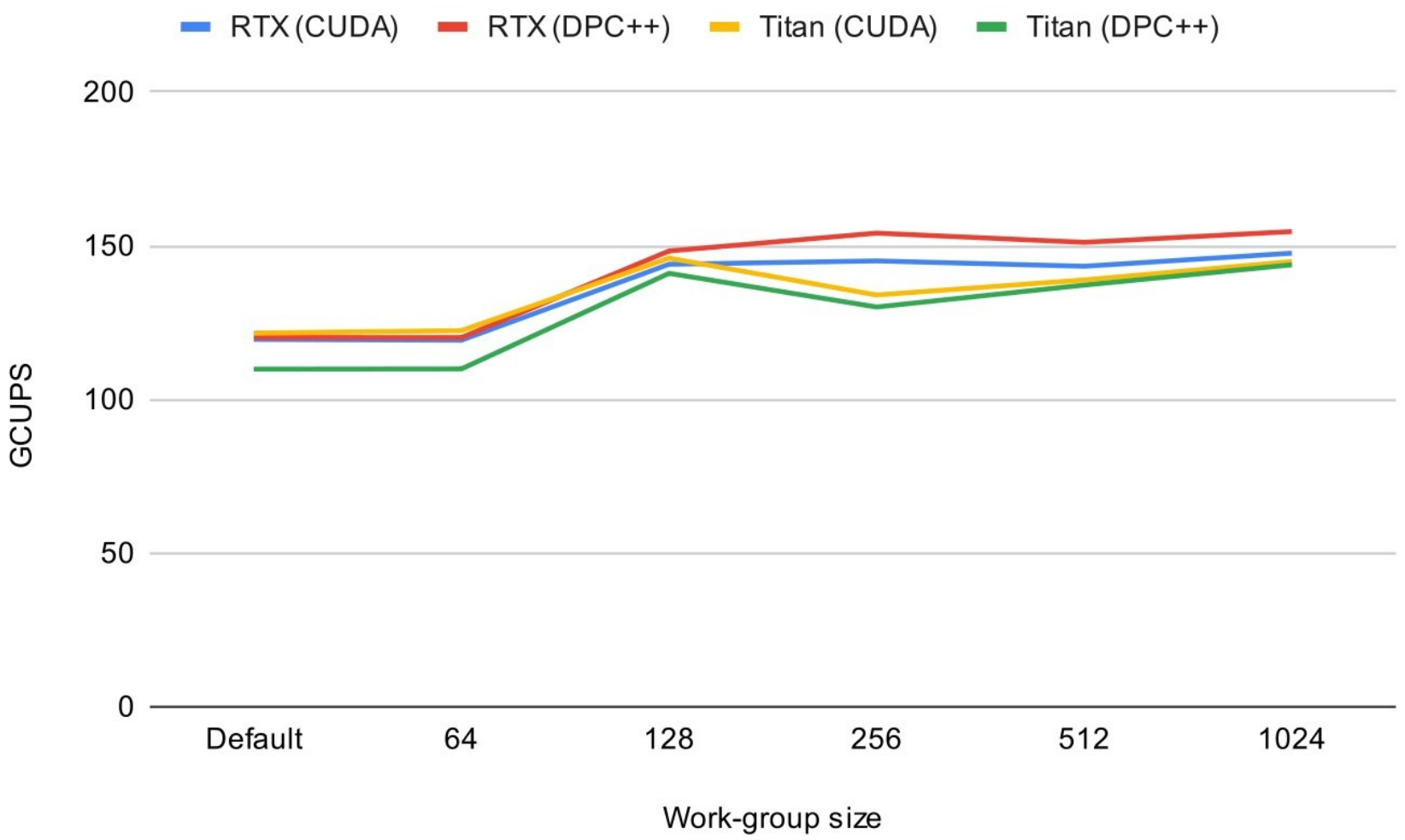
Experimental Results - Varying work group size



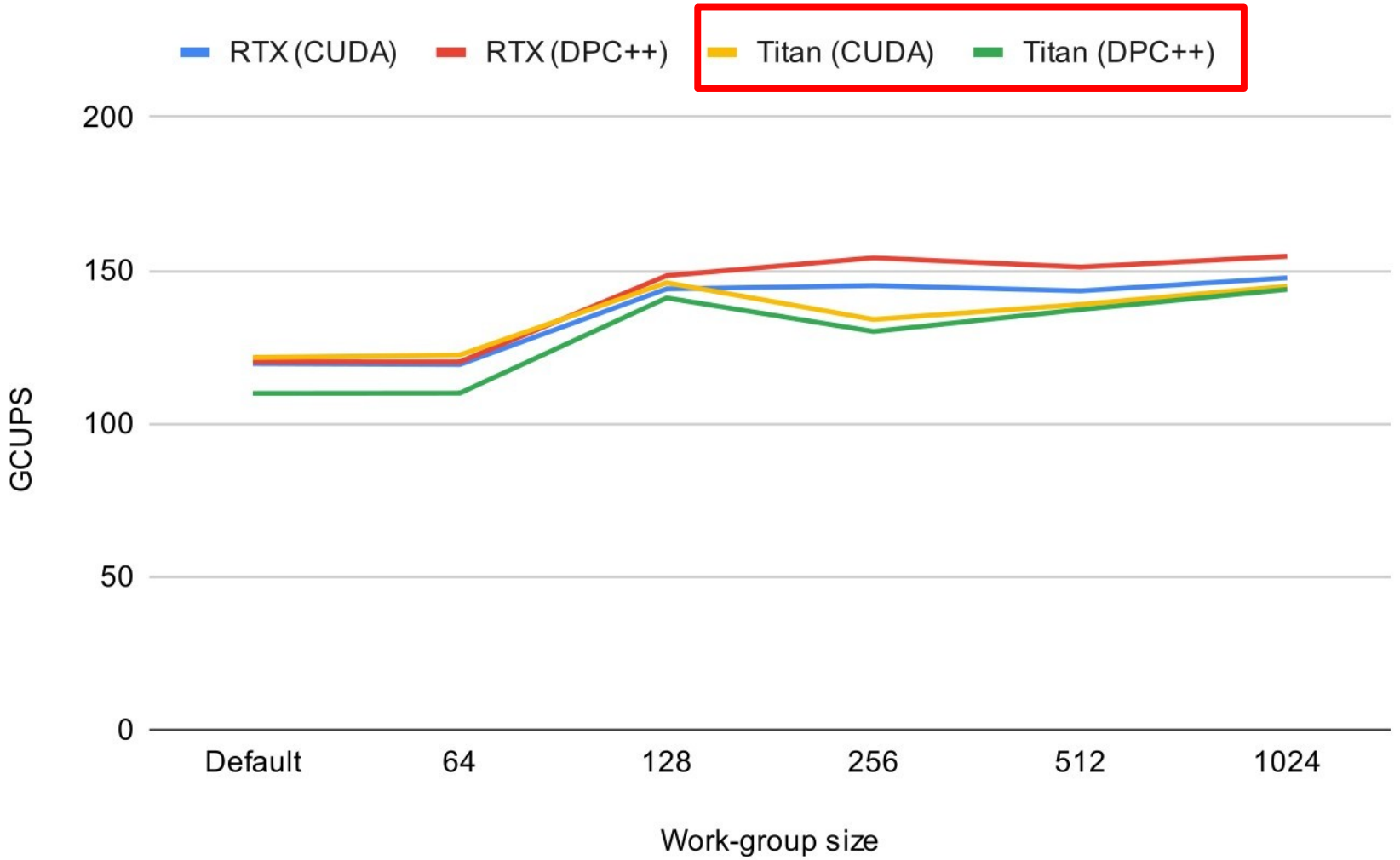
Experimental Results - Varying work group size



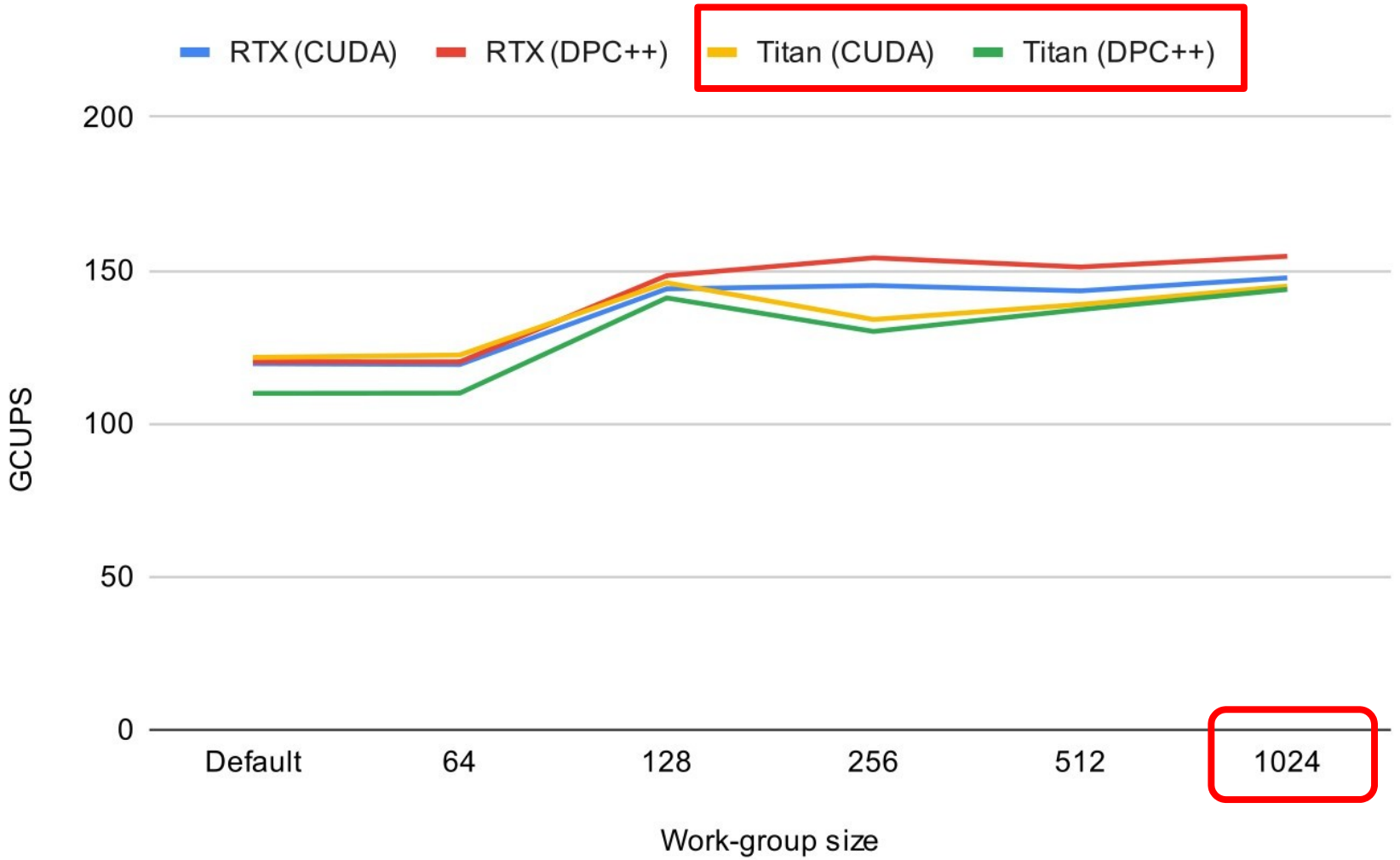
Experimental Results - Varying work group size



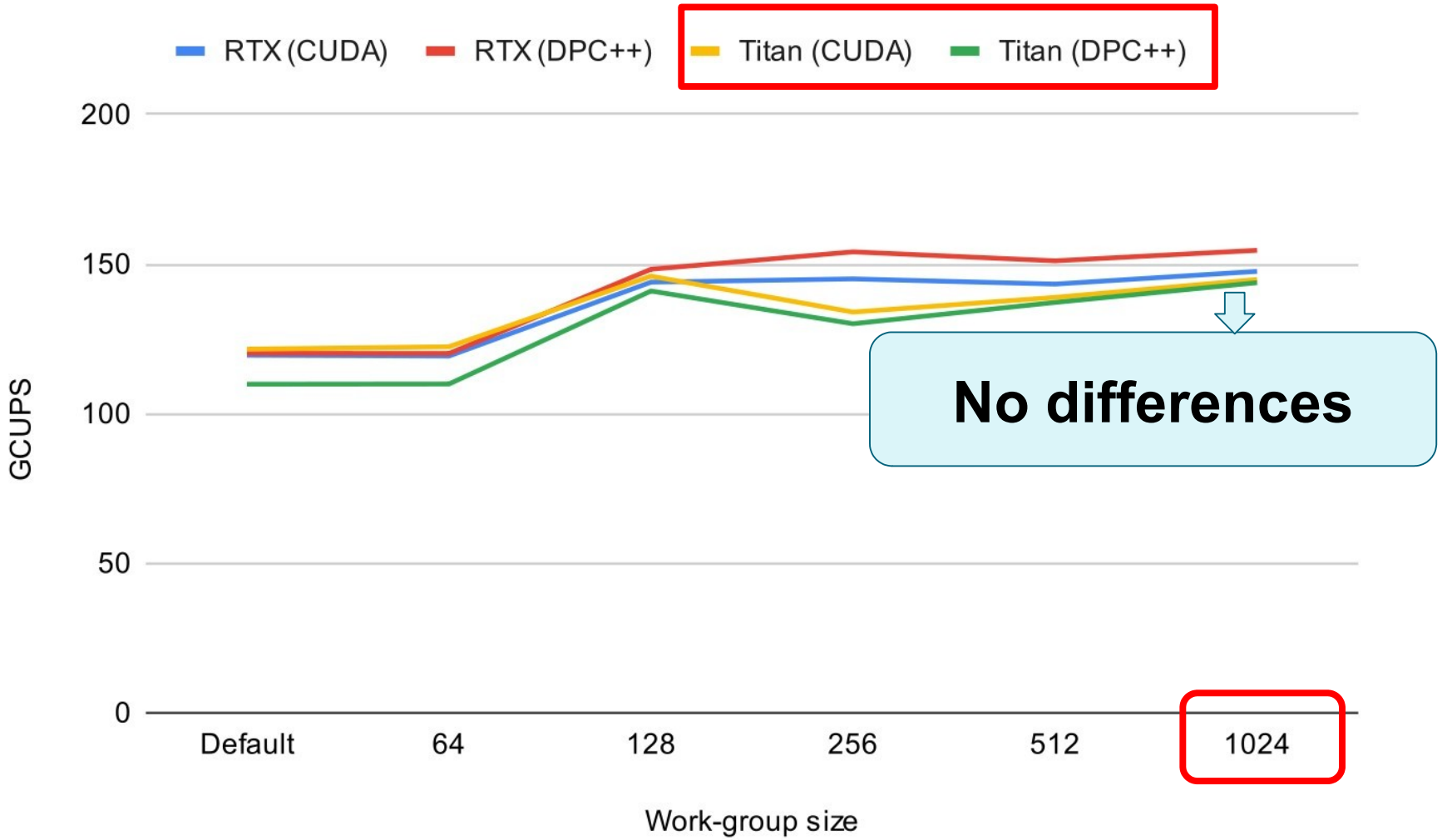
Experimental Results - Varying work group size



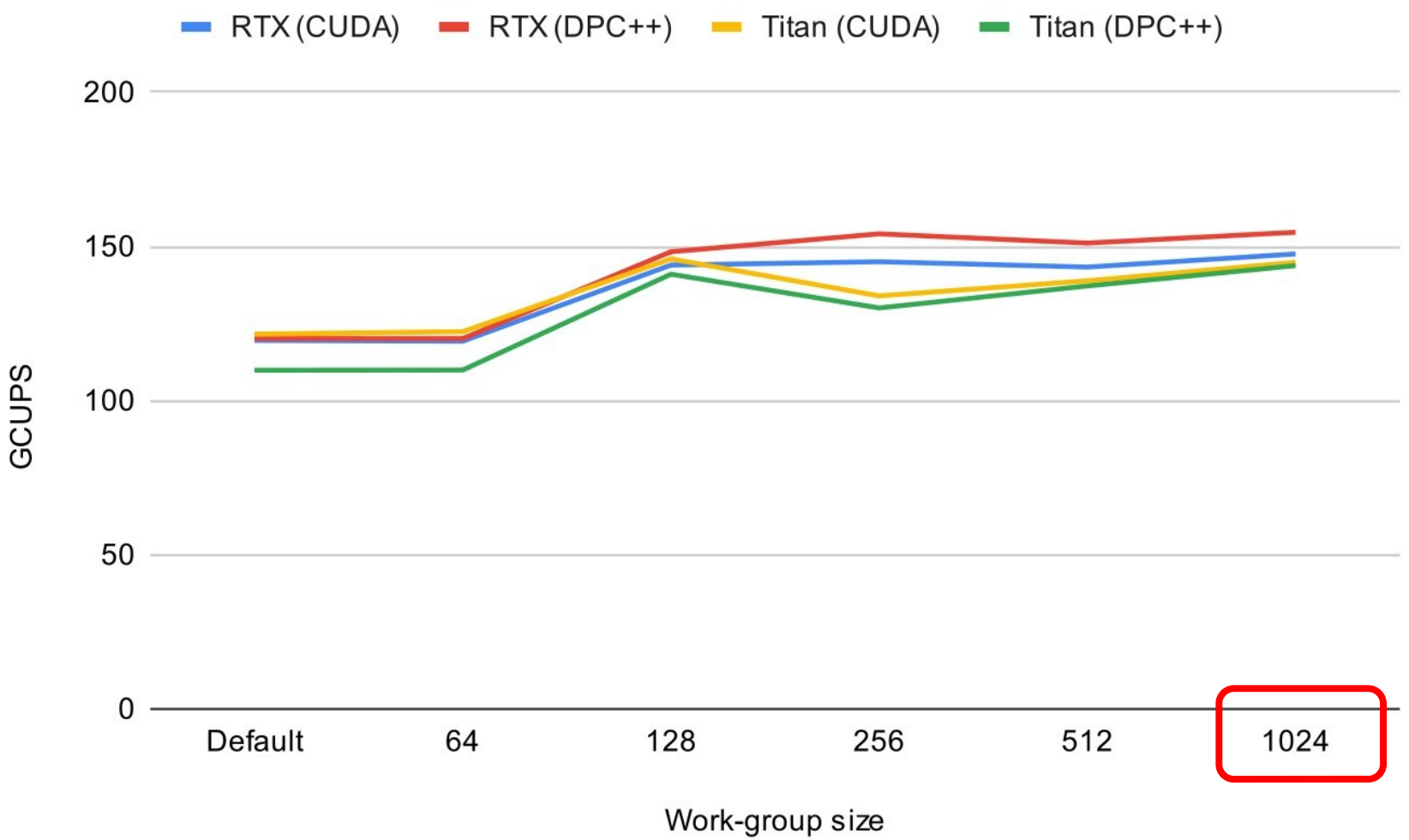
Experimental Results - Varying work group size



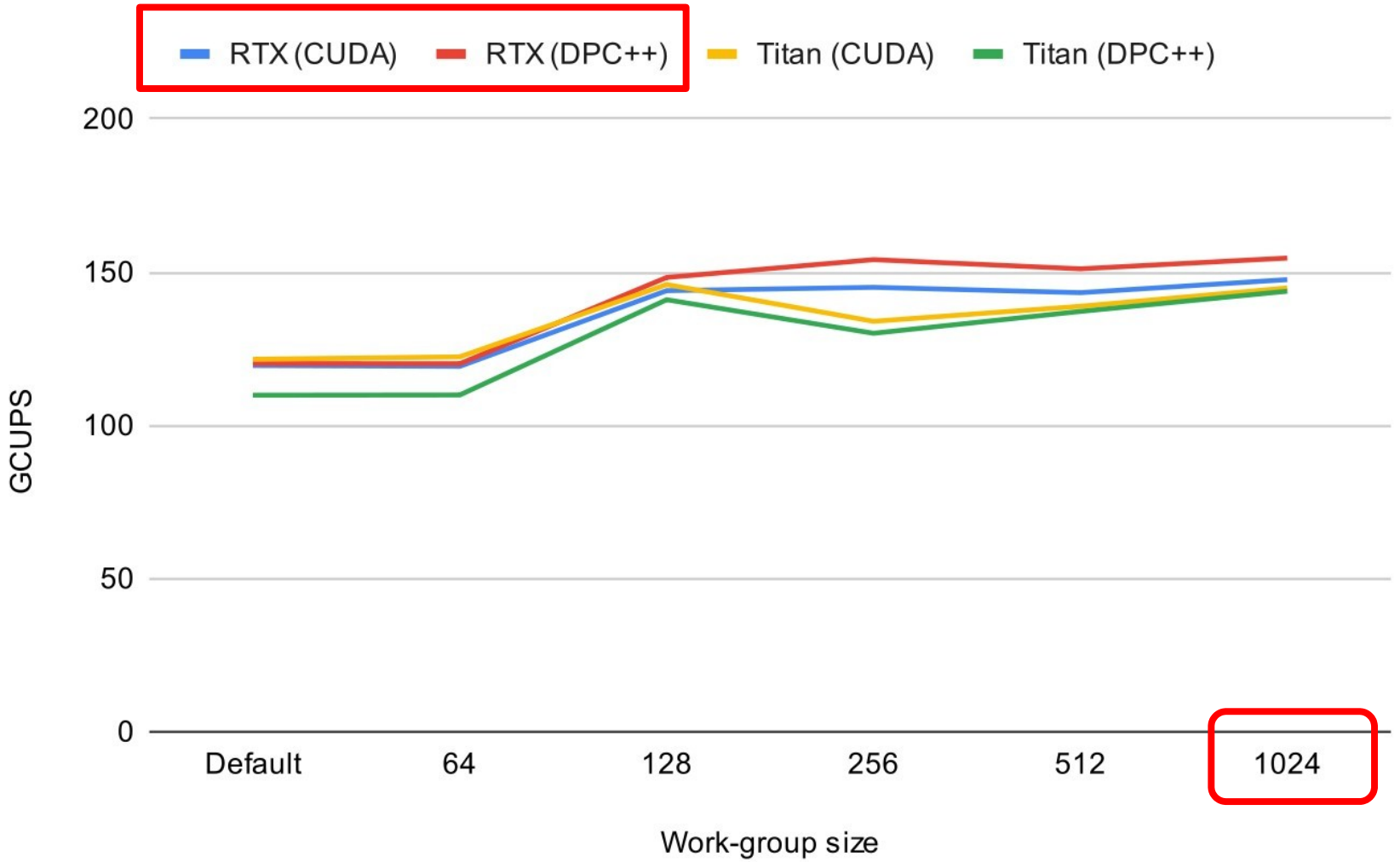
Experimental Results - Varying work group size



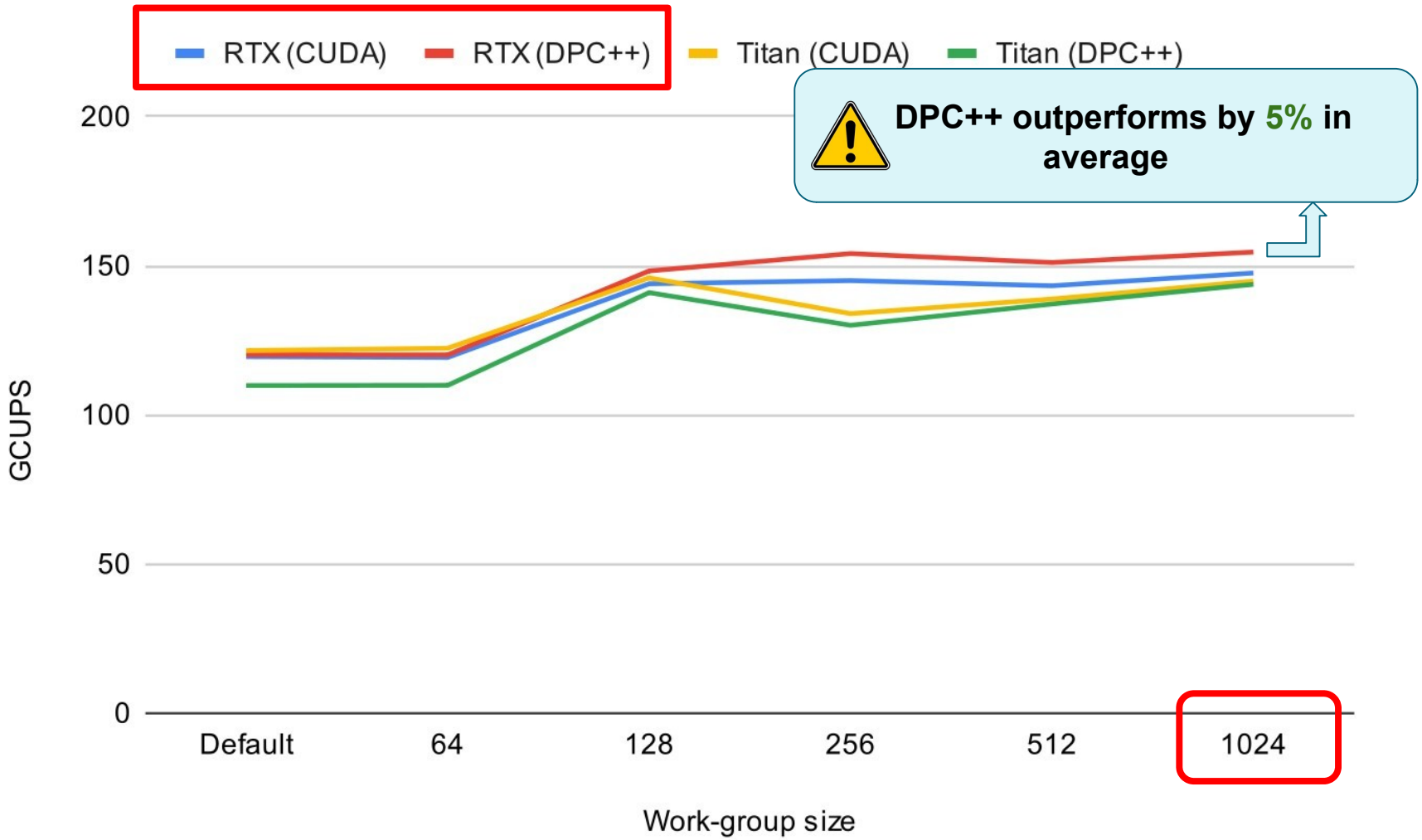
Experimental Results - Varying work group size



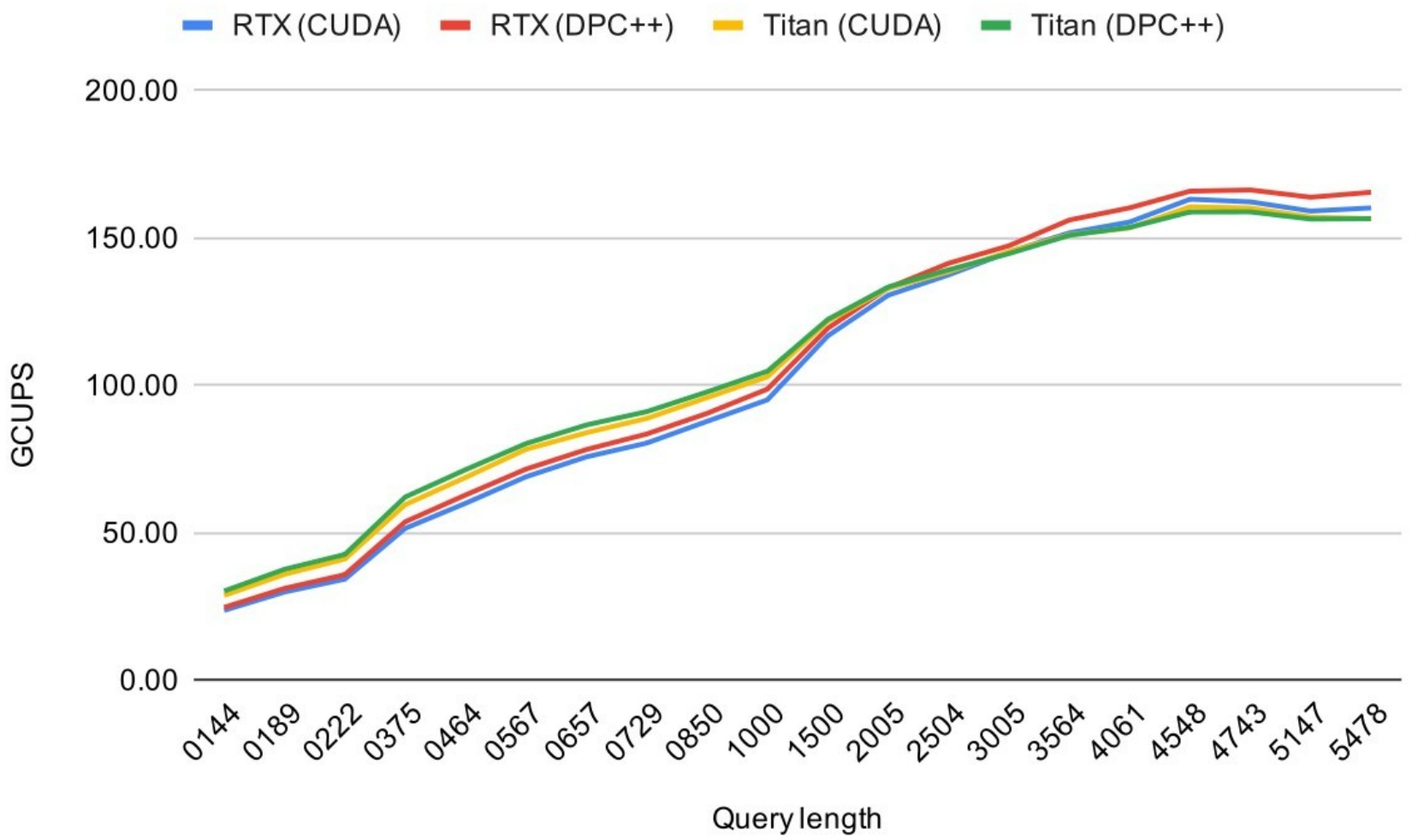
Experimental Results - Varying work group size



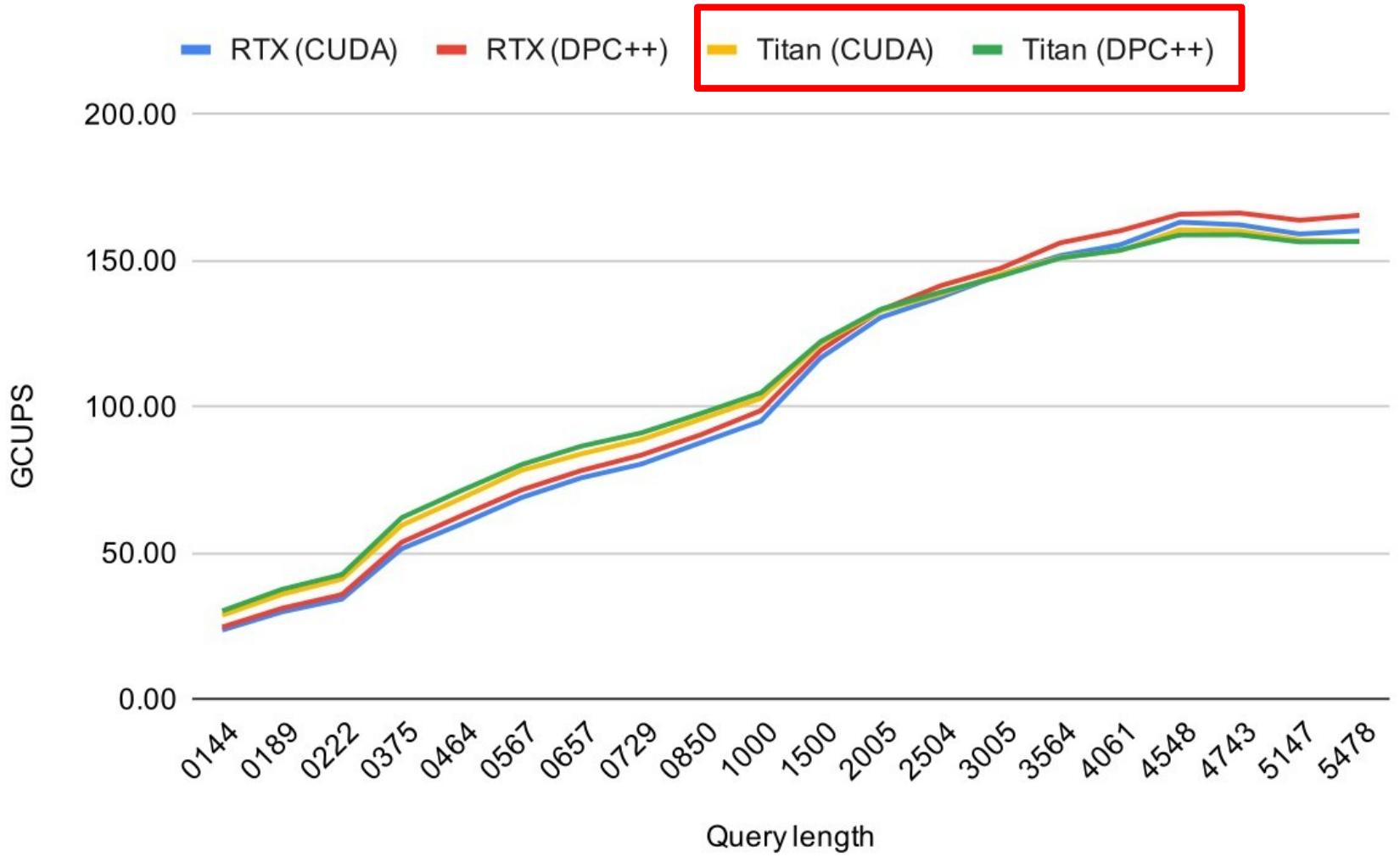
Experimental Results - Varying work group size



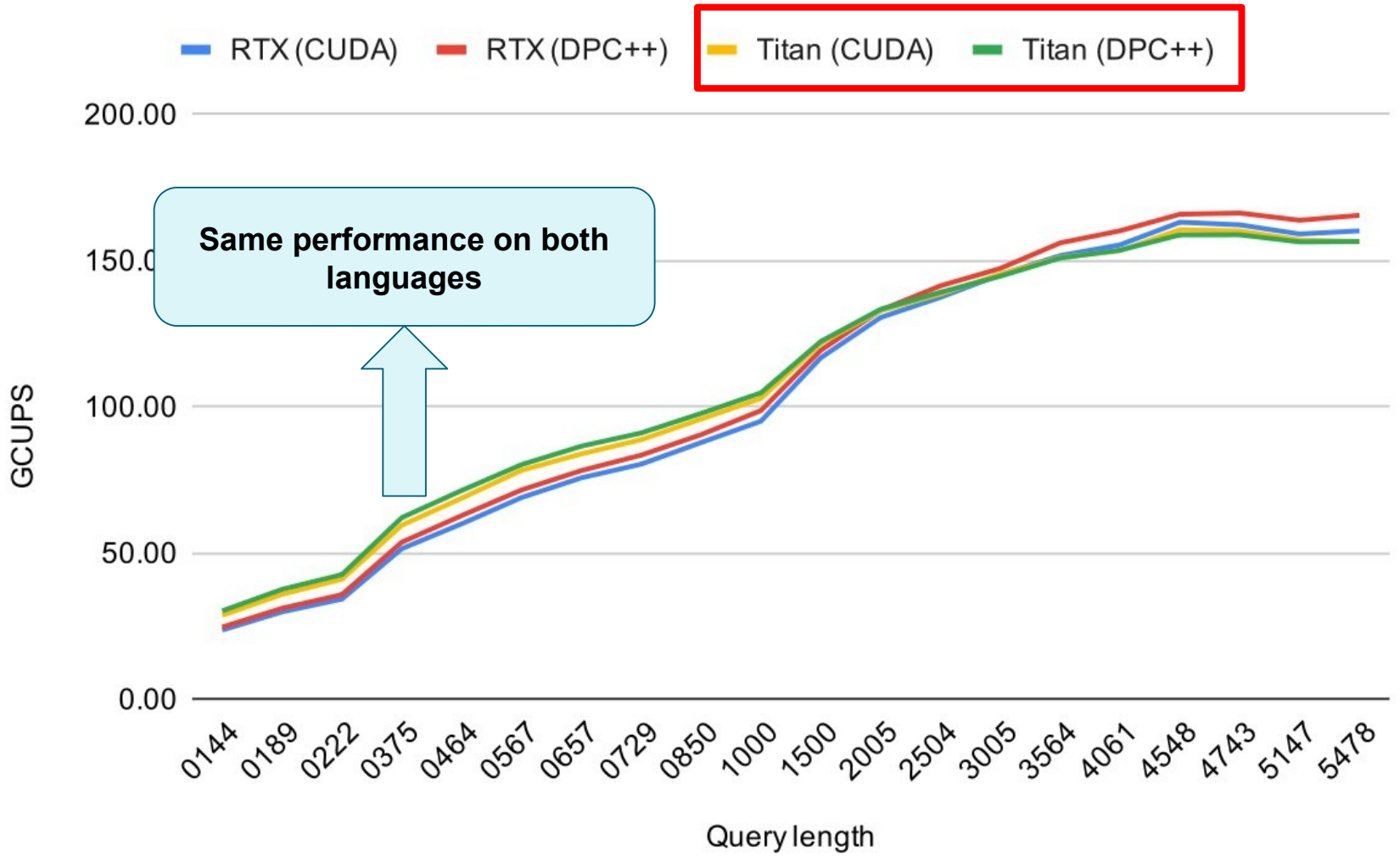
Experimental Results - Varying query length



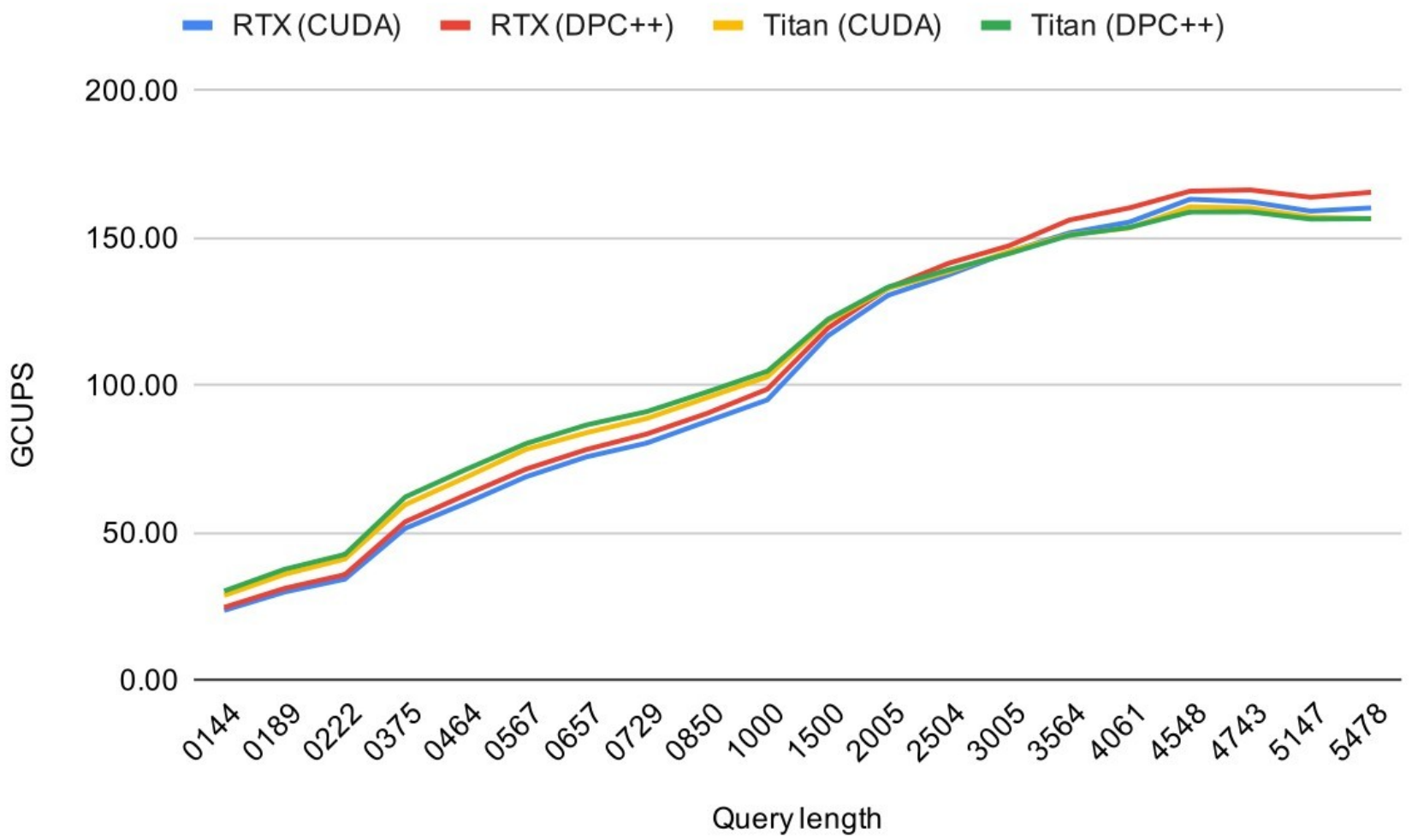
Experimental Results - Varying query length



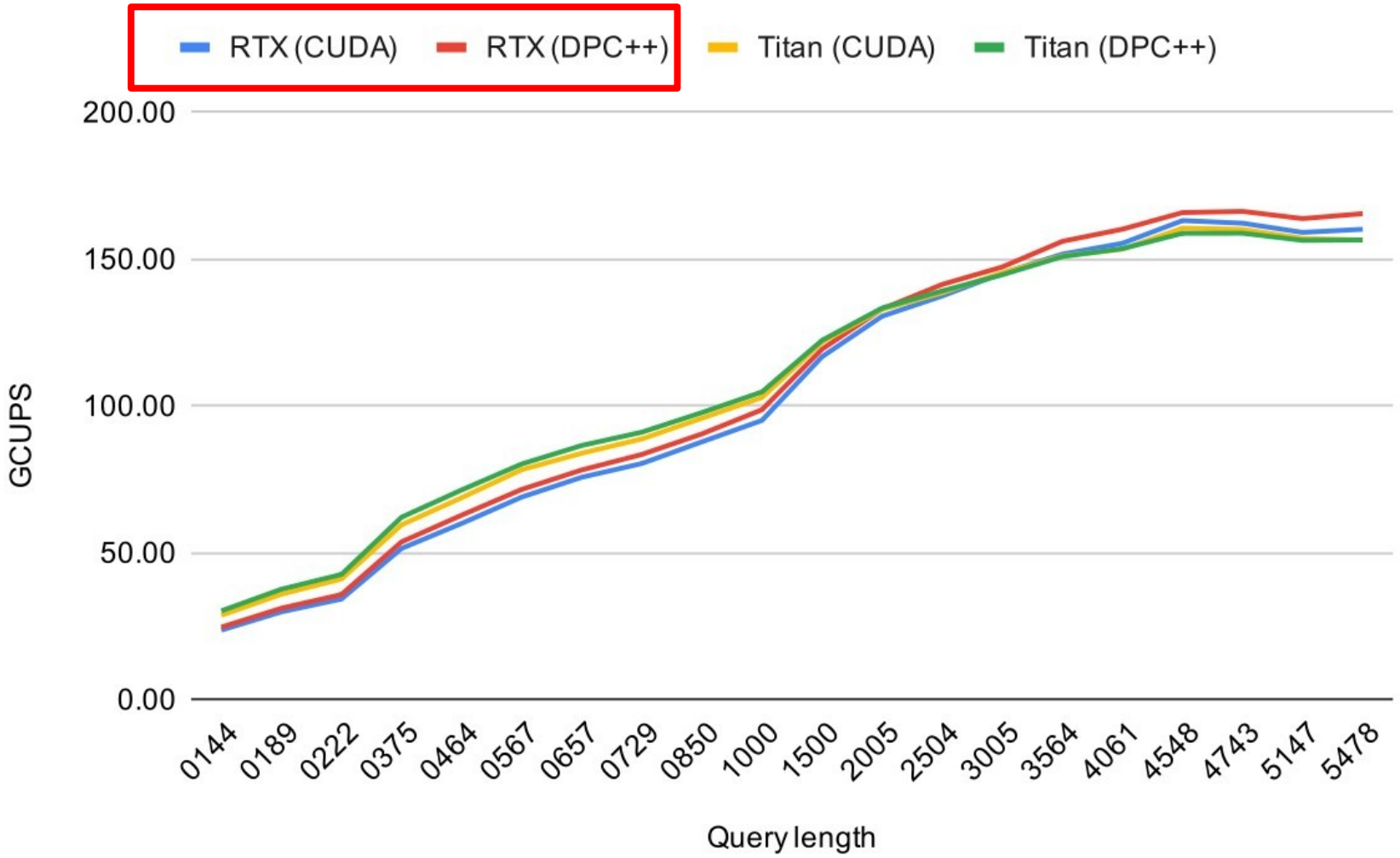
Experimental Results - Varying query length



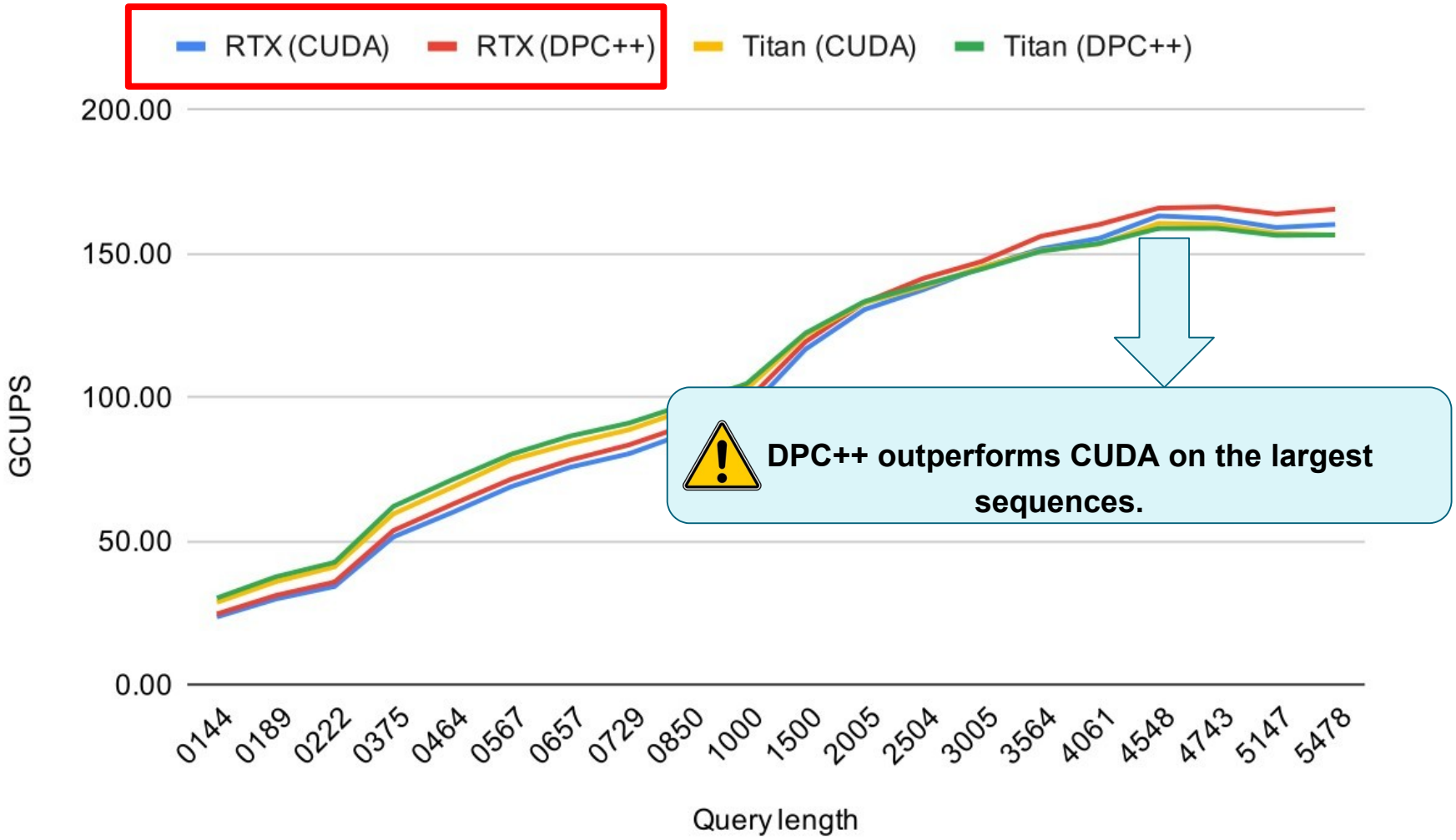
Experimental Results - Varying query length



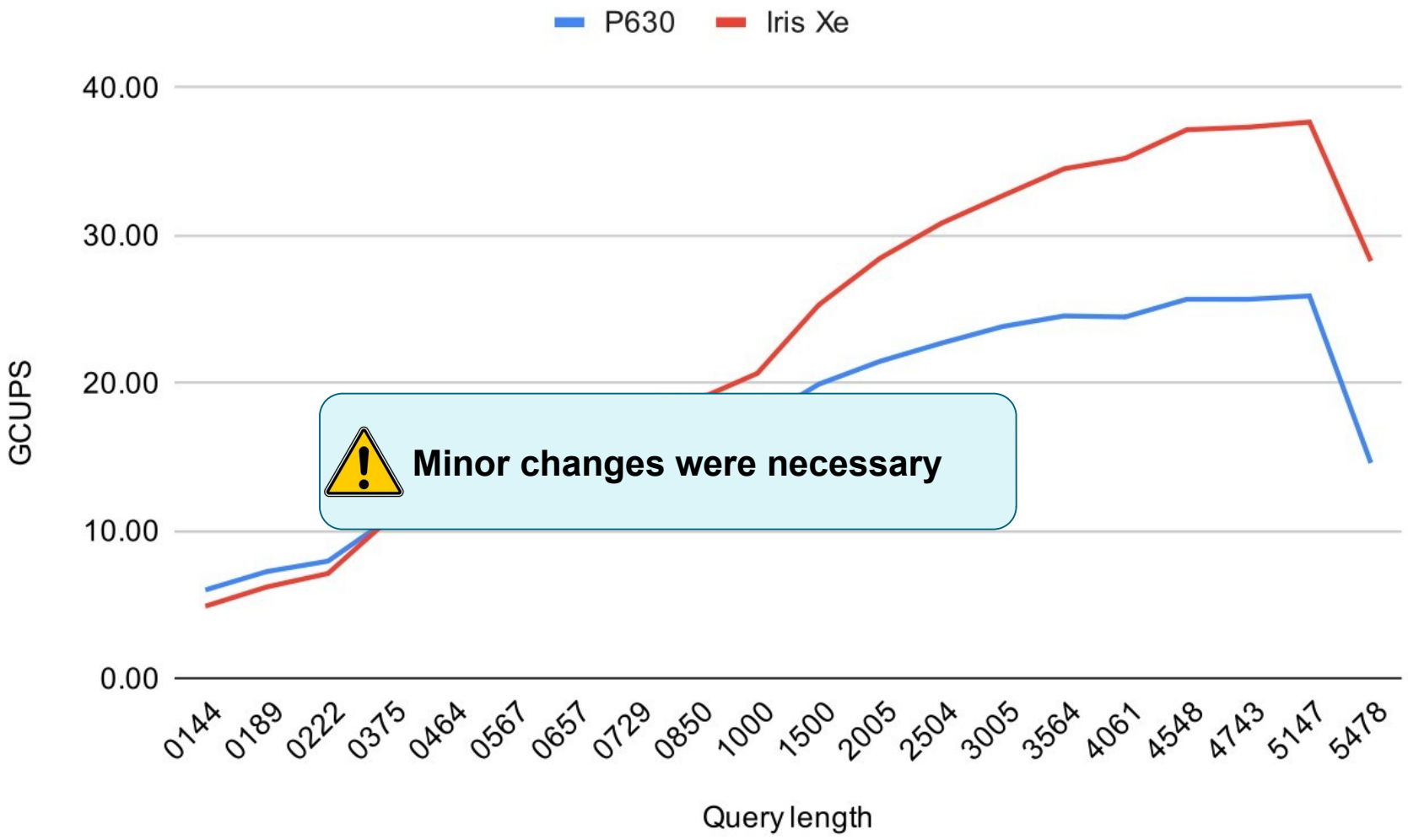
Experimental Results - Varying query length



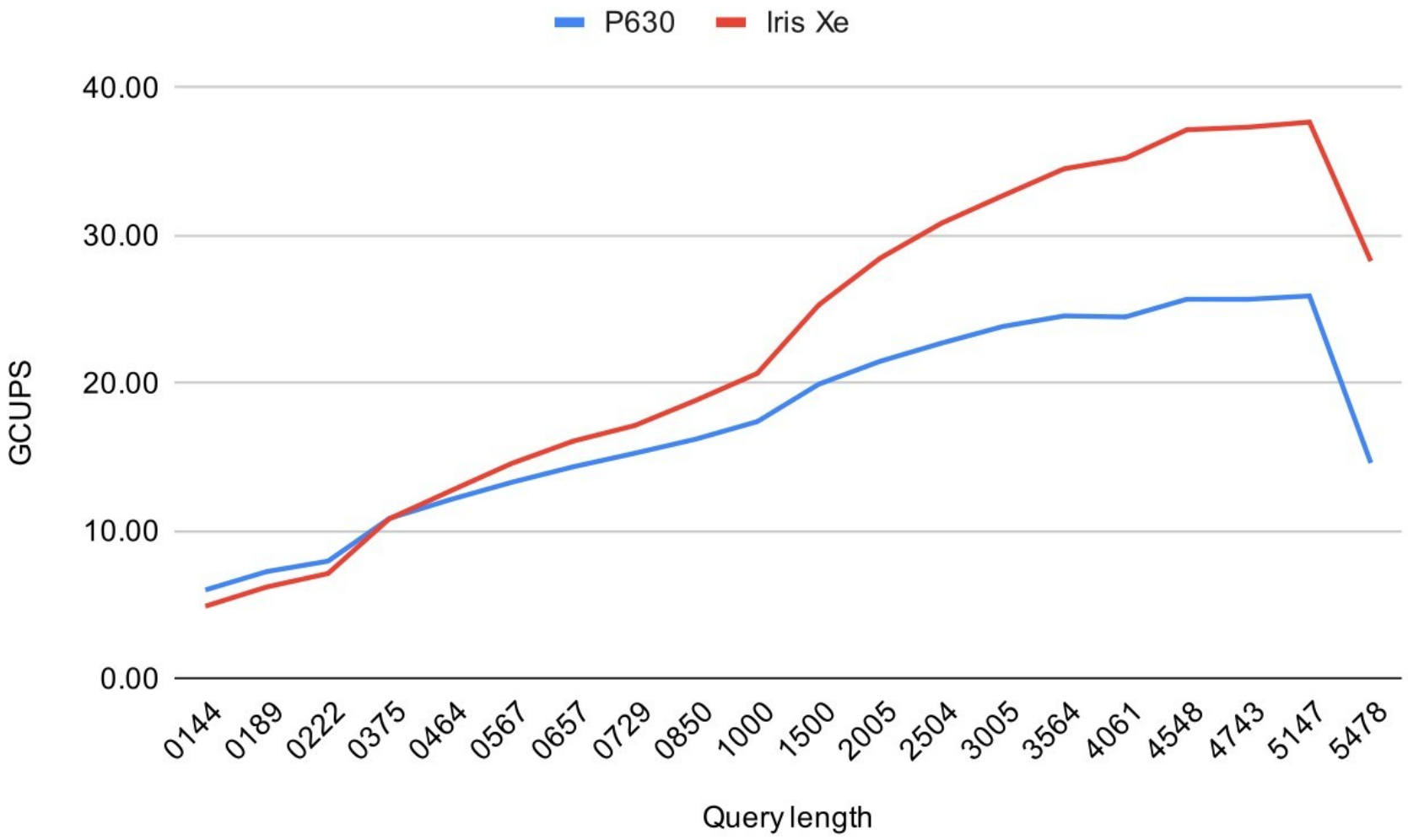
Experimental Results - Varying query length



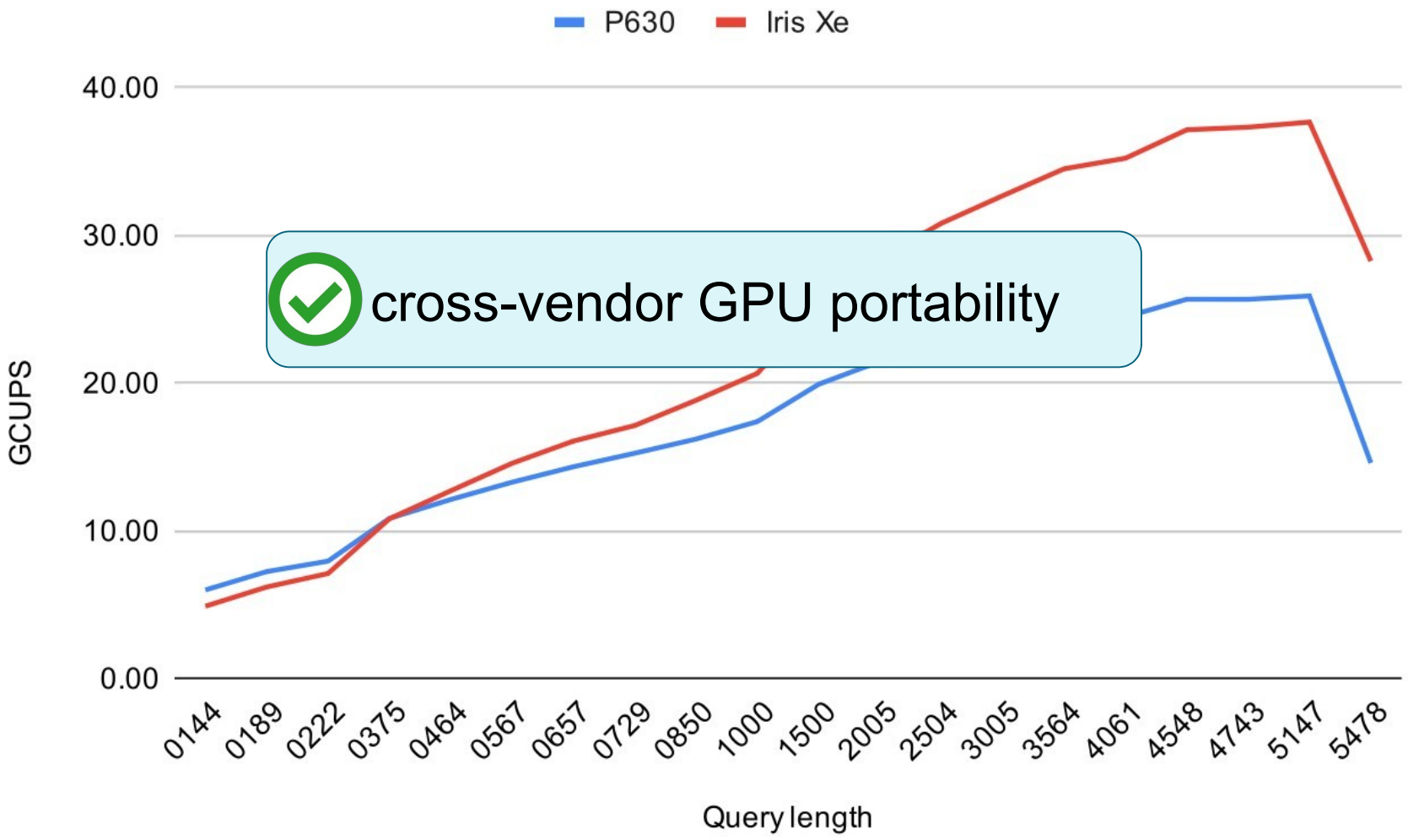
Experimental Results - Cross-GPU vendor portability



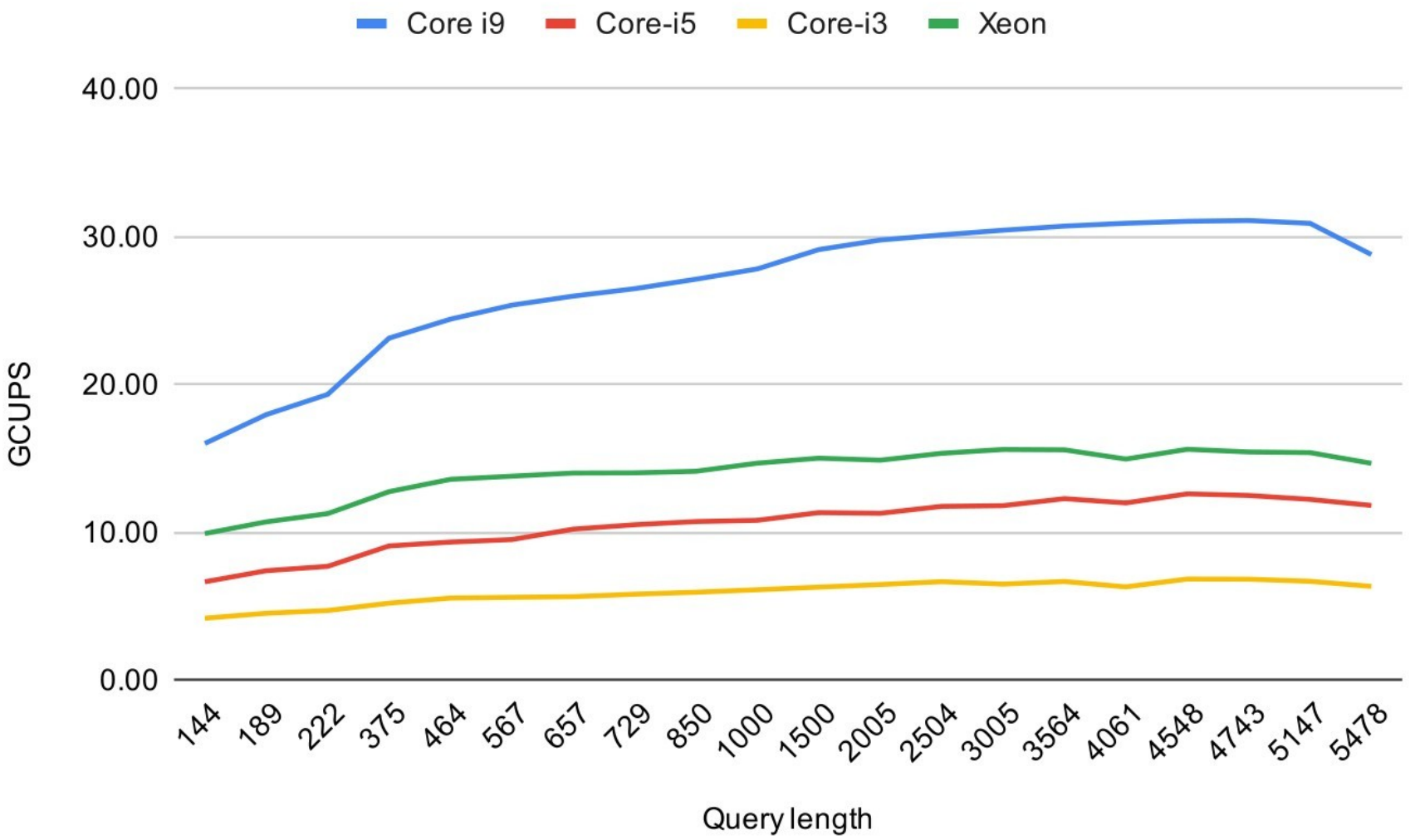
Experimental Results - Cross-GPU vendor portability



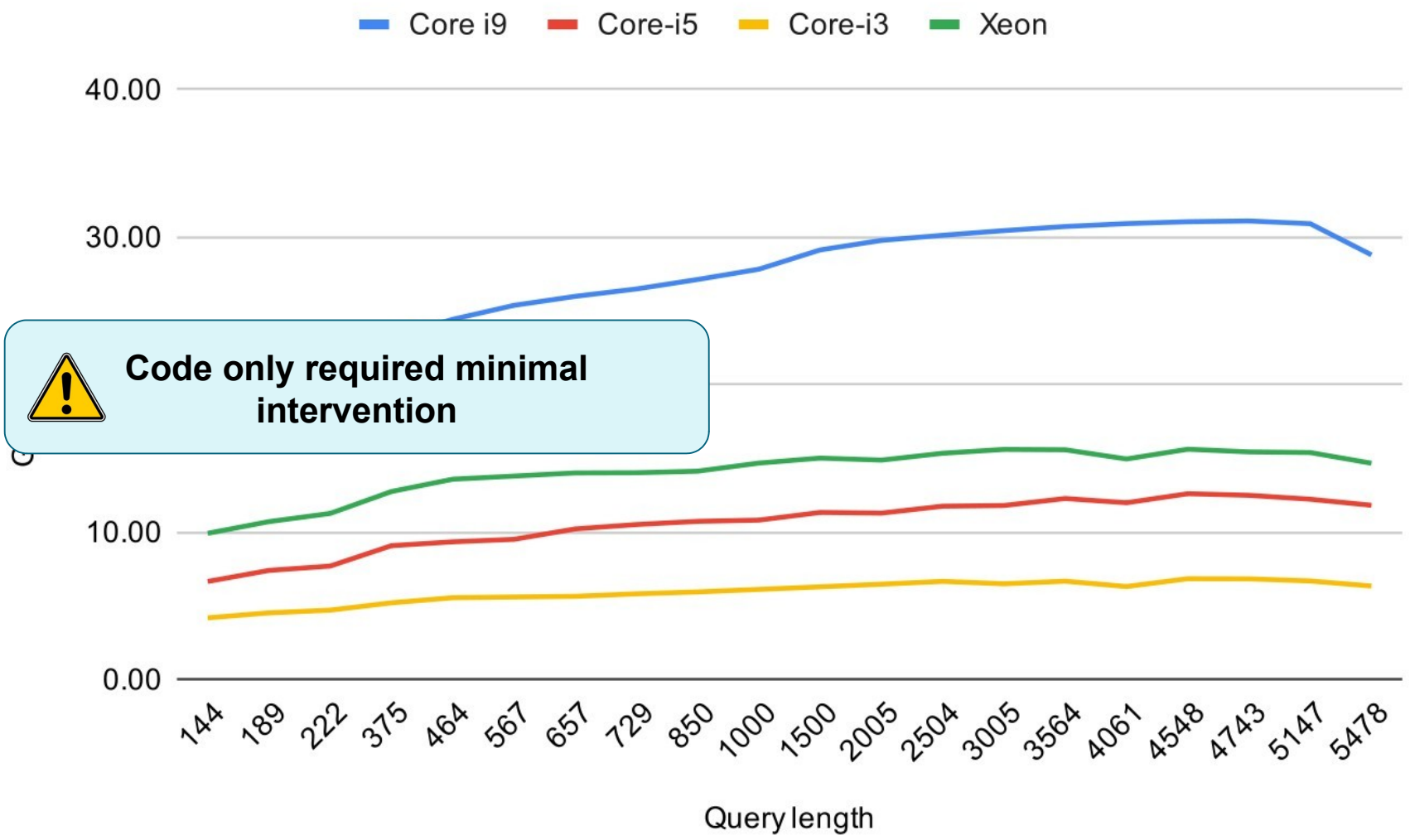
Experimental Results - Cross-GPU vendor portability



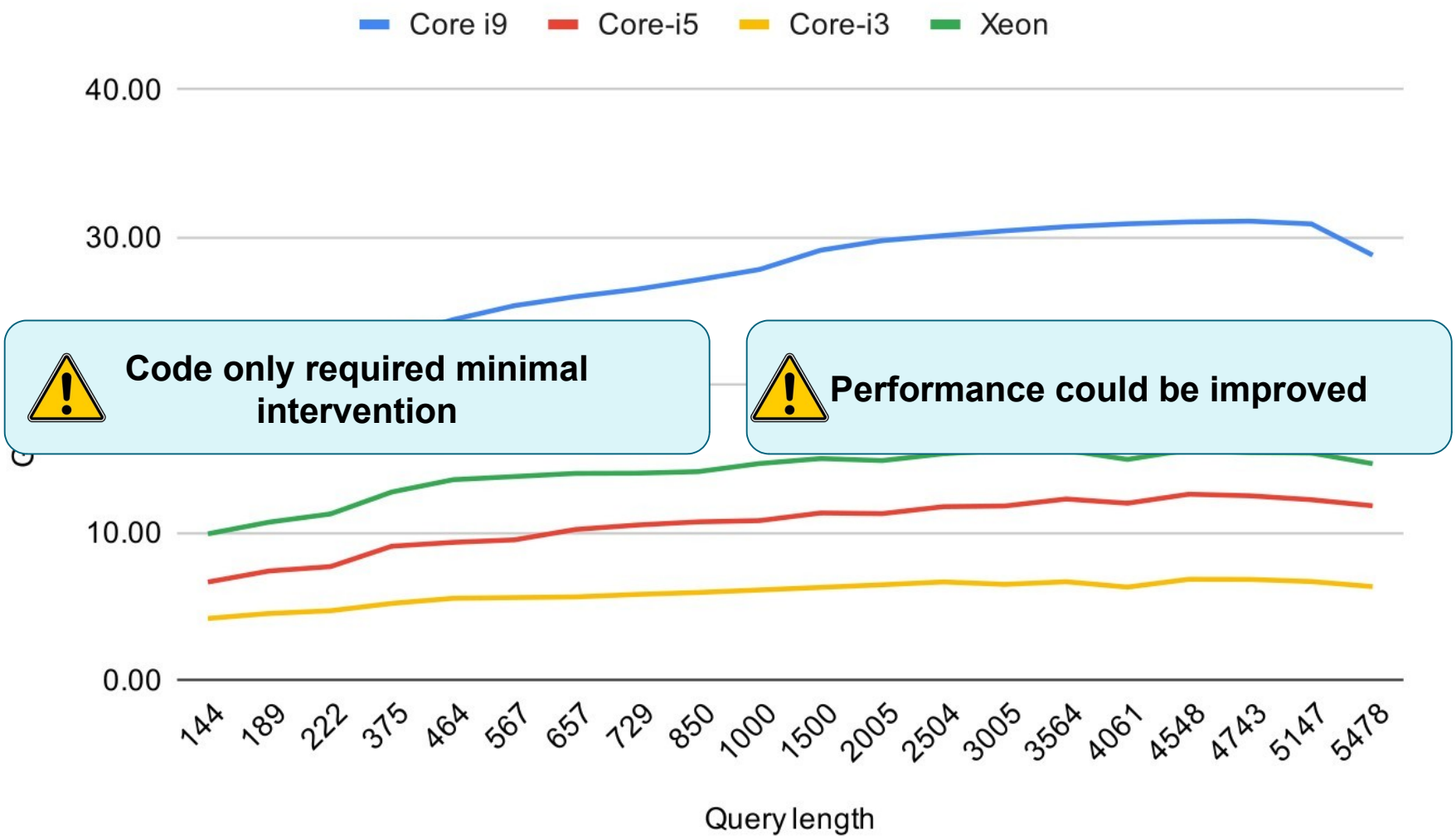
Experimental Results - Cross-architecture portability



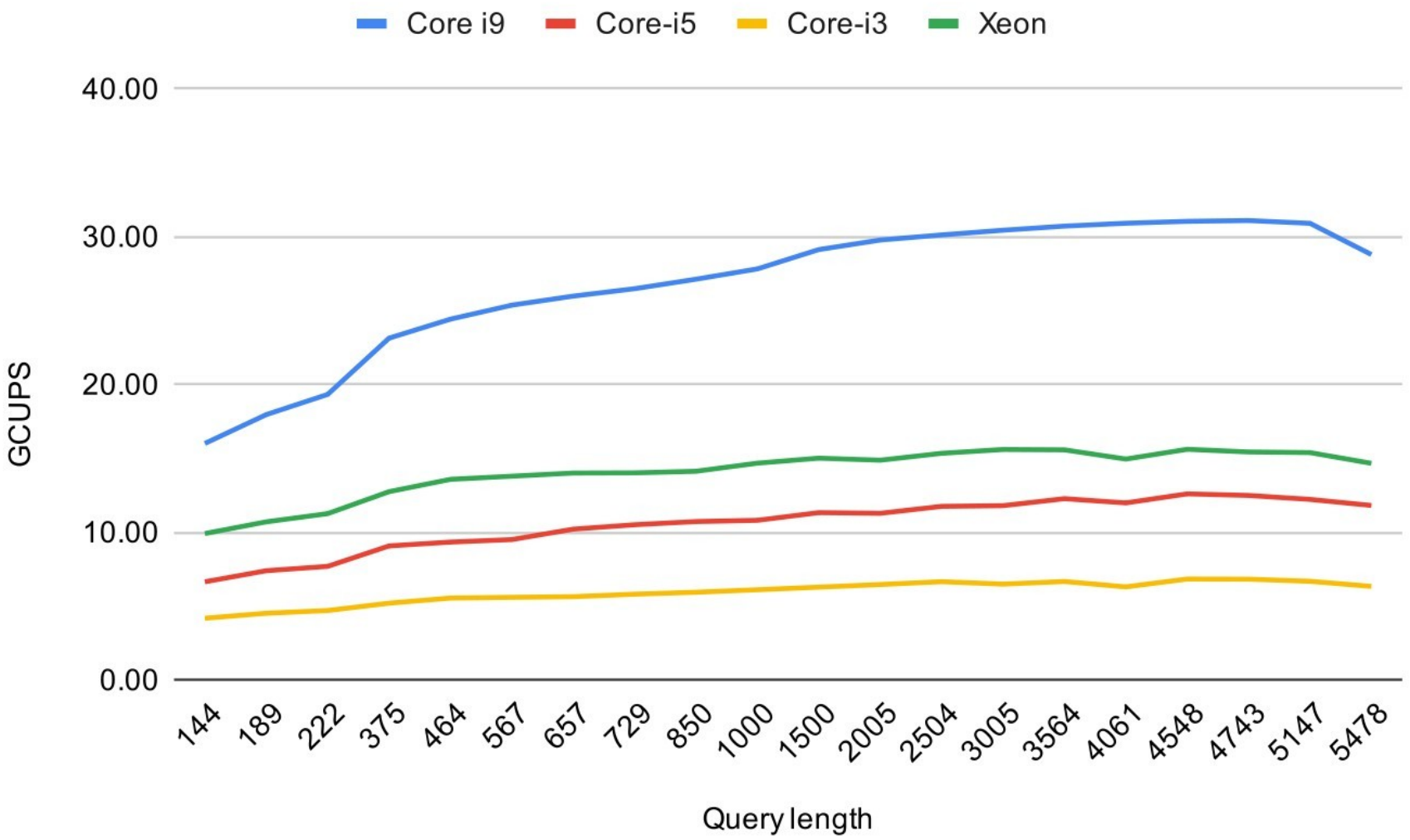
Experimental Results - Cross-architecture portability



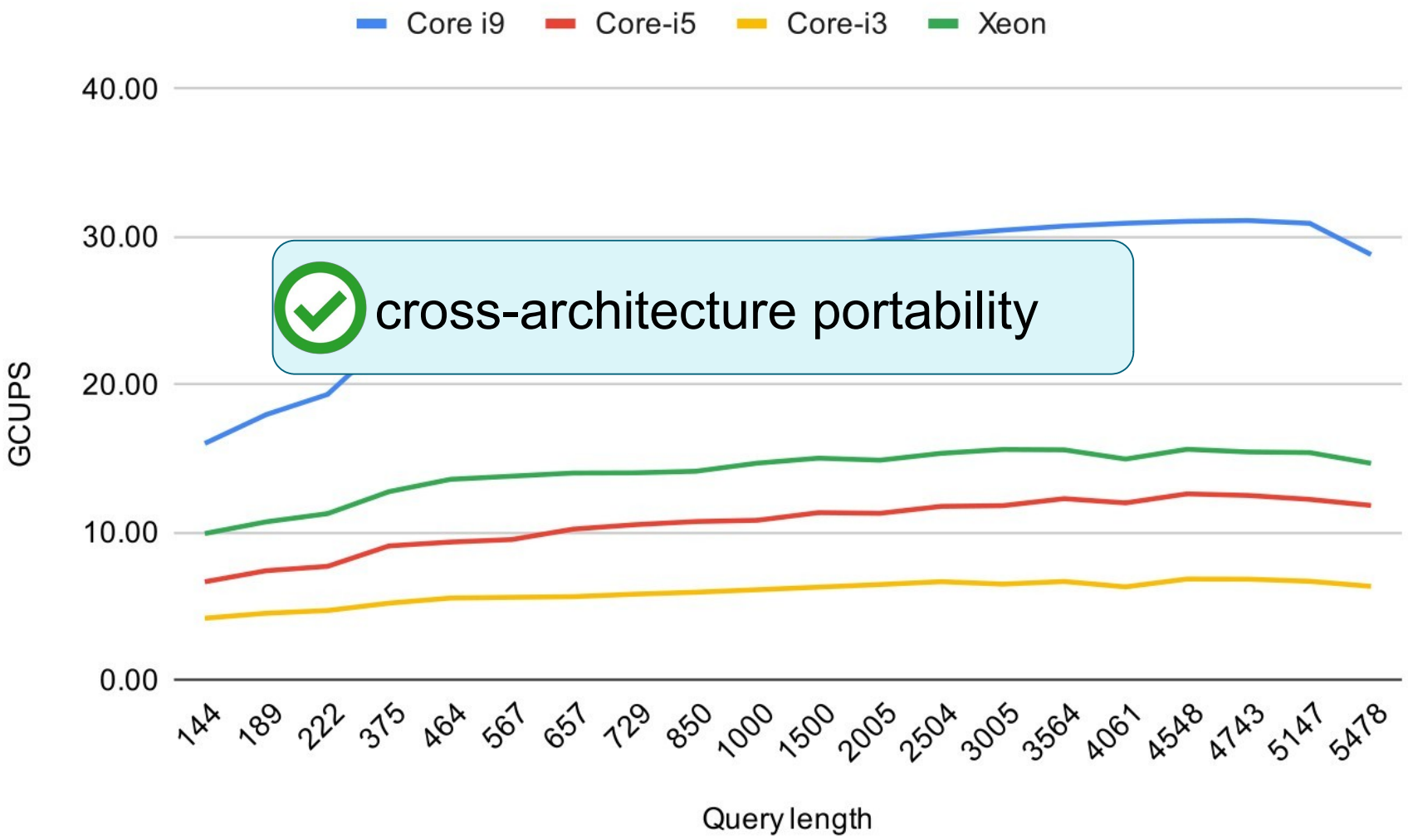
Experimental Results - Cross-architecture portability



Experimental Results - Cross-architecture portability



Experimental Results - Cross-architecture portability



Agenda

- Motivation and Goal
- oneAPI
- Smith-Waterman
- Experimental Work
- Experimental Results
- **Conclusions**

Conclusions

This paper presents our experiences migrating a CUDA-based, biological software tool to DPC++ using the oneAPI framework.

The main contributions are:

Conclusions

This paper presents our experiences migrating a CUDA-based, biological software tool to DPC++ using the oneAPI framework.

The main contributions are:

1. DPCT proved to be an effective tool for code migration to DPC++

Conclusions

This paper presents our experiences migrating a CUDA-based, biological software tool to DPC++ using the oneAPI framework.

The main contributions are:

1. DPCT proved to be an effective tool for code migration to DPC++

2. The migrated code could be successfully executed on CPUs and also GPUs from different vendors, demonstrating its cross-architecture, cross-vendor GPU portability

Conclusions

This paper presents our experiences migrating a CUDA-based, biological software tool to DPC++ using the oneAPI framework.

The main contributions are:

1. DPCT proved to be an effective tool for code migration to DPC++
2. The migrated code could be successfully executed on CPUs and also GPUs from different vendors, demonstrating its cross-architecture, cross-vendor GPU portability
3. The performance results showed that the migrated DPC++ code is comparable to the original CUDA one.

Conclusions - Future Work

1. Understanding the gap in performance between DPC++ and CUDA code, and optimizing DPC++ code to reach its maximum performance.

Conclusions - Future Work

1. Understanding the gap in performance between DPC++ and CUDA code, and optimizing DPC++ code to reach its maximum performance.

2. Carrying out more exhaustive experimental work.

Conclusions - Future Work

1. Understanding the gap in performance between DPC++ and CUDA code, and optimizing DPC++ code to reach its maximum performance.

2. Carrying out more exhaustive experimental work.

3. Running the DPC++ code on other architectures such as FPGAs, to verify its cross-architecture portability.

¡Thank you very much for your time!

IWBBIO
2022