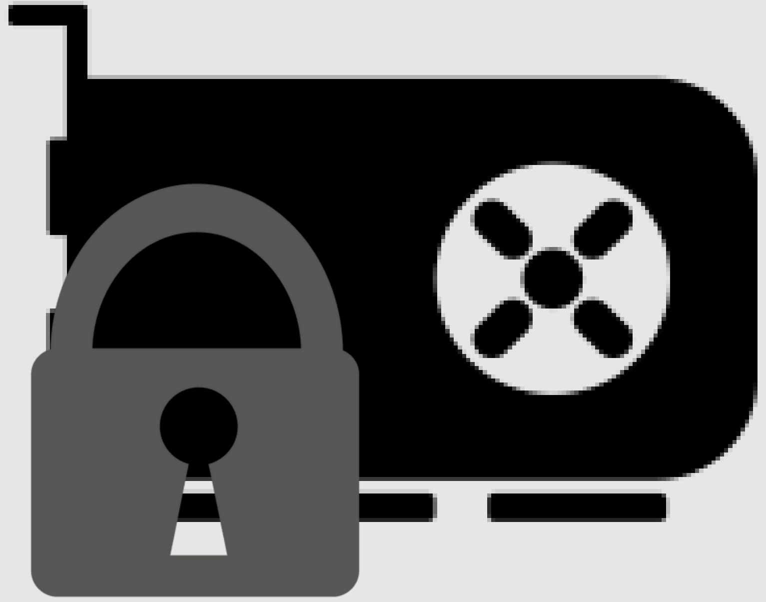
GSYNC : A GPGPU Synchronization library

CS527 – Parallel Computer Architecture

CSD3171 – Georgios Anagnopoulos



GSYNC

GSYNC Overview:

- GPU Locks
- GPU Barriers
- Merge library with M4 MACROS

GPU Architecture

The Pascal GP102 Architecture:

GP102 is composed of :

- Graphics Processing Clusters (GPCs)
- Streaming Multiprocessors (SMs)
- and Memory Controllers.

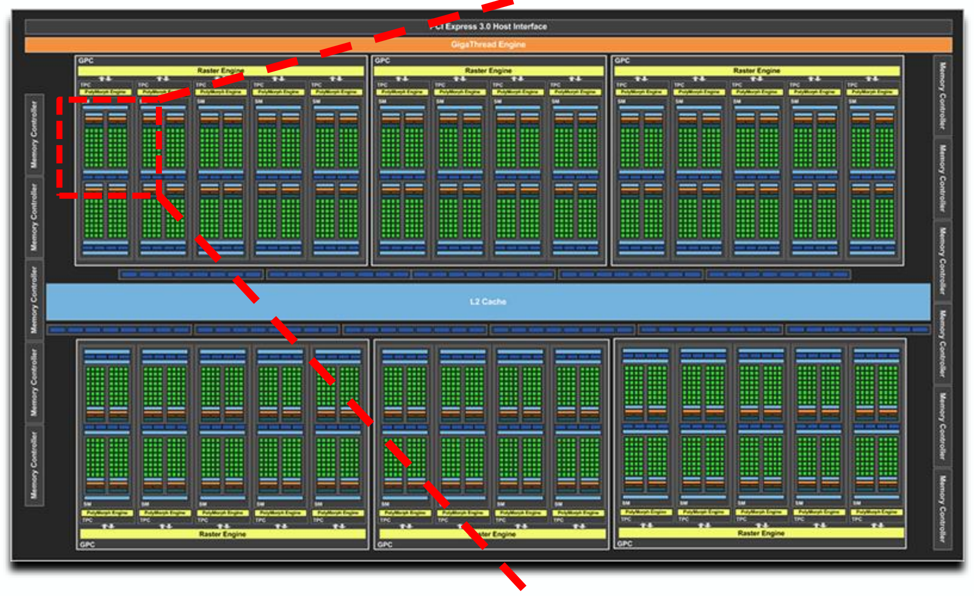
GPU	GeForce GTX 1080 (Pascal)
SMs	20
CUDA Cores	2560
Base Clock	1607 MHz
GPU Boost Clock	1733 MHz
GFLOPs	8873[1]
Texture Units	160
Texel fill-rate	277.3 Gigatexels/sec
Memory Clock (Data Rate)	10,000 MHz
Memory Bandwidth	320 GB/sec
ROPs	64
L2 Cache Size	2048 KB
TDP	180 Watts
Transistors	7.2 billion
Die Size	314 mm^2
Manufacturing Process	16 nm

Background

Pascal Architecture

GP100 Chip

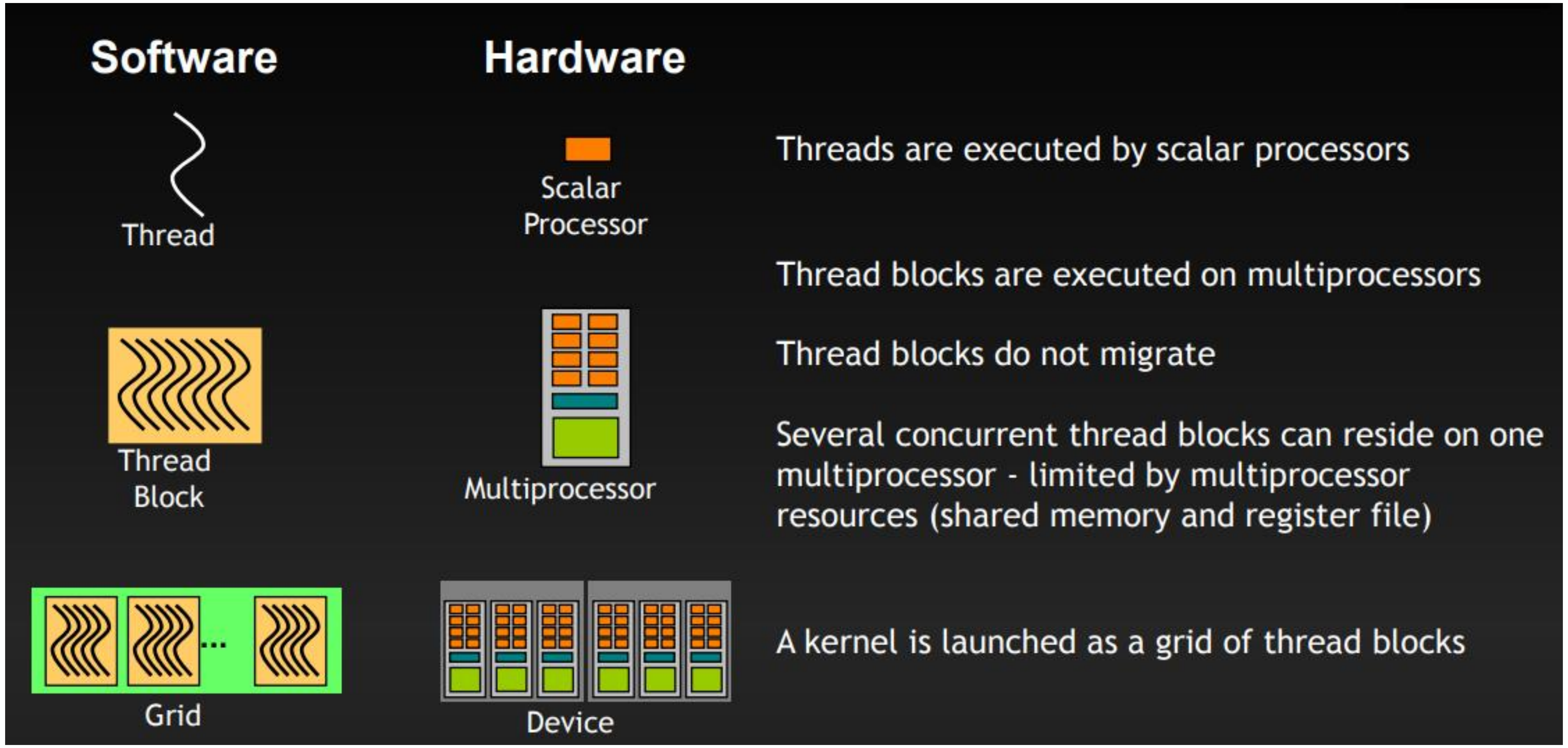
GPU Hardware Architecture



Streaming Multiprocessor

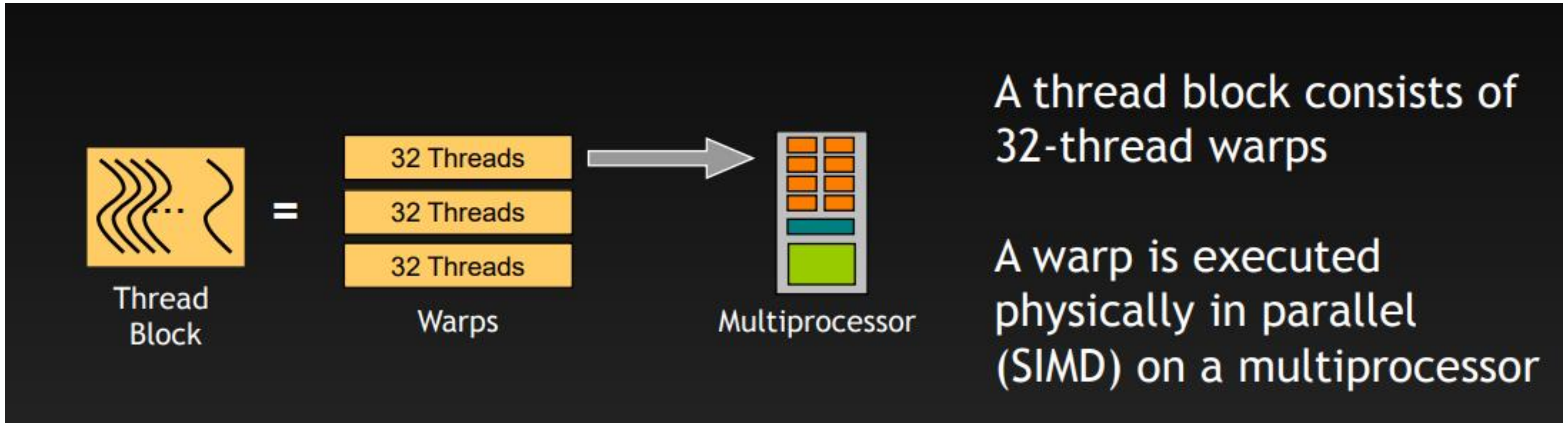
Background

GPU Hardware Architecture



Background

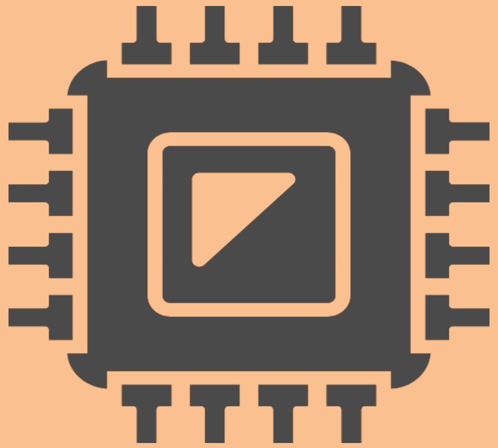
GPU Hardware Architecture



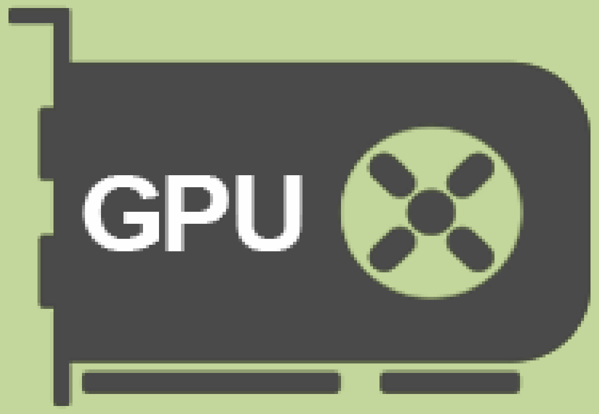
GPU Programming Model

Introducing two new terms:

- Host



- Device



GPU Programming Model

```
__device__ int index(){
     return threadId();
}

__global__ void kernel(int n){
   printf("Hello World from %d\n",index());
   return ;
}

__host__ int main(int argc, char ** argv){

     int blocks = 2;
     int threads_per_block = 2;

     kernel<<<blocks,threads_per_block>>>();

     return 0;
}
```

And three new keywords:

- host
- global
- device

GPU Programming Model

```
__device__ int index(){
     return threadId();
}
```

parallel code

```
__global__ void kernel(int n){
   printf("Hello World from %d\n",index());
   return ;
}
```

parallel code

```
__host__ int main(int argc, char ** argv){

     int blocks = 2;
     int threads_per_block = 2;
```

serial code

```
     kernel<<<blocks,threads_per_block>>>();
```

parallel code

```
     return 0;
}
```

serial code

GPU Synchronization Library

GSYNC implementation:

1. Atomic functions
2. Spinlocks
3. Ticketlocks
4. Barriers
5. Merge library with M4 MACROS

GPU Synchronization Library

Atomic operations:

- FETCH_AND_ADD
- INCREMENT
- FETCH_AND_INCREMENT
- DECREMENT
- FETCH_AND_DECREMENT
- XCHG
- CMPXCHG

Summary
Background
• GSYNC
Results
Conclusion

GSYNC Locks

SPINLOCK:

__host__ *gpulock_t* * **gspinlock_init**(*gpulock_t* * lock);

__device__ *void* **gspinlock_lock**(*gpulock_t* * lock);

__device__ *void* **gspinlock_unlock**(*gpulock_t* * lock);

GSYNC

GSYNC Locks

TICKETLOCK:

```
__host__ gputlock_t * gticketlock_init(gputlock_t * lock);

__device__ void gticketlock_lock(gputlock_t * lock);

__device__ void gticketlock_unlock(gputlock_t * lock);
```

GSYNC BARRIER

```
__host__ gbarrier_t * gbarrier_init(
gbarrier_t * barrier,
unsigned int init_value);
```

```
__device__ void gbarrier_wait(gbarrier_t * barrier);
```

```
__device__ void gbarrier_destroy(gbarrier_t * barrier);
```

GSYNC & M4 MACROS

Usage Functions:

- **G_MALLOC(** destination, size**)**
  -- malloc -> cudaMallocManaged

- **CREATE(**func,p,args...**)** --> func <<< p,1 >>> (p,args)

  -- function must be first argument
  -- number of total threads must be second argument
  -- rest of args

GSYNC & M4 MACROS

Lock Functions:

- **LOCKDEC(** lock **)**
- **LOCKINIT(** lock **)**
- **LOCK(** lock **)**
- **UNLOCK(** lock **)**

Barrier Functions:

- **BARDEC (** barrier **)**
- **BARINIT(** barrier, *int* num **)**
- **BARRIER(** barrier **)**

GSYNC & M4 MACROS

Helper Functions:

- **WAIT_FOR_END(** *int* threads **)**
- **GET_PID (** *int* pid **)**
- **GCLOCK_START(** start **)**
- **GCLOCK_END(** end **)**
- **GCLOCK_DIFF(** *float* ms_elapsed **)**

Results

My TestApplication:

- Spinlocks only:

  lock_demo

```
__global__ void mykernel(
int P,
gpulock_t * lock) {
    int pid;
    GET_PID(pid);
    LOCK(lock);
    printf("[D] (cs%d)\n",pid);
    UNLOCK(lock);
}
```

- Spinlocks & Barriers:

  lock&bar_demo

```
__global__ void mykernel(
int P,
gpulock_t * lock,
gbarrier_t * barrier) {

    int pid;
    GET_PID(pid);
    LOCK(lock);
    printf("[D] (cs%d)\n",pid);
    UNLOCK(lock);
    printf("[D] (before %d,%u)\n",pid,*barrier);
    BARRIER(barrier);
    printf("[D] (after %d,%u)\n",pid,*barrier)
}
```

Results

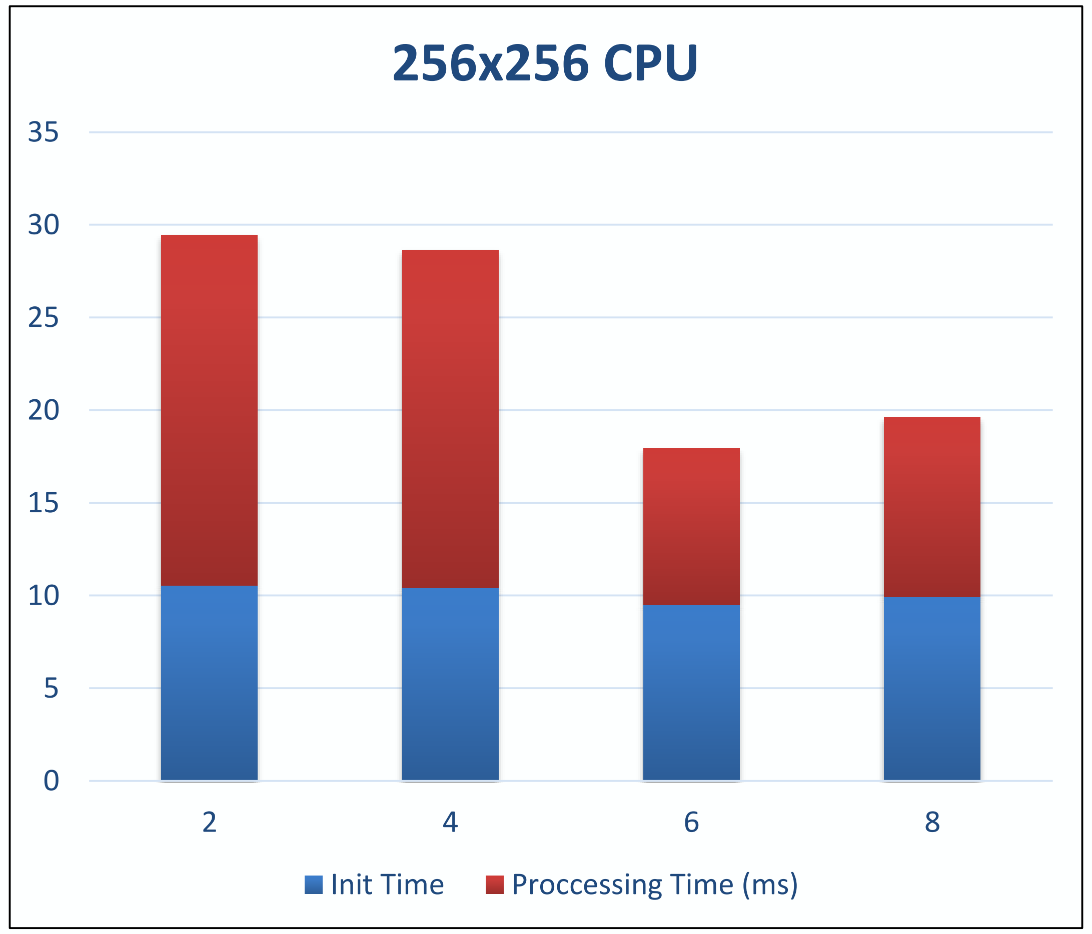
NOTE

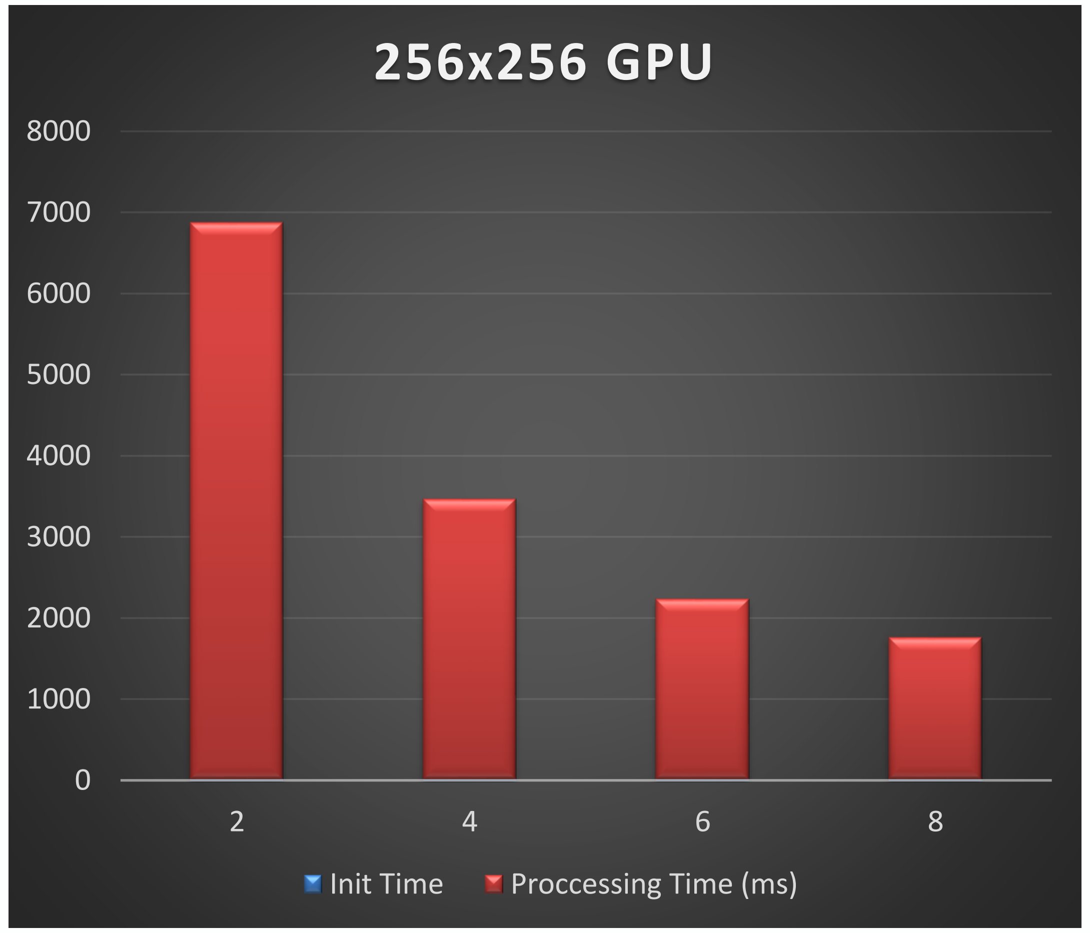
“ Internally printf() uses a shared data structure and so ***it is possible that calling printf() might change the order of execution of threads***. In particular, a thread which calls printf() might take a longer execution path than one which does not call printf(), and that path length is dependent upon the parameters of the printf(). Note, however, that CUDA makes no guarantees of thread execution order except at explicit __syncthreads() barriers, so ***it is impossible to tell whether execution order has been modified by printf() or by other scheduling behaviour in the hardware***.“

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#formatted-output

Results

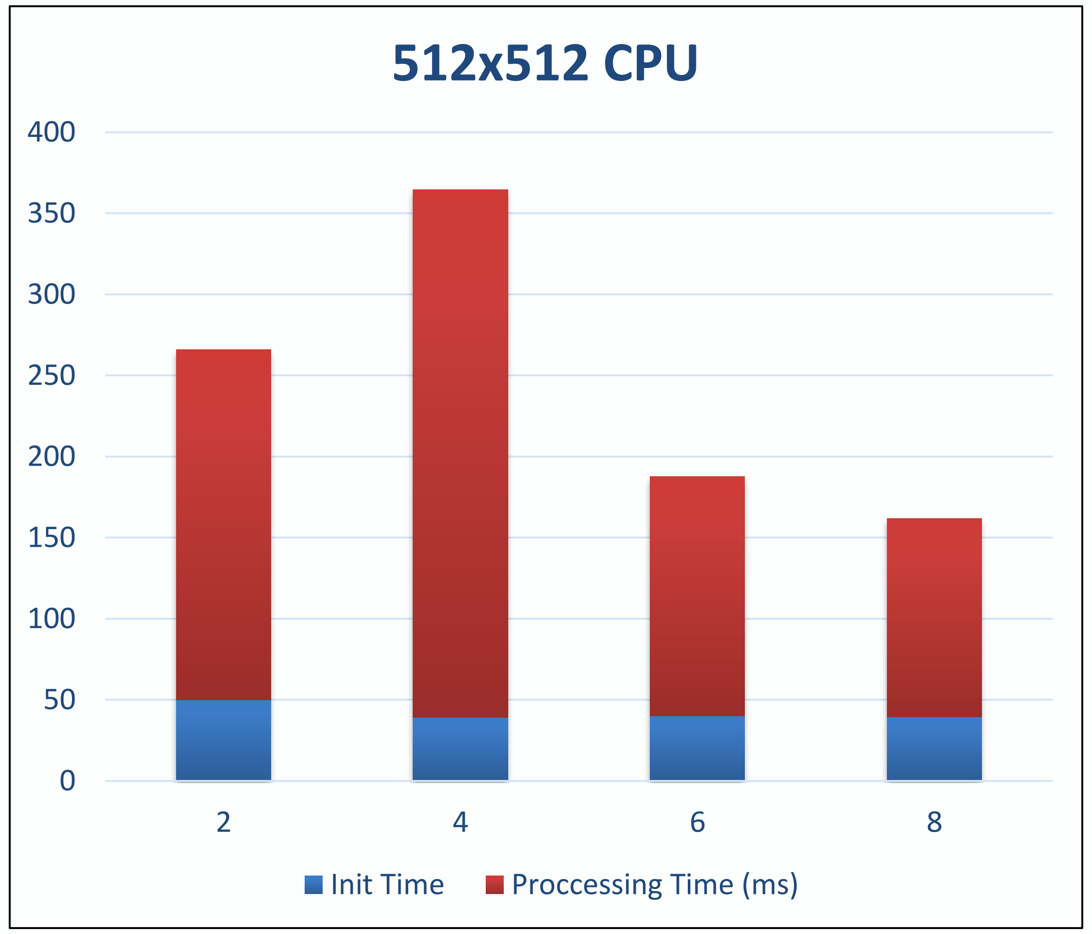
GSYNC on Matrix Multiplication 256 x 256

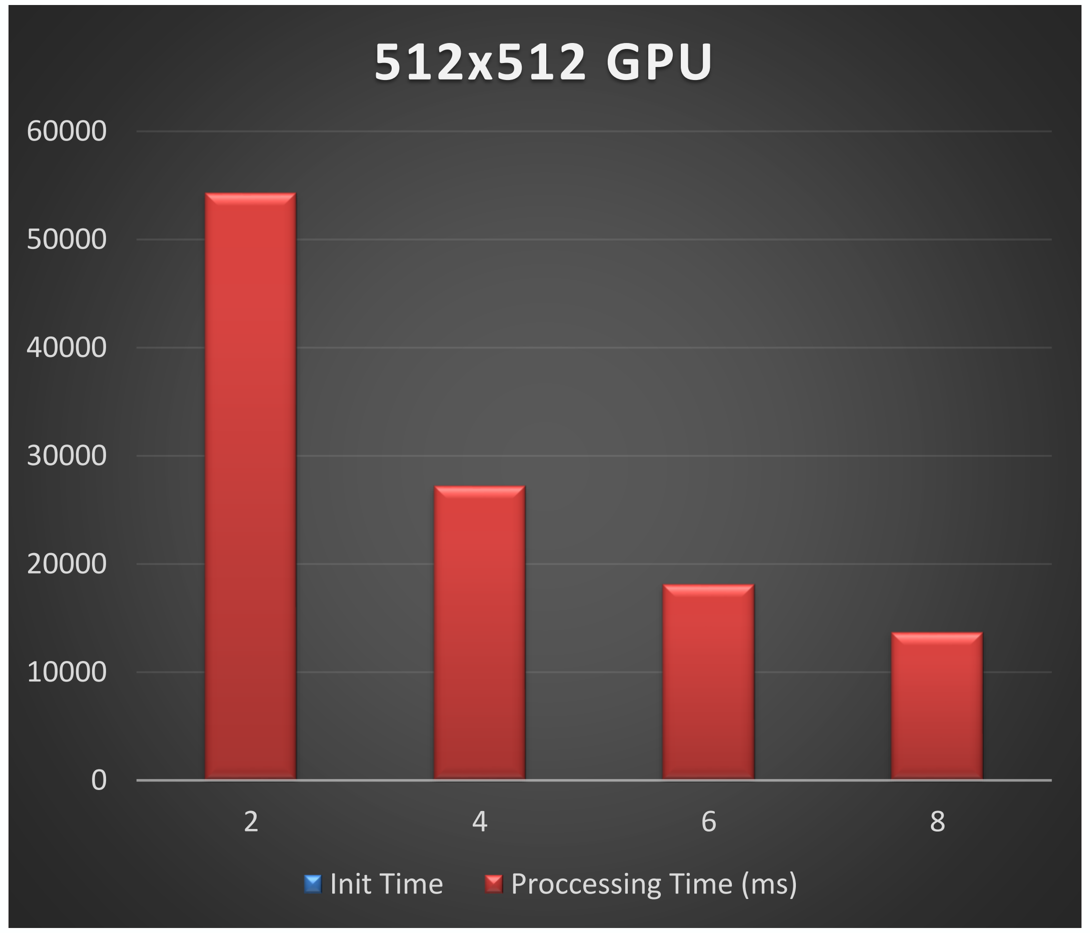




Results

GSYNC on Matrix Multiplication 512 x 512

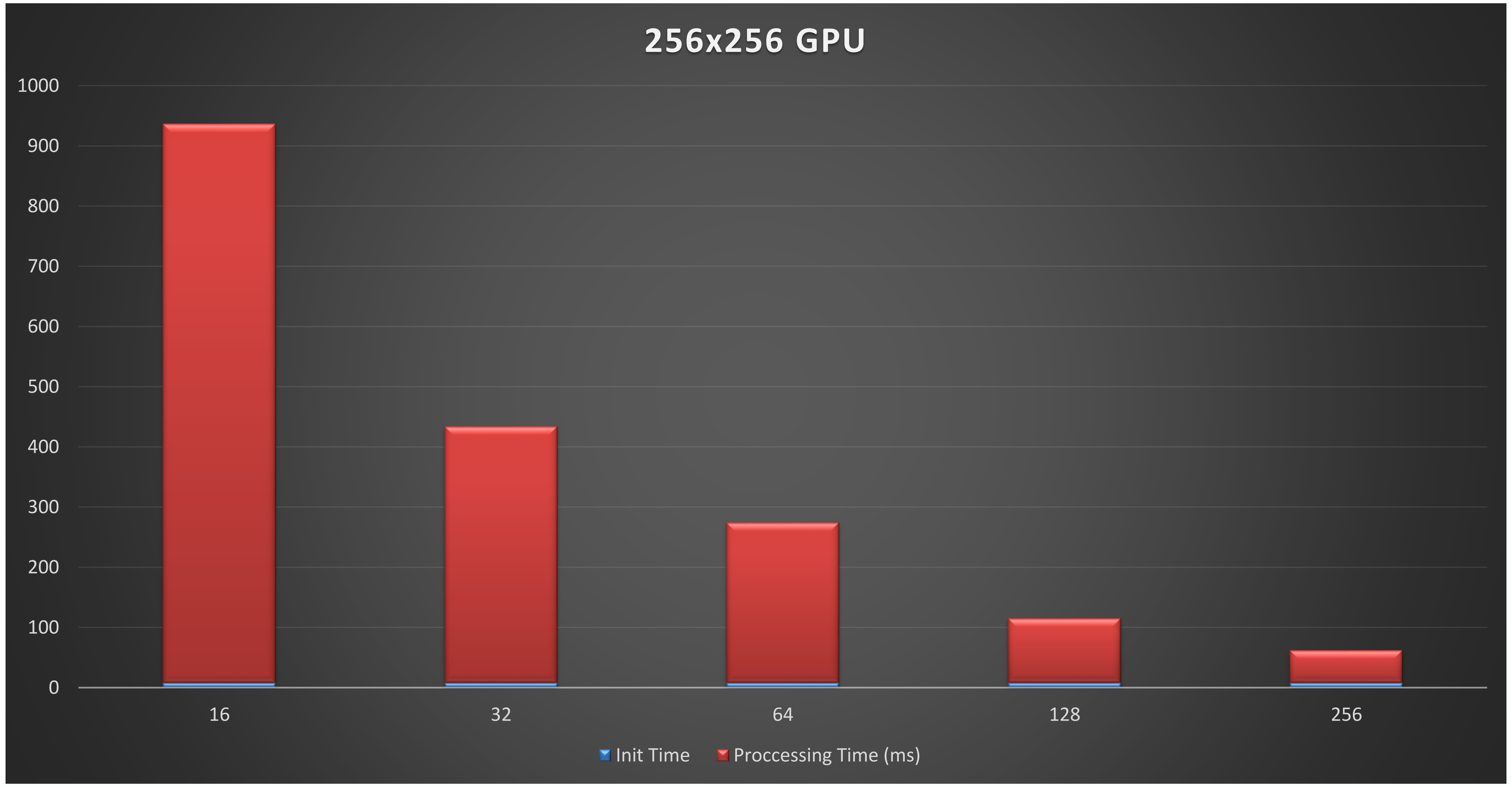




Results

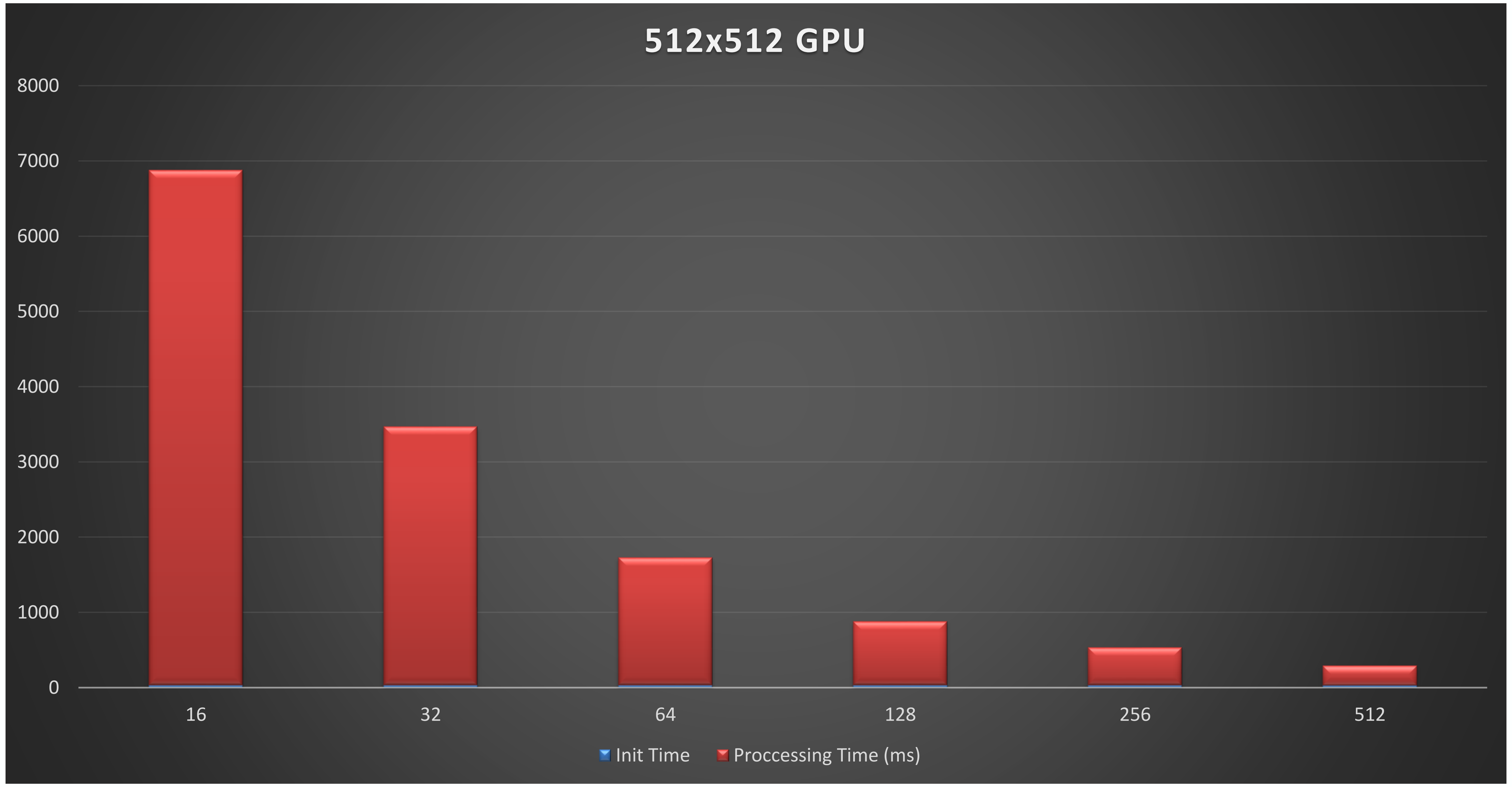
Results

GSYNC on Matrix Multiplication 256 x 256 (more threads)



Results

GSYNC on Matrix Multiplication 512 x 512 (more threads)



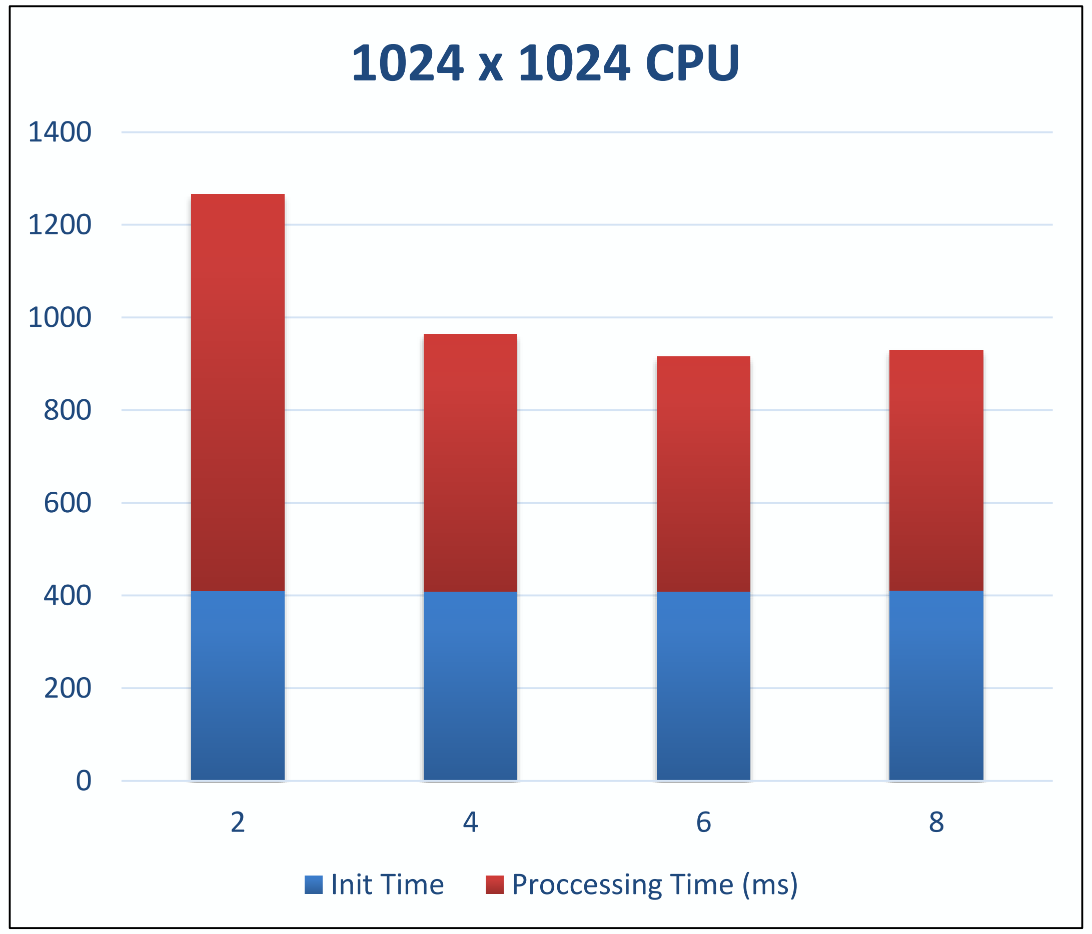
Results

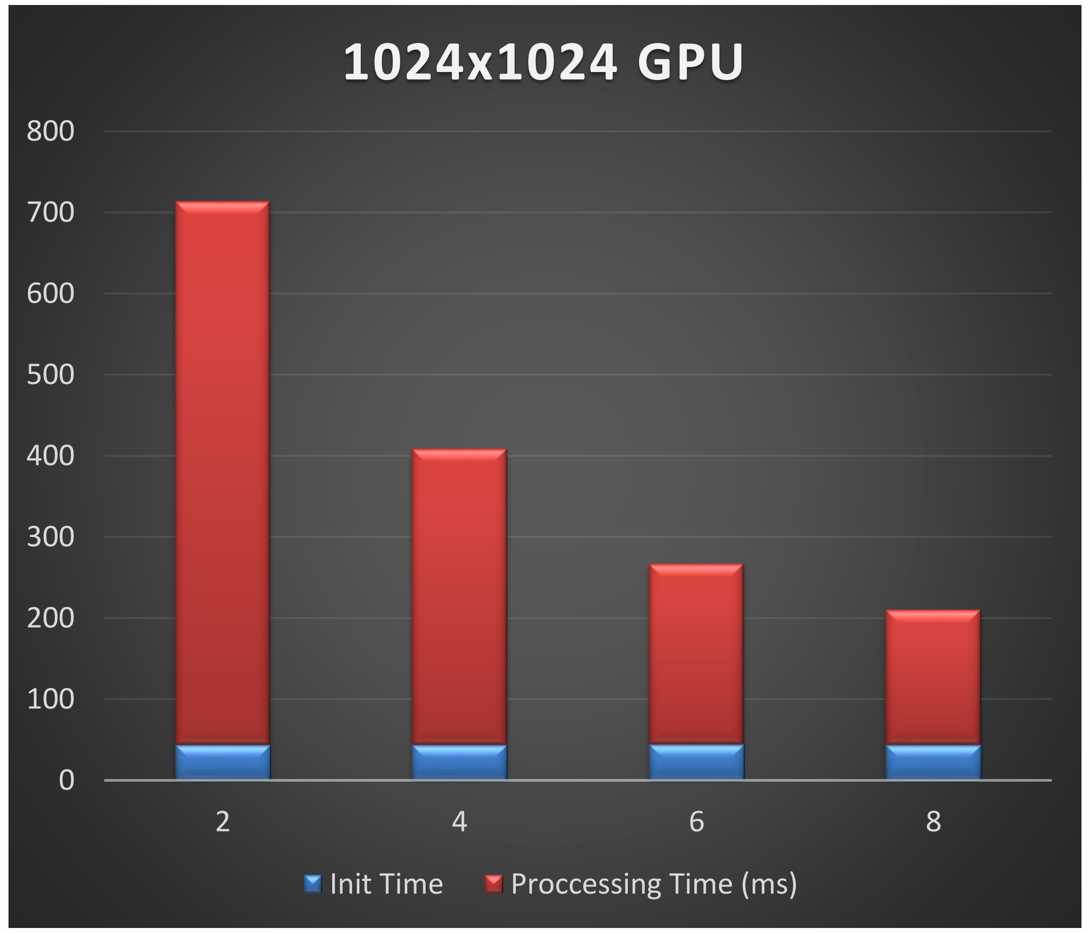
Ported GSYNC **SPLASH-2** Programs:

- LU
- Radix
- FFT (working partially)

Results

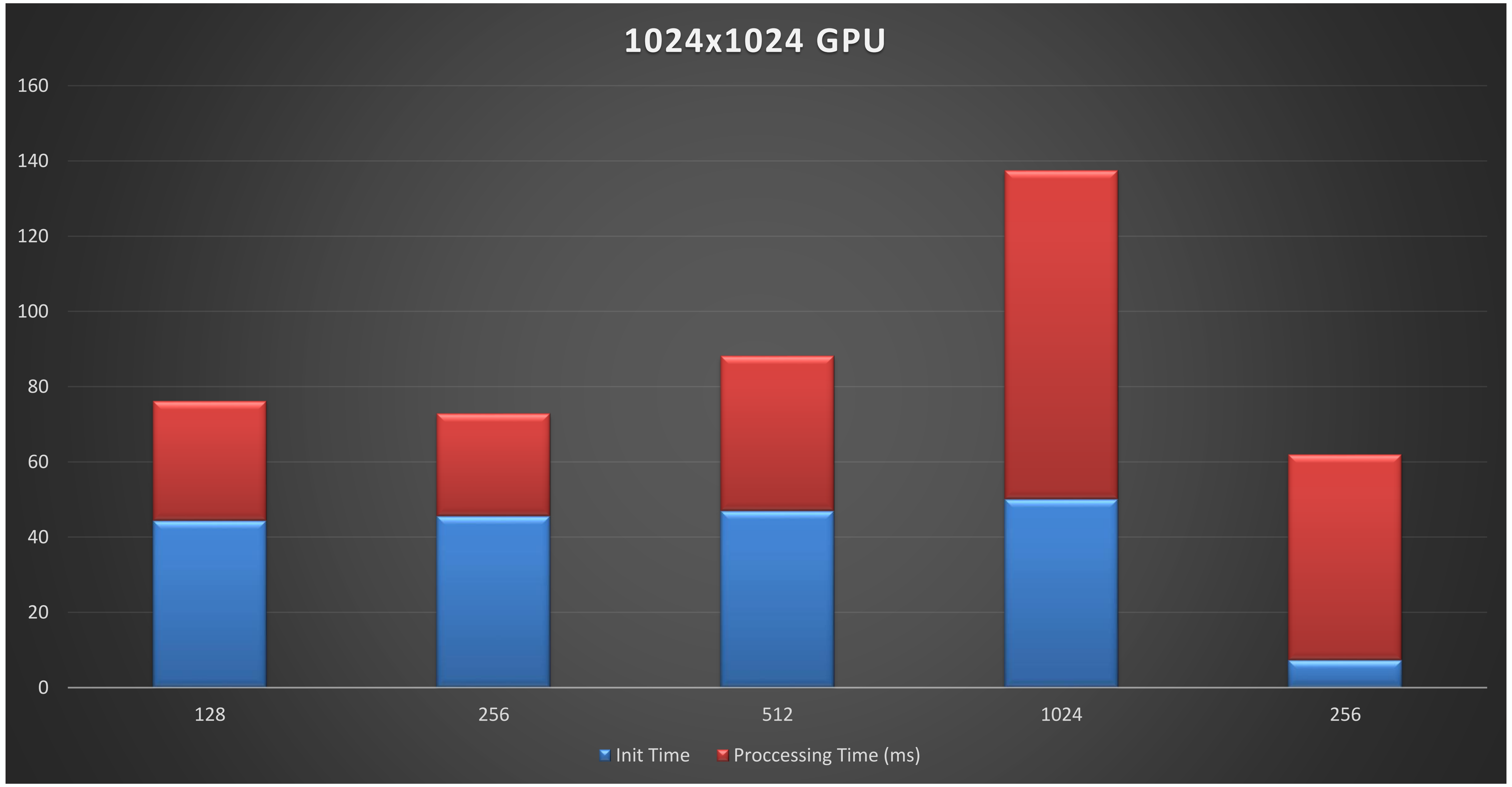
GSYNC on LU 1024 x 1024





Results

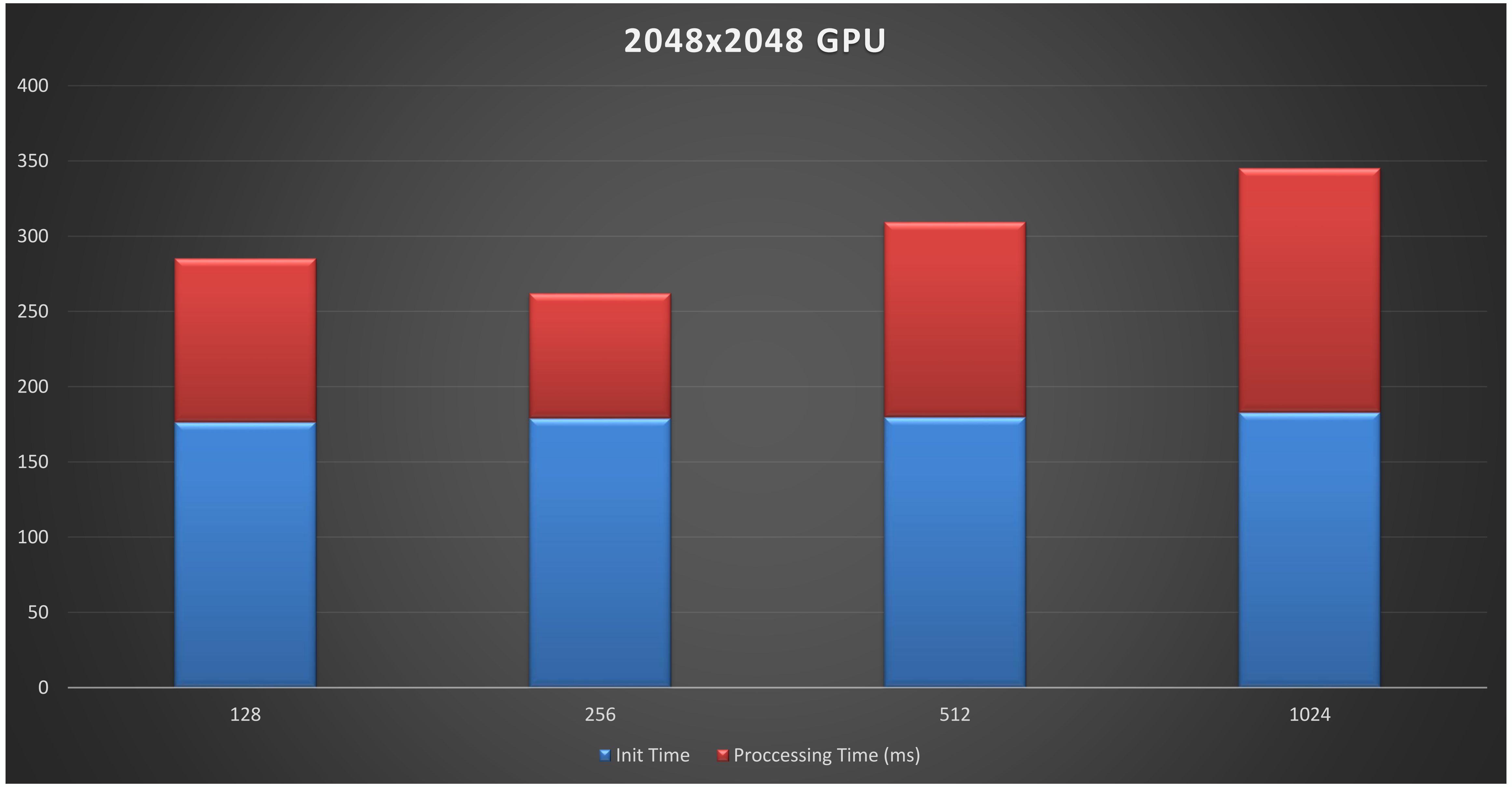
GSYNC on LU 1024 x 1024 (more threads)



Results

Results

GSYNC on LU 2048 x 2048 (more threads)



Conclusion

GSYNC is a library that utilizes the atomic operations provided by the CUDA API and offers synchronization between the GPU threads.

Also emerged with the M4 Macros.

GSYNC disadvantages:

- SIMD in Blocks.
  create many blocks <<< create many threads

- Access to memory allocated by *cudaMallocManaged* is slow.

Conclusion

Conclusion

Do not buy GSYNC