Karlsruher Institut für Technologie

Fakultät für Informatik
Institut für Technische Informatik
Chair for Embedded Systems

Lab: Low Power Design and Embedded Systems

Lab Report

Alkistis Aikaterini Sigourou
upwkh@student.kit.edu
Matrikelnummer 2394538

March 7, 2022

Contents

Abstract

Main goal of this lab is to get acquainted with the combination of efficiency with best timings, prices, durability and complexity in a application. As while as to see the big impact that temperature plays to a system. Focusing on loop transformations, hybrid and hardware implementation and thermal observance.

1 Introduction

In modern days as the everyday requirements are rising the need of evolving our IT systems is rising with. The aspects of a system, that we are looking more, nowadays is the combination of efficiency,of course, with best timings, prices, durability and complexity.

During this lab, we will explore the power of loop transforming in a application, how easy to reverse a software application in a hardware one and what effects it will have the characteristics that we talked above. Finally, we will see a representation of the thermal attitude of a processor while a benchmark is running over it.

The applications that we are going to focus in the first and second part of this lab are: **Matrix Multiply**, **FFT**, **ADPCM**, **AES** and **JPEG** and on the third part we are getting acquaintance with the PARSEC suite and the benchmarks **CANNEAL** and **FLUIDANIMATE**.

2 Software for Low Power

2.1 Description and Goals

In the first part of the lab we were asked to perform Loop transformations and compile optimizations to 3 different applications in order to succeed lower power consumption, better execution time and cache performance.

We emphasized in 5 different transformations (Generic Loop, Permutation, Unrolling, Tilling and Fission/Fusion Loop).

2.2 Results

In the tables 1, 2 and 3, in the 4 first rows we see the results of the optimizations per application in the spectrum of power consumption, execution time, executed instructions and the cache performance due to miss rate. In the last row of the tables we see the results after no optimization(O0) but with the loop transformation.

Optimization	Power Consumption	# Executed Instructions	Execution Time	Miss Rate dl1 & dl2
O0	11,693,104.46	347,843	178,198	0.0067 0.7462
O1	5,572,379.69	140,519	84921	0.0311 0.6919
O2	4,307,992.76	116,495	65652	0.0312 0.6935
O3	4,152,673.52	110,356	63285	0.0312 0.6929
Loop Trans	11,313,107.11	320,628	172,407	0.0072 0.7386

Table 1: Matrix Multiply - Metrics

Optimization	Power Consumption	# Executed Instructions	Execution Time	Miss Rate
O0	2,891,680.76	65,980	44,068	0.0215 0.7845
O1	1,779,510.99	38,000	27,119	0.0653 0.7929
O2	1,741,320.96	36,830	26,537	0.0654 0.7980
O3	1,407,125.4	27,142	21,444	0.0829 0.7956
Loop Trans	2,825,775.08	64,844	43,978	0.0216 0.7889

Table 2: FFT - Metrics

Optimization	Power Consumption	# Executed Instructions	Execution Time	Miss Rate
O0	12,216,281.61	245,953	186,171	0.0073 0.1483
O1	4,893,835.68	123,825	74,580	0.0233 0.6286
O2	4,785,236.89	121,424	72,925	0.0232 0.6277
O3	4,740,681.85	120,874	72,246	0.0233 0.6286
Loop Trans				

Table 3: ADPCM - Metrics

In the Figures 1 and 2 we can see the results due to the above aspects for our 5 loop transformations.

2.3 Analysis

For the Matrix Multiply we choose the technique of unrolling for the first loop, so we can achieve the same result in a shortest time. We used unroll once. And then, in the main function we applied the tilling technique in the loop.

16x16	Power Consumption	# Ex Instructions	Execution Time	Cache Performance(Miss Rate)
Gen Loop	969449.2938	13243	14774	0,0832 0,7622
Perm Loop	971286.6148	13247	14802	0,0872 0,7624
Unroll Loop	950419.8979	12366	14484	0,0871 0,7635
Tilling Loop	1059281.1662	16190	16143	0,0673 0,7677
Fision Loop	1340588.1327	24272	20430	0,0568 0,7671

(a)

32x32	Power Consumption	# Ex Instructions	Execution Time	Cache Performance(Miss Rate)
Gen Loop	1489148.658	31339	22694	0,0509 0,7222
Perm Loop	1490985.979	31343	22722	0,0561 0,7224
Unroll Loop	1381927.8584	27774	21060	0,0561 0,7235
Tilling Loop	1634625.1088	38206	24911	0,0387 0,7271
Fision Loop	2878688.2712	74880	43870	0,0263 0,7268

(b)

64x64	Power Consumption	# Ex Instructions	Execution Time	Cache Performance(Miss Rate)
Gen Loop	3546160.7375	102974	54042	0,0302 0,6153
Perm Loop	3546160.7375	102951	54042	0,1594 0,1401
Unroll Loop	3086961.7285	88534	47044	0,0345 0,6188
Tilling Loop	3803845.0055	122182	57969	0,0226 0,6187
Fision Loop	9077021.8499	275940	138830	0,0216 0,4012

(c)

128x128	Power Consumption	# Ex Instructions	Execution Time	Cache Performance(Miss Rate)
Gen Loop	11684180.327	387123	178062	0,0220 0,5281
Perm Loop	11689692.29	387127	178146	0,1647 0,0833
Unroll Loop	981955.7137	329734	149652	0,0255 0,5325
Tilling Loop	12172382.7601	449756	185502	0,0166 0,5292
Fision Loop	33654208.4746	1077303	512876	0,0178 0,2855

(d)

Figure 1: Loop Transformations in cache
(a) 16x16 (b) 32x32 (c) 64x64 (d) 128x128

O1	Power Consumption	# Ex Instructions	Execution Time	Cache Performance(Miss Rate)
Gen Loop	936574.3719	15589	14273	0.1177 0,7215
Perm Loop	959409.647	16553	14621	0,1177 0,7218
Unroll Loop	908161.5152	13464	13464	0,1176 0,7222
Tilling Loop	1012363.8625	17461	15428	0,1177 0,7273
Fision Loop	1250559.4044	23962	23962	0.0824 0,7273

(a)

O2	Power Consumption	# Ex Instructions	Execution Time	Cache Performance(Miss Rate)
Gen Loop	884801.2913	12375	13484	0,1177 0,7204
Perm Loop	896350.166	14697	13660	0,1177 0,7218
Unroll Loop	857635.1882	12008	13070	0,1176 0,7216
Tilling Loop	972795.8427	16083	14825	0,1177 0,7276
Fision Loop	1130083.5427	21084	17222	0,0824 0,7255

(b)

O3	Power Consumption	# Ex Instructions	Execution Time	Cache Performance(Miss Rate)
Gen Loop	884801.2913	12375	13484	0,1177 0,7205
Perm Loop	896350.166	14697	13660	0,1177 0,7218
Unroll Loop	857635.1882	12008	13070	0,1176 0,7216
Tilling Loop	972795.8427	16083	14825	0,1177 0,7276
Fision Loop	1130083.6427	21084	17222	0,0824 0,7255

(c)

Figure 2: Optimization on the 32x32 cache
(a) O1 (b)O2 (c) O3

From the results we noticed that with these techniques we can achieve better results than the regular code.

For the FFT we choose the technique of unrolling for two loops, so we can achieve the same result in a shorter time. We used unroll three times in the loops in the main function.

From the results we noticed that with these techniques we could achieve better results than the regular code (O0) but not good enough to overcome O1 optimization.

For ADPCM code unfortunately, all of our tries to achieve something better than the original code was a dead end. Due to complexity and the amount of functions that the code provide the right profiling of that wasn't possible.

The way that we choose which loop transformation is better for which loop is due to the results on the Figures 1 and 2.

3 Low Power Hardware/Software Co-design

3.1 Description and Goals

In the second part of the lab, we have been asked to run 5 applications in 3 different ways. At first we have to run the application as a pure Software application, then as pure Hardware application and finally as a Hybrid application (some functions implemented in the software and some in hardware).

Our goal was to find the best implementation for each of the applications in the aspect of execution time, of area-delay and thermal power.

3.2 Results

In the tables 4, 5 6,7, 8 we see the results of the three types of executions per application.

Category	Pure SW	Hybrid	Pure HW
Clock Cycles	233247	112043	33243
Fmax (MHz)	71.88	73.38	112.75
Execution Time (us)	3,244.95	1,526.89	294,84
Speedup over SW	1	2.12	11
Area (ALMs)	4.375	5.060	375
Area-delay Product	14,196,655.88	7,726,050.42	110,564.30
Percentage vs SW (%)	1	0.54	0.01
Logic Utilization(%)	15	16	1
Total registers	5,556	6,236	599
Total RAM Blocks (%)	5	5	2
Total DSP Blocks (%)	7	9	2
Total thermal power (mW)	525.01	528.87	554.34
Power-Delay product	1,703,631.16	807,524.96	163,440.57
Percentage vs SW (%)	1	0.47	0.1

Table 4: Matrix Multiply - Metrics

3.3 Analysis

In some applications, in order to be able to recognize which function of their implementation takes the most time to execute, we used the profiling tool valgrind. Let see now one by one the applications.

3.3.1 Matrix Multiply

In the Matrix Multiply one, based to the results presented in the Table 4 **Pure Hardware** Implementation provides the best results. The execution time is 10 times faster, when the Area-delay product and the total thermal power needed is one hundred and ten times lower than the pure software one.

As far as the hybrid implementation we have better results than software but still not enough to overcome the hardware implement.

Category	Pure SW	Hybrid	Pure HW
Clock Cycles	31,526	19,654	2,647
Fmax (MHz)	71.88	74.42	99.43
Execution Time (us)	438.6	264.1	26.62
Speedup over SW	1	1.7	16.5
Area (ALMs)	4,735	11,665	1,815
Area-delay Product	2,076,733.58	3,080,676.03	48,318.47
Percentage vs SW (%)	1	1.48	0.02
Logic Utilization (%)	15	36	6
Total registers	5,556	13,559	2,990
Total RAM Blocks (%)	5	5	1
Total DSP Blocks (%)	7	16	9
Total thermal power (mW)	525.01	566.03	744.3
Power-Delay product	230,265.24	149,486.07	19,814.56
Percentage vs SW (%)	1	0.65	0.09

Table 5: FFT - Metrics

Category	Pure SW	Hybrid	Pure HW
Clock Cycles	192,575	101,576	16,417
Fmax (MHz)	71.88	67.74	87.57
Execution Time (us)	2,679.12	1,499.5	187.47
Speedup over SW	1	1.79	14.3
Area (ALMs)	4,735	11,406	7,186
Area-delay Product	12,685,623.61	17,103,275.11	1,347,180.11
Percentage vs SW (%)	1	1.35	0.10
Logic Utilization (%)	15	36	22
Total registers	5,556	14,443	9,179
Total RAM Blocks (%)	5	9	18
Total DSP Blocks (%)	7	100	100
Total thermal power (mW)	525.01	623.42	1,987.27
Power-Delay product	1,406,563.73	934,817.09	372,559.23
Percentage vs SW (%)	1	0.66	0.26

Table 6: ADPCM - Metrics

3.3.2 FFT

In the FFT one, based to the results presented in the Table 5 **Pure Hardware** Implementation provides again the best results. The execution time is more than 15 times faster, when the Area-delay product and the total thermal power needed is almost one hundred and ten times lower than the pure software one.

As far as the hybrid implementation we have better results than software but still not enough to overcome the hardware implement. Interesting is here the fact that in hybrid implementation we need almost the triple amount of ALMs as we want for the pure Software.

Category	Pure SW	Hybrid	Pure HW
Clock Cycles	63,381	62,953	9,059
Fmax (MHz)	71.88	70.23	67.89
Execution Time (us)	881.76	896.38	133.43
Speedup over SW	1	0.98	6.6
Area (ALMs)	4,735	5,187	5,655
Area-delay Product	4,175,139.61	4,649,540.24	754,583.08
Percentage vs SW (%)	1	1.11	0.18
Logic Utilization (%)	15	16	18
Total registers	5,556	6,397	9,124
Total RAM Blocks (%)	5	5	3
Total DSP Blocks (%)	7	9	0
Total thermal power (mW)	525.01	531.05	1,086.52
Power-Delay product	462,933.48	476,024.35	144,981.36
Percentage vs SW (%)	1	1.03	0.31

Table 7: AES - Metrics

Category	Pure SW	Hybrid	Pure HW
Clock Cycles	4,753,314	3,894,190	1,310,962
Fmax (MHz)	71.88	75.88	68.54
Execution Time (us)	66,128.46	51,320.37	19,126.96
Speedup over SW	1	1.29	3.46
Area (ALMs)	1,735	9,850	18,649
Area-delay Product	313,118,277.5	505,505,686.6	356,698,721
Percentage vs SW (%)	1	1.61	1.14
Logic Utilization (%)	15	24	58
Total registers	5,556	9,850	21,216
Total RAM Blocks (%)	5	5	30
Total DSP Blocks (%)	7	7	100
Total thermal power (mW)	525.01	551.59	551.59
Power-Delay product	34,718,104.94	28,307,805.25	10,550,241.17
Percentage vs SW (%)	1	0.81	0.30

Table 8: JPEG - Metrics

3.3.3 ADPCM

In the same aspect as the previous two applications, the best implementation for ADPCM is **Pure Hardware** (Table 6).The execution time is almost 15 times faster, when the Area-delay product and the total thermal power needed is almost one tenth and one fourth times lower than the pure software one.

As far as the hybrid implementation we have better results than software in execution time and power delay product but area delay relation is worst than pure's software ones.

As far of the total thermal power in the pure hardware we notice an almost 4 times more than software ones but the execution time is so small that the power delay relation

is much better than software's.

For hybrid implementation we took into hardware the function "adpcm_main". Due to profiling as we see in Figure 3 the functions that need the most execution time is **upzero, encode and decode**. Nontheless in the try of taking into hardware the three of them or two or either one of them the results that we have gather were much worst than the ones in the table 6.

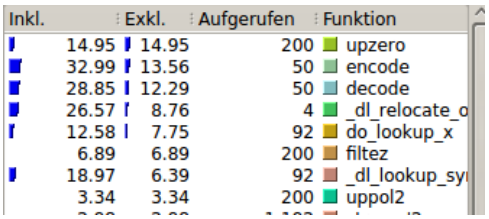


Figure 3: ADPCM Profiling

3.3.4 AES

In the AES, is the first time that we notice a hybrid implementation to be the worst in every aspect (Table 7). **Pure Hardware** once again is the best solution that we have. The execution time is 6 times faster, while the Area-delay product and the total thermal power needed is almost one fifth and one third times lower than the pure software one.

After the profiling of AES (Figure 4) we have gather the functions that involve more in the implementation of the application. We have taken into hardware the function **AddRoundKey** and the results in comparison of the software results are promising.

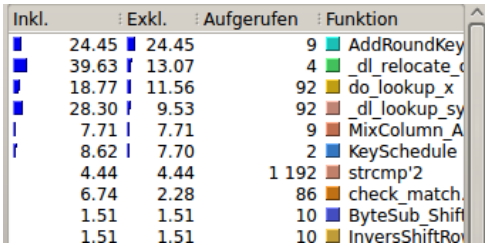


Figure 4: AES Profiling

3.3.5 JPEG

In the JPEG application for first time we notice a software only implementation to have better aspects than the hybrid or the hardware implementation in ALMs (Table 8).

After the profiling of JPEG (Figure 5) we have gather the functions that involve more in the implementation of the application. We have taken into hardware the function **DecodeHuffMCU** and the results in comparison of the software results are promising and better than when we took into hardware the function **YuvToRgb**.

As final thoughts, we have seen that both hybrid and hardware implementations are usually a better if not the best solution for our applications. Tthough we should always take consideration the finance aspect. In order to implement all of these function in hardware you will need more physical items e.g. registers, ALU etc. SOmething like that raise a lot the cost of the implementation.

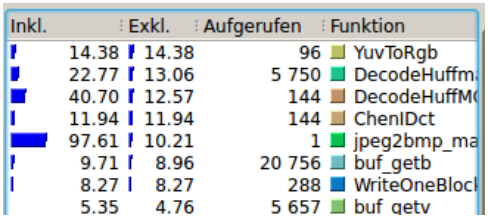


Figure 5: JPEG Profiling

4 Thermal Lab Experiment

4.1 Description and Goals

In the third part of the lab, we have been asked to observe the changes in temperature and execution time aspect of a processor, when 2 benchmarks are running on it, with different workloads.

For this experiment we choose the applications: CANNEAL and FLUIDANIMATE.

4.2 Results

We used the thermal camera software to get images from the temperature distribution of the processor during the running of the benchmarks. On the table 9, we present the average temperatures and time of the benchmark CANNEAL and on the figures 6,7,8 we see the visual representation of the heat on the processor. Correspondingly the table 10 and the figures 9,10,11 we present the results of the benchmark FLUIDANIMATE.

		Workload		Workload		Workload	
No.threads		**simmedium**	**time**	**simlarge**	**time**	**native**	**time**
	Temperature (°C)						
1	**High**	34.8		36.3			
	Low	24.5		25.6			
	Average	**29.65**	5.614 s	**30.95**	12.444 s		
2	**High**	35.1		37			
	Low	24.3		25.8			
	Average	**29.7**	4.708 s	**31.4**	10.369 s		
4	**High**	36.3		38.8		45.2	
	Low	24.4		25.8		28.5	
	Average	**30.35**	4.239 s	**32.3**	9.340 s	**36.8**	2m 17.450 s
8	**High**	37.1		41.5		51.8	
	Low	25.9		26		29.2	
	Average	**31.5**	4.073 s	**32.3**	9.340s	**36.8**	2m 17.450 s
16	**High**	36.4		40.8			
	Low	24.6		25.9			
	Average	**30.5**	4.107 s	**33.35**	8.979 s		
32	**High**	37.7		40.4		48.2	
	Low	24.8		26.2		29.5	
	Average	**31.25**	4.165 s	**33.3**	9.102 s	**38.85**	2m 0.486 s
128	**High**	36.1		38.8			
	Low	25		26.3			
	Average	**30.55**	4.488 s	**32.55**	9.733 s		

Table 9: CANNEAL - Metrics

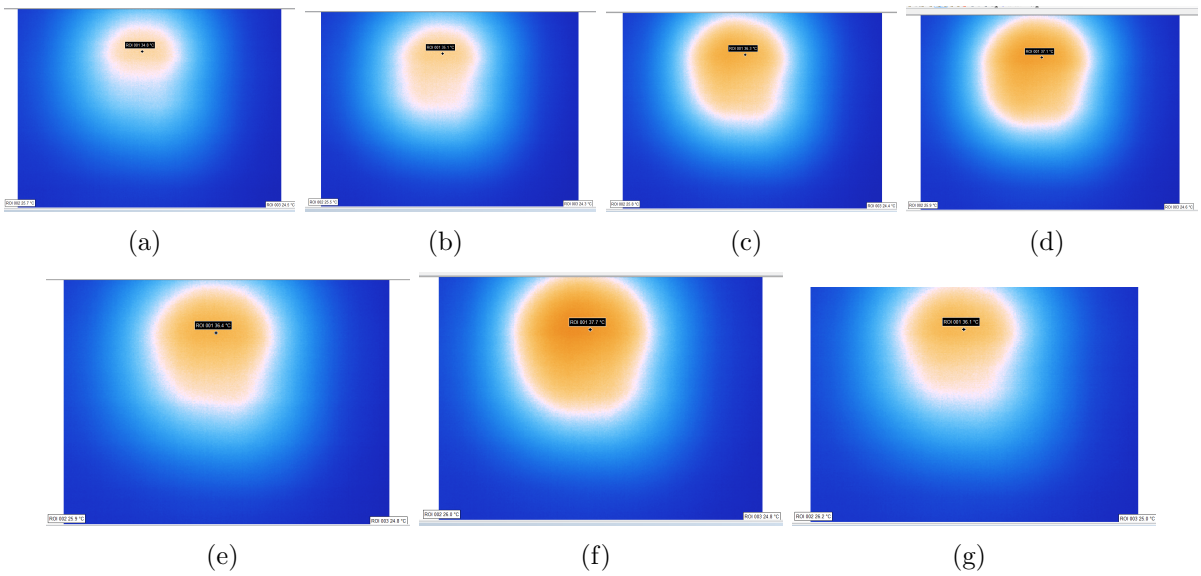


Figure 6: CANNEAL simmedium
(a) 1 thread (b) 2 threads (c) 4 threads (d) 8 threads
(e) 16 threads (f) 32 threads (g) 128 threads

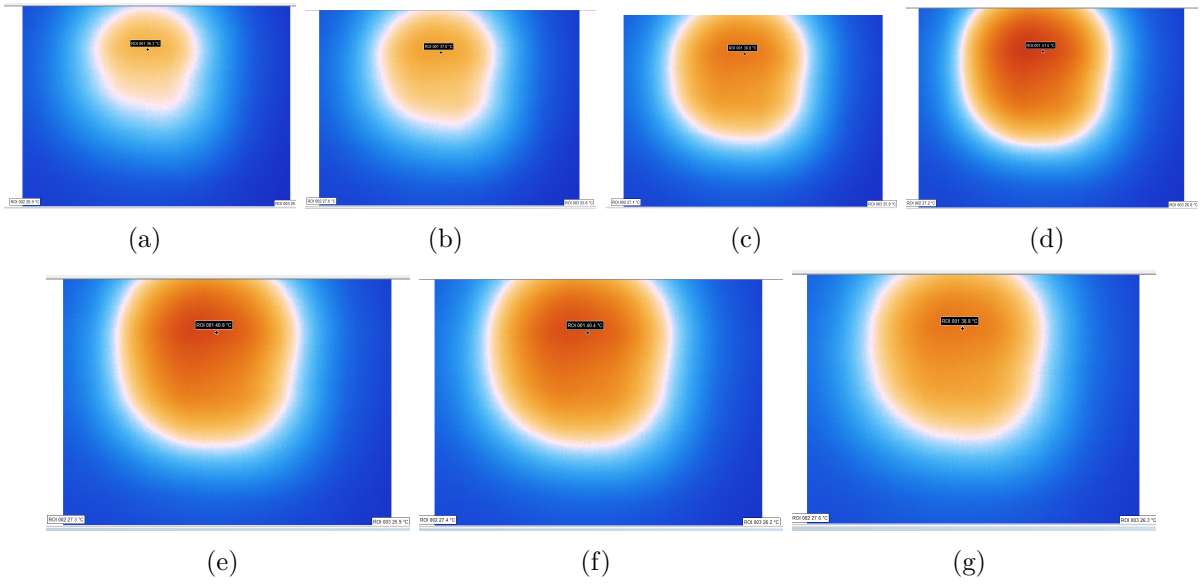
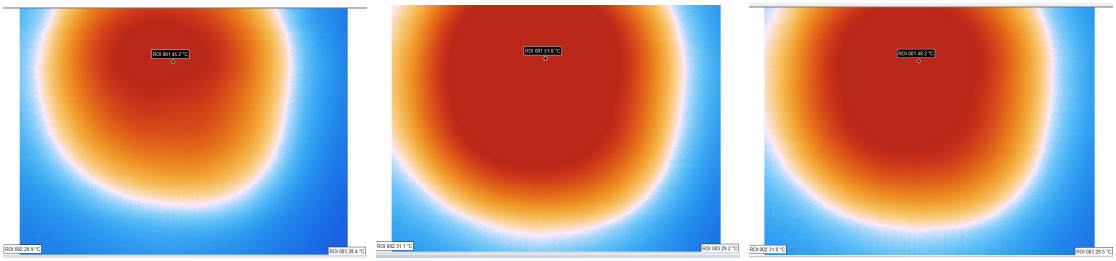


Figure 7: CANNEAL simlarge
(a) 1 thread (b) 2 threads (c) 4 threads (d) 8 threads
(e) 16 threads (f) 32 threads (g) 128 threads

4.3 Analysis

The system that we are working on has the characteristics that we seen in Figure 12. We saw that, we had an 8-core processor with 1 thread per processor (8 in total). So the results that we are expecting to present is an improvement in the timing as we add more threads (2,4,8) and some anomalies when we try to process in more than 8 threads, cause then 2 or more processes are trying to execute in the same core and we have collisions that lead to raise of the timing and the temperature.

Looking in the tables of the results, we can say that the expected results are happening. The timings in both benchmarks in every workload are decreasing as we go from 1 to 8 threads and then increasing as we are trying to add more threads than 8.

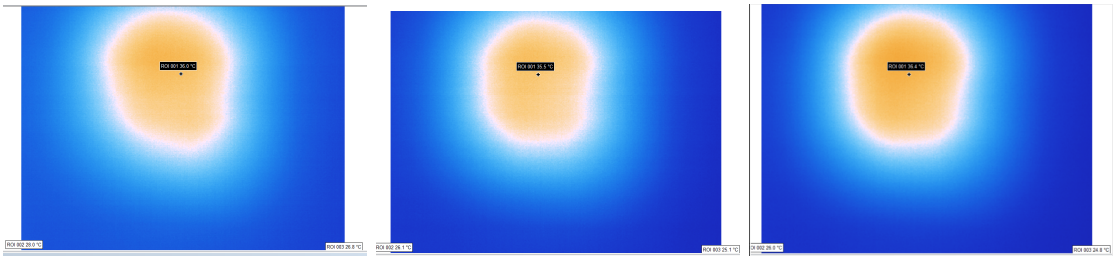


(a) (b) (c)

Figure 8: CANNEAL native
(a) 4 threads (b) 8 threads (c) 32 threads

		Workload		Workload		Workload	
No.threads		**simmedium**	**time**	**simlarge**	**time**	**native**	**time**
	Temperature (°C)						
1	**High**	36.6				38.9	
	Low	27.3				29.4	
	Average	**31.95**	2.210 s		6.753 s	**29.4**	16m 38.543 s
2	**High**	36		36.8		40	
	Low	26.8		26		29	
	Average	**31.4**	1.283 s	**31.4**	3.823 s	**34.5**	8 m 34.019 s
4	**High**	35.5		38.2		43.4	
	Low	25.1		25.6		29.3	
	Average	**30.3**	0.805	**31.4**	3.823 s	**36.35**	4m 30.984 s
8	**High**	36.4		39.9		56.6	
	Low	24.8		25.2		33.5	
	Average	**30.6**	0.620 s	**32.55**	1.664 s	**45.05**	2m 33.690 s
64	**High**					57.7	
	Low					35	
	Average					**46.35**	3m 1417s

Table 10: FLUIDANIMATE - Metrics



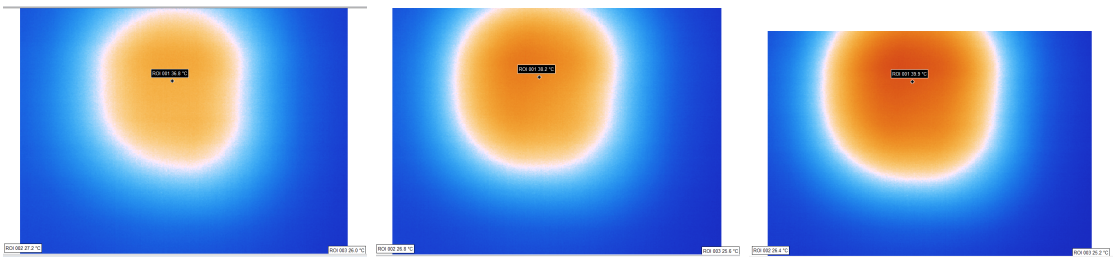
(a) (b) (c)

Figure 9: FLUIDANIMATE simmedium
(a) 2 threads (b) 4 threads (c) 8 threads

Interesting is that in the benchmark FLUIDANIMATE every workload the time fall is inversely proportional as the raise of the threads but if we take a look in the CANNEAL we only have a small improvement as we add more threads.

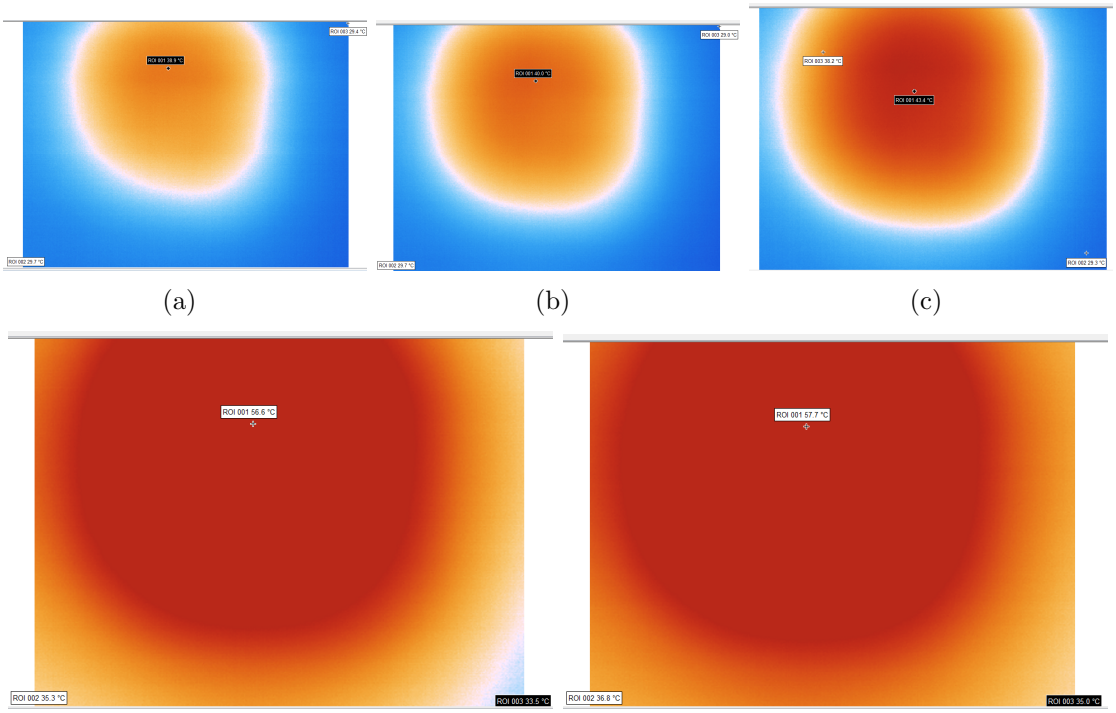
In the aspect of the temperature raise or falling, we can discuss that as we add more threads more process are paralyzed in result the temperature to rise. Really interesting is that from 1 thread of simmedium workload to 8 threads in native workload we have 10 °Craise.

So as engineers we should always take the right decision for our system. If we want the best timing we raise the amount of the threads but if we have a processor with not



(a) (b) (c)

Figure 10: FLUIDANIMATE simlarge
(a) 2 threads (b) 4 threads (c) 8 threads



(a) (b) (c)

(d) (e)

Figure 11: FLUIDANIMATE native
(a) 1 thread (b) 2 threads (c) 4 threads
(d) 8 threads (e) 64 threads

enough strength in the heat aspect we should not try to achieve the best time cause we are in dangerous spot of the processor to ”explode”.

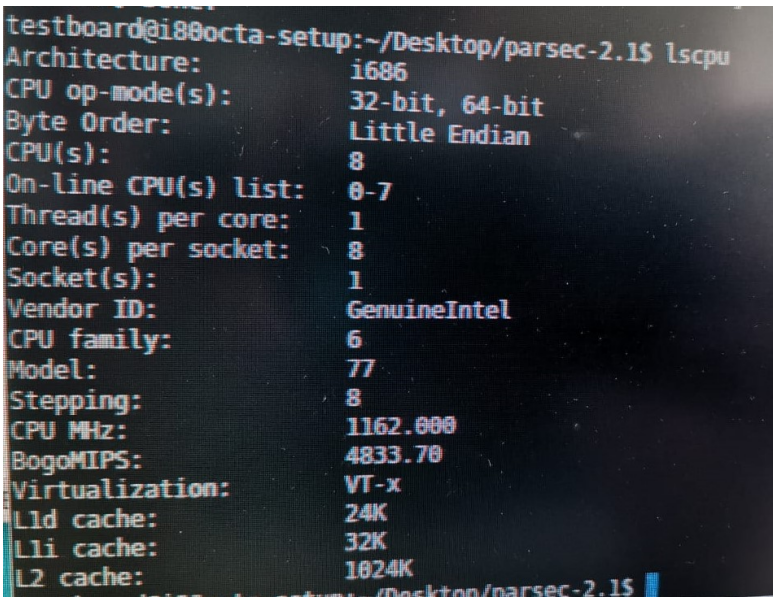


Figure 12: Processor's characteristics

5 Conclusions

To sum up, in these exercises of this lab we experienced the difference of software, hybrid and hardware implementation, how loop transformations can help to the cache performance as much as how temperature plays significant role in the life span and efficiency of an application/processor.

From the five looping techniques that we analyze we came to the conclusion that loop unrolling gives the best results in the most situations and the implementation of loops in that way can give results as good as after optimization.

For the dilemma between software or hardware implementation, we can answer that hybrid in the most cases can proceed with better results as the software ones and at the same time it offers a cheaper solution than pure hardware. Though, when the economic criteria is not in the way, choosing that technique it looks like one way road. We should always be careful though with the power consumption of the hardware implemented systems.

As we saw and in the lab no.3 as much we paralyze our code (while we are not using more than the available threads) the results of timing execution are getting better and better. Unfortunately that's something that is not true also for the temperature resistance of the system.