University POLITEHNICA of Bucharest
Advanced Computing in Embedded Systems
Distributed and High-Performance Computing

Heat Transfer

Introduction

Physical simulations can be among the most computationally challenging problems to solve. Fundamentally, there is often a trade-off between accuracy and computational complexity. As a result, computer simulations have become more and more important in recent years, thanks in large part to the increased accuracy possible because of the parallel computing revolution. Since many physical simulations can be parallelized quite easily, we will look at a very simple simulation model in this example.

In this paper, it simulated the heat transfer in a Microfluidic Device. The program receives an image with sensor model, computes heat transfer for a specific number of steps and make a short video with heat animation.

Objective

• Implementation of the serial implementation of the algorithm

• Implementation of the algorithm to check for the correctitude of the proposed method

• Implementation of the parallel implementation of the algorithm and optimize the results for the proposed paradigm

Background

The implementation of this application requires:

- C++ skills
- basic OpenCV skills
- Math
- Image Processing

Resources

- https://computing.llnl.gov/tutorials/parallel_comp/
- Barbara Chapman et al. , "Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)" , The MIT Press, 2007.
- Gregory V. Wilson, Paul Lu, "Parallel programming using C++" , The MIT Press, 1996.
- https://www.coursera.org/learn/parallelism-ia/home/welcome

Mathematical model

The law of heat conduction, also known as Fourier's law, states that the rate of heat transfer through a material is proportional to the negative gradient in the temperature and to the area, at right angles to that gradient, through which the heat flows.

The differential form of Fourier's law of thermal conduction shows that the local heat flux density q, is equal to the product of thermal conductivity, k, and the negative local temperature gradient, - ∇T . The heat flux density is the amount of energy that flows through a unit area per unit time.

$$q = -k\nabla T$$

where (including the SI units)
q is the local heat flux density, $W \cdot m^{-2}$
k is the material's conductivity, $W \cdot m^{-1} \cdot K^{-1}$,
∇T is the temperature gradient, $K \cdot m^{-1}$.

In my heat conduction model, we will compute the heat in every pixel of the picture. For each point of mesh we will solve numerical heat as a sum of old temperature and product of material's conductivity and the difference between current temperature and the temperatures of its neighbor (right, left, top, bottom).

$$q = T_{old} + k(T_{right} + T_{left} + T_{top} + T_{bottom} - 4 \cdot T_{old})$$

Algorithm description

Import Input Data

In this project, the algorithm will take the geometric model data from a bmp file. Where the black contour will represent the geometric shape of the heat resistance with max temperature T_MAX, and the white spaces, will represent the environment with the minimum temperature T_MIN.

```
pix     = cv::imread("mems.bmp", 1);
pix_out = pix;

if (pix.empty())
       std::cout << "Fail to open!" << std::endl;
else
       std::cout << "Image opened fine!" << std::endl;

picture_in = (float*)malloc(DIM * DIM * sizeof(float));
picture_out = (float*)malloc(DIM * DIM * sizeof(float));
picture_ct = (float*)malloc(DIM * DIM * sizeof(float));


for (int i = 0; i < DIM; i++) {
       for (int j = 0; j < DIM; j++) {
              picture_ct[DIM * i + j] = picture_in[DIM * i + j] =
              (pix.at<cv::Vec3b>(i, j)[0] == 0) ? T_MAX : T_MIN;
}
}
```

To create the short video animation we used the VideoWriter class of the OpenCV 3.0 library.

```
VideoWriter video("outcpp.avi", cv::VideoWriter::fourcc('M', 'J', 'P', 'G'), 10, Size(DIM, DIM));
```

The video is MJPEG compress, with a frame rate of 10 fps and DIM * DIM resolution.

The algorithm is structured in 2 main parts: heat transfer computing and conversion of numerical values in RGB space.

Heat Transfer Computation

Before calculating the heat transfer, I need to save the initial data in a constant image, which will be kept throughout the simulation so as not to lose the initial conditions. This will be used every step of the way to calculate heat transfer.

```
void copy_values_from_previous_step(float* in, float* ct) {
	for (int i = 0; i < DIM * DIM; i++) {
		if (ct[i] != T_MIN)
			in[i] = ct[i];}
}
```

Computation of the output temperatures is based on previous temperatures. In function update_values will be compute and save the temperatures. Then we will swap the input array with the output array.

```
void update_values(float* out, float* in) {
long long left, right, top, bottom;

for (long long i = 0; i < DIM * DIM; i++) {
	left = (i % DIM == 0) ? i : i - 1;
	right = (i % DIM == DIM - 1) ? i : i + 1;
	top = (i < DIM) ? i : i - DIM;
	bottom = (i >= DIM * (DIM - 1)) ? i : i + DIM;

	out[i] = in[i] + 0.1 * (in[top] + in[left] + in[right] + in[bottom] - in[i] * 4);
	}
}
```

Conversion from Float to RGB

Next step is conversion of floating-point temperatures values in RGB space. We will assume that temperatures values will be in range [0.0001,1] where 1 will be the fraction between maximum temperature and maximum temperature and 0.0001 the ratio between minimum temperature and maximum temperature. In color heat transfer I will consider that we work in HLS space and we will convert HLS values to RGB. In our color space we consider:

- lightness = temperature,
- saturation = 1,
- hue = (180 + (int)(360.0f * lightness)) % 360

```
unsigned char value(float n1, float n2, int hue)
{
	hue = (hue > 360) ? (hue - 360) : ((hue < 0) ? hue + 360 : hue);
	if (hue < 60)
		return (unsigned char)(255 * (n1 + (n2 - n1) * hue / 60));
	if (hue < 180)
		return (unsigned char)(255 * n2);
	if (hue < 240)
		return (unsigned char)(255 * (n1 + (n2 - n1) * (240 - hue) / 60));
	return (unsigned char)(255 * n1);
}
```

```
void float_to_color(Mat ptr_out, float* out){
	// Find pixel position
	for (int i = 0; i < DIM; i++)
		for(int j = 0; j < DIM; j++){
			float lightness = out[DIM * i + j];
			float saturation = 1;
			int   hue = (180 + (int)(360.0f * lightness)) % 360;
			float m1, m2;

			m2 = (lightness <= 0.5f) ? lightness * (1 + saturation) :
			lightness + saturation - lightness * saturation;

			m1 = 2 * lightness - m2;

			ptr_out.at<cv::Vec3b>(i,j)[2] = value(m1, m2, hue + 120);
			ptr_out.at<cv::Vec3b>(i,j)[1] = value(m1, m2, hue);
			ptr_out.at<cv::Vec3b>(i,j)[0] = value(m1, m2, hue - 120);
	}
}
```

Main Function

```
bool dest = true;
for (int i = 0; i < 100; i++) {
	for (int i = 0; i < CONDUCTIVITY; i++) {
		if (dest) {
			copy_values_from_previous_step(picture_in, picture_ct);
			update_values(picture_out, picture_in);
		}
		else {
			copy_values_from_previous_step(picture_out, picture_ct);
			update_values(picture_in, picture_out);
		}
		dest = !dest;
	}
	float_to_color(pix_out, picture_out);
	video.write(pix_out);
}
video.release();
```

Profiling for Sequential Program

Profiling for Sequential Program

I used Intel VTune for profiling.

Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: C:\Users\bianca.palade\source\repos\Heat_Transfer_Sequential\x64\Debug\Heat_Transfer_Sequential.exe
Environment Variables:
Operating System: Microsoft Windows 10
Computer Name: DESKTOP-TFFSL40
Result Size: 3 MB
Collection start time: 14:36:40 29/11/2021 UTC
Collection stop time: 14:36:47 29/11/2021 UTC
Collector Type: User-mode sampling and tracing
Finalization mode: Fast. If the number of collected samples exceeds the threshold, this mode limits the number of processed samples

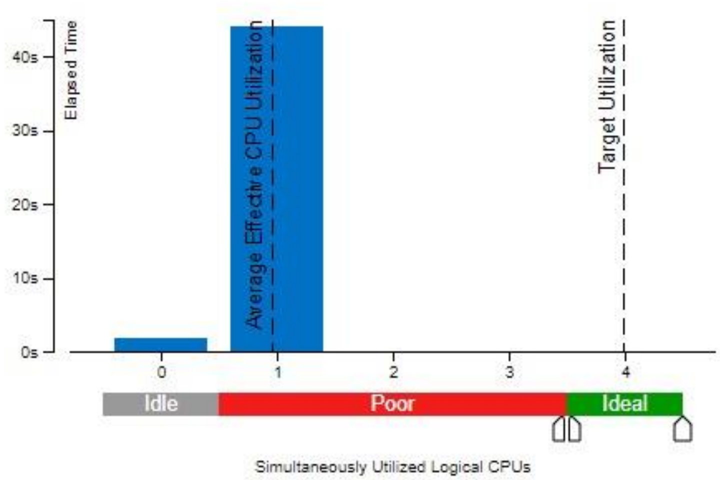
CPU
Name: Intel(R) Processor code named Kabylake ULX
Frequency: 2.7 GHz
Logical CPU Count: 4

Cache Allocation Technology
Level 2 capability: not detected
Level 3 capability: not detected

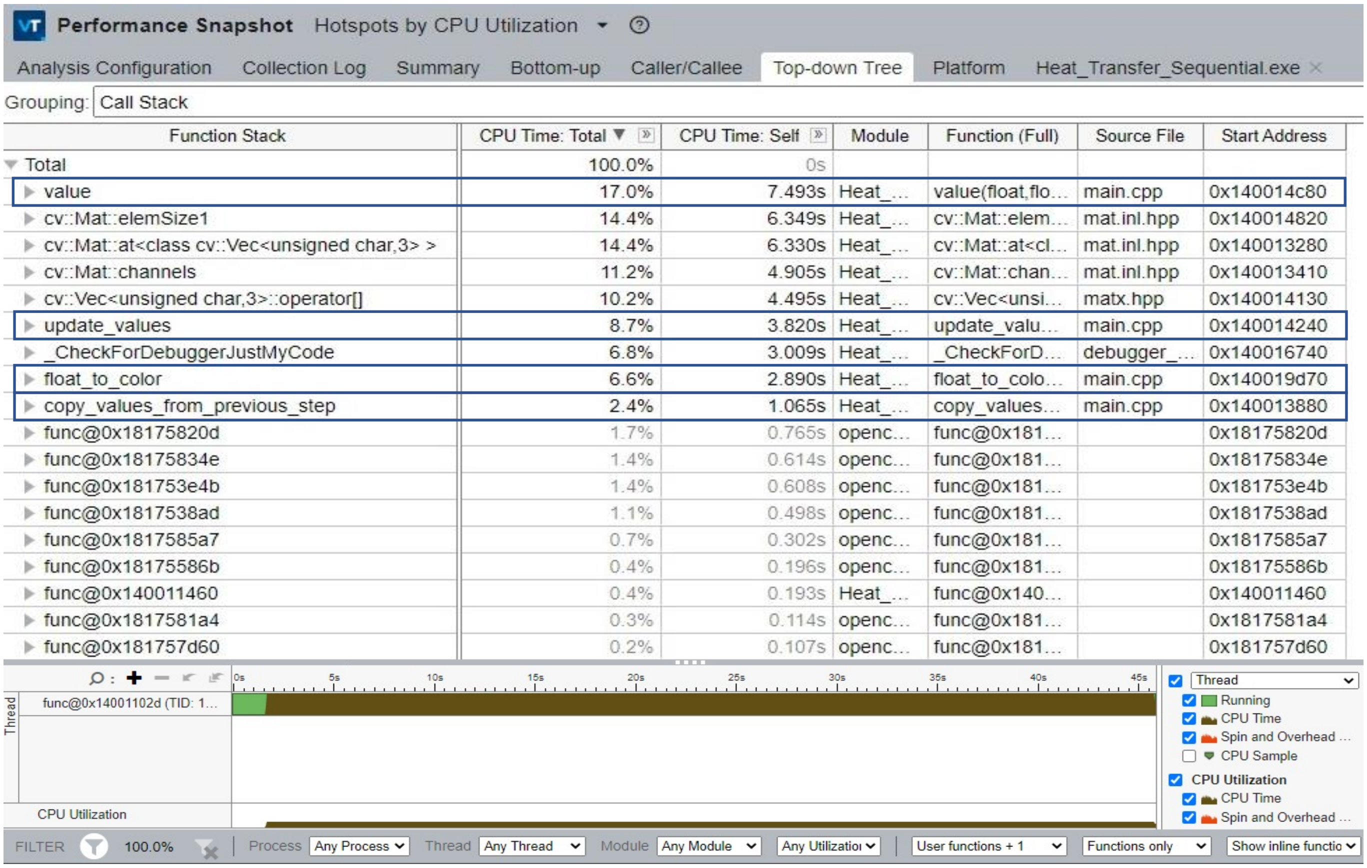
Hotspots by CPU Utilization

Elapsed Time: 45.827 s
Total Thread : 1
Paused Time : 0 s

Effective CPU Utilization Histogram

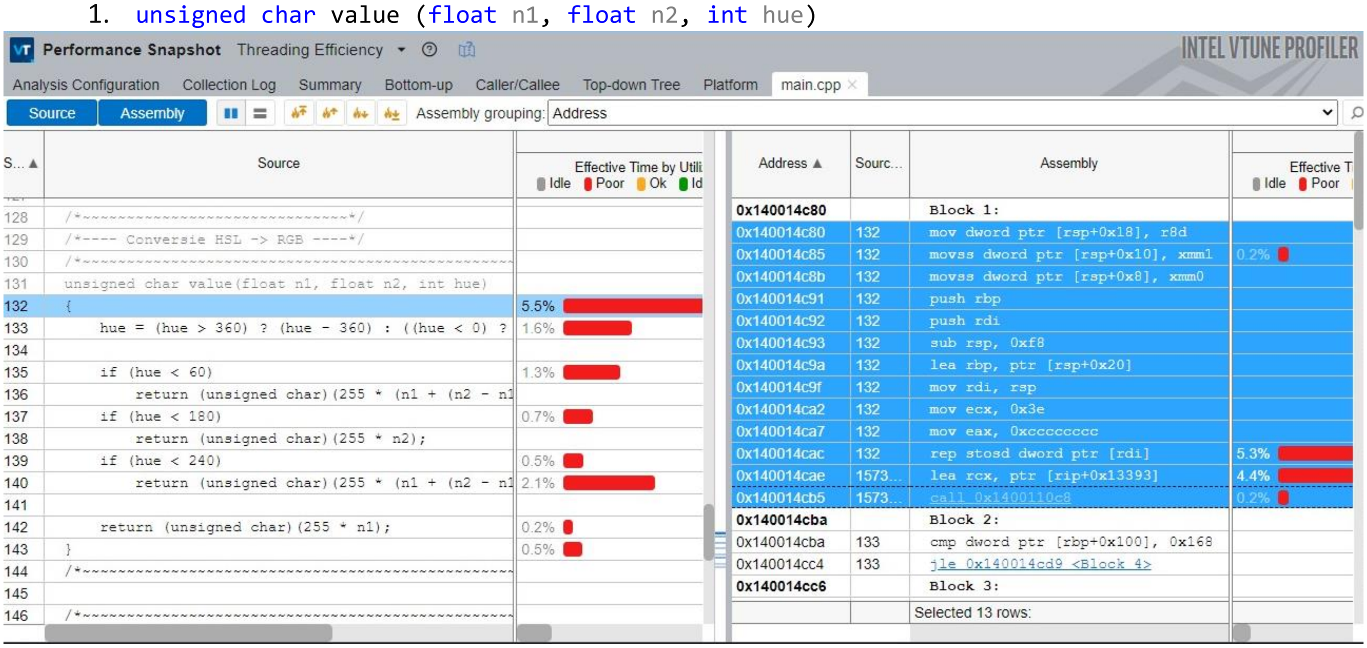


Top-down Tree

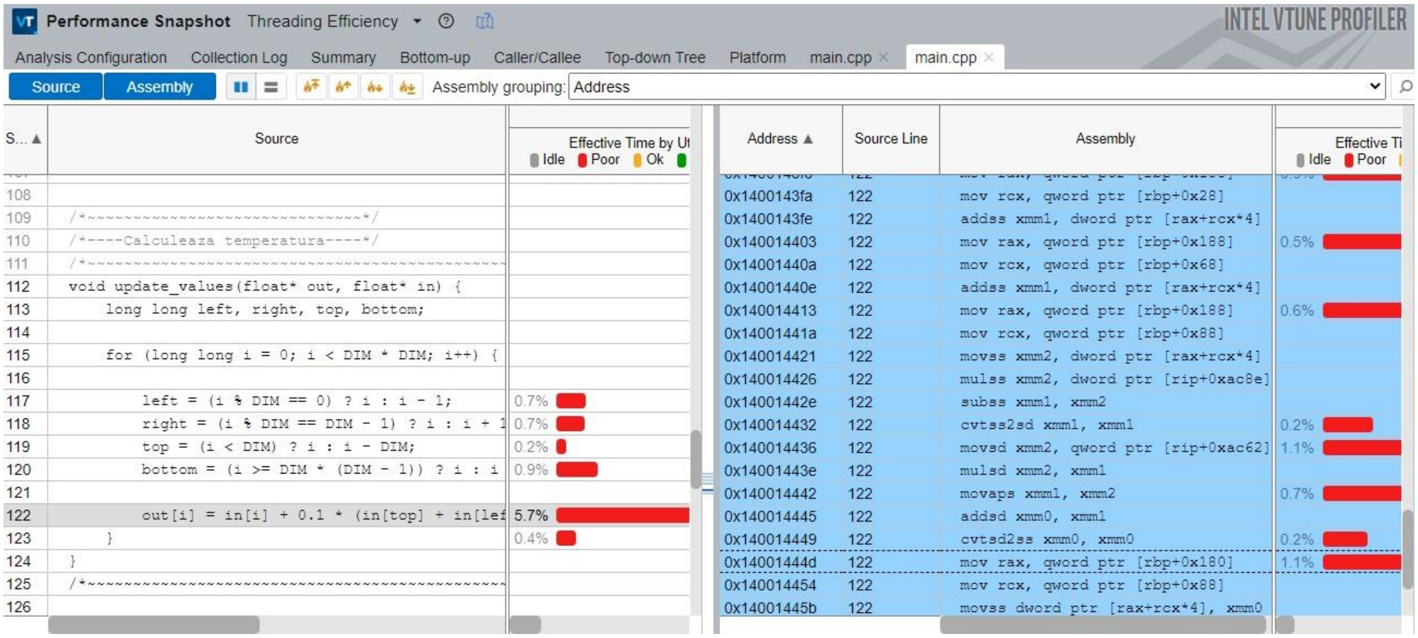


Hotspots Functions (source + assembly)

1. unsigned char value (float n1, float n2, int hue)



2. `void update_values(float* out, float* in)`



Parallel Algorithm

To parallelize the algorithm, I used OpenMP. I tried to improve the execution time by parallelizing the hotspots functions

1. `void float_to_color(Mat ptr_out, float* out)`

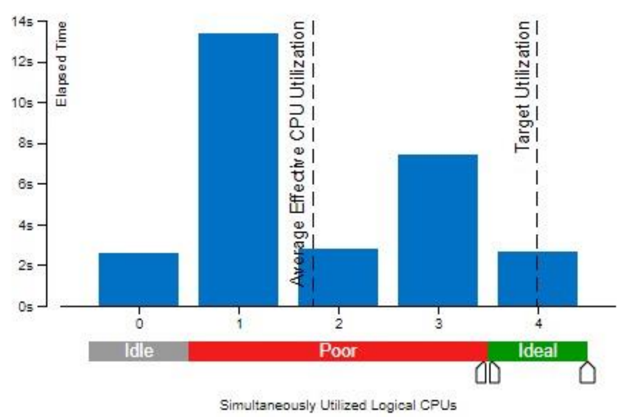
The value function value() cannot be parallelized efficiently, because it only checks conditions and

```
void float_to_color(Mat ptr_out, float* out)
{
	int i, j;
	#pragma omp parallel for num_threads(MAX_THREADS) private(i,j)
	for (i = 0; i < DIM; i++)
		for (j = 0; j < DIM; j++) {
			float lightness = out[DIM * i + j];
			float saturation = 1;
			int   hue = (180 + (int)(360.0f * lightness)) % 360;
			float m1, m2;

			m2 = (lightness <= 0.5f) ? lightness * (1 + saturation) : lightness + 
saturation - lightness * saturation;
			m1 = 2 * lightness - m2;

			ptr_out.at<cv::Vec3b>(i, j)[2] = value(m1, m2, hue + 120);
			ptr_out.at<cv::Vec3b>(i, j)[1] = value(m1, m2, hue);
			ptr_out.at<cv::Vec3b>(i, j)[0] = value(m1, m2, hue - 120);}}
```

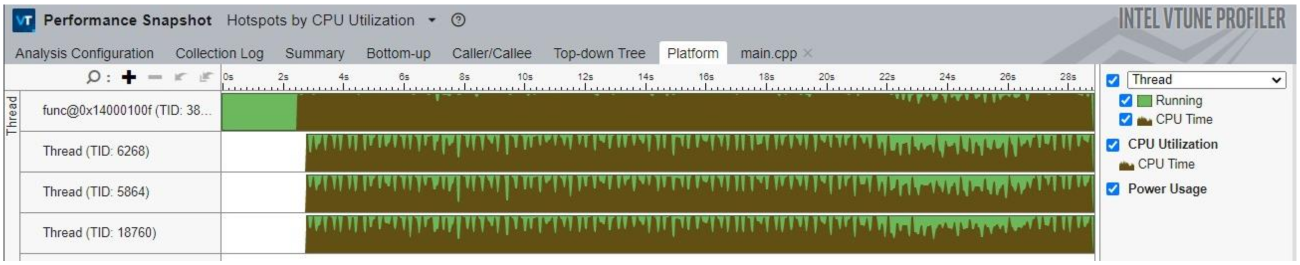
does not contain any repetitive instructions, but the `float_to_color()` function, can be parallelized because it calls the value () function for DIM * DIM * 3 times.

Profiling

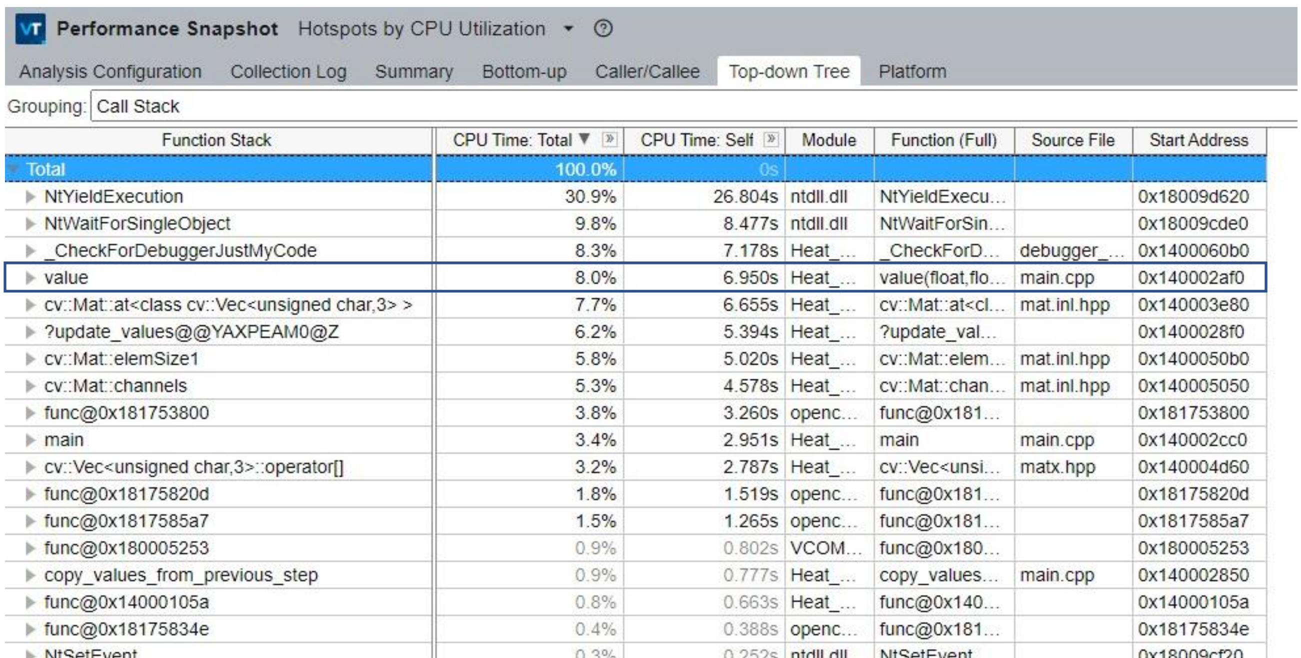
Hotspots by CPU Utilization
Effective CPU Utilization Histogram



Elapsed Time: 28.852 s
Total Thread : 4
Paused Time : 0 s



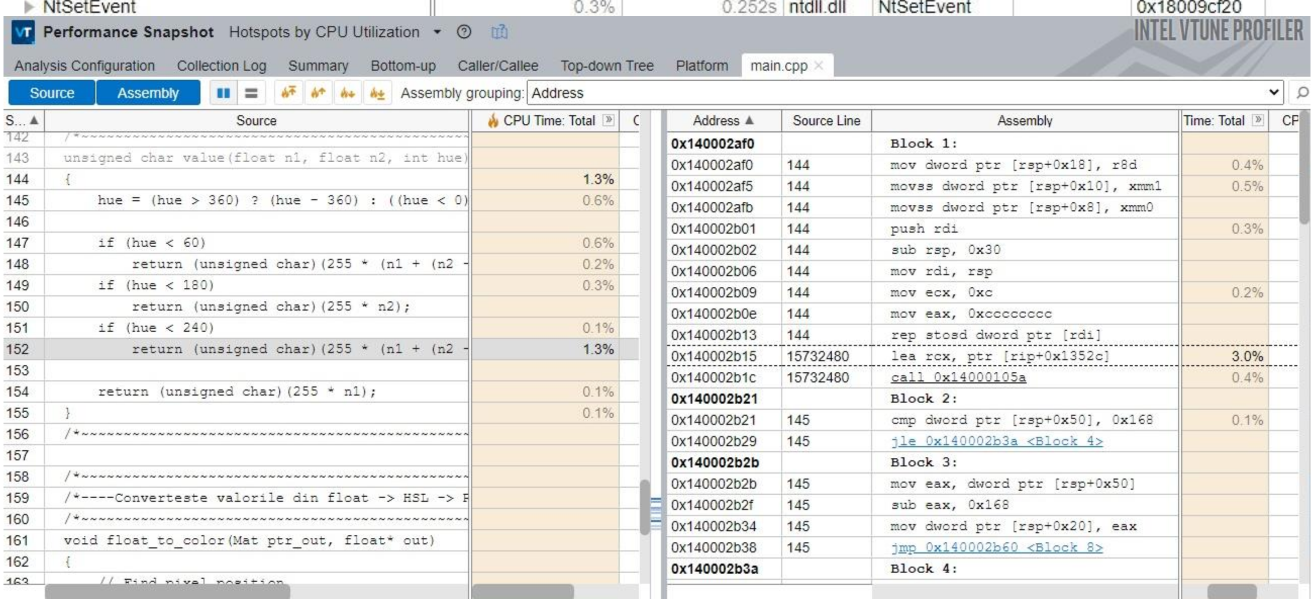
Hotspots Functions (source + assembly)


Performance Snapshot Hotspots by CPU Utilization

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

Grouping: Call Stack

Function Stack	CPU Time: Total	CPU Time: Self	Module	Function (Full)	Source File	Start Address
Total	100.0%	0s				
NtYieldExecution	30.9%	26.804s	ntdll.dll	NtYieldExecu...		0x18009d620
NtWaitForSingleObject	9.8%	8.477s	ntdll.dll	NtWaitForSin...		0x18009cde0
_CheckForDebuggerJustMyCode	8.3%	7.178s	Heat_...	_CheckForD...	debugger_...	0x1400060b0
value	8.0%	6.950s	Heat_...	value(float,flo...	main.cpp	0x140002af0
cv::Mat::at<class cv::Vec<unsigned char,3> >	7.7%	6.655s	Heat_...	cv::Mat::at<cl...	mat.inl.hpp	0x140003e80
?update_values@@YAXPEAM0@Z	6.2%	5.394s	Heat_...	?update_val...		0x1400028f0
cv::Mat::elemSize1	5.8%	5.020s	Heat_...	cv::Mat::elem...	mat.inl.hpp	0x1400050b0
cv::Mat::channels	5.3%	4.578s	Heat_...	cv::Mat::chan...	mat.inl.hpp	0x140005050
func@0x181753800	3.8%	3.260s	openc...	func@0x181...		0x181753800
main	3.4%	2.951s	Heat_...	main	main.cpp	0x140002cc0
cv::Vec<unsigned char,3>::operator[]	3.2%	2.787s	Heat_...	cv::Vec<unsi...	matx.hpp	0x140004d60
func@0x18175820d	1.8%	1.519s	openc...	func@0x181...		0x18175820d
func@0x1817585a7	1.5%	1.265s	openc...	func@0x181...		0x1817585a7
func@0x180005253	0.9%	0.802s	VCOM...	func@0x180...		0x180005253
copy_values_from_previous_step	0.9%	0.777s	Heat_...	copy_values...	main.cpp	0x140002850
func@0x14000105a	0.8%	0.663s	Heat_...	func@0x140...		0x14000105a
func@0x18175834e	0.4%	0.388s	openc...	func@0x181...		0x18175834e
NtSetEvent	0.3%	0.252s	ntdll.dll	NtSetEvent		0x18009cf20



2. `void update_values(float* out, float* in)`

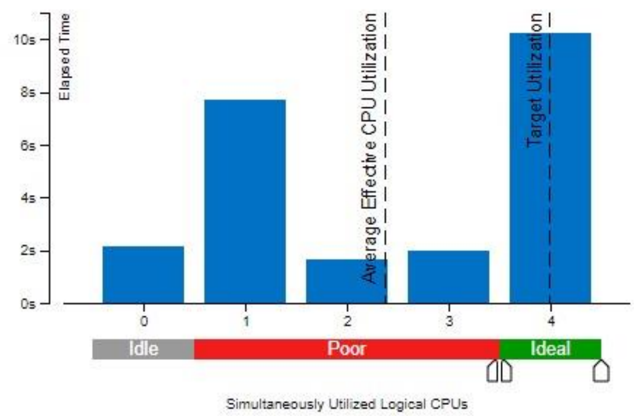
The second most time consuming is the *update_values ()* function. To parallelize it, I divided the vector *out* into the maximum number of threads.

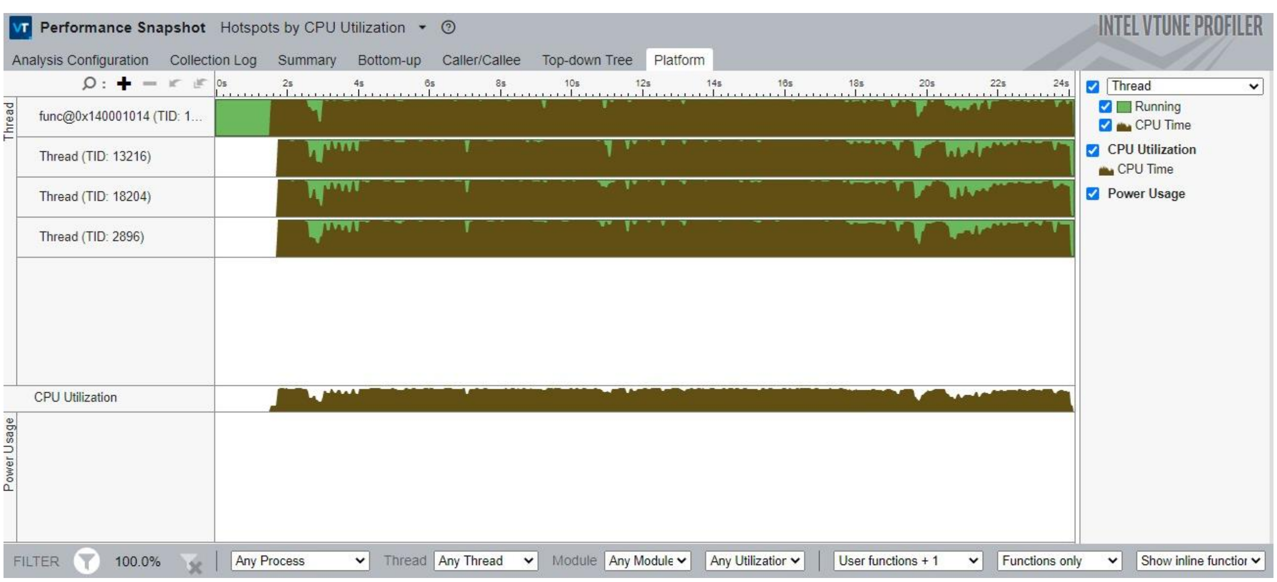
```
void update_values(float* out, float* in) {
	#pragma omp parallel
	for (long long i = omp_get_thread_num() * DIM * DIM / MAX_THREADS; i <
omp_get_thread_num() * DIM * DIM / MAX_THREADS + DIM * DIM / MAX_THREADS; ++i) {
		long long left = (i % DIM == 0) ? i : i - 1;
		long long right = (i % DIM == DIM - 1) ? i : i + 1;
		long long top = (i < DIM) ? i : i - DIM;
		long long bottom = (i >= DIM * (DIM - 1)) ? i : i + DIM;
		out[i] = in[i] + 0.1f * (in[top] + in[left] + in[right] + in[bottom] - in[i]
	* 4);
}
}
```

Hotspots by CPU Utilization

Effective CPU Utilization Histogram

Elapsed Time: 23.587 s
Total Thread : 4
Paused Time : 0 s





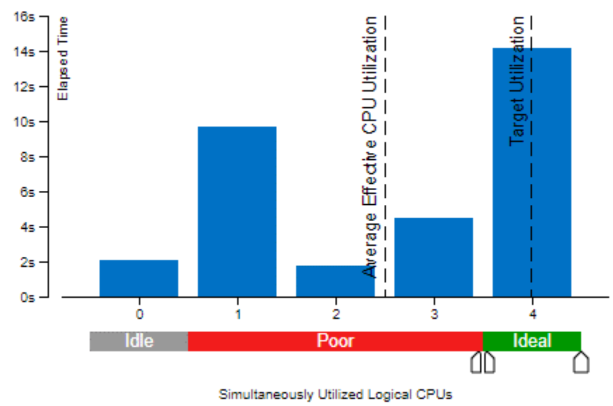
Initially, I tried to use *#pragma omp parallel for ordered,* but the program run much slower than the sequential version of the function, because when OpenMP distributes the work among threads there is a lot of administration/synchronisation going on to ensure the values in my vector are not corrupted somehow.

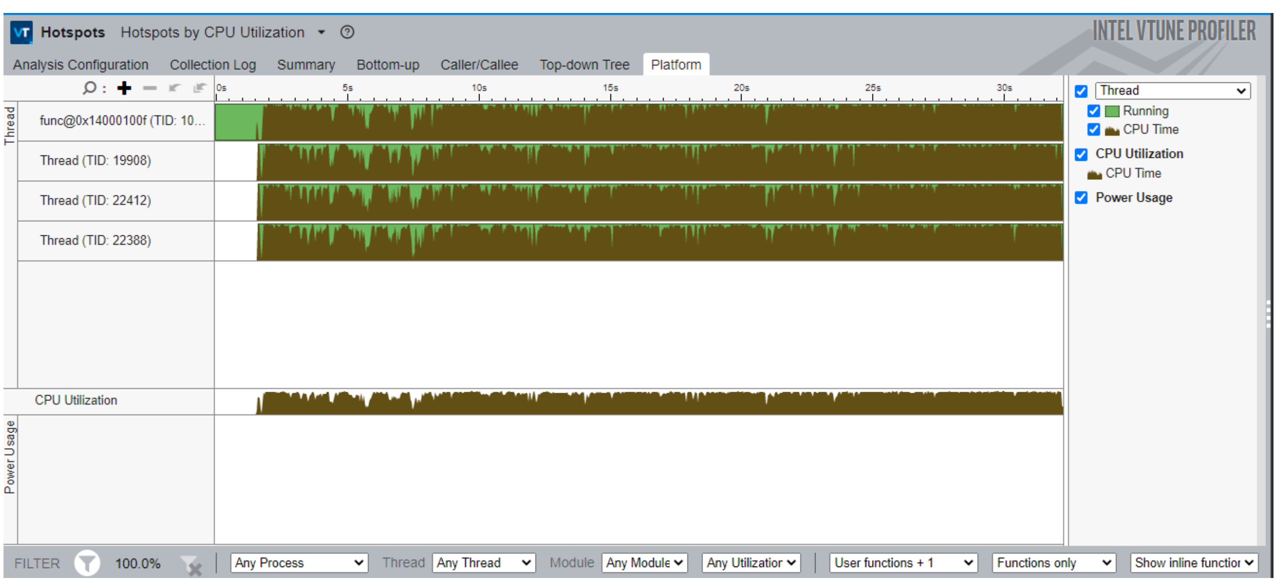
```
#pragma omp parallel for ordered
       for (long long i = 0; i < DIM * DIM; i++) {
              left = (i % DIM == 0) ? i : i - 1;
              right = (i % DIM == DIM - 1) ? i : i + 1;
              top = (i < DIM) ? i : i - DIM;
              bottom = (i >= DIM * (DIM - 1)) ? i : i + DIM;

              out[i] = in[i] + 0.1 * (in[top] + in[left] + in[right] + in[bottom] - in[i]
* 4);}
```

Hotspots by CPU Utilization

Elapsed Time: 32.215 s
Total Thread : 4
Paused Time : 0 s

Effective CPU Utilization Histogram



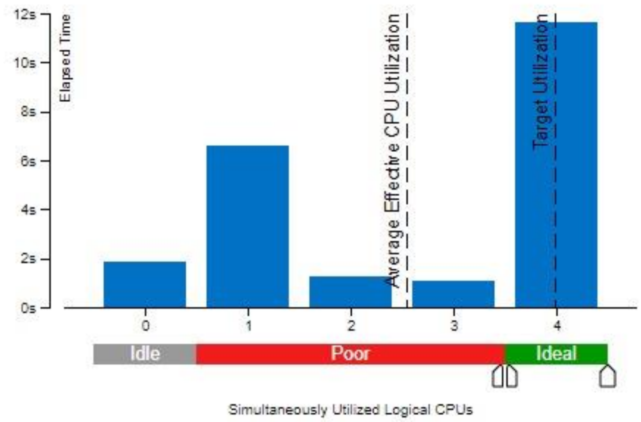


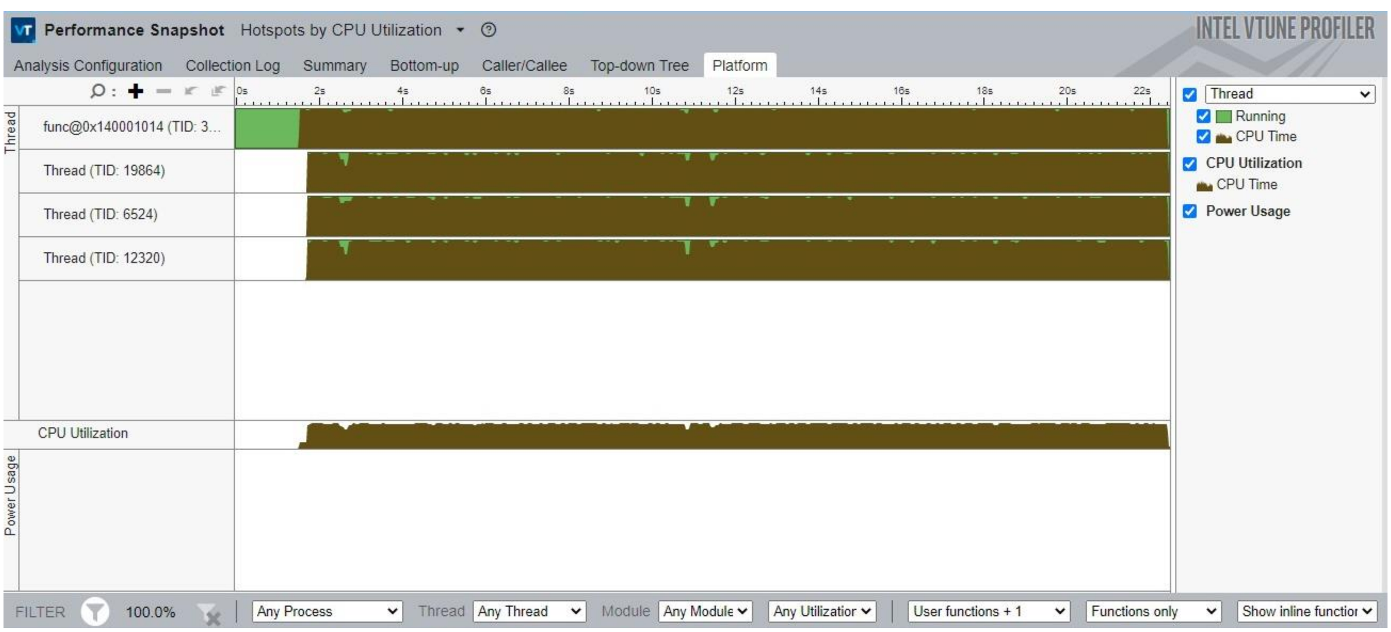
3. `void copy_values_from_previous_step(float* in, float* ct)`

```
void copy_values_from_previous_step(float* in, float* ct) {
        #pragma omp parallel for num_threads(MAX_THREADS)
        for (int i = 0; i < DIM * DIM; i++) {
                if (ct[i] != T_MIN)
                        in[i] = ct[i];
        }
}
```

Hotspots by CPU Utilization
Effective CPU Utilization Histogram

Elapsed Time: 22.452 s
Total Thread : 4
Paused Time : 0 s



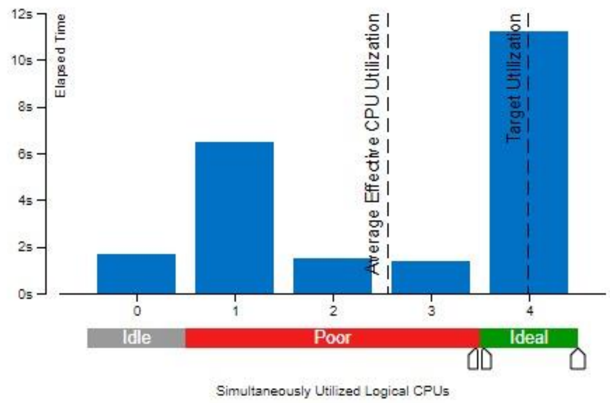


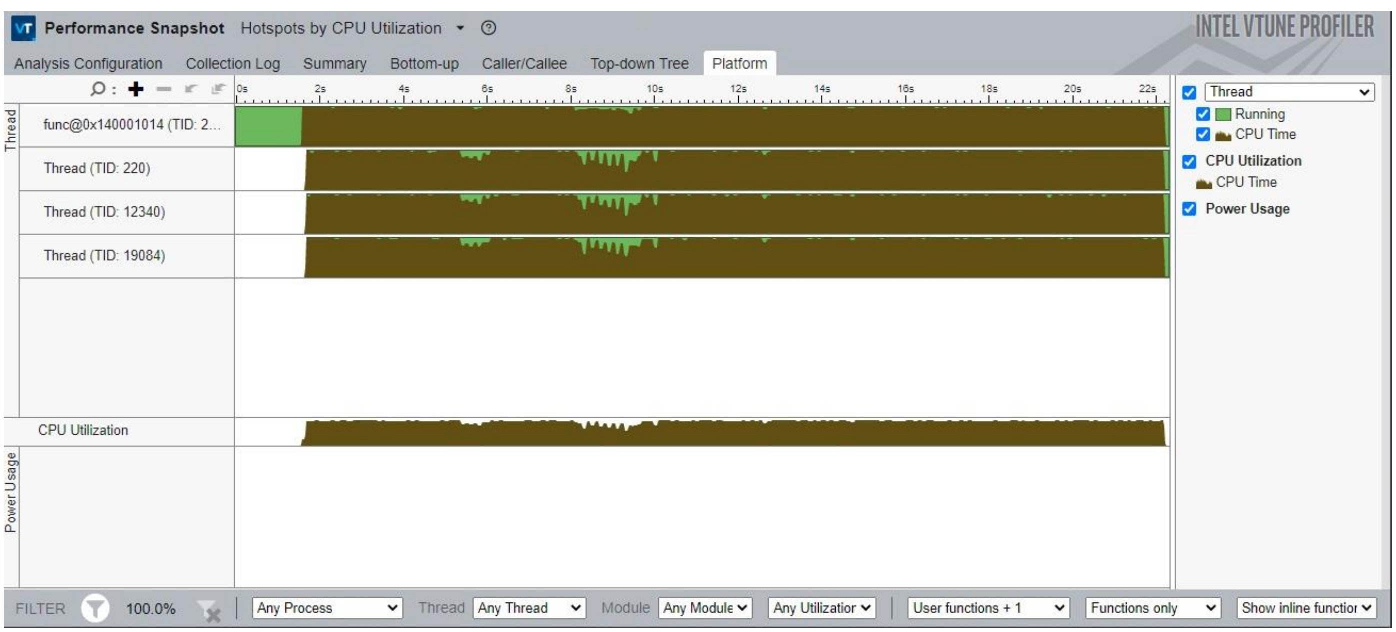
```
#pragma omp parallel for  num_threads(MAX_THREADS) private(i,j)
for (i = 0; i < DIM; i++) {
        for (j = 0; j < DIM; j++) {
                picture_in[DIM * i + j] = (pix.at<cv::Vec3b>(i, j)[0] == 0) ? T_MAX : T_MIN;
                picture_ct[DIM * i + j] = picture_in[DIM * i + j];
        }
}
```

4. from int main I parallelized image read

Hotspots by CPU Utilization
Effective CPU Utilization Histogram

Elapsed Time: 22.303 s
Total Thread : 4
Paused Time : 0 s





Hotspots Functions

Performance Snapshot Hotspots by CPU Utilization

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

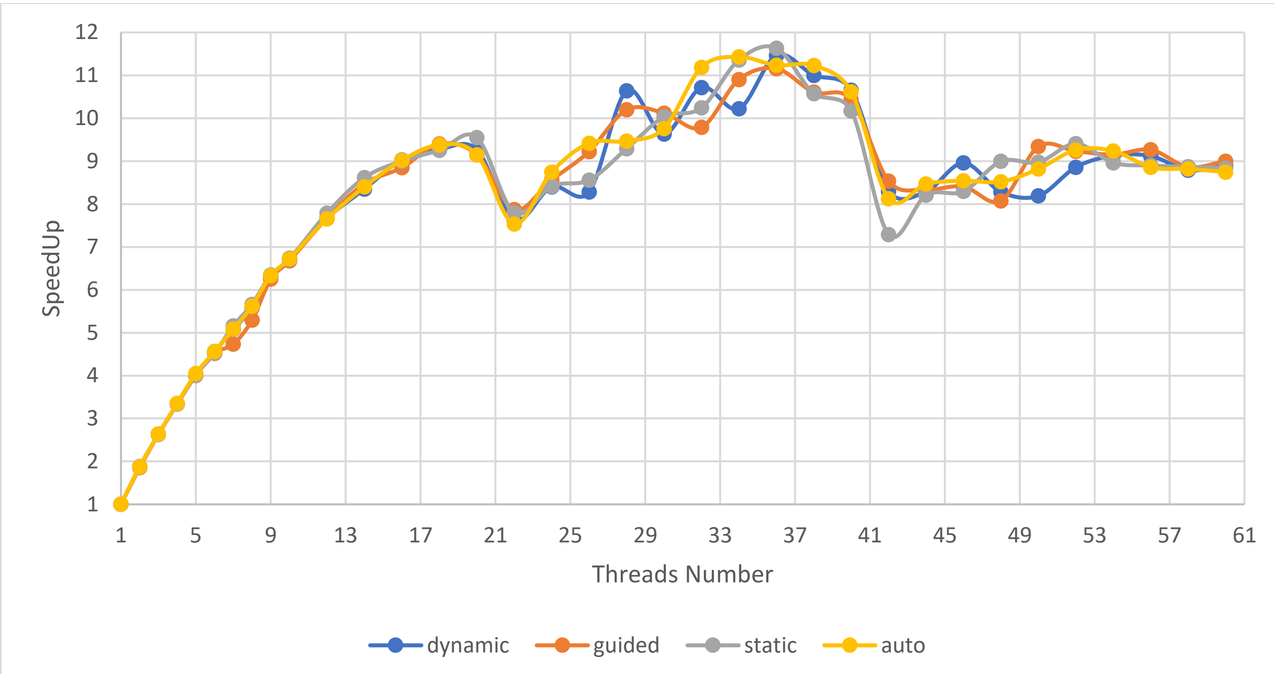
Grouping: Call Stack

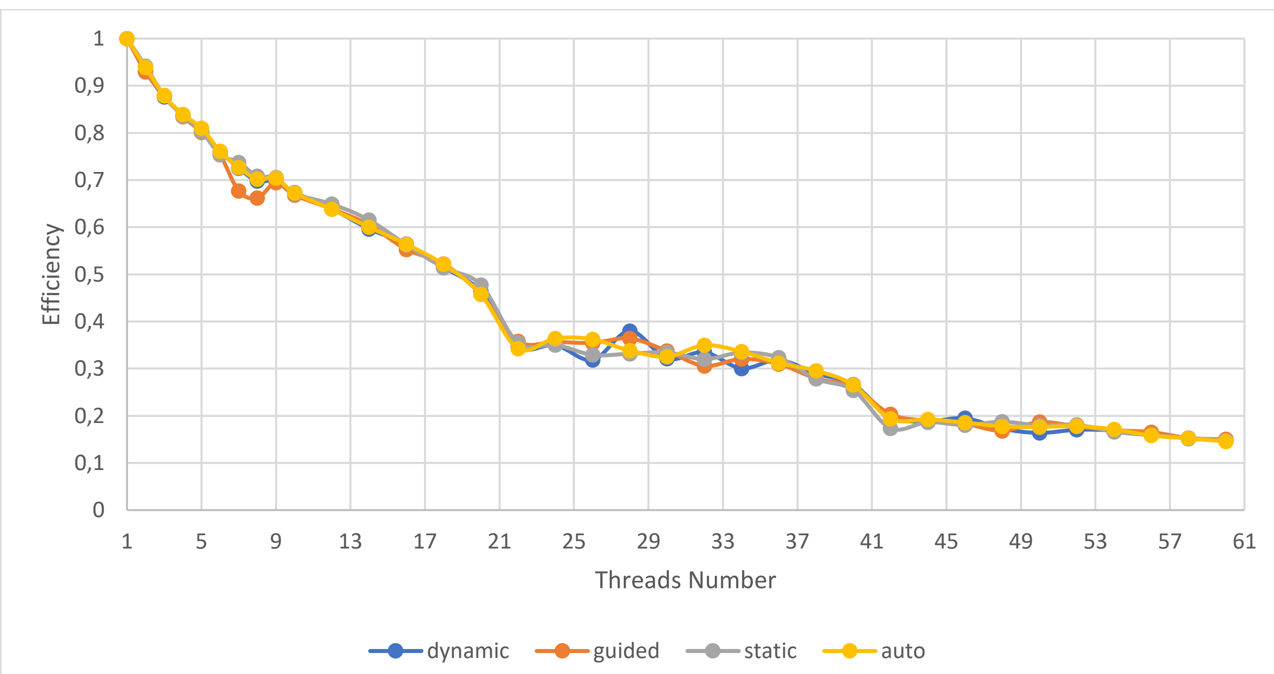
Function Stack	CPU Time: Total ▼	CPU Time: Self	Module	Function (Full)	Source File	Start Address
Total	100.0%	0s				
NtYieldExecution	25.2%	19.783s	ntdll.dll	NtYieldExecu...		0x18009d620
main	13.7%	10.772s	Heat_...	main	main.cpp	0x140009570
_CheckForDebuggerJustMyCode	10.1%	7.929s	Heat_...	_CheckForD...	debugger_...	0x1400060b0
cv::Mat::at<class cv::Vec<unsigned char,3> >	9.9%	7.800s	Heat_...	cv::Mat::at<cl...	mat.inl.hpp	0x140003e80
value	8.6%	6.785s	Heat_...	value(float,flo...	main.cpp	0x1400093a0
cv::Mat::channels	6.3%	4.929s	Heat_...	cv::Mat::chan...	mat.inl.hpp	0x140003e30
cv::Vec<unsigned char,3>::operator[]	4.9%	3.855s	Heat_...	cv::Vec<unsi...	matx.hpp	0x140004aa0
cv::Mat::elemSize1	4.6%	3.637s	Heat_...	cv::Mat::elem...	mat.inl.hpp	0x140004090
func@0x181753e4b	2.4%	1.901s	openc...	func@0x181...		0x181753e4b
func@0x1817538f5	2.1%	1.649s	openc...	func@0x181...		0x1817538f5
func@0x18175820d	1.9%	1.470s	openc...	func@0x181...		0x18175820d
func@0x180027390	1.4%	1.098s	KERN...	func@0x180...		0x180027390
func@0x14000105a	1.3%	1.060s	Heat_...	func@0x140...		0x14000105a
omp_get_thread_num	1.3%	1.007s	VCOM...	omp_get_thr...		0x1800059f0
func@0x1817585a7	1.1%	0.848s	openc...	func@0x181...		0x1817585a7
func@0x18175834e	1.0%	0.779s	openc...	func@0x181...		0x18175834e
func@0x180005254	0.9%	0.676s	VCOM...	func@0x180...		0x180005254
?elemSize1@Mat@cv@@QEBA_KXZ	0.6%	0.496s	Heat_...	?elemSize1...		0x140004090
TlsGetValue	0.6%	0.491s	KERN...	TlsGetValue		0x180015540
RtlGetCurrentUmsThread	0.4%	0.304s	ntdll.dll	RtlGetCurren...		0x180058990
SwitchToThread	0.4%	0.294s	KERN...	SwitchToThr...		0x1800676a0
func@0x1400011fe	0.3%	0.200s	Heat_...	func@0x140...		0x1400011fe
GetTickCount	0.2%	0.188s	KERN...	GetTickCount		0x180015640
func@0x181fa7f30	0.1%	0.110s	openc...	func@0x181f...		0x181fa7f30
func@0x181fa7cd0	0.1%	0.108s	openc...	func@0x181f...		0x181fa7cd0
func@0x181757250	0.1%	0.106s	openc...	func@0x181757250		0x181757250
SwitchToThread	0.1%	0.100s	KERN...	SwitchToThr...		0x18001b400
func@0x181758552	0.1%	0.093s	openc...	func@0x181...		0x181758552
func@0x1817558a0	0.1%	0.091s	openc...	func@0x181...		0x1817558a0

Speedup, Efficiency, CPU-time

Running without O3 on HPSL cluster

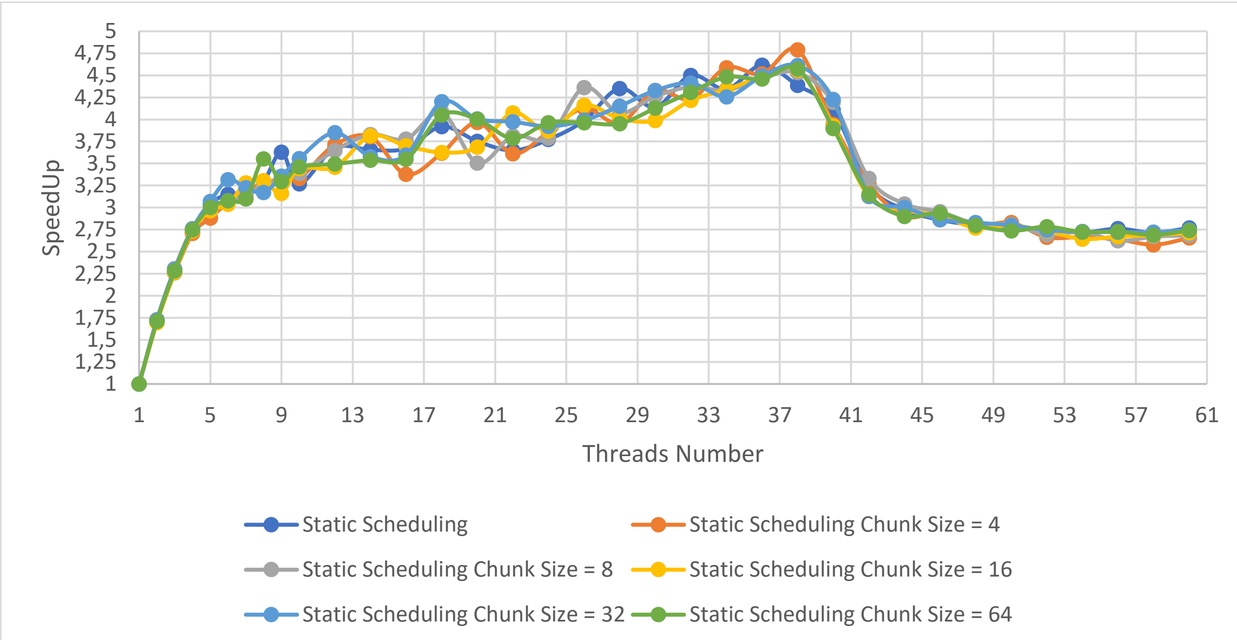
I ran the program 3 times for each condition, and I did the arithmetic mean of the values

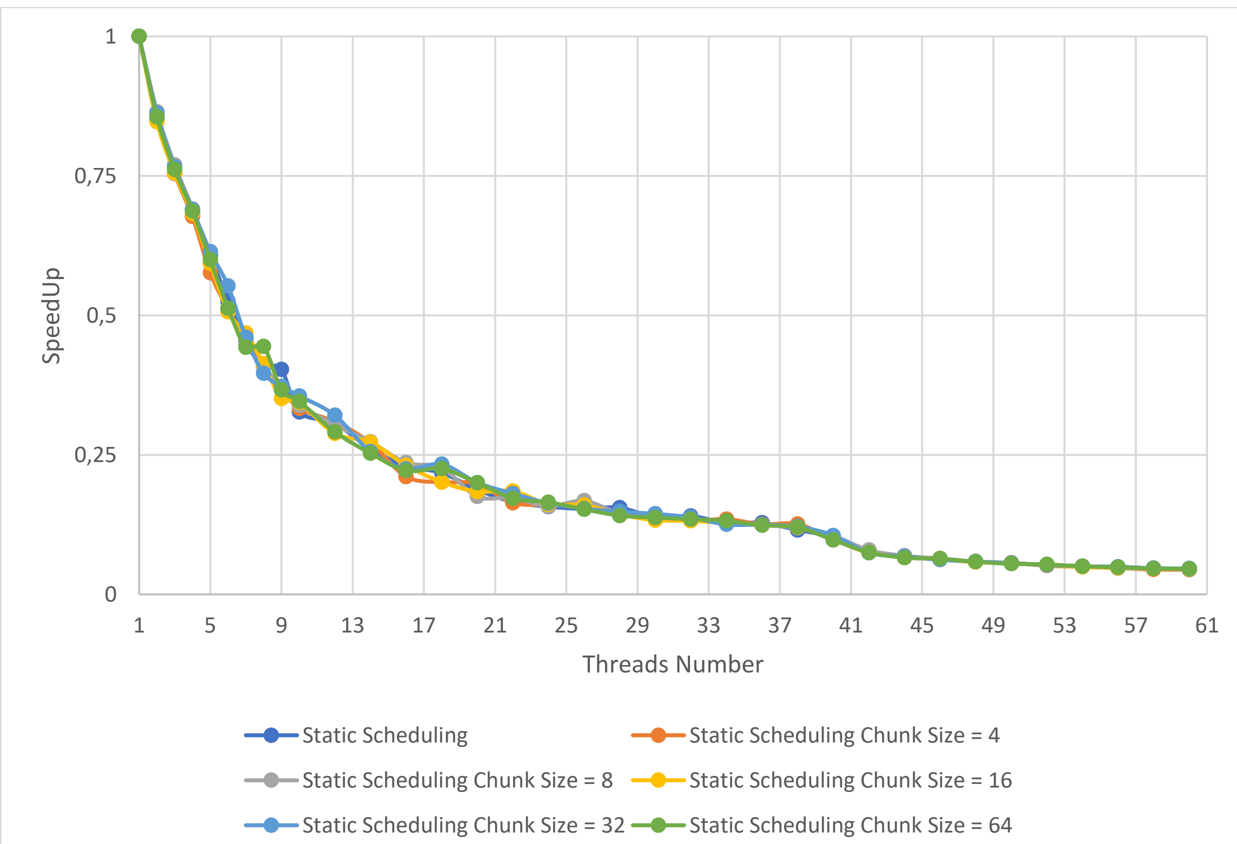




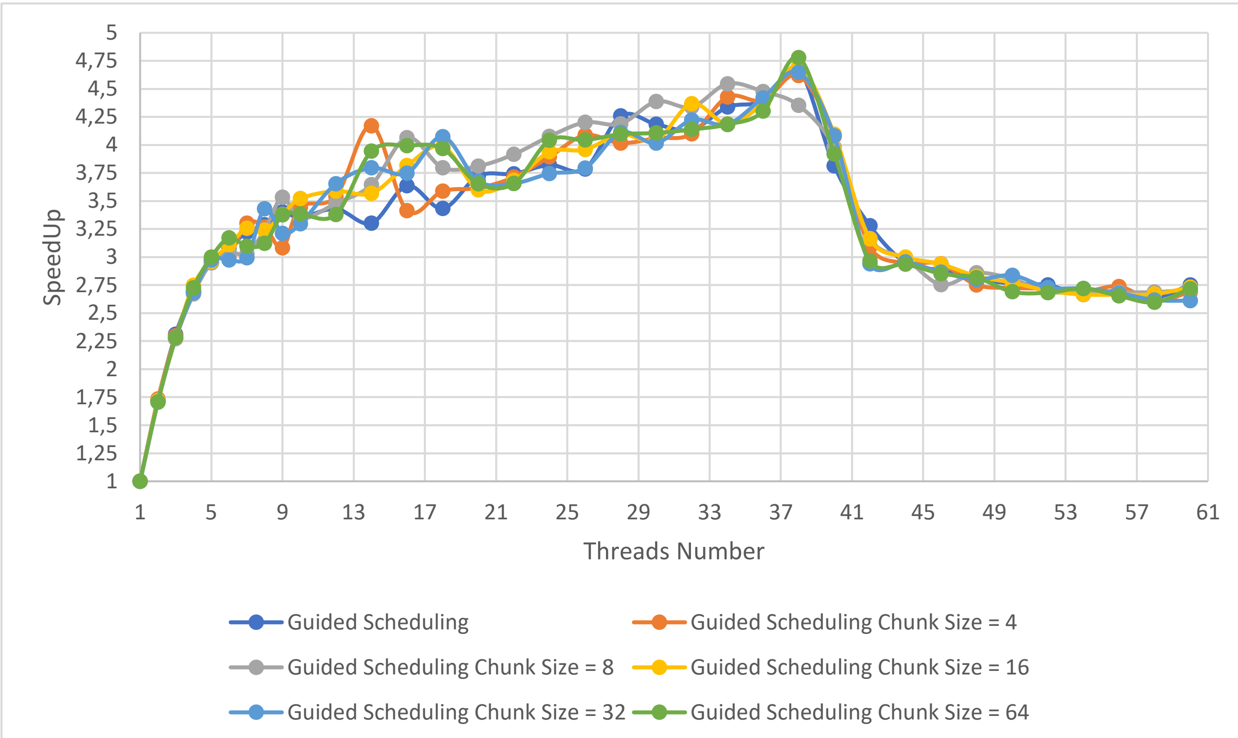
Running with -O3 on HPSL cluster

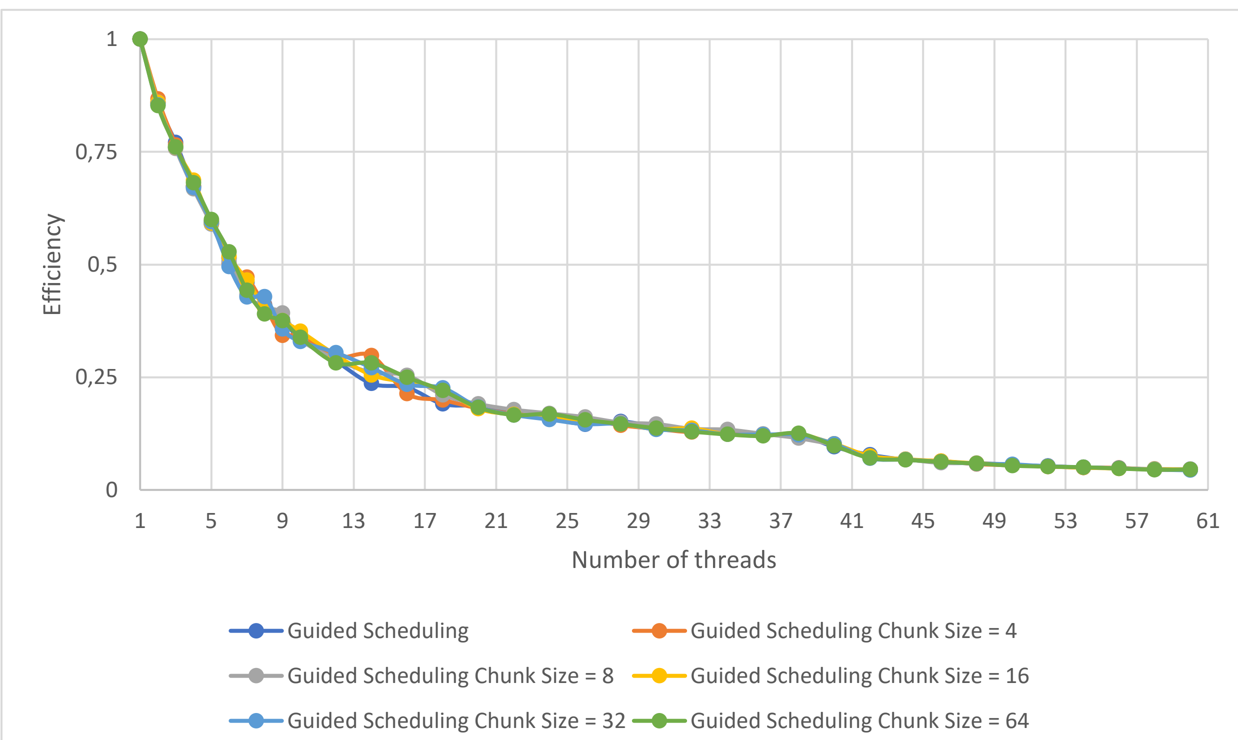
1. Static, Chunk Size = 4,8,16,32,64 Speedup + Efficiency



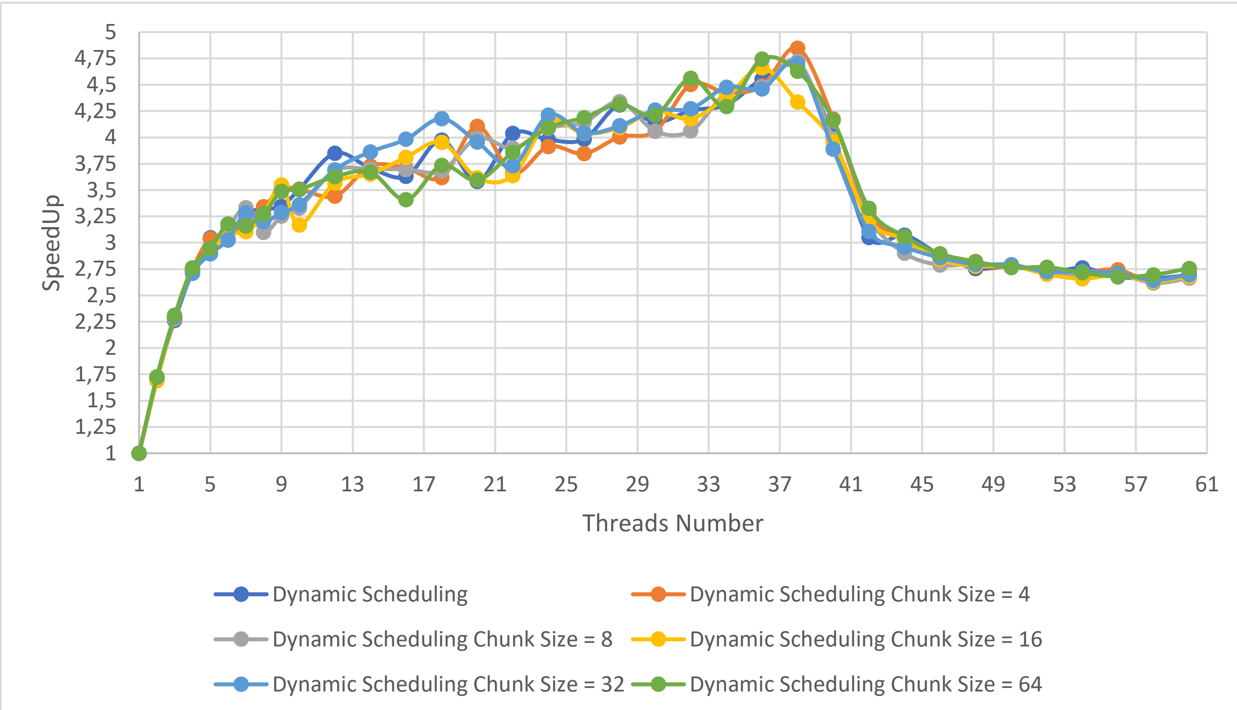


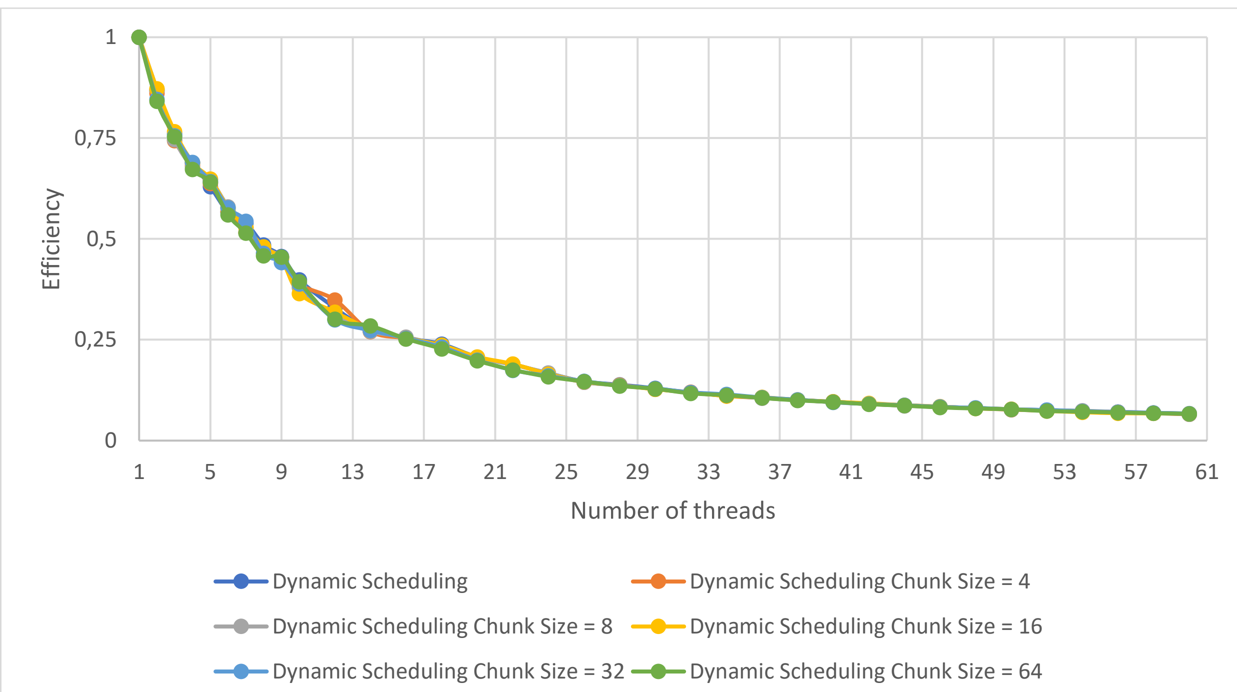
2. Guided, Chunk Size = 4,8,16,32,64 Speedup + Efficiency





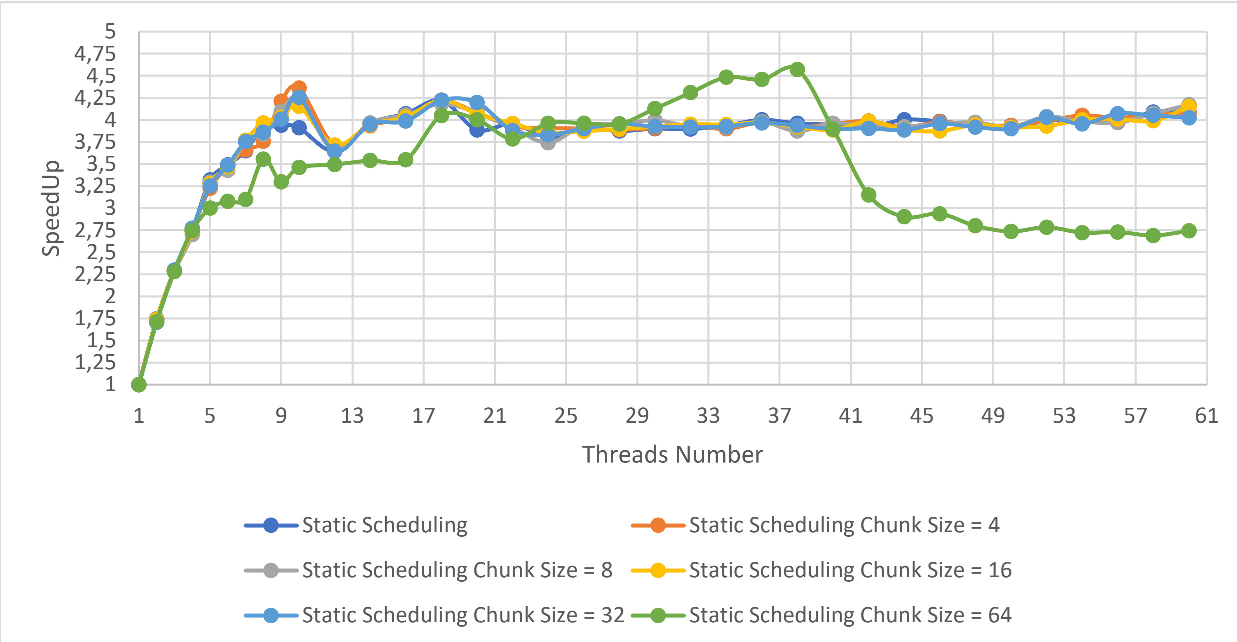
3. Dynamic, Chunk Size = 4,8,16,32,64 Speedup + Efficiency

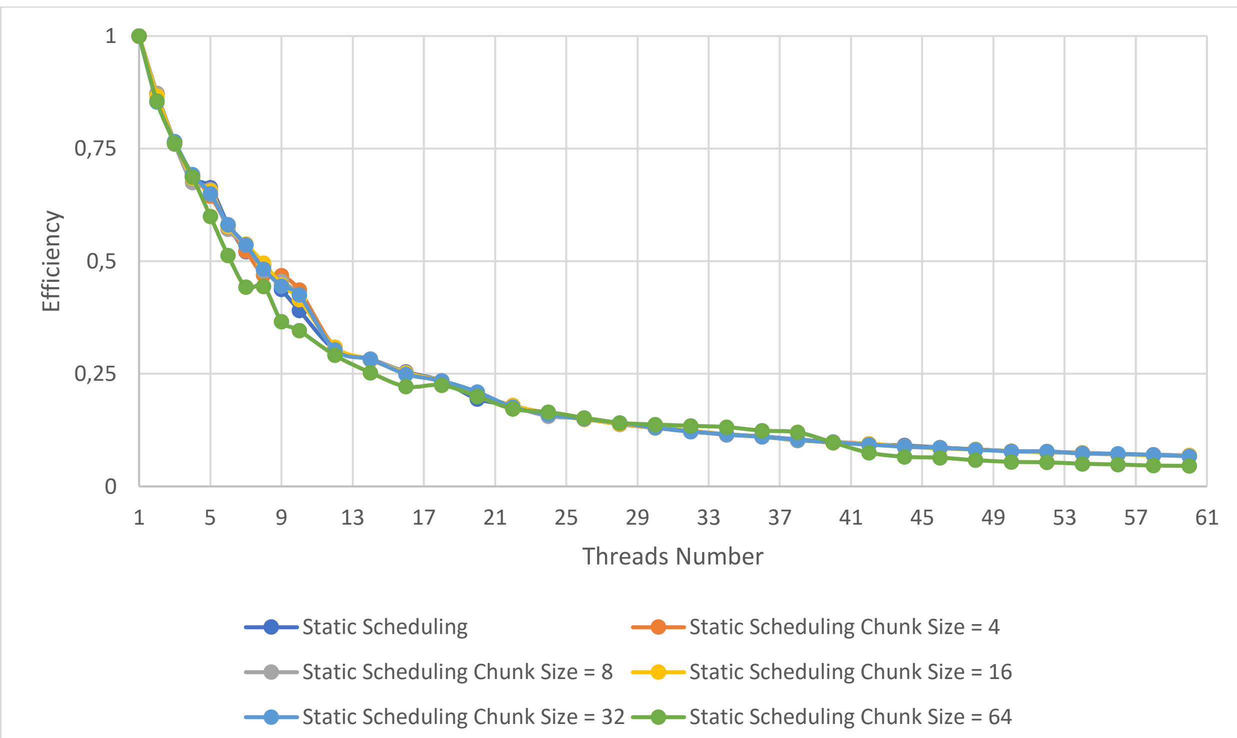




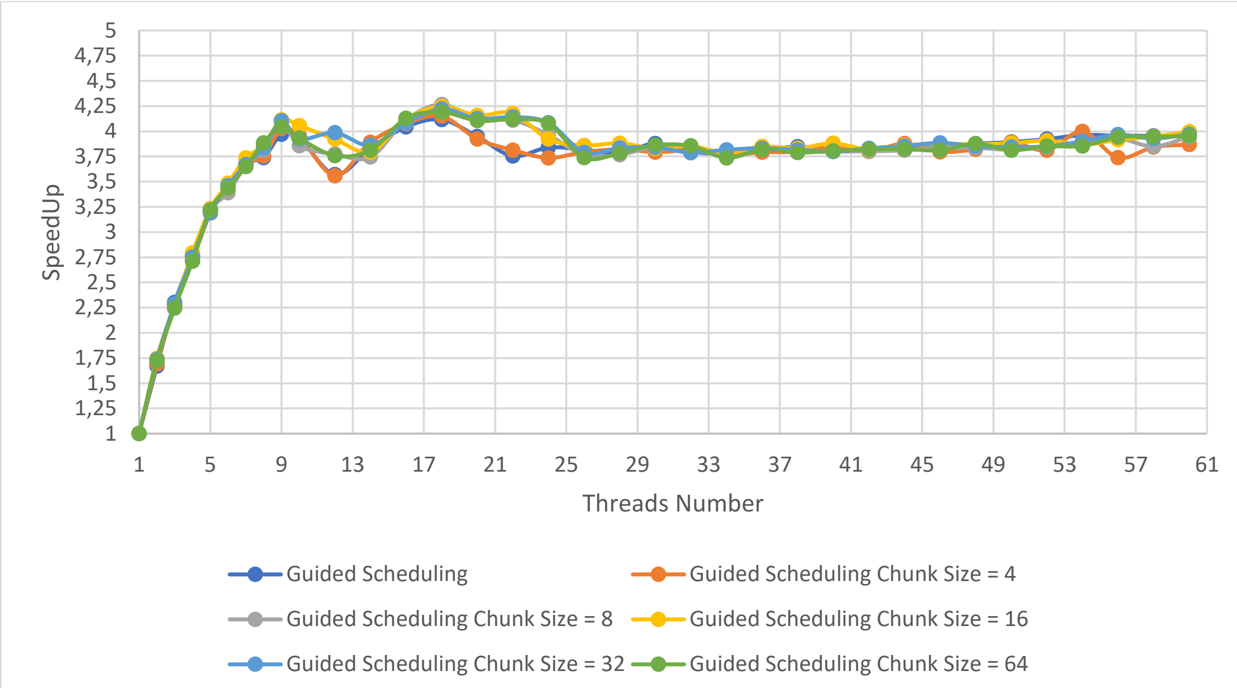
Running with -O3 on IBM cluster

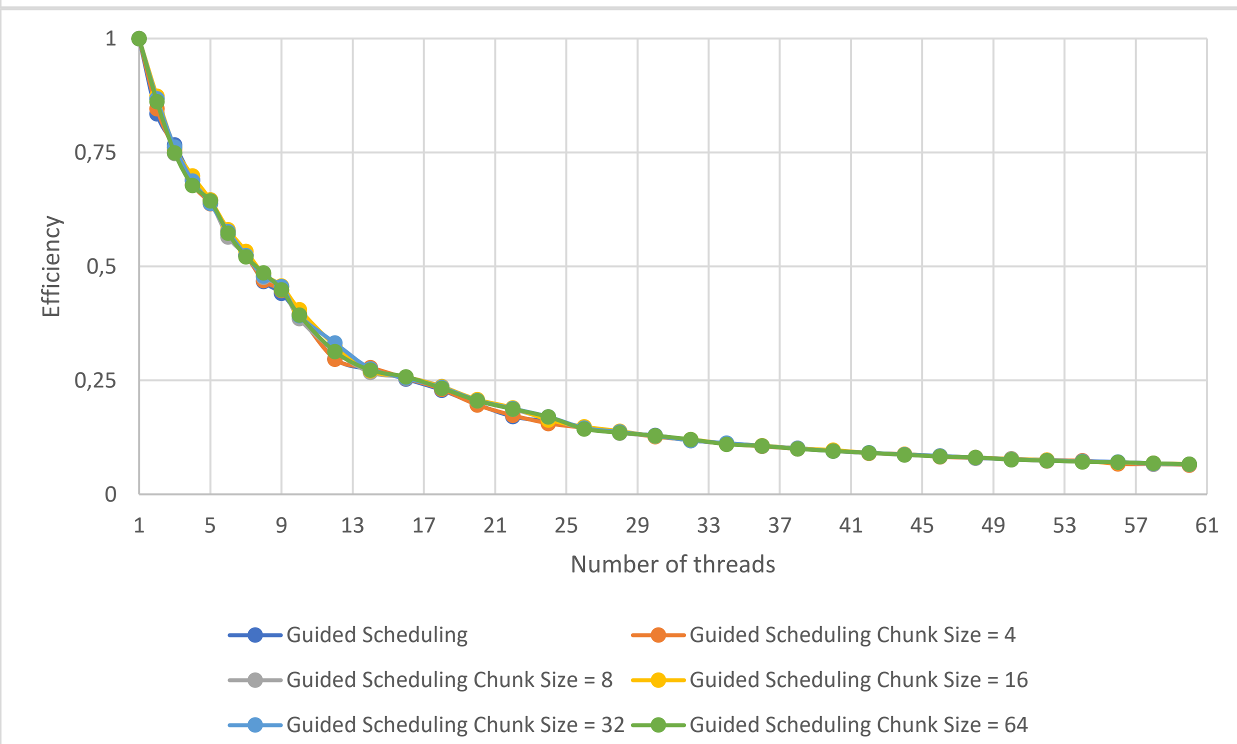
1. Static, Chunk Size = 4,8,16,32,64 Speedup + Efficiency



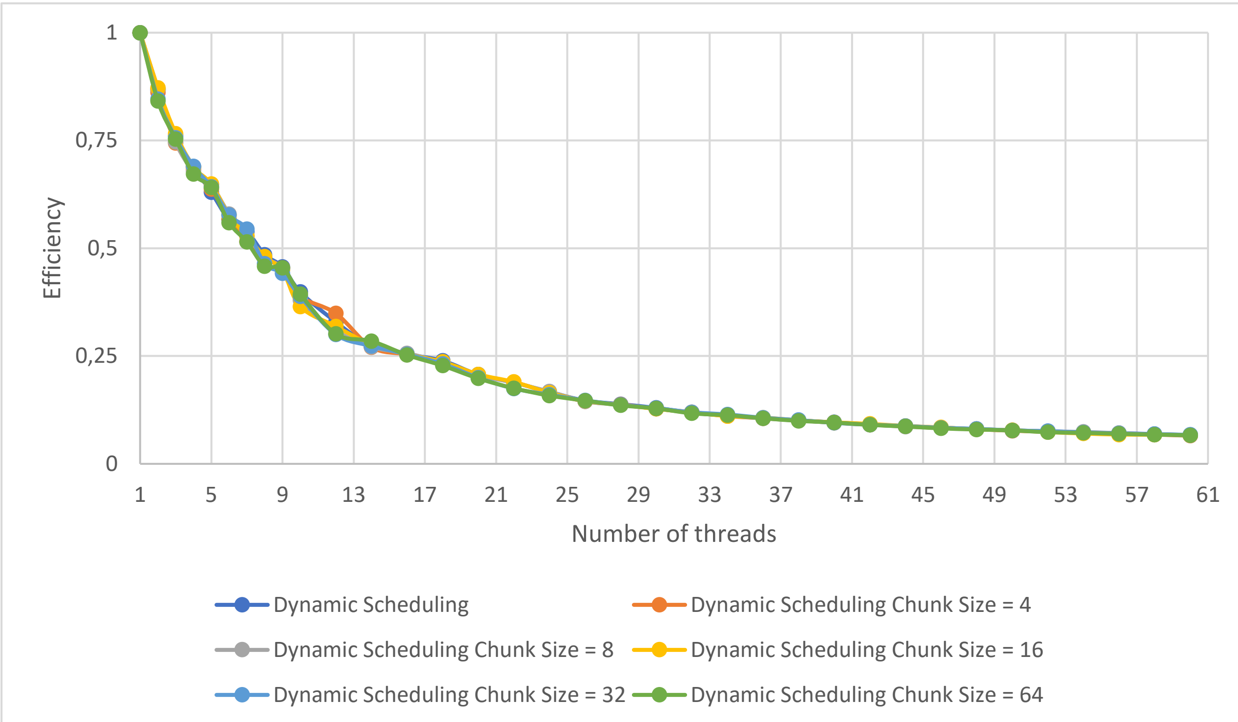


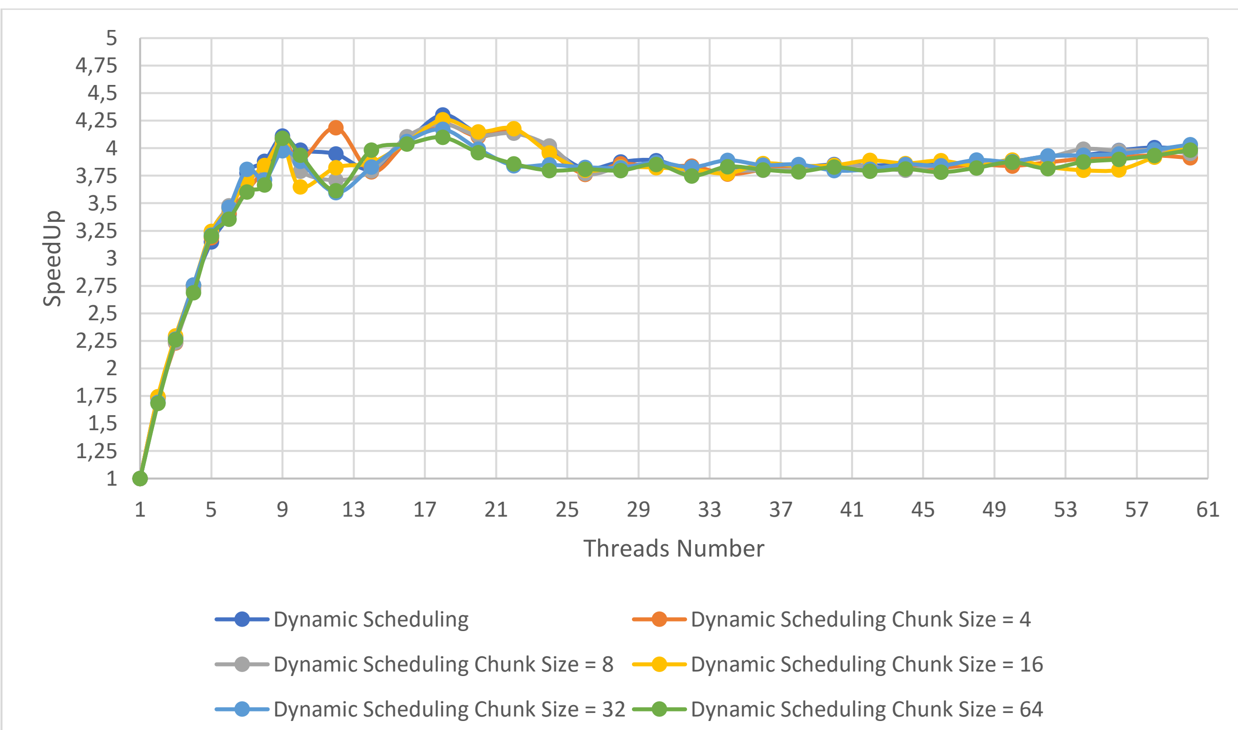
2. Guided, Chunk Size = 4,8,16,32,64 Speedup + Efficiency





3. Dynamic, Chunk Size = 4,8,16,32,64 Speedup + Efficiency





Conclusion

Speed Up												
Chunk-size / Scheduling	1		4		8		16		32		64	
	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM
static	4.616	4.221	4.788	4.360	4.530	4.240	4.558	4.223	4.611	4.256	4.571	4.571
	T:36	T: 18	T:38	T: 18	T:38	T: 18	T:38	T: 18	T:38	T: 18	T:38	T: 18
dynamic	4.665	4.303	4.848	4.221	4.722	4.219	4.660	4.258	4.701	4.171	4.744	4.098
	T:38	T:18	T:38	T:18	T:38	T:18	T:36	T:18	T:38	T:18	T:36	T:18
guided	4.671	4.116	4.619	4.149	4.545	4.265	4.700	4.246	4.646	4.220	4.778	4.189
	T : 38	T:18	T:38	T:18	T:34	T:18	T:38	T:18	T:38	T:18	T:38	T:18
auto	HPSL : 4.799											
	T:38											

CPU - time												
Chunk-size / Scheduling	1		4		8		16		32		64	
	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM
static	12.341	13.940	11,806	13.481	12.528	13.859	12.403	13.874	12.459	13.834	12.449	12.449
	T:36	T:18	T:38	T:18	T:38	T:18	T:38	T:18	T:38	T:18	T:38	T:18
dynamic	12.164	13.813	11.733	13.888	12.033	13.888	12.156	13.811	12.088	13.930	12.022	13.989
	T:38	T:18	T:38	T:18	T:38	T:18	T:36	T:18	T:38	T:18	T:36	T:18
guided	12.254	14.101	12.436	13.894	12.562	13.648	12.051	13.887	12.189	13.992	12.824	13.983
	T:38	T:18	T:38	T:18	T:34	T:18	T:38	T:18	T:38	T:18	T:38	T:18
auto												

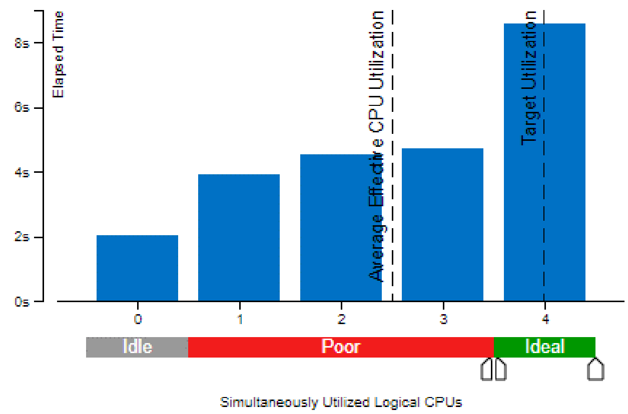
Parallel Algorithm (II)

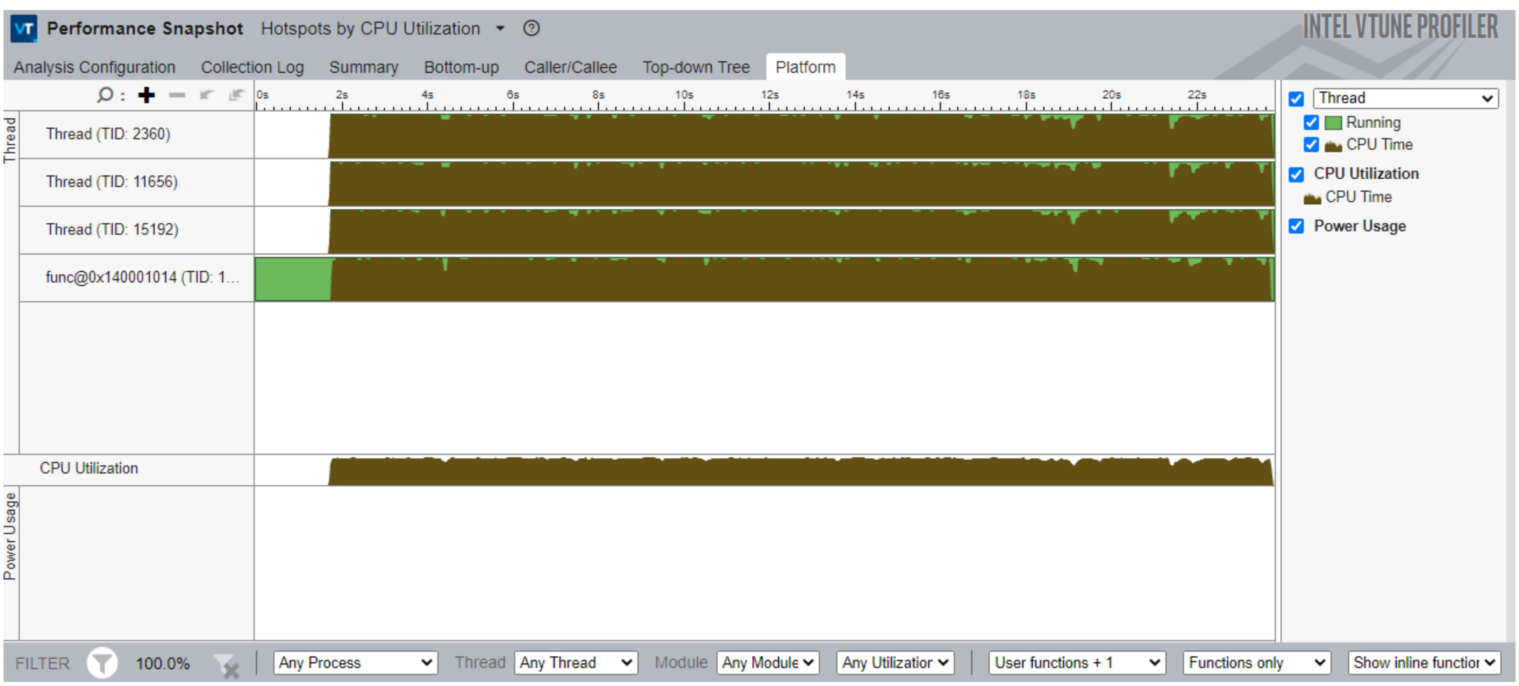
To parallelize the algorithm, I used OpenMP to split image in number of threads.

Hotspots by CPU Utilization

Effective CPU Utilization Histogram

Elapsed Time: 23.798 s
Total Thread : 4
Paused Time : 0 s





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

	. . .

	//Citeste valorile din imagine
	int i, j;
	#pragma omp parallel for num_threads(MAX_THREADS) private(i,j)
	for (i = 0; i < DIM; i++) {
		for (j = 0; j < DIM; j++) {
			picture_ct[DIM * i + j] = picture_in[DIM * i + j] =
(pix.at<cv::Vec3b>(i, j)[0] == 0) ? T_MAX : T_MIN;
		}
	}

	#pragma omp parallel
	for (int i = 0; i < 100; i++) {
		for (int i = 0; i < CONDUCTIVITY; i++) {
			if (out_pic) {
				copy_values_from_previous_step(picture_in, picture_ct);
				update_values(picture_out, picture_in);
			}
			else {
				copy_values_from_previous_step(picture_out, picture_ct);
				update_values(picture_in, picture_out);
			}
			#pragma omp barrier
			#pragma omp single
			out_pic = !out_pic;
		}

		float_to_color(pix_out, picture_out);
		#pragma omp barrier
		#pragma omp single
		video.write(pix_out);
	}
	video.release();

	...

	return 0;
}
```

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*----Copiaza temperatura de la pasul anterior----*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~*/
void copy_values_from_previous_step(float* in, float* ct) {
	for (long long i = omp_get_thread_num() * DIM * DIM / MAX_THREADS; i <
omp_get_thread_num() * DIM * DIM / MAX_THREADS + DIM * DIM / MAX_THREADS; ++i) {
		if (ct[i] != T_MIN)
			in[i] = ct[i];
	}
}
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
```

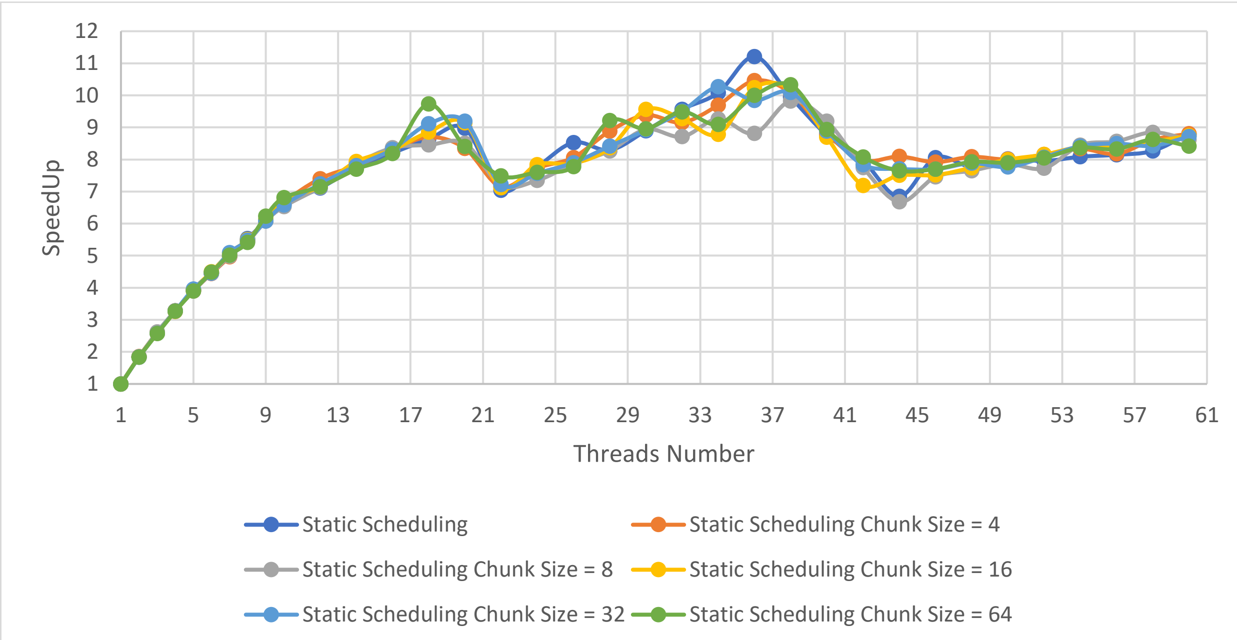
```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*----Calculeaza temperatura----*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~*/
void update_values(float* out, float* in) {
       for (long long i = omp_get_thread_num() * DIM * DIM / MAX_THREADS; i <
omp_get_thread_num() * DIM * DIM / MAX_THREADS + DIM * DIM / MAX_THREADS; ++i) {
              long long left = (i % DIM == 0) ? i : i - 1;
              long long right = (i % DIM == DIM - 1) ? i : i + 1;
              long long top = (i < DIM) ? i : i - DIM;
              long long bottom = (i >= DIM * (DIM - 1)) ? i : i + DIM;
              out[i] = in[i] + 0.1f * (in[top] + in[left] + in[right] + in[bottom] - in[i] * 4);
       }
}
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
```

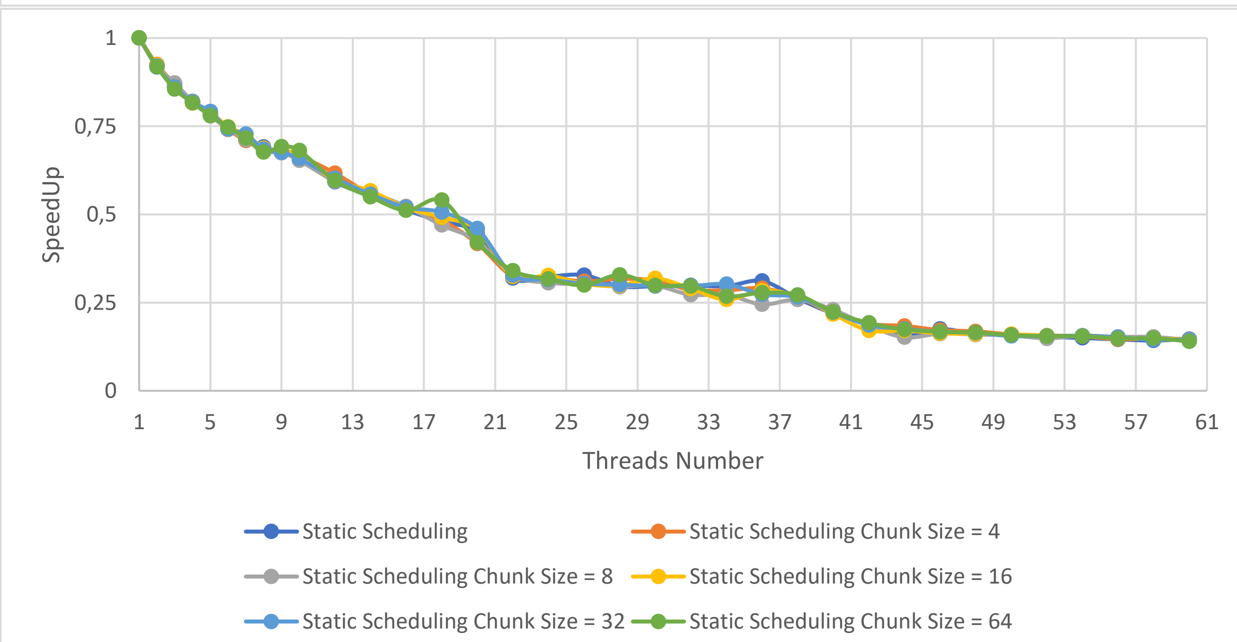
```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*----Converteste valorile din float -> HSL -> RGB ----*/
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
void float_to_color(Mat ptr_out, float* out)
{
       // Find pixel position
       for (long long a = omp_get_thread_num() * DIM * DIM / MAX_THREADS; a <
omp_get_thread_num() * DIM * DIM / MAX_THREADS + DIM * DIM / MAX_THREADS; ++a) {
              int i = a / DIM;
              int j = a % DIM;

              float lightness = out[DIM * i + j];
              float saturation = 1;
              int   hue = (180 + (int)(360.0f * lightness)) % 360;
              float m1, m2;

              m2 = (lightness <= 0.5f) ? lightness * (1 + saturation) : lightness +
saturation - lightness * saturation;
              m1 = 2 * lightness - m2;

              ptr_out.at<cv::Vec3b>(i, j)[2] = value(m1, m2, hue + 120);
              ptr_out.at<cv::Vec3b>(i, j)[1] = value(m1, m2, hue);
              ptr_out.at<cv::Vec3b>(i, j)[0] = value(m1, m2, hue - 120);
       }
}
/********************************************************************************/
```

Speedup, Efficiency, CPU-time

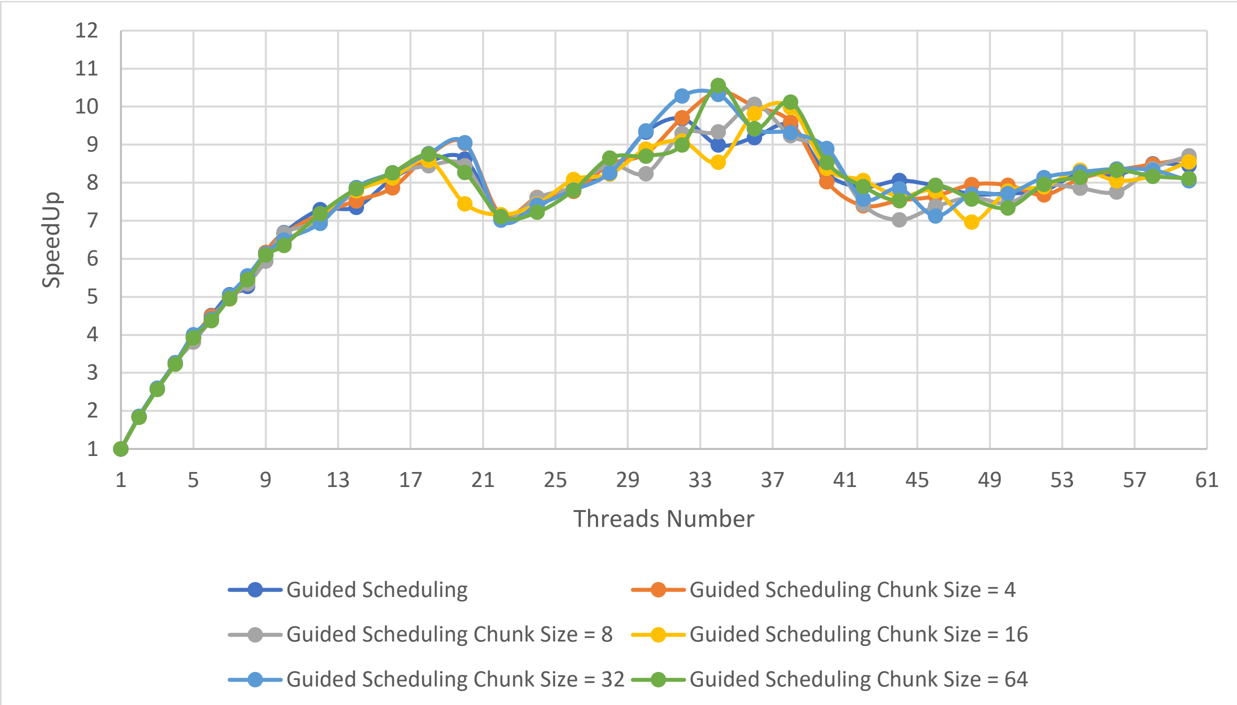
Running with -O3 on HPSL cluster

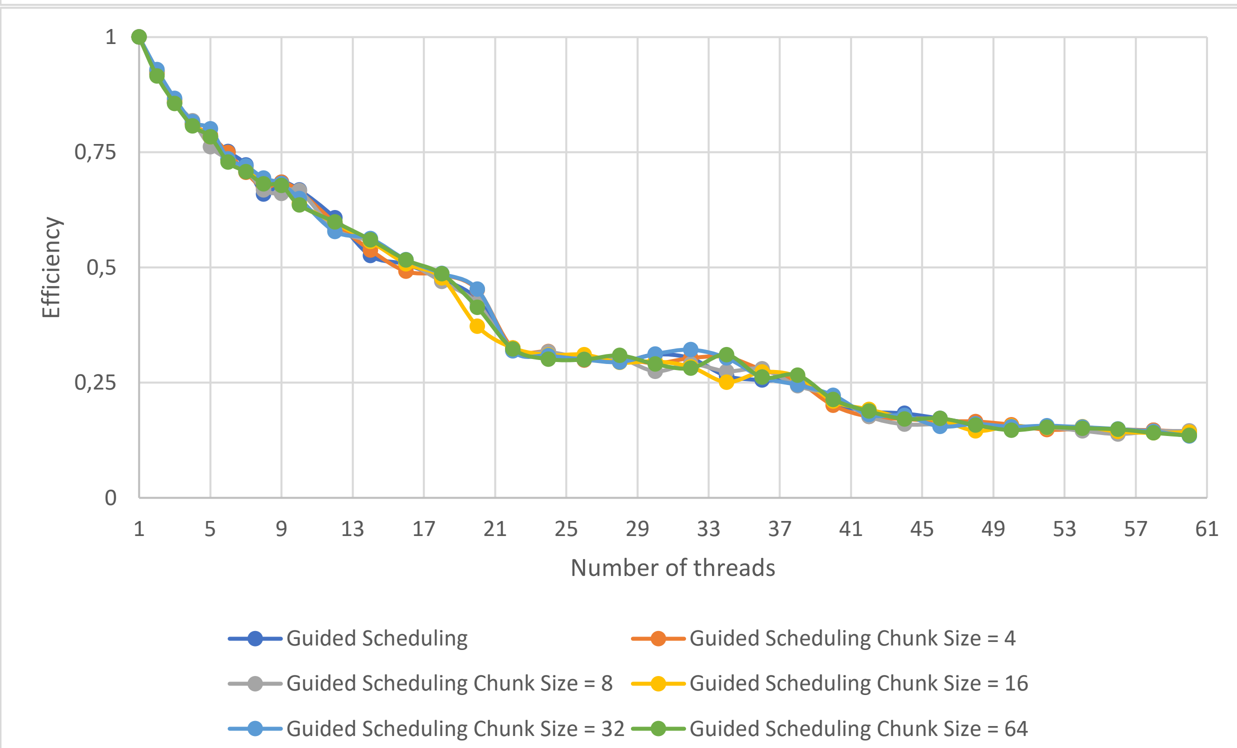
1. Static, Chunk Size = 4,8,16,32,64 Speedup + Efficiency



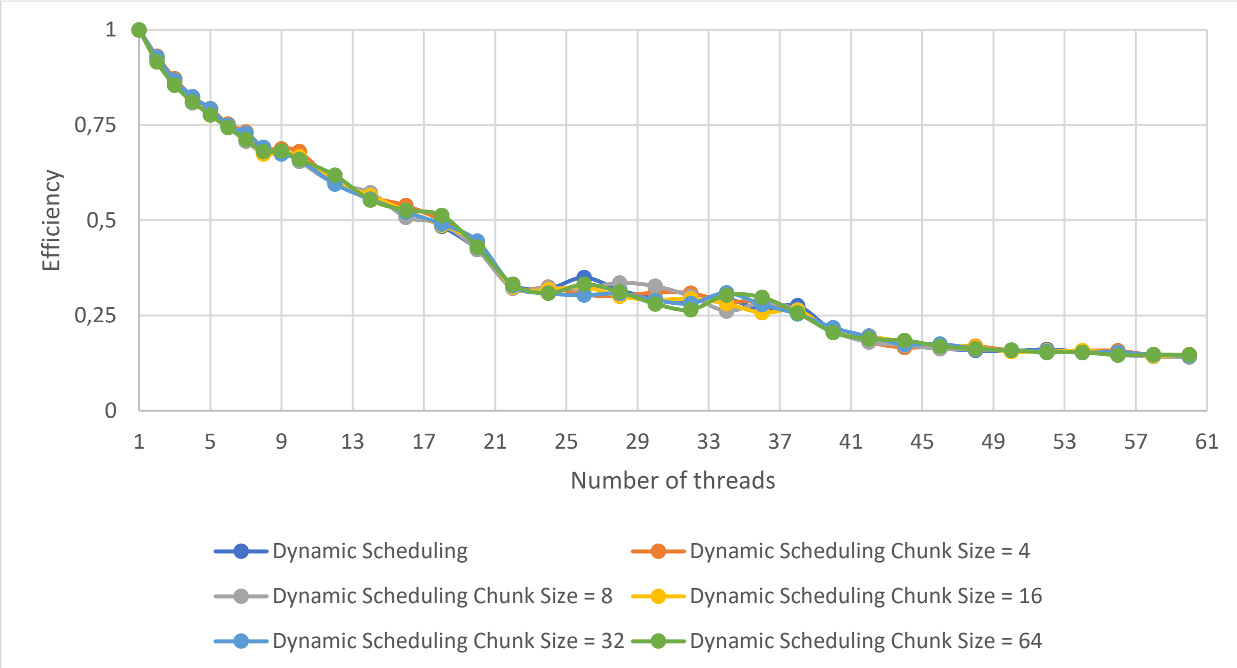


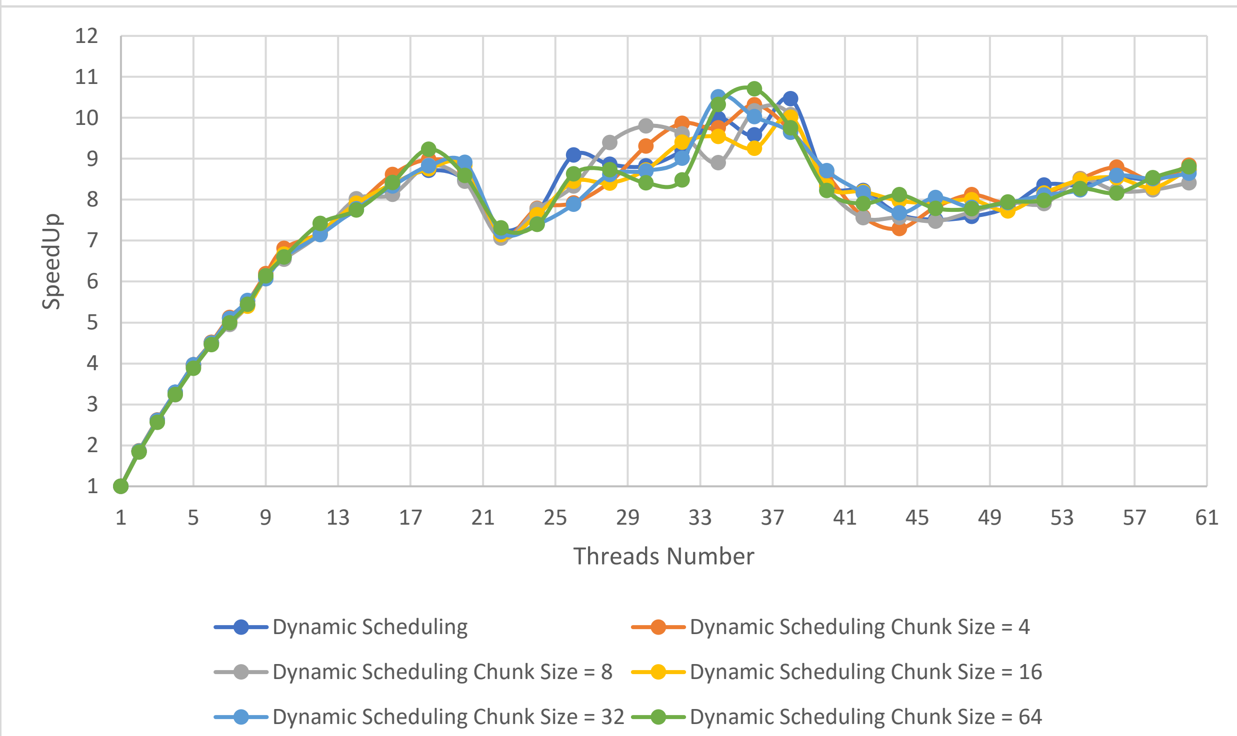
2. Guided, Chunk Size = 4,8,16,32,64 Speedup + Efficiency





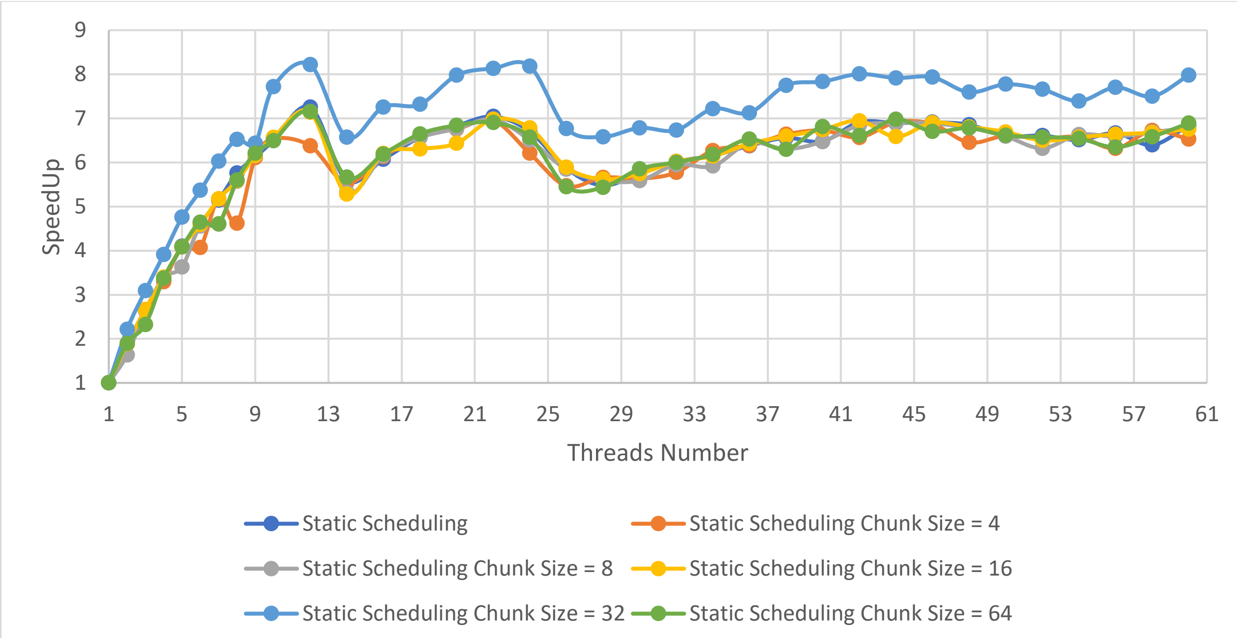
3. Dynamic, Chunk Size = 4,8,16,32,64 Speedup + Efficiency

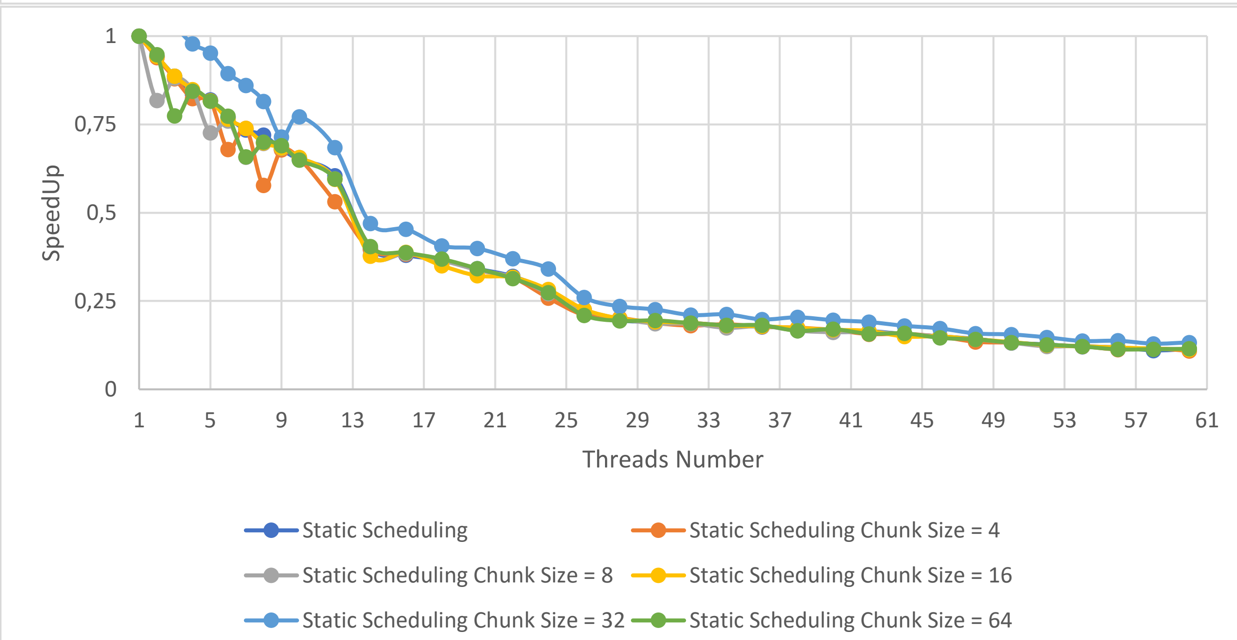




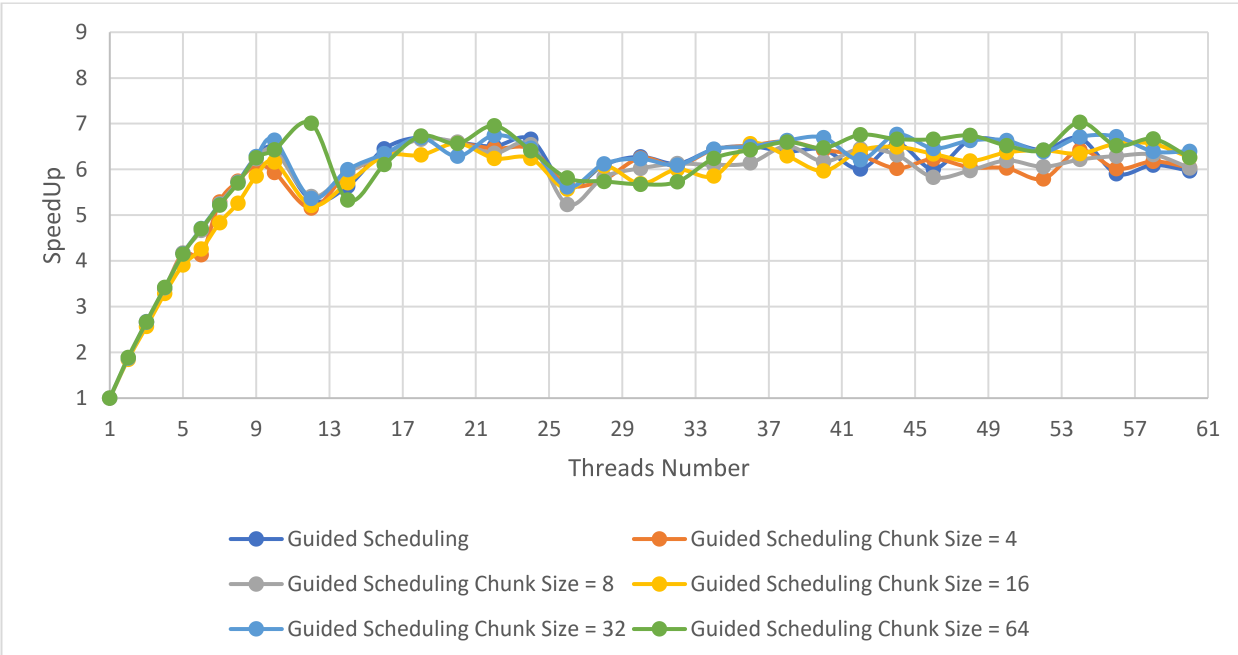
Running with -O3 on IBM cluster

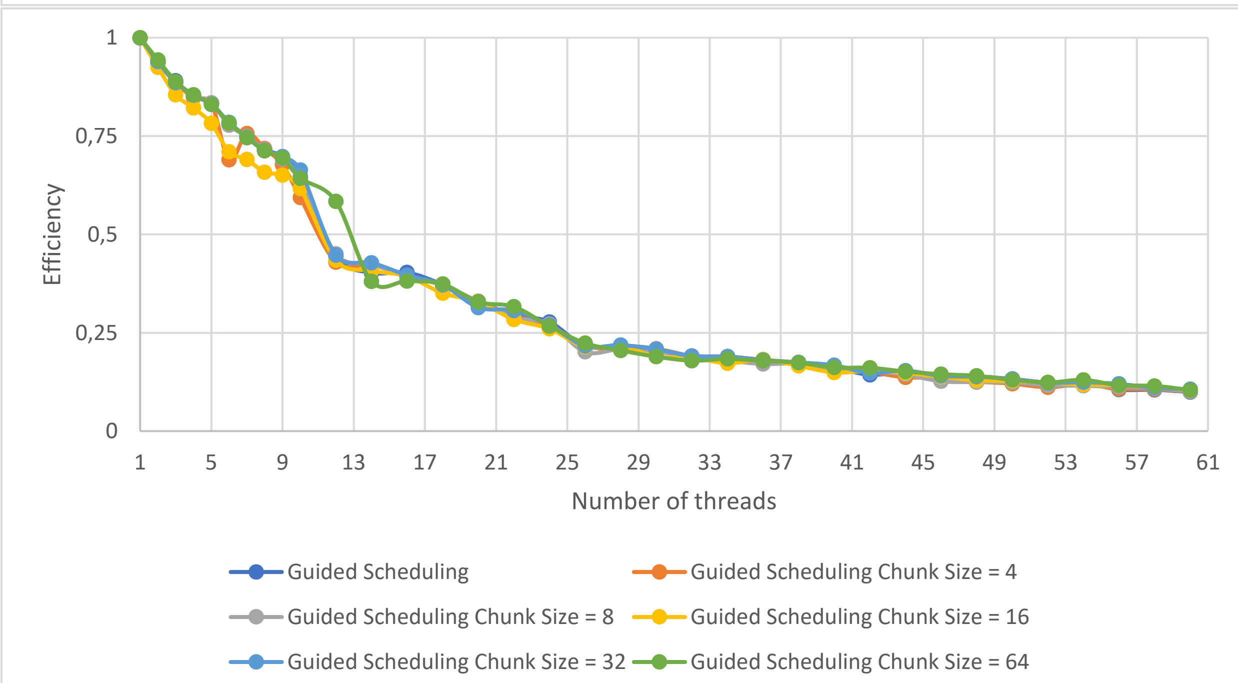
1. Static, Chunk Size = 4,8,16,32,64 Speedup + Efficiency



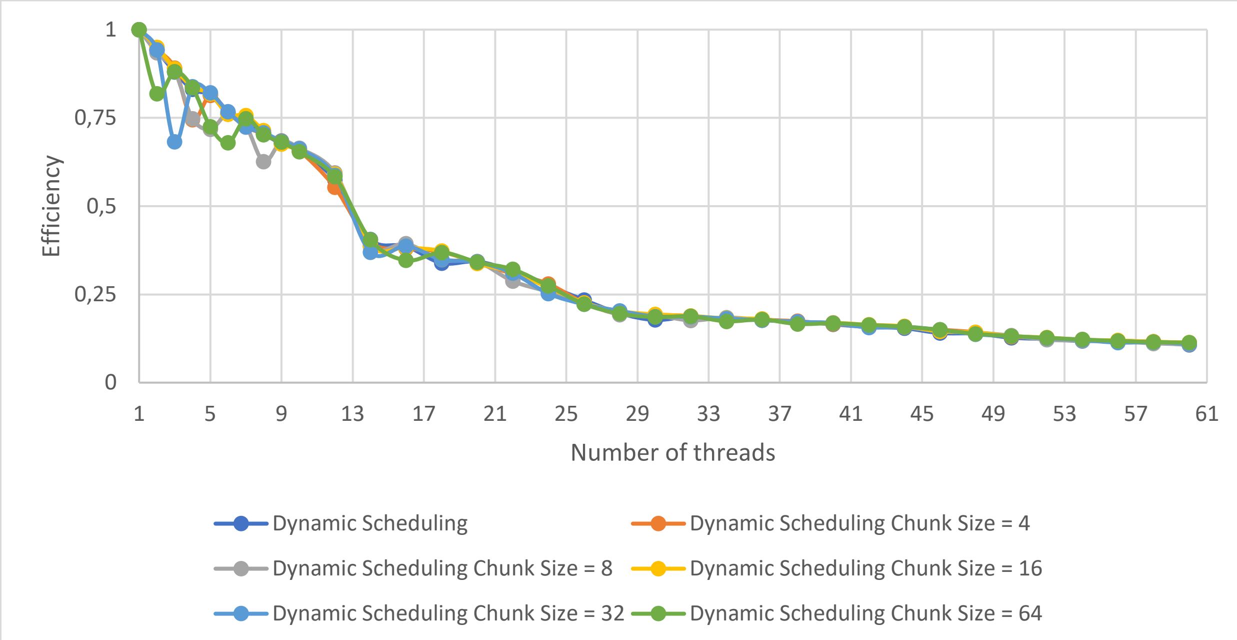


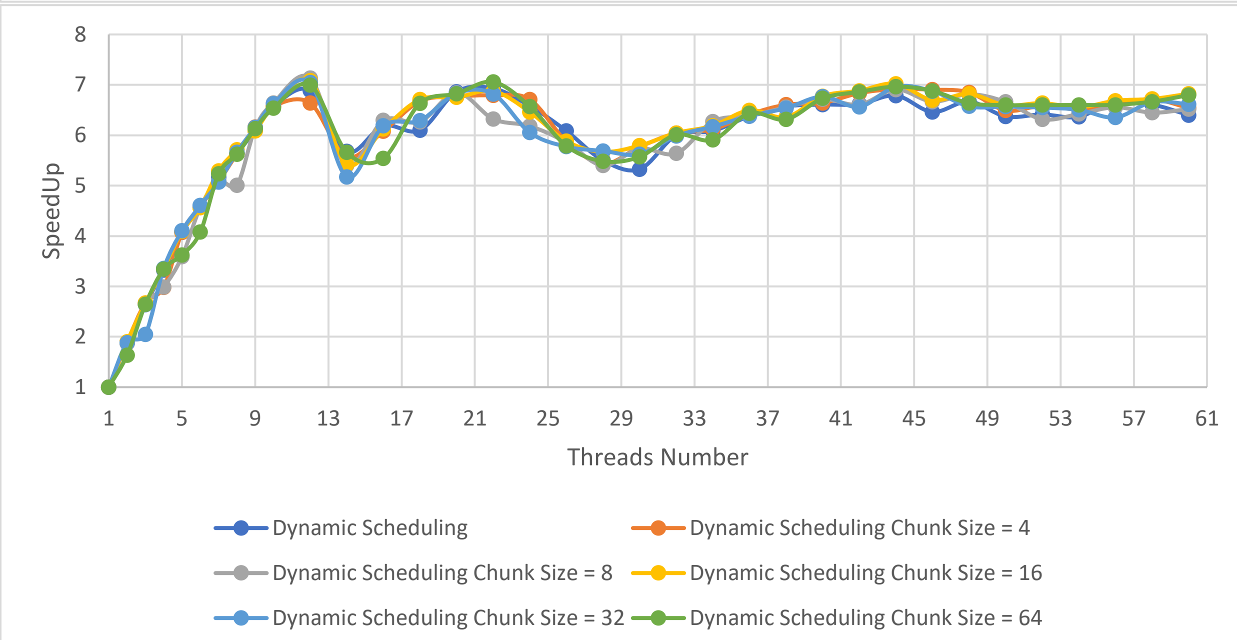
2. Guided, Chunk Size = 4,8,16,32,64 Speedup + Efficiency





3. Dynamic, Chunk Size = 4,8,16,32,64 Speedup + Efficiency





Speed Up												
Chunk-size / Scheduling	1		4		8		16		32		64	
	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM
static	11.20	7.254	10.45	6.954	9.823	7.153	10.23	7.139	10.26	8.216	10.32	7.151
	T:36	T:12	T:36	T:22	T:38	T:12	T:36	T:12	T:34	T:12	T:38	T:12
dynamic	10.46	6.929	10.31	6.914	10.16	7.134	10.01	7.083	10.51	7.042	10.70	7.057
	T:38	T:12	T:36	T:12	T:36	T:12	T:38	T:12	T:34	T:12	T:36	T:12
guided	9.688	6.704	10.39	6.679	10.06	6.634	9.981	6.572	10.32	6.745	10.56	7.032
	T:32	T:18	T:34	T:18	T:36	T:18	T:38	T:20	T:34	T:22	T:34	T:12

CPU - time												
Chunk-size / Scheduling	1		4		8		16		32		64	
	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM	HPSL	IBM
static	13.848	20.405	14.831	21.255	15.742	20.748	15.201	20.861	15.134	21.118	14.968	20.798
	T:36	T:12	T:36	T:22	T:38	T:12	T:36	T:12	T:34	T:12	T:38	T:12
dynamic	14.89	21.352	15.13	21.508	15.167	20.772	15.519	21.04	14.785	21.039	14.394	21.010
	T:38	T:12	T:36	T:12	T:36	T:12	T:38	T:12	T:34	T:12	T:36	T:12
guided	16.059	22.125	14.975	22.227	15.287	22.351	15.611	22.705	15.135	21.920	14.657	21.199
	T:32	T:18	T:34	T:18	T:36	T:18	T:38	T:20	T:34	T:22	T:34	T:12

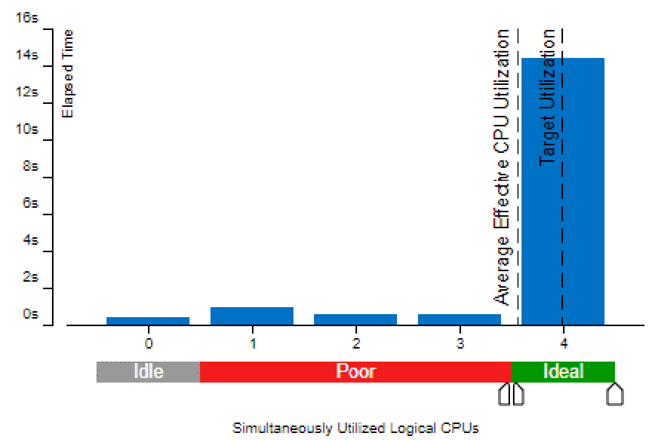
Parallel Algorithm (III) – implementation without opencv

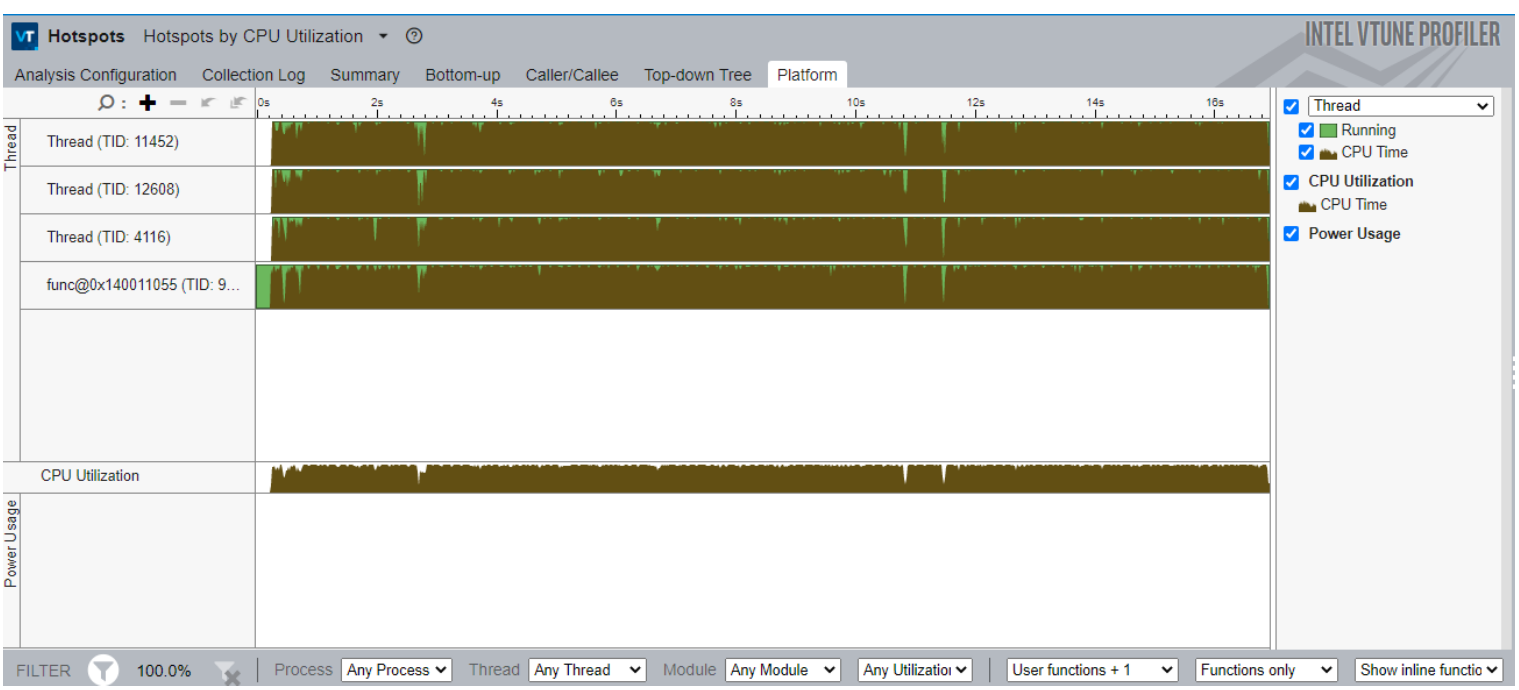
I was unable to replace or create a function to write a movie in MJPEG format. Instead, I saved every second of the video as a bitmap image. For reading and writing BMP files I created a structure below for BMP header, copy BMP footer, check if file is a BMP file, read file, copy data and write data.

Hotspots by CPU Utilization

Elapsed Time: 16.909 s
Total Thread : 4
Paused Time : 0 s

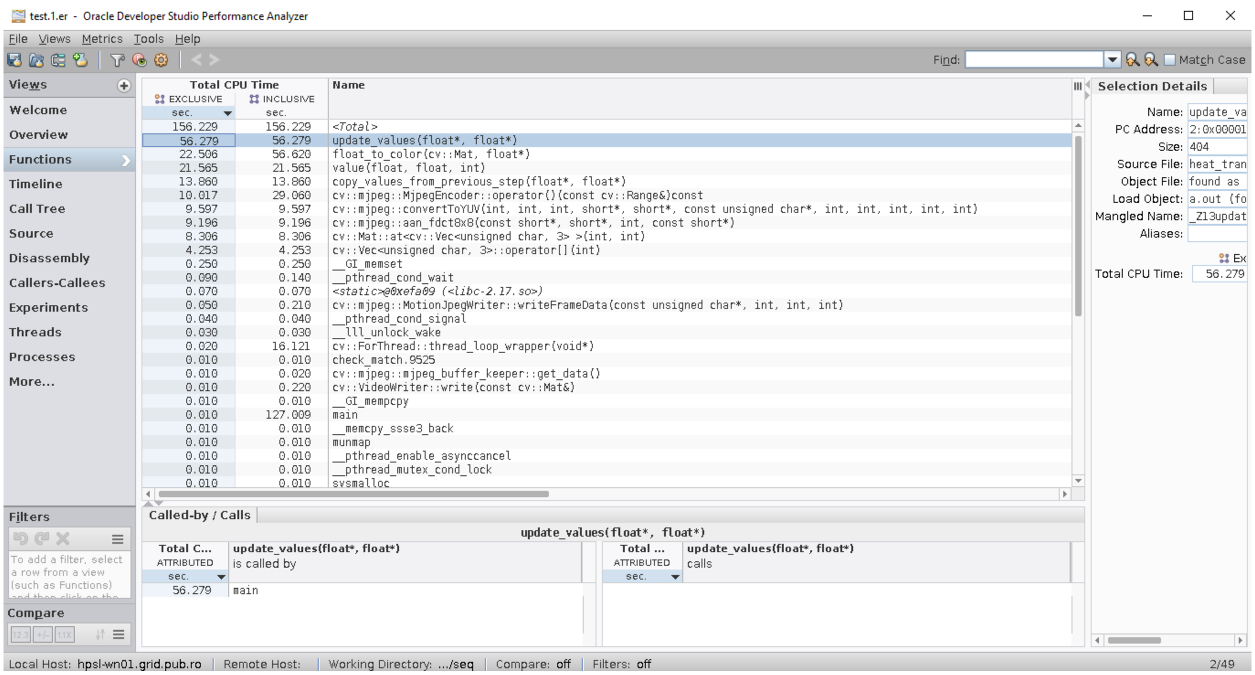
Effective CPU Utilization Histogram

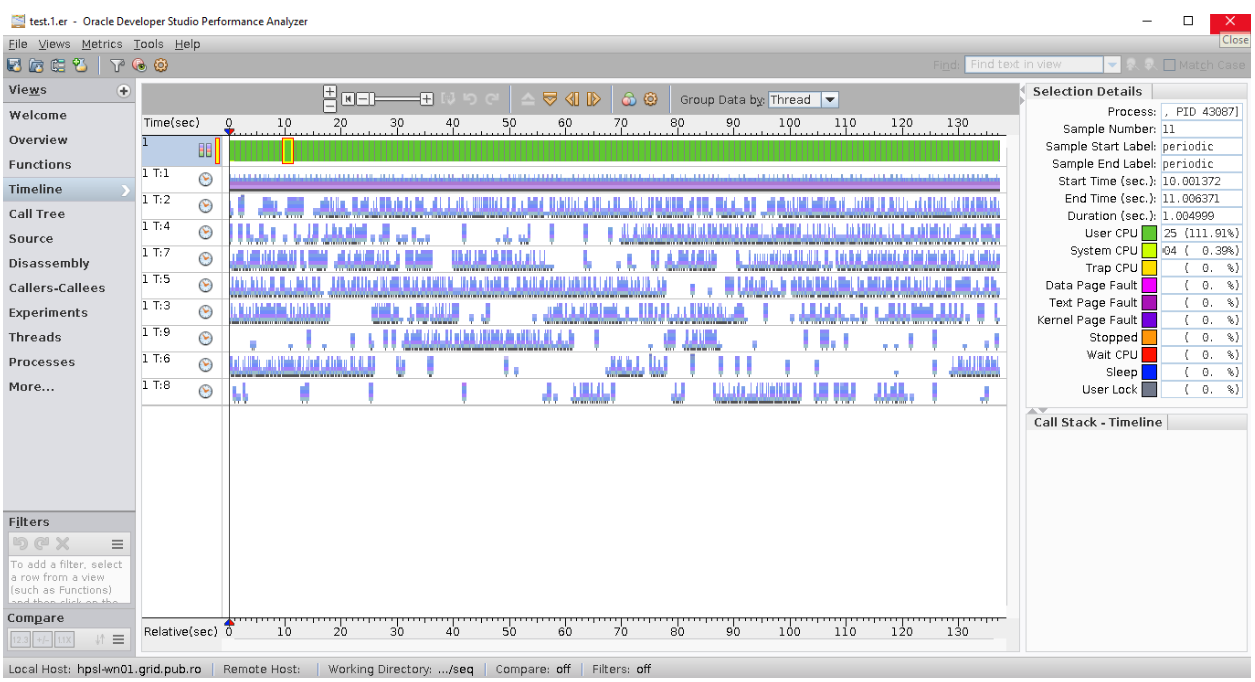


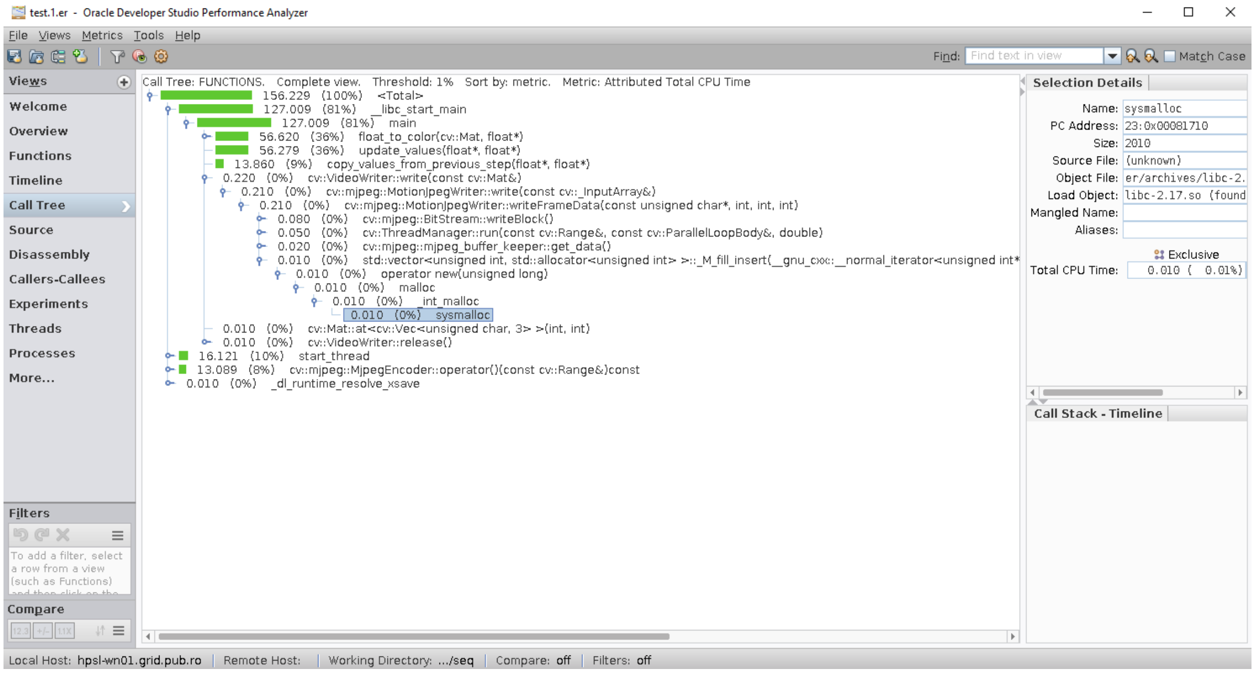


Profiling with Solaris Studio

Profiling for Sequential Algorithm

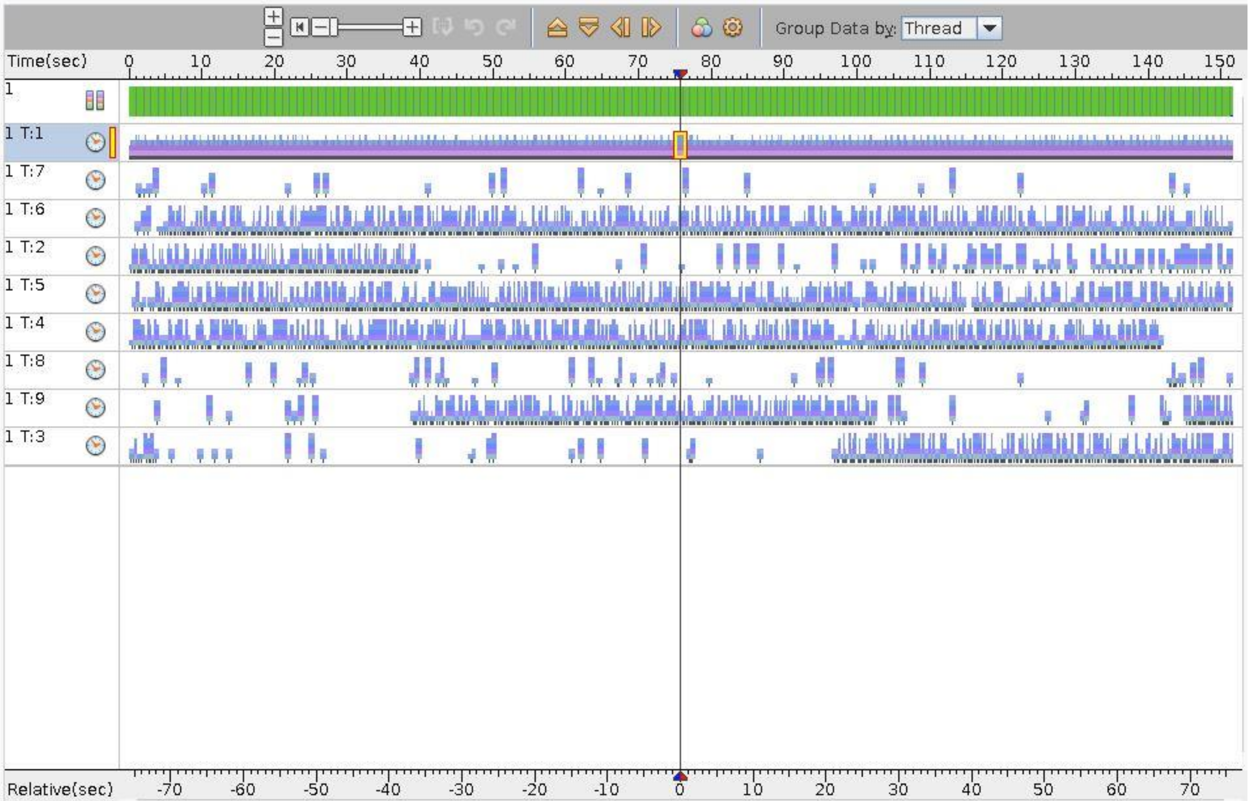


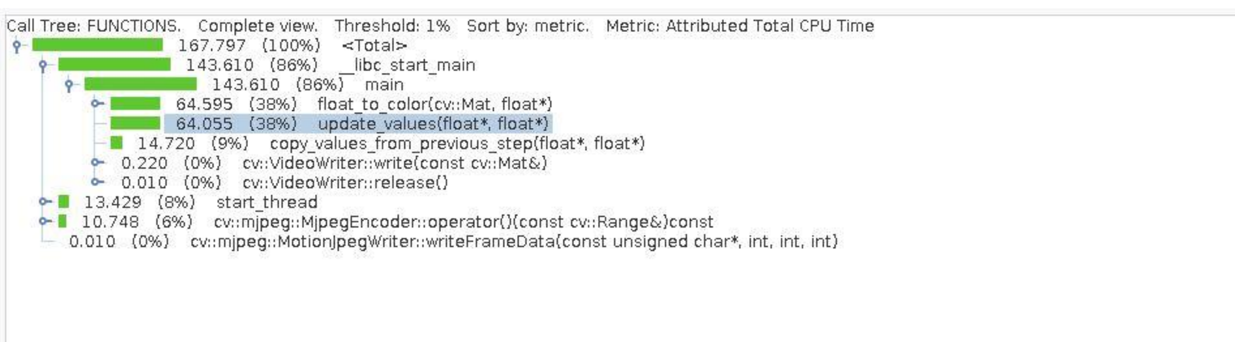




Profiling for Sequential Algorithm II

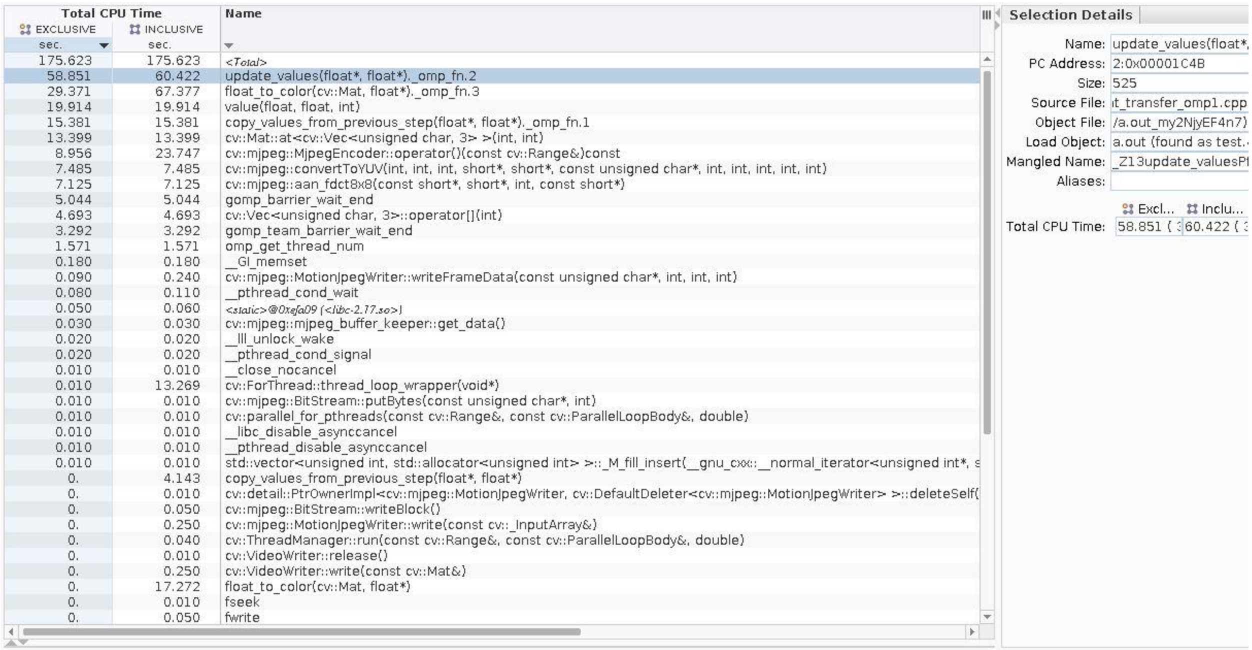
Total CPU Time EXCLUSIVE sec.	Total CPU Time INCLUSIVE sec.	Name
167.797	167.797	<Total>
64.055	64.055	update_values(float*, float*)
25.017	64.595	float_to_color(cv::Mat, float*)
20.855	20.855	value(float, float, int)
14.720	14.720	copy_values_from_previous_step(float*, float*)
13.860	13.860	cv::Mat::at<cv::Vec<unsigned char, 3> >(int, int)
10.597	24.157	cv::mjpeg::MjpegEncoder::operator()(const cv::Range&)const
6.775	6.775	cv::mjpeg::aan_fdct8x8(const short*, short*, int, const short*)
6.555	6.555	cv::mjpeg::convertToYUV(int, int, int, short*, short*, const unsigned char*, int, int, int, int, int)
4.863	4.863	cv::Vec<unsigned char, 3>::operator[](int)
0.230	0.230	__GI_memset
0.080	0.080	cv::mjpeg::mjpeg_buffer_keeper::get_data()
0.060	0.210	cv::mjpeg::MotionJpegWriter::writeFrameData(const unsigned char*, int, int, int)
0.040	0.040	__pthread_cond_wait
0.030	0.030	<static>@0xefa09 (<libc-2.17.so>)
0.020	0.020	__pthread_cond_signal
0.010	0.010	ftell
0.010	0.010	_IO_new_file_seekoff
0.010	143.610	main
0.010	0.010	munmap
0.	0.010	cv::detail::PtrOwnerImpl<cv::mjpeg::MotionJpegWriter, cv::DefaultDeleter<cv::mjpeg::MotionJpegWriter> >::deleteSelf(
0.	13.429	cv::ForThread::thread_loop_wrapper(void*)
0.	0.030	cv::mjpeg::BitStream::writeBlock()
0.	0.220	cv::mjpeg::MotionJpegWriter::write(const cv::_InputArray&)
0.	0.040	cv::ThreadManager::run(const cv::Range&, const cv::ParallelLoopBody&, double)
0.	0.010	cv::VideoWriter::release()
0.	0.220	cv::VideoWriter::write(const cv::Mat&)
0.	0.030	fwrite
0.	0.030	_IO_new_file_write
0.	0.030	_IO_new_file_xsputn
0.	143.610	__libc_start_main
0.	13.429	start_thread
0.	13.429	<static>@0x27dde (<libcollector.so>)
0.	0.010	std::deque<cv::mjpeg::mjpeg_buffer, std::allocator<cv::mjpeg::mjpeg_buffer> >::_M_destroy_data_aux(std::_Deque_it





```
Call Tree: FUNCTIONS.  Complete view.  Threshold: 1%  Sort by: metric.  Metric: Attributed Total CPU Time
167.797  (100%)  <Total>
  143.610  (86%)  __libc_start_main
    143.610  (86%)  main
      64.595  (38%)  float_to_color(cv::Mat, float*)
      64.055  (38%)  update_values(float*, float*)
      14.720  (9%)  copy_values_from_previous_step(float*, float*)
      0.220  (0%)  cv::VideoWriter::write(const cv::Mat&)
      0.010  (0%)  cv::VideoWriter::release()
  13.429  (8%)  start_thread
  10.748  (6%)  cv::mjpeg::MjpegEncoder::operator()(const cv::Range&)const
  0.010  (0%)  cv::mjpeg::MotionJpegWriter::writeFrameData(const unsigned char*, int, int, int)
```

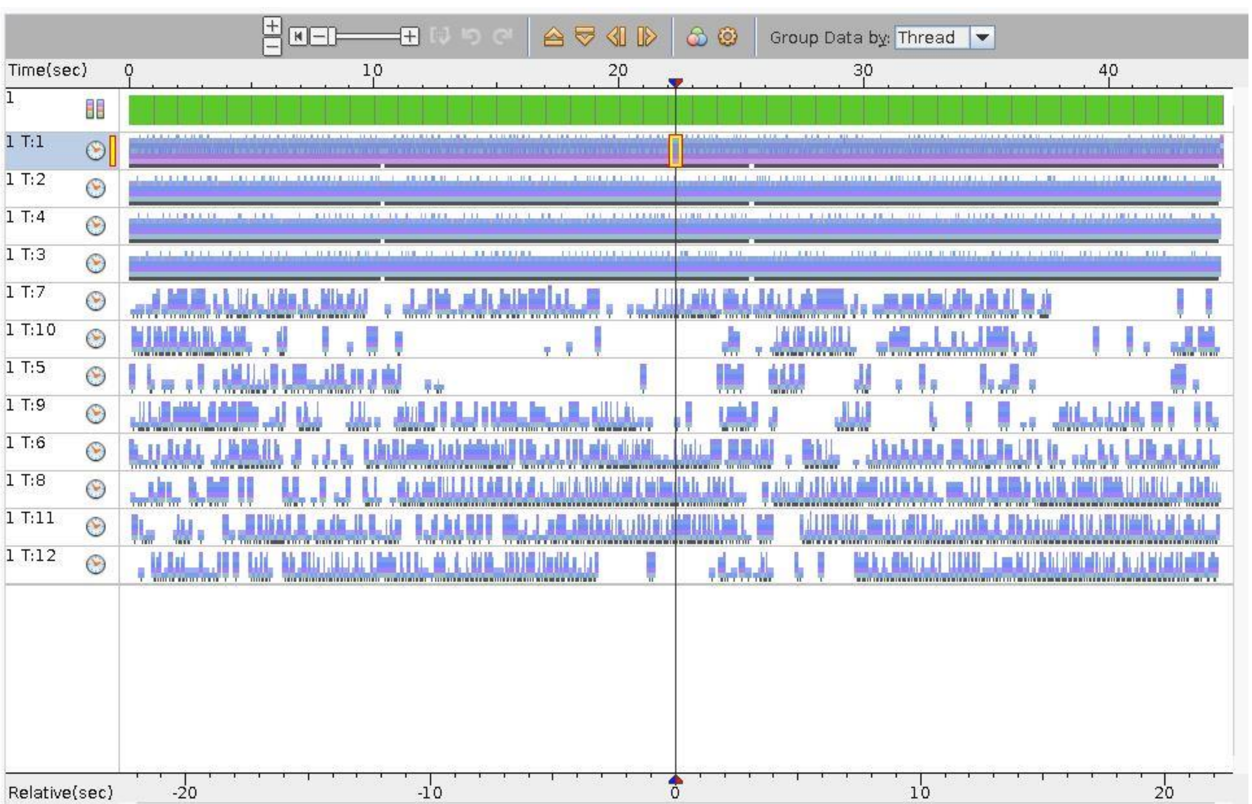
Profiling for Parallel Algorithm I



Total CPU Time EXCLUSIVE sec.	Total CPU Time INCLUSIVE sec.	Name
175.623	175.623	<Total>
58.851	60.422	update_values(float*, float*)._omp_fn.2
29.371	67.377	float_to_color(cv::Mat, float*)._omp_fn.3
19.914	19.914	value(float, float, int)
15.381	15.381	copy_values_from_previous_step(float*, float*)._omp_fn.1
13.399	13.399	cv::Mat::at<cv::Vec<unsigned char, 3> >(int, int)
8.956	23.747	cv::mjpeg::MjpegEncoder::operator()(const cv::Range&)const
7.485	7.485	cv::mjpeg::convertToYUV(int, int, int, short*, short*, const unsigned char*, int, int, int, int, int)
7.125	7.125	cv::mjpeg::aan_fdct8x8(const short*, short*, int, const short*)
5.044	5.044	gomp_barrier_wait_end
4.693	4.693	cv::Vec<unsigned char, 3>::operator[](int)
3.292	3.292	gomp_team_barrier_wait_end
1.571	1.571	omp_get_thread_num
0.180	0.180	__GI_memset
0.090	0.240	cv::mjpeg::MotionJpegWriter::writeFrameData(const unsigned char*, int, int, int)
0.080	0.110	__pthread_cond_wait
0.050	0.060	<static>@0xefa09 (<libc-2.17.so>)
0.030	0.030	cv::mjpeg::mjpeg_buffer_keeper::get_data()
0.020	0.020	__lll_unlock_wake
0.020	0.020	__pthread_cond_signal
0.010	0.010	__close_nocancel
0.010	13.269	cv::ForThread::thread_loop_wrapper(void*)
0.010	0.010	cv::mjpeg::BitStream::putBytes(const unsigned char*, int)
0.010	0.010	cv::parallel_for_pthreads(const cv::Range&, const cv::ParallelLoopBody&, double)
0.010	0.010	__libc_disable_asynccancel
0.010	0.010	__pthread_disable_asynccancel
0.010	0.010	std::vector<unsigned int, std::allocator<unsigned int> >::_M_fill_insert(__gnu_cxx::__normal_iterator<unsigned int*, s
0.	4.143	copy_values_from_previous_step(float*, float*)
0.	0.010	cv::detail::PtrOwnerImpl<cv::mjpeg::MotionJpegWriter, cv::DefaultDeleter<cv::mjpeg::MotionJpegWriter> >::deleteSelf(
0.	0.050	cv::mjpeg::BitStream::writeBlock()
0.	0.250	cv::mjpeg::MotionJpegWriter::write(const cv::_InputArray&)
0.	0.040	cv::ThreadManager::run(const cv::Range&, const cv::ParallelLoopBody&, double)
0.	0.010	cv::VideoWriter::release()
0.	0.250	cv::VideoWriter::write(const cv::Mat&)
0.	17.272	float_to_color(cv::Mat, float*)
0.	0.010	fseek
0.	0.050	fwrite

Selection Details

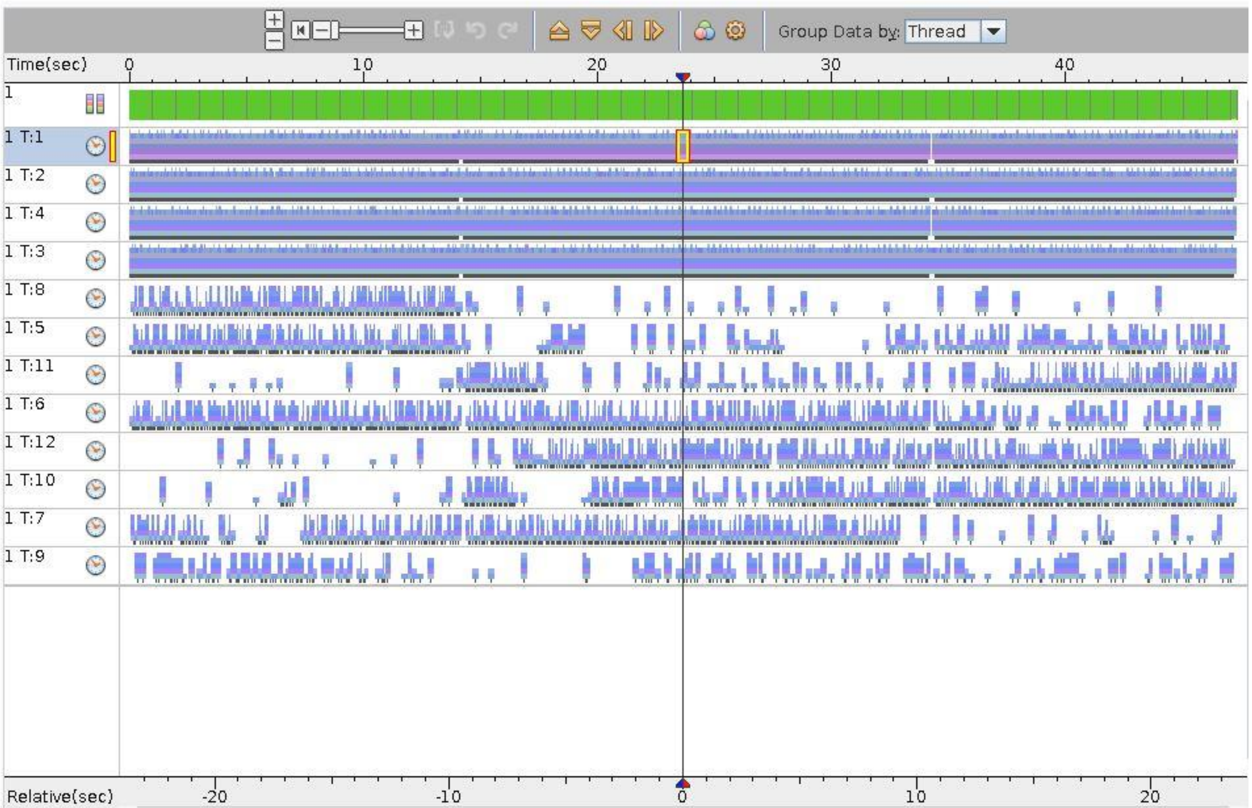
Name:	update_values(float*,	
PC Address:	2:0x00001C4B	
Size:	525	
Source File:	ıt_transfer_omp1.cpp	
Object File:	/a.out_my2NjyEF4n7)	
Load Object:	a.out (found as test.	
Mangled Name:	_Z13update_valuesPf	
Aliases:		
	Excl...	Inclu...
Total CPU Time:	58.851 (3	60.422 (3



```
Call Tree: FUNCTIONS.   Complete view.   Threshold: 1%   Sort by: metric.   Metric: Attributed Total CPU Time
175.623  (100%)   <Total>
  128.020  (73%)   start_thread
    128.020  (73%)   <static>@0x27dde (<libcollector.so>)
      114.750  (65%)   gomp_thread_start
        50.916  (29%)   float_to_color(cv::Mat, float*)._omp_fn.3
        45.252  (26%)   update_values(float*, float*)._omp_fn.2
          1.181  (1%)   omp_get_thread_num
        11.788  (7%)   copy_values_from_previous_step(float*, float*)._omp_fn.1
        5.034  (3%)   gomp_barrier_wait_end
        1.761  (1%)   gomp_team_barrier_wait_end
      13.269  (8%)   cv::ForThread::thread_loop_wrapper(void*)
  37.026  (21%)   __libc_start_main
  10.577  (6%)   cv::mjpeg::MjpegEncoder::operator()(const cv::Range&)const
```

Profiling for Parallel Algorithm II

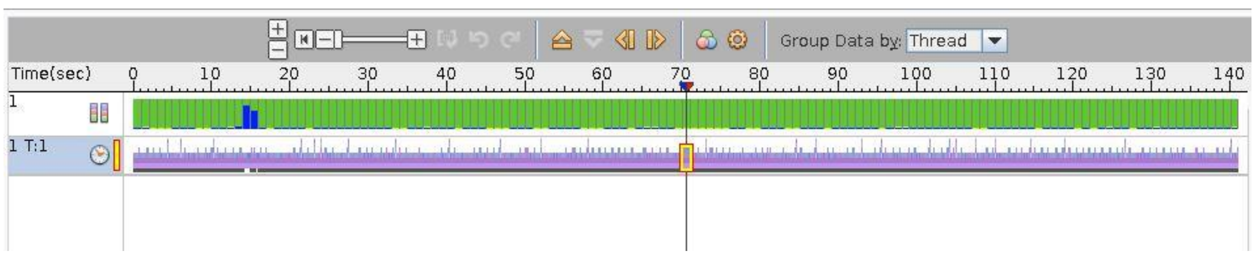
Total CPU Time EXCLUSIVE sec.	INCLUSIVE sec.	Name
185.780	185.780	<Total>
58.141	59.682	update_values(float*, float*)
31.992	72.321	float_to_color(cv::Mat, float*)
21.065	22.616	copy_values_from_previous_step(float*, float*)
20.925	20.925	value(float, float, int)
13.870	13.870	cv::Mat::at<cv::Vec<unsigned char, 3> >(int, int)
9.246	24.397	cv::mjpeg::MjpegEncoder::operator()(const cv::Range&)const
7.805	7.805	cv::mjpeg::convertToYUV(int, int, int, short*, short*, const unsigned char*, int, int, int, int, int)
7.165	7.165	cv::mjpeg::aan_fdct8x8(const short*, short*, int, const short*)
6.585	6.585	gomp_team_barrier_wait_end
4.933	4.933	cv::Vec<unsigned char, 3>::operator[](int)
3.693	3.693	omp_get_thread_num
0.180	0.180	__GI_memset
0.080	0.130	cv::mjpeg::MotionJpegWriter::writeFrameData(const unsigned char*, int, int, int)
0.020	0.020	cv::mjpeg::mjpeg_buffer_keeper::get_data()
0.020	0.020	__pthread_cond_wait
0.010	0.010	__close_nocancel
0.010	0.010	fwrite
0.010	161.343	main._omp_fn.1
0.010	0.010	pthread_mutex_unlock
0.010	0.010	<static>@0xef769 (<libc-2.17.so>)
0.010	0.010	std::vector<unsigned int, std::allocator<unsigned int> >::_M_fill_insert(__gnu_cxx::__normal_iterator<unsigned int*, s
0.	0.010	cv::createMotionJpegWriter(const cv::String&, double, cv::Size_<int>, bool)
0.	0.010	cv::detail::PtrOwnerImpl<cv::mjpeg::MotionJpegWriter, cv::DefaultDeleter<cv::mjpeg::MotionJpegWriter> >::deleteSelf(
0.	13.890	cv::ForThread::thread_loop_wrapper(void*)
0.	0.010	cv::makePtr<cv::mjpeg::MotionJpegWriter, cv::String, double, cv::Size_<int>, bool>(const cv::String&, const double&,
0.	0.010	cv::mjpeg::BitStream::writeBlock()
0.	0.130	cv::mjpeg::MotionJpegWriter::write(const cv::_InputArray&)
0.	0.010	cv::ThreadManager::run(const cv::Range&, const cv::ParallelLoopBody&, double)
0.	0.010	cv::VideoWriter::open(const cv::String&, int, double, cv::Size_<int>, bool)
0.	0.010	cv::VideoWriter::release()
0.	0.010	cv::VideoWriter::VideoWriter(const cv::String&, int, double, cv::Size_<int>, bool)
0.	0.130	cv::VideoWriter::write(const cv::Mat&)
0.	0.010	_fopen_internal
0.	40.068	GOMP_parallel
0.	121.275	gomp_thread_start
0.	0.010	_IO_new_file_close_it




```
Call Tree: FUNCTIONS.   Complete view.   Threshold: 1%   Sort by: metric.   Metric: Attributed Total CPU Time
185.780  (100%)   <Total>
  135.165  (73%)   start_thread
    135.165  (73%)   <static>@0x27dde (<libcollector.so>)
      121.275  (65%)   gomp_thread_start
        121.275  (65%)   main._omp_fn.1
          54.308  (29%)   float_to_color(cv::Mat, float*)
          44.651  (24%)   update_values(float*, float*)
            1.101  (1%)   omp_get_thread_num
          17.282  (9%)   copy_values_from_previous_step(float*, float*)
          4.933  (3%)   gomp_team_barrier_wait_end
          0.090  (0%)   cv::VideoWriter::write(const cv::Mat&)
      13.890  (7%)   cv::ForThread::thread_loop_wrapper(void*)
  40.088  (22%)   __libc_start_main
  10.527  (6%)   cv::mjpeg::MjpegEncoder::operator()(const cv::Range&)const
```

Profiling for Sequential Program without OpenCV library

Total CPU Time EXCLUSIVE sec.	Total CPU Time INCLUSIVE sec.	Name
135.625	135.625	<Total>
64.775	64.775	update_values(float*, float*)
25.748	52.987	float_to_color(std::vector<unsigned char, std::allocator<unsigned char> >&, float*)
23.216	23.216	value(float, float, int)
14.670	14.670	copy_values_from_previous_step(float*, float*)
4.033	4.033	std::vector<unsigned char, std::allocator<unsigned char> >::operator[](unsigned long)
1.761	1.761	writev
0.801	0.801	__open_nocancel
0.590	0.590	__close_nocancel
0.010	0.010	_IO_link_in
0.010	0.010	operator new(unsigned long)
0.010	0.010	std::ios_base::ios_base()
0.	3.182	BMP::writeBMP(const char*)
0.	1.761	BMP::write_headers_and_data(std::basic_ofstream<char, std::char_traits<char> >&)
0.	0.811	__fopen_internal
0.	0.590	_IO_new_file_close_it
0.	0.801	_IO_new_file_fopen
0.	0.010	_IO_new_file_init_internal
0.	135.625	__libc_start_main
0.	135.625	main
0.	0.590	__new_fclose
0.	0.590	std::basic_filebuf<char, std::char_traits<char> >::close()
0.	0.010	std::basic_filebuf<char, std::char_traits<char> >::_M_allocate_internal_buffer()
0.	0.821	std::basic_filebuf<char, std::char_traits<char> >::open(const char*, std::_Ios_Openmode)
0.	1.761	std::basic_filebuf<char, std::char_traits<char> >::xsputn(const char*, long)
0.	0.590	std::__basic_file<char>::close()
0.	0.811	std::__basic_file<char>::open(const char*, std::_Ios_Openmode, int)
0.	1.761	std::__basic_file<char>::xsputn_2(const char*, long, const char*, long)
0.	0.590	std::basic_ofstream<char, std::char_traits<char> >::~basic_ofstream()
0.	0.831	std::basic_ofstream<char, std::char_traits<char> >::basic_ofstream(const char*, std::_Ios_Openmode)
0.	1.761	std::basic_ostream<char, std::char_traits<char> >::write(const char*, long)




```
Call Tree: FUNCTIONS.   Complete view.   Threshold: 1%   Sort by: metric.   Metric: Attributed Total CPU Time
135.625  (100%)   <Total>
  135.625  (100%)   __libc_start_main
    135.625  (100%)   main
      64.775  (48%)   update_values(float*, float*)
      52.987  (39%)   float_to_color(std::vector<unsigned char, std::allocator<unsigned char> >&, float*)
      14.670  (11%)   copy_values_from_previous_step(float*, float*)
      3.182  (2%)   BMP::writeBMP(const char*)
      0.010  (0%)   std::vector<unsigned char, std::allocator<unsigned char> >::operator[](unsigned long)
```