GPU COMPUTING

Difference between CPU and GPU Architecture

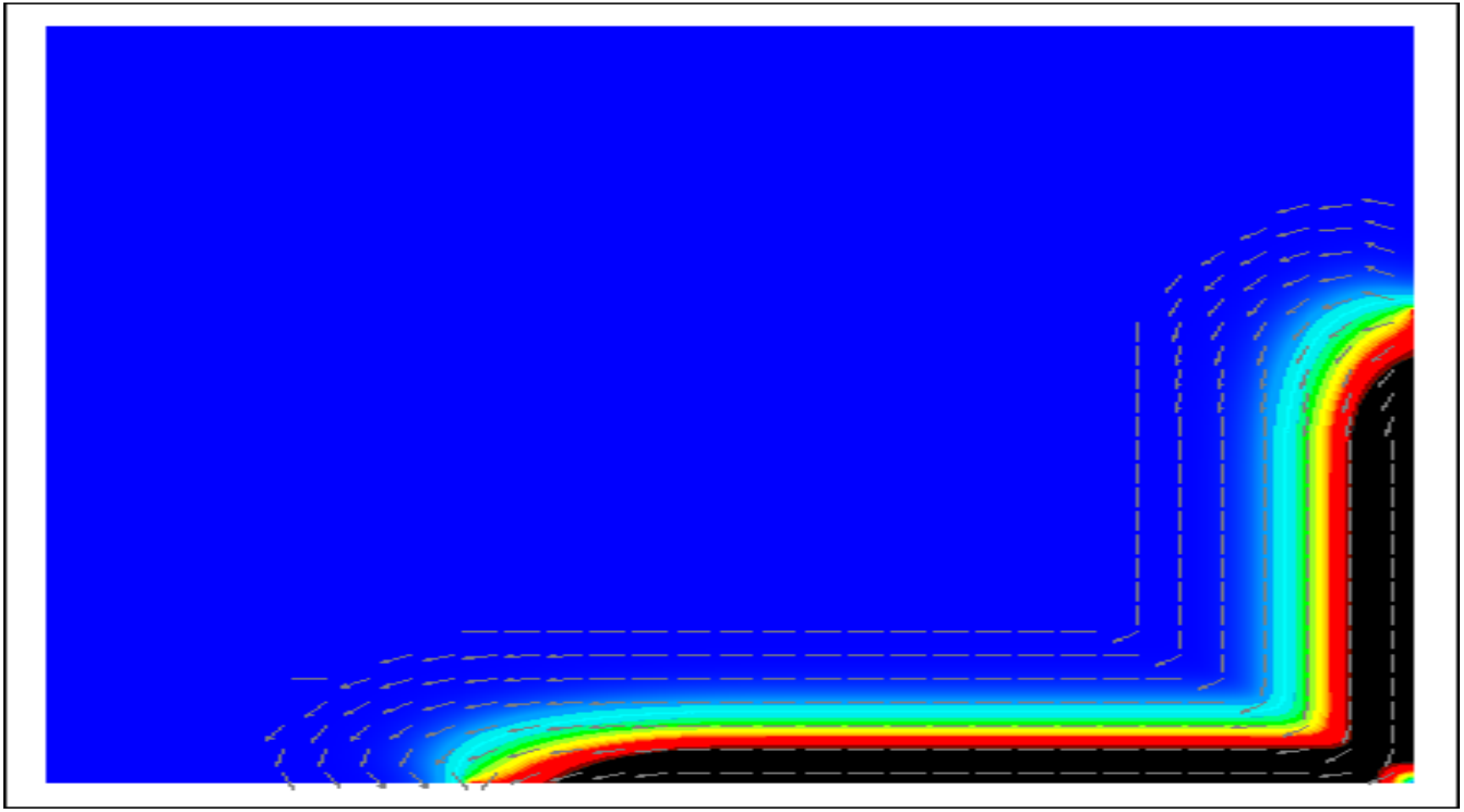
CPU

- More suitable for low latency workloads
- CPU has bigger memory caches with lower memory bandwidth
- Higher Processor Clock Rate
- Execution Model SIMD/SMT
- Data don’t need to moved from Host to Device to be processed

GPU

- More suitable for high throughput workloads
- GPU has less memory capacity with higher main memory bandwidth
- Lower Processor Clock Rate
- Execution Model SIMT
- Data Transfer between host and device is a Limiting performance factor

PDE Calculation Plot (32x32 Grid 5000 iter)



OpenACC
More Science, Less Programming

KNOW YOUR APPLICATION

```
10  id jacobistep(double **psinew, double **psi, int m, int n)
11
12  int i, j;
13
14  for(i=1;i<=m;i++)
15    {
16     for(j=1;j<=n;j++)
17    {
18      psinew[i][j]=0.25*(psi[i-1][j]+psi[i+1][j]+psi[i][j-1]+psi[i][j+1]);
19       }
20   }
21
```

Time(%)	Total Time (ns)	Instances	NVTX
33.9	18,600,340,372	1	total
33.1	18,136,889,464	1	main_simulation_loop
20.6	11,304,100,596	5,000	jacobi
12.4	6,822,128,866	5,000	swap
0.0	8,352,028	1	init
0.0	1,970,085	5,000	calculate_error

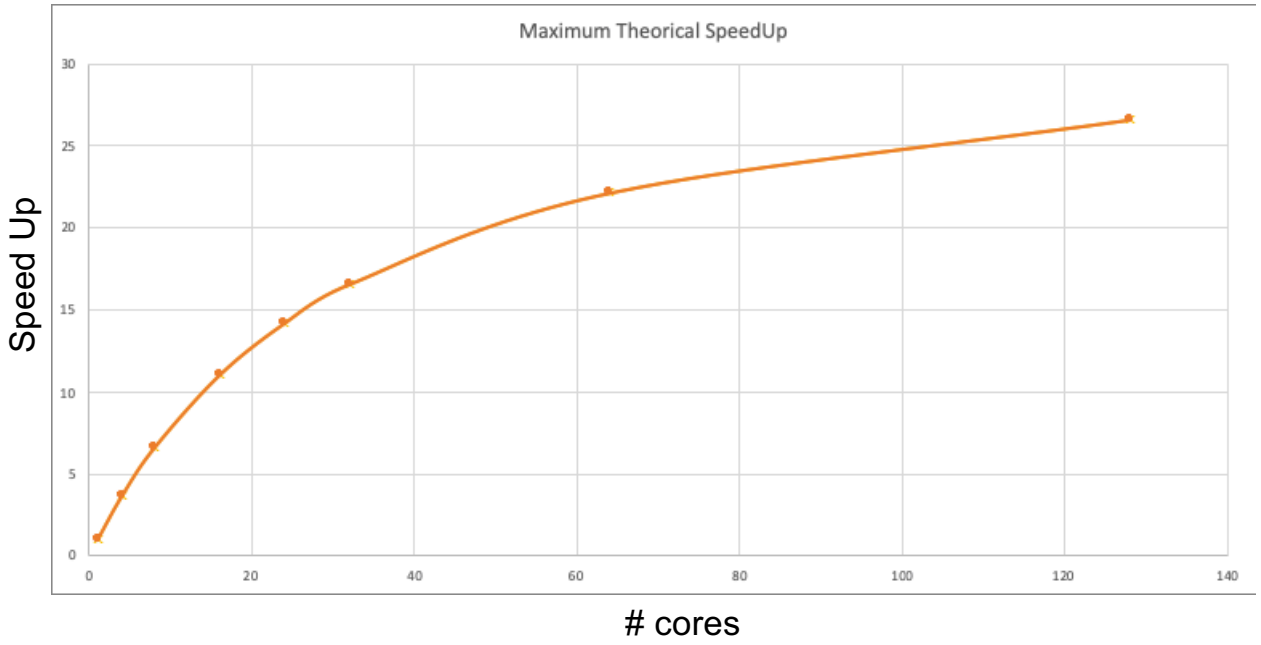
The more important kernel in terms of execution time is jacobi. In the nested for-loop at line (18) for one multiplication and 3 additions 4 double values are fetched from global memory and the result is stored back in global memory. The compute-to-global-memory-access ratio is 4/10 = 0.4. The execution time is limited by the speed at which double-precision operands are fetched from global memory. The application is memory bound.

Theorical Speed Up using Amdahls Law

Time(%)	Total Time (ns)	Instances	Average (ns)	Minimum (ns)	Maximum (ns)	StdDev (ns)	NVTX
33.9	18,600,340,372	1	18,600,340,372.0	18,600,340,372	18,600,340,372	0.0	total
33.1	18,136,889,464	1	18,136,889,464.0	18,136,889,464	18,136,889,464	0.0	main_simulation_loop
20.6	11,304,100,596	5,000	2,260,820.1	2,175,354	9,488,915	137,055.8	jacobi
12.4	6,822,128,866	5,000	1,364,425.8	1,338,596	6,271,088	140,306.2	swap
0.0	8,352,028	1	8,352,028.0	8,352,028	8,352,028	0.0	init
0.0	1,970,085	5,000	394.0	136	746,645	10,559.8	calculate_error

$$S(n) = \frac{1}{(1-P) + \frac{P}{n}}$$

# cores [n]	Theorical SpeedUp [S(n)]
1	1
4	3.669724771
8	6.611570248
16	11.03448276
24	14.20118343
32	16.58031088
64	22.14532872
128	26.61122661
Parallizable Fraction[P]	0.97



MULTICORE ACCELERATION

Multicore parallelization results.

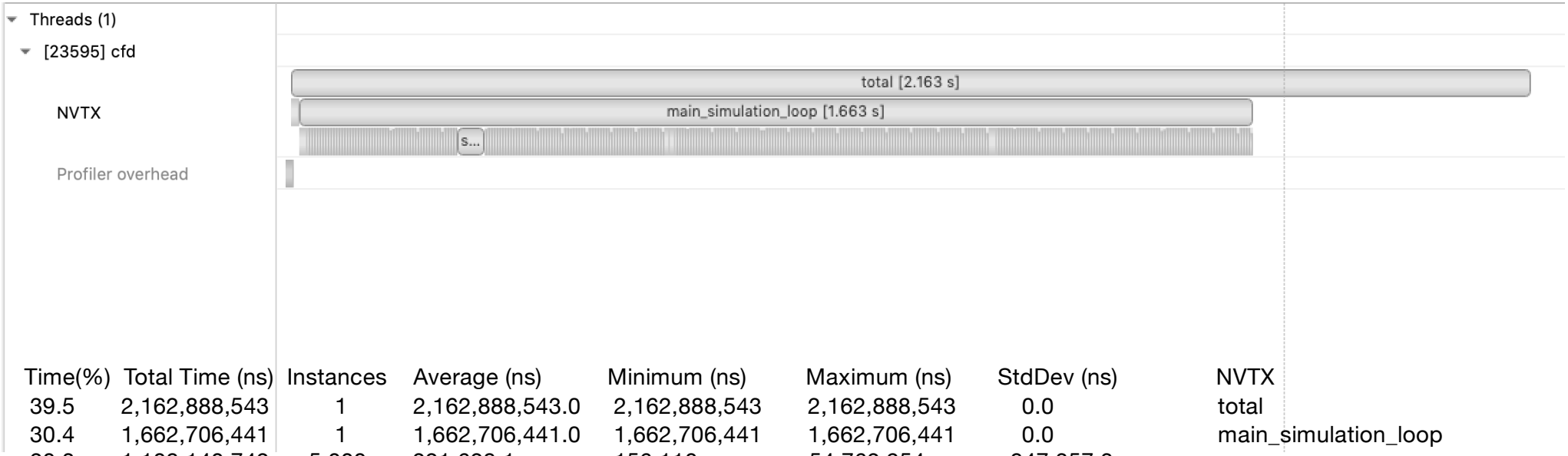
Mode.	Total Time(s)	main_simulation_loop Time(s)	Speedup
Serial	18.495	18.026	0
Multicore	2.163	1.663	10x

Percentage of speedup : 10x

Multicore Programming Model - OpenACC

CPU Node 48-core Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz, 192GB RAM

MULTICORE ACCELERATION - Profiling



Time(%)	Total Time (ns)	Instances	Average (ns)	Minimum (ns)	Maximum (ns)	StdDev (ns)	NVTX
39.5	2,162,888,543	1	2,162,888,543.0	2,162,888,543	2,162,888,543	0.0	total
30.4	1,662,706,441	1	1,662,706,441.0	1,662,706,441	1,662,706,441	0.0	main_simulation_loop
20.3	1,108,140,742	5,000	221,628.1	156,110	54,762,954	847,357.3	swap
9.5	520,805,772	5,000	104,161.2	74,351	19,069,796	395,376.7	jacobi
0.3	15,087,815	1	15,087,815.0	15,087,815	15,087,815	0.0	init
0.0	1,874,269	5,000	374.9	266	102,343	1,444.6	calculate_error

GPU ACCELERATION

GPU parallelization results.

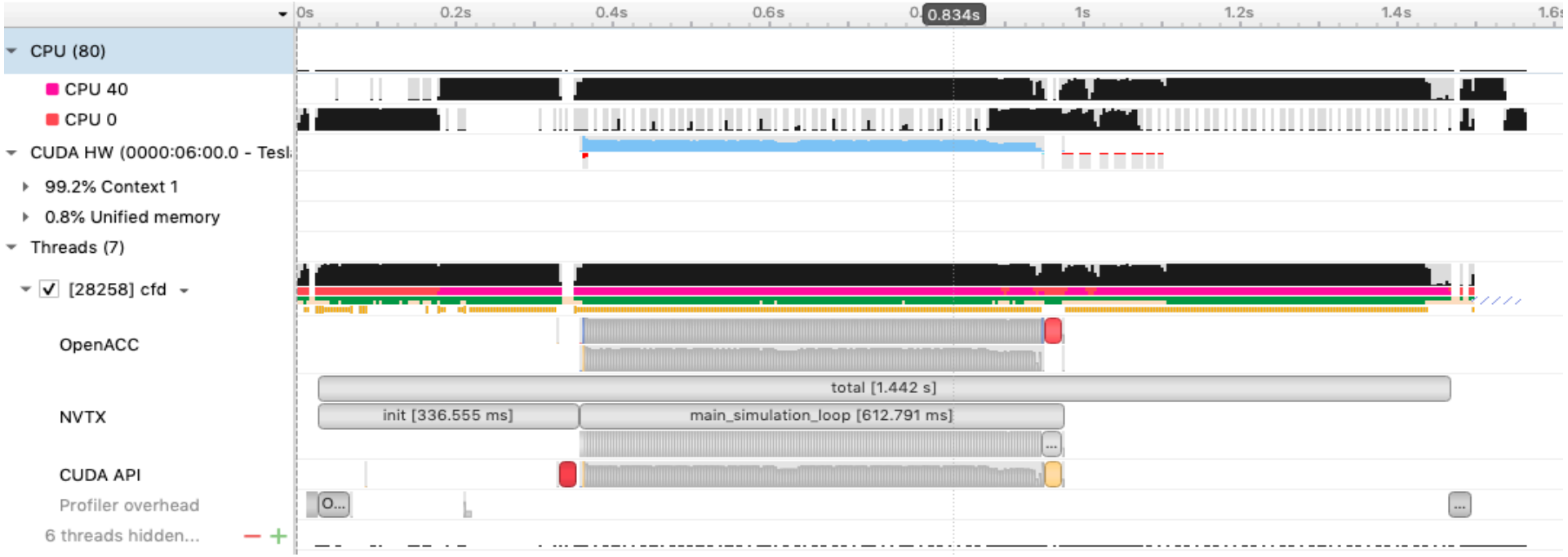
Mode.	Total Time(s)	main_simulation_loop Time(s)	Speedup
Serial	18.495	18.026	0
Multicore	2.163	1.663	10x
GPU	1.42	0.612	29x

Percentage of speedup : 29x

GPU Programming Model – OpenACC using TILE/managed memory

1 NVIDIA VOLTA 5120 FP32 cores / 2560 FP64 cores 16GB RAM

GPU ACCELERATION – nsys profiling

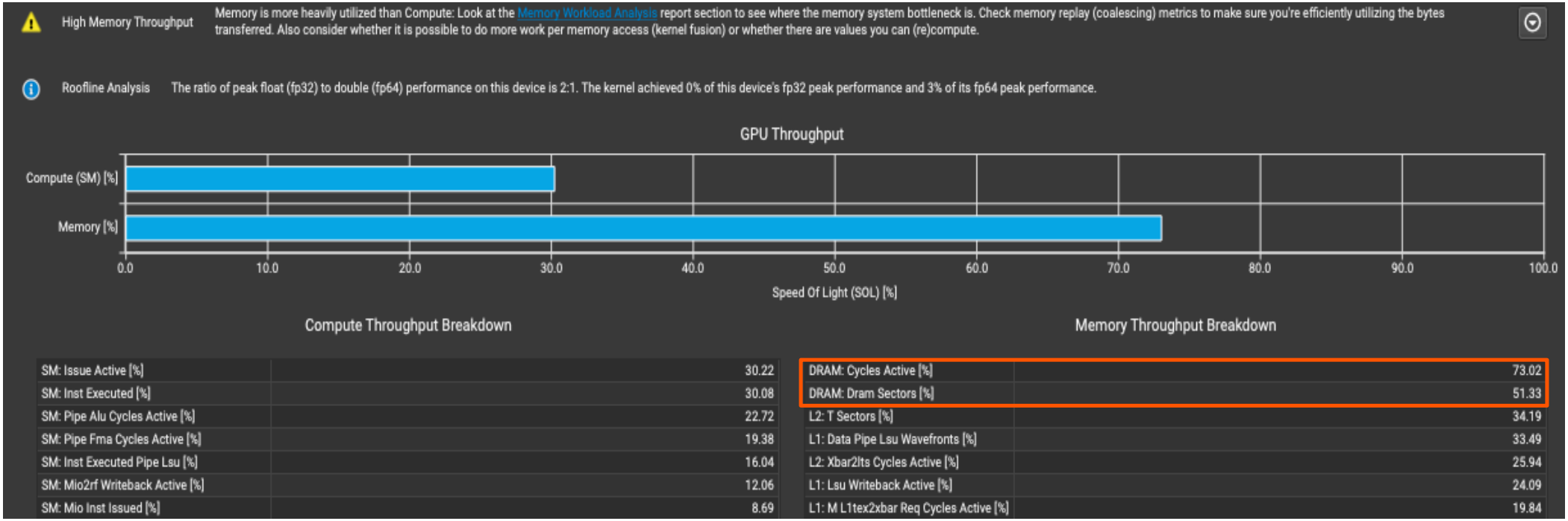


Time(%)	Total Time (ns)	Instances	Average (ns)	Minimum (ns)	Maximum (ns)	StdDev (ns)	NVTX
38.4	3,282,267,380	1	3,282,267,380.0	3,282,267,380	3,282,267,380	0.0	total
33.0	2,824,520,551	1	2,824,520,551.0	2,824,520,551	2,824,520,551	0.0	main_simulation_loop
14.3	1,223,043,756	5,000	244,608.8	227,902	1,236,437	38,328.4	jacobi
14.2	1,210,402,922	5,000	242,080.6	226,149	477,625	33,842.8	swap
0.1	7,693,156	1	7,693,156.0	7,693,156	7,693,156	0.0	init
0.0	1,636,038	5,000	327.2	136	547,247	7,740.7	calculate_error

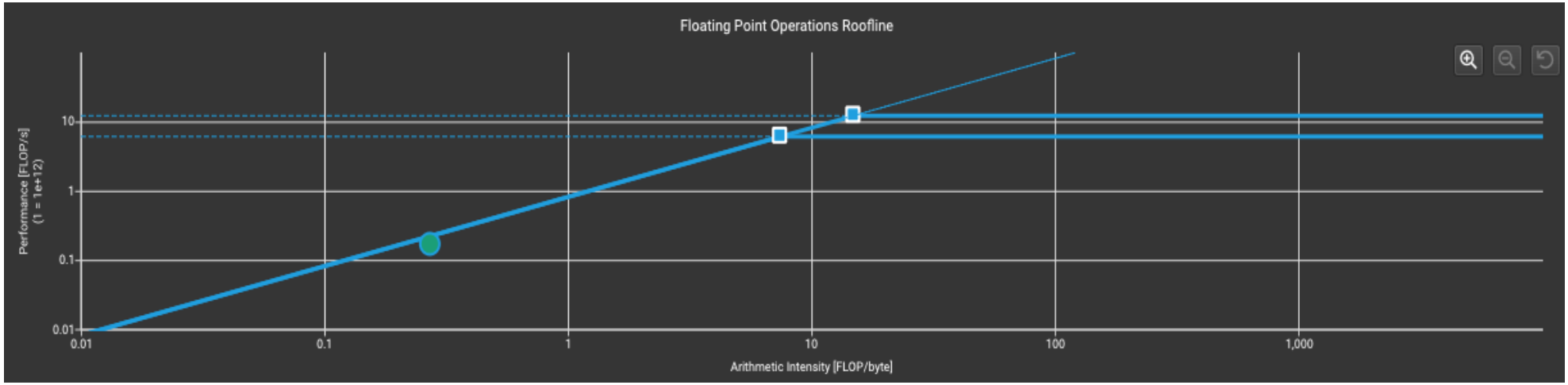
GPU ACCELERATION – Profiling

Compute (SM) is only used 30% vs Memory is dominant with more than 70% utilization

The kernel jacobistep is memory bound and rely too much on global memory access.

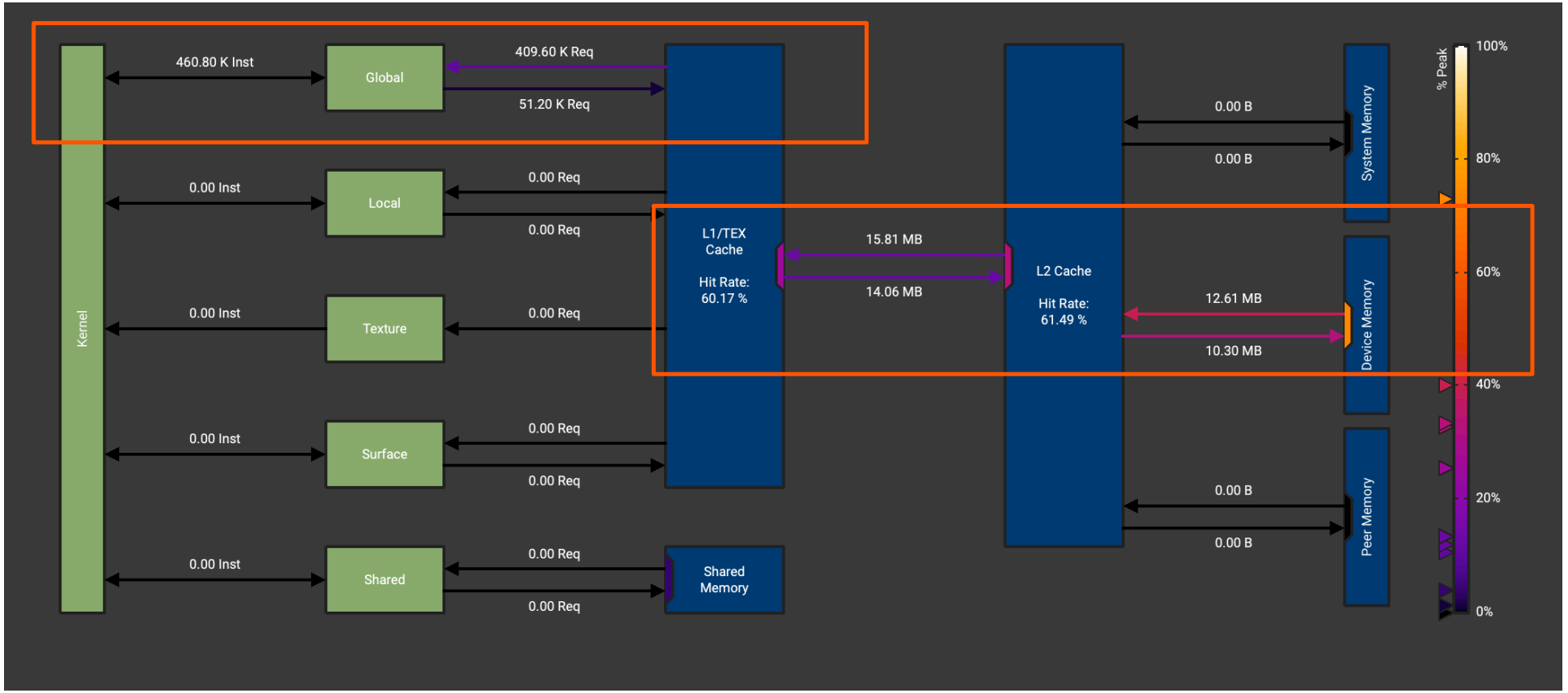


The memory-bound kernel jacobi peak performance is far from good when running on a V100 GPU device. Achieved only a 3% of fp64 theorical peak performance.

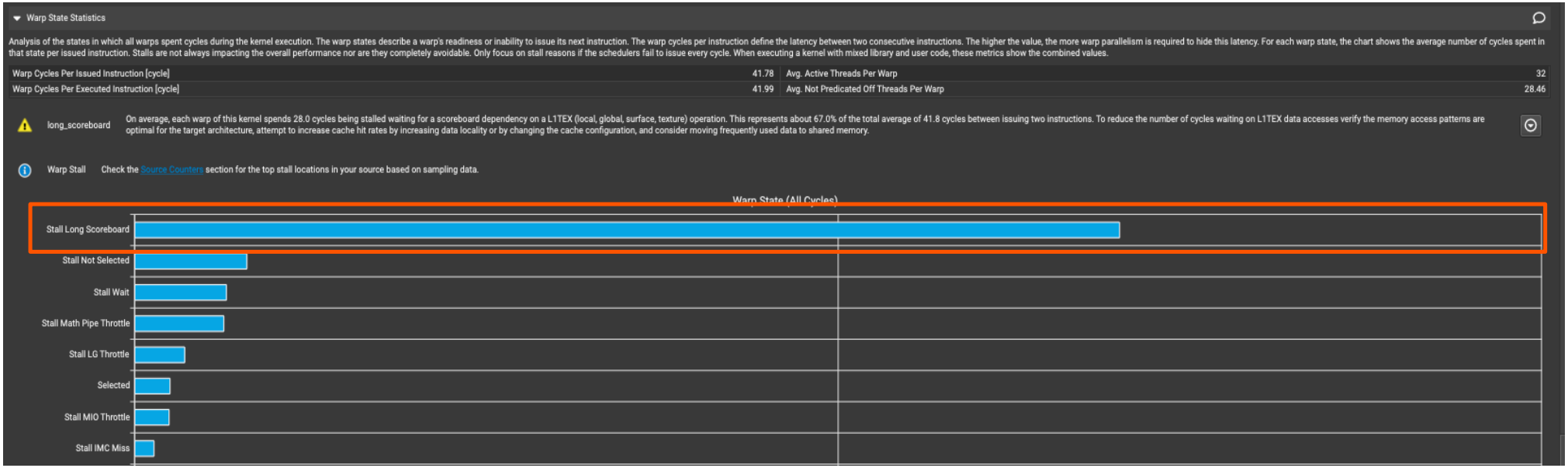


	Arithmetic Intensity	Peak Performance
Theorical Fp64 Roof Line	7.45	6 TFLOPS
Effective	0.27	166 GFLOPS

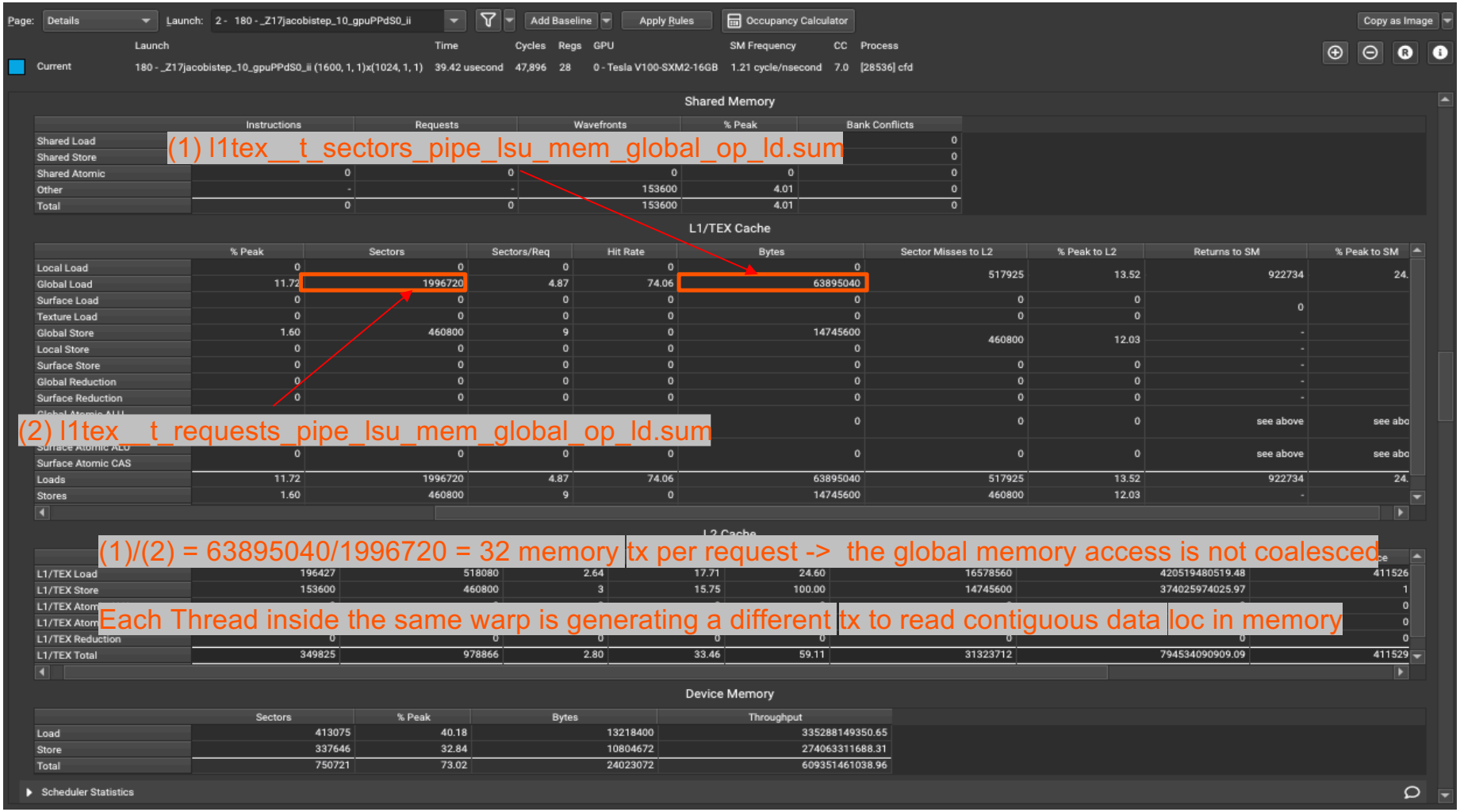
The optimization focus is the global memory access, no other memory is used by the kernel jacobistep



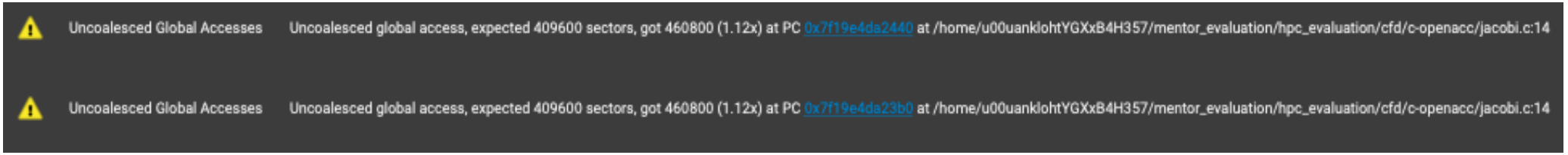
Stall Long Scoreboard top warp stall reason due to excessive number of global memory access

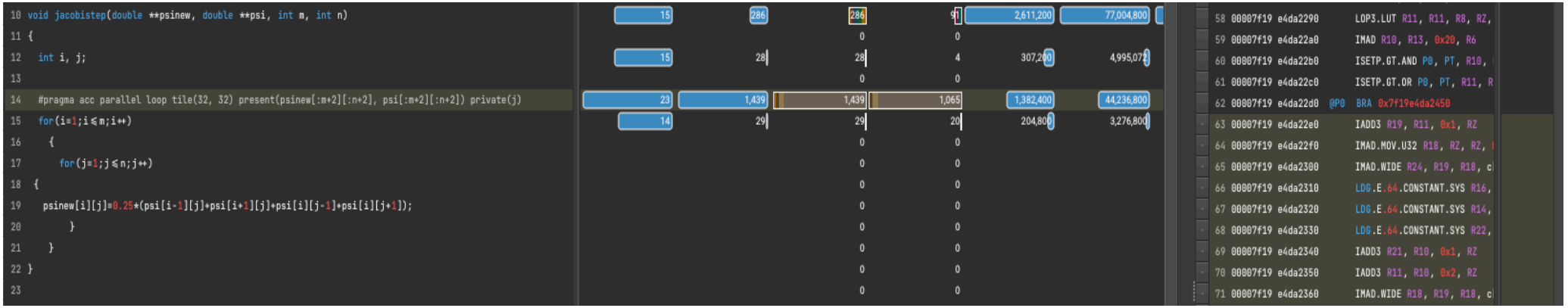


Not efficient Global Memory Access



Uncoalesced Global Memory Access at jacobistep line 14





OpenACC
More Science, Less Programming

Further Optimization

Change Loop Order to improve memory coalescing in function jacobistep at file jacobi.c

- Index i should be moved as the last dimension for coalesced memory access
- Parallelize the external loop line (15) and distribute the execution across multiple threads blocks
- Vectorize the nested loop in line (18)

```
14 #pragma acc parallel loop copyout(psinew[:m+2][:n+2]) copyin(psi[:m+2][:n+2])
15 for(j=1;j<=n; j++)
16  {
17 #pragma acc loop vector
18      for(i=1;i<=m;i++)
19      {
20          psinew[j][i]=0.25*(psi[j][i-1]+psi[j][i+1]+psi[j-1][i]+psi[j +1][i] );
21      }
22 }
```

GPU parallelization results.

Mode.	Total Time(s)	Speedup
Serial	18.495	
Multicore	2.163	10x
GPU(tile)	1.42	29x
GPU(gang/mem coalescing)	0.45	40x

Swap pointers psi and psitmp instead of copy the elements from psitmp to psi in cfd.c's main function

Reimplement the jacobi kernel using CUDA and load the entire tile in shared memory to improve the memory access pattern and speed up the load of double precision operands.

HPC APPLICATION EVALUATION

Michel Herrera Sanchez
11/12/2021

OpenACC
More Science, Less Programming

GPU HACKATHONS

Learn more at

WWW.GPUHACKATHONS.ORG

OpenACC
More Science, Less Programming

GPU HACKATHONS