GCC vs. ICC comparison using PARSEC Benchmarks

Abedalmuhdi Almomany, Afnan Alquraan, Lakshmy Balachandran

Abstract--Our goal is to compare the impact of various compiler optimizations on program performance using two widely used state-of-the-art compiler suites: GNU C Compiler and Intel's C/C++ Compiler using PARSEC benchmarks. Compiler optimization is the process of tuning the output of a compiler to minimize or maximize some of the attributes of an executable computer program. Optimization of a compiler can be done by turning on optimization flags. In this paper, we investigate the chances of enhancing the program performance by better utilization of the existing architectural features such as compiler optimization. Proper utilization of such architectural features would not only enhance the program performance, but also reduce the need for costly upgrades as well as the system cost under development.

Index Terms -- compiler, icc, gcc, PARSEC.

I. INTRODUCTION

"Good, fast and cheap" is one of the major slogans for this century. This is applicable to each and every field we are dealing with. In the field of Microprocessors also, we could see that architectural researchers are immensely working to fulfill the same principle. This can be only achieved by increasing the performance of microprocessors. Hence architectural researches are mostly relying on improving the performance of microprocessors. Analysis of the performance improvement can be only done by comparing the novel with existing versions using widely accepted benchmarks. Basis of our study is a similar study [1] that conducted to compare performance of various optimizations using matrix multiplication program as the benchmark. As per our review, a detailed evaluation of the characteristics of gcc and icc compliers using PARSEC benchmark suites would certainly help the ongoing architectural researches. The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC was originally written as the compiler for the GNU operating system. GCC stands for "GNU Compiler Collection". The GNU system was developed to be 100% free software, free in the sense that it respects the user's freedom. Then Intel C++ Compiler, also known as icc , is a group of C and C++ compilers from Intel available for OS X, Linux, Windows and Intel-based Android devices.

We consider six binaries generated by gcc and icc compilers for the given analysis. Namely, we consider three binaries generated by gcc: (1) gcc –O0 assumes no optimization, (2) gcc –O2 assumes level 2 optimization, and (3) gcc –O3 assumes level 3, the highest optimization level. Similarly, we consider four binaries generated by icc: (4) icc – O0 assumes no optimization, (5) icc –O2 assumes level 2 optimization, (6) icc –O3 assumes level 3 optimization.

II. PARSEC BENCHMARK

The Princeton Application Repository for Shared-Memory Computers (PARSEC) is a benchmark suite composed of multithreaded programs. The suite focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors. The benchmark suite with all its applications and input sets is available as open source free of charge. Most of the workloads inside the benchmark suite were by researchers from Intel and Princeton University. Several other Benchmark suites including SPEC OMP2001, SPLASH-2, ALP Bench, BioParallel etc were widely used before the introduction of PARSEC benchmark suites but each one has its own limitations. The lack of good benchmark suites can adversely affect the parallel architecture research as well as reduce its impact. To address this problem, PARSEC benchmark suite was created with a goal to create a suite of emerging workloads that can drive CMP research. Due to the high importance of PARSEC benchmark suites in Computer Architectural field, a lot of researches are conducting in the areas of PARSEC benchmark suites with an intention to help the ongoing parallel architecture research.

III. BENCHMARKS

The current version of PARSEC suite contains the following 13 programs from many different areas such as computer vision, video encoding, financial analytics, animation physics and image processing. This will help to use the PARSEC as a benchmark suite for wide variety of applications.

Manuscript Received on December 2014.

Abedalmuhdi Almomany, A PHD candidate in the University of Alabama in Huntsville ,he has a Master Degree in Embedded Systems from Yarmouk University - Jordan.

Afnan Alquraan, has a bachelor Degree in Computer Engineering from the Yarmouk University- Jordan.

Lakshmy Balachandran, MSE in Computer Engineering, University of Alabama in Huntsville, USA.

Table: 1 PARSEC workloads

Programs	Application Domain	Parallelization
Blackscholes	Financial Analysis	Data-Parallel
Bodytrack	Computer Vision	Data-Parallel
Canneal	Engineering	Unstructured
Dedup	Enterprise Storage	Pipeline
Facesim	Animation	Data-Parallel
Ferrret	Similarity Search	Pipeline
Fluidanimate	Animation	Data-Parallel
Freqmine	Data Mining	Data-Parallel
Raytrace	Rendering	Data-Parallel
Streamcluster	Data Mining	Data-Parallel
Swaptions	Financial Analysis	Data-Parallel
Vips	Media Processing	Data-Parallel
X264	Media Processing	Data-Parallel

IV. EXPERIMENTAL SETUP ENVIRONMENTS

- CPU info:
  - processor : 0-7
  - vendor_id : GenuineIntel
  - CPU family : 6
  - model : 42
  - model name : Intel(R) Xeon(R) CPU E31270 @ 3.40GHz
  - stepping : 7
  - microcode : 0x23
  - CPU MHz : 1600.000
  - cache size : 8192 KB

- OS: Ubuntu 12.04
- Linux kernal: 3.2.0-60-generic
- gcc version: 4.6.3
- icc version: 12.1.5
- PARSEC version : 3.0
- Gcc Optimizations : gcc-O2,O3 (-funroll-loops -fprefetch-loop-arrays -fpermissive -fno-exceptions)
- Icc Optimizations : icc-O2,O3 (-funroll-loops -opt-prefetch -fpermissive -fno-exceptions)

V. IMPACT OF COMPILER OPTIMIZATIONS: A CASE STUDY USING PARSEC BENCHMARKS

Evaluating the impact compiler optimizations may have on benchmark program performance is a rather challenging task. A number of factors such as execution time, cache hierarchy, CPU utilization and their interactions may influence program performance. Here, we are performing a detailed study of different parameters that affect the performance of different compilers using PARSEC benchmarks.

VI. EXECUTION TIME FOR THE WORKLOADS USING GCC AND ICC

As mentioned the introduction, our plan is to determine program performance of the gcc and icc compilers when using common compiler optimizations. We consider six binaries generated by gcc and icc compilers for the analysis. Namely, we consider three binaries generated by gcc: (1) gcc –O0 assumes no optimization, (2) gcc –O2 assumes level 2 optimization, and (3) gcc –O3 assumes level 3, the highest optimization level. Similarly, we consider three binaries generated by icc: (4) icc – O0 assumes no optimization, (5) icc –O2 assumes level 2 optimization, (6) icc –O3 assumes level 3 optimization.

Note 1: We ran the benchmark program using different inputs such as simsmall, simlarge and native. We observed a proportional pattern in the generated output for all the above mentioned inputs. Hence we are using the output pattern of native to explain our results.

Note 2: We conducted our study for different number of threads namely 1, 2, 4 and 8. A detailed evaluation of the results for different number of threads shows that there is a proportional behavior in the output for these different numbers of threads. Hence we are explaining the pattern of '8' number of threads in detail.

Note 3: Both Note 1 and Note 2 are applicable to all the results that we are explain here.

Table: 2 Illustration - Execution Time of different threads for Blackscholes (native)

Number of Threads	gcc-00	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
1	197.53s	133.8230s	137.765s	175.435s	91.646s	91.743s
2	108.977s	76.481s	78.355s	97.827s	54.865s	54.707s
4	65.09s	47.887s	48.926s	59.084s	36.260s	36.224s
8	51.158s	37.591s	41.431s	47.040s	31.277s	31.214s

Table: 2 explains the execution time of blackscholes with native inputs for 1, 2, 4 and 8 threads. As per the table values, we can observe that for 2,4 and 8 threads execution time is less for icc-O3 optimization with values 54.707s, 36.224s, 31.214s respectively. For 1 thread, the execution time is less for icc-O2 but there is only a very small minimal different between the execution time of icc-O2 and icc-O3 here. Hence we can conclude that for Blacksholes, the execution time is less for icc optimization than gcc and there is only a small difference between icc-O2 and icc-O3.

Table: 3 Execution Time for the Programs (native-8 threads)

Program	gcc-00	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
Blackscholes	51.158s	37.591s	41.431s	47.04s	31.277s	31.21s
Bodytrack	156.47s	37.375s	36.089s	181.244s	29.657s	30.04s
Canneal	77.417s	76.831s	77.288s	86.234s	86.119s	87.96s
Facesim	88.362s	87.798s	88.7s	87.812s	87.715s	87.84s
Ferrret	169.205s	62.684s	70.577s	170.239s	63.251s	63.87s
Fluidanimate	374.82s	75.046s	75.564s	402.196s	76.319s	77.97s
Freqmine	224.665s	85.917s	84.244s	239.321s	87.671s	89.11s
Raytrace	341.335s	253.986s	87.156s	383.028s	69.929s	67.6s
Streamcluster	235.31s	82.572s	83.486s	222.877s	78.485s	78.75s
Swaptions	99.828s	43.122s	42.999s	96.187s	31.034s	31.04s
Vips	22.034s	21.771s	21.257s	24.113s	21.315s	63.84s
X264	32.357s	23.351s	23.88s	36.713s	25.128s	23.21s
Total Execution Time	**1872.961s**	**888.044s**	**732.671s**	**1977.004s**	**687.9S**	**732.4s**

Table: 3 explains the execution time for the all workloads in PARSEC benchmarks (except Dedup) for different optimizations. For example, we can see for workload Bodytrack, the minimum execution time is for icc-O2 and maximum is for gcc-O0. Also, for X264 the minimum execution time is for icc-O3 and maximum is for gcc-O0. The last row of Table: 3 is the total execution time of all the workloads (we have not considered Dedup workload here) in PARSEC benchmarks. The result shows that icc-O2 has least overall execution time compared to all other optimizations we have considered. Speed up could be calculated from the execution time. The Table.4 below shows the speed up factor of each benchmark for different number of optimizations. The speedup factor is calculated with reference to gcc-O0 of one thread.

Speed Up = (Execution time of gcc-O0) / (Execution time of gcc/icc optimized).

Table: 4 Speed-Ups for different optimizations normalized to the gcc-O0 of 1 thread (native)

Program	1 Thread –Speed UP						2 Threads –Speed UP					
	gcc-O0	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3	gcc-O0	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
Blackscholes	1.00	1.48	1.48	1.13	2.16	3.61	1.81	2.58	2.52	2.02	3.60	3.61
Bodytrack	1.00	4.61	4.61	0.86	5.80	10.26	1.86	8.08	8.15	1.86	10.25	10.26
Canneal	1.00	1.01	1.01	0.89	0.89	1.42	1.61	1.62	1.61	1.33	1.44	1.42
Facesim	1.00	1.00	1.00	1.00	1.01	1.86	1.85	1.86	1.86	1.85	1.89	1.86
Ferrret	1.00	2.37	2.37	1.00	2.36	4.53	1.90	4.52	4.11	1.91	4.50	4.53
Fluidanimate	1.00	6.65	6.65	0.99	6.06	11.14	1.91	12.05	11.96	1.90	10.87	11.14
Freqmine	1.00	2.70	2.70	0.91	2.57	4.92	1.92	5.23	5.30	1.77	4.93	4.92
Raytrace	1.00	0.52	0.52	0.84	0.06	8.22	1.31	0.94	4.26	1.20	8.02	8.22
Streamcluster	1.00	2.71	2.71	0.99	5.08	8.93	1.95	5.03	4.90	2.01	8.96	8.93
Swaptions	1.00	2.16	2.16	1.05	3.10	6.09	1.95	4.19	4.26	2.04	6.05	6.09
Vips	1.00	0.97	0.97	1.06	1.06	0.71	1.94	1.87	1.93	2.05	2.07	0.71
X264	1.00	1.13	1.13	0.61	1.12	2.80	1.95	2.93	2.90	1.62	2.62	2.80
	4 Threads -Speed UP						8 Threads –Speed UP					
Blackscholes	3.03	4.12	4.04	3.34	5.47	5.45	3.86	5.25	4.77	4.20	6.32	6.33
Bodytrack	3.21	12.30	12.29	2.83	16.08	15.70	3.93	16.46	17.05	3.39	20.75	20.48
Canneal	2.34	2.35	2.36	2.09	2.08	2.05	2.72	2.74	2.72	2.44	2.44	2.39
Facesim	3.08	3.10	3.09	3.09	3.08	3.10	3.49	3.52	3.48	3.52	3.52	3.51
Ferrret	3.13	7.85	7.07	3.10	7.79	7.72	3.51	9.48	8.42	3.49	9.40	9.31
Fluidanimate	3.54	20.53	20.32	3.53	18.91	19.01	4.67	23.32	23.16	4.35	22.93	22.44
Freqmine	3.67	9.73	9.93	3.35	9.20	9.23	4.65	12.15	12.39	4.36	11.91	11.72
Raytrace	1.55	1.55	5.88	1.40	8.01	8.23	1.64	2.20	6.42	1.46	8.01	8.28
Streamcluster	3.65	8.77	8.62	3.75	12.80	12.87	4.10	11.70	11.57	4.33	12.30	12.26
Swaptions	3.69	7.89	8.02	3.87	11.45	11.57	4.49	10.39	10.42	4.66	14.44	14.43
Vips	3.65	3.29	3.66	3.44	4.02	1.22	4.25	4.30	4.41	3.89	4.40	1.47
X264	3.08	5.04	4.90	2.68	4.09	4.50	3.90	5.41	5.29	3.44	5.02	5.44

From the Table. 4, we could see that for Blacksholes the speed up is maximum for icc-O3 and minimum for gcc-O0. Similarly for streamcluster the speed up of icc-O2 is 12.30 times gcc- without any optimization.

VII. COMPARISON OF CACHE MISSES OF WORKLOADS FOR DIFFERENT COMPILER OPTIMIZATIONS

We used Perf Tool to analyze the factors like cache misses, page fault, CPU utilized etc. The cache misses counted by the perf tool is the number of memory access that cannot be served by any level of caches. Table: 5 shows the cache misses of different threads for Blackscholes (native).

Table: 5 Illustration – Cache Misses of different threads for Blackscholes (native)

Number of Threads	gcc-00	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
1	36398320	34228256	34624479	37894909	34228256	159124
2	35745956	33577988	34813256	37562361	33577988	30657717
4	40393484	36345194	40919286	39323089	36345194	30499104
8	37972212	32955782	92835740	36733608	32209138	31194108

Table: 6 Cache Misses for different optimizations normalized to the gcc-O0 (native – 8 threads)

Programs	gcc-O0	gccc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
Blackscholes	1.000	0.868	2.445	0.967	0.848	0.821
Bodytrack	1.000	0.992	0.935	0.991	1.002	0.971
Canneal	1.000	0.958	0.958	0.962	0.963	1.001
Facesim	1.000	0.992	0.989	0.989	0.987	0.988
Ferrret	1.000	0.777	0.771	0.966	0.989	1.021
Fluidanimate	1.000	1.064	1.080	0.975	1.170	1.175
Freqmine	1.000	1.043	1.038	0.996	1.026	1.034
Raytrace	1.000	6.570	7.522	0.956	0.869	0.800
Streamcluster	1.000	0.966	0.970	1.002	1.016	1.018
Swaptions	1.000	0.942	1.205	1.566	0.738	0.695
Vips	1.000	0.959	0.968	0.926	0.905	0.911
X264	1.000	0.972	0.978	0.939	0.962	0.958

From Table: 6 we can understand that which optimization has least number of cache misses compared to others. For instance, workload Bodytrack has least number of cache misses in gcc-O3 while for Raytrace the cache misses are minimal for icc-O3.

VIII. COMPARISON OF NUMBER OF CYCLES IN THE WORKLOADS FOR DIFFERENT COMPILER OPTIMIZATIONS

The performance of a compiler also depends upon the number of cycles used for running the program. Table:8 shows the Number of Cycles for different optimizations normalized to the gcc -O0 (native – 8 threads).

Table: 7 Illustration – Number of Cycles of different threads for Blackscholes (native – normalized to the gcc-O0)

Number of Threads	gcc-00	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
1	1.000	0.677	0.695	0.889	0.464	0.464
2	1.000	0.679	0.696	0.890	0.465	0.465
4	1.000	0.686	0.698	0.889	0.465	0.465
8	1.000	0.648	0.645	0.903	0.481	0.481

Table: 8 Number of Cycles for different optimizations normalized to the gcc -O0 (native – 8 threads)

Program	gcc-00	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
Blackscholes	1.000	0.648	0.646	0.904	0.481	0.481
Bodytrack	1.000	0.203	0.200	1.210	0.167	0.167
Canneal	1.000	0.996	0.995	1.092	1.090	1.103

Facesim	1.000	1.007	0.995	1.004	1.003	1.004
Ferrret	1.000	0.370	0.417	1.006	0.375	0.377
Fluidanimate	1.000	0.202	0.203	1.081	0.205	0.209
Freqmine	1.000	0.380	0.371	1.068	0.388	0.395
Raytrace	1.000	1.748	0.404	1.060	0.097	0.095
Streamcluster	1.000	0.351	0.351	0.951	0.336	0.337
Swaptions	1.000	1.004	1.001	2.236	0.722	0.723
Vips	1.000	0.998	0.998	0.923	0.922	0.923
X264	1.000	0.695	0.697	1.126	0.685	0.686

The overall numbers of cycles are considered as the total number of cycles for execution of all the programs (except Dedup) in the PARSEC benchmarks. The overall number of cycles is less for icc-O2 and high for icc-O0 .

IX. COMPARISON OF AMOUNT OF PAGE FAULTS OBSERVED IN THE WORKLOADS FOR DIFFERENT COMPILER OPTIMIZATIONS.

Using perf tool we identified the amount of page faults in different workloads for different optimizations.

Table: 10 illustrate the page faults for different optimizations normalized to the gcc-O0 (native – 8 threads). As per the analysis the Overall page faults is less for icc-O3 and high for gcc-O3.

Table: 9 Illustration – Page Faults of different threads for Blackscholes (native - normalized to the gcc-O0)

Number of Threads	gcc-00	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
1	1	0.99998	0.99998	1.00069	1.00070	1.00069
2	1	0.999987	0.99994	1.000658	1.00066	1.000670
4	1	1.00001	1.00002	1.00067	1.00068	1.00068
8	1	0.9999	1.00012	1.00058	1.00062	1.00063

Table: 10 Page Faults for different optimizations normalized to the gcc-O0 (native – 8 threads)

Benchmark	gccO0	gcccO2	gccO3	iccO0	iccO2	iccO3
Blackscholes	1.000	1.000	1.000	1.001	1.001	1.001
Bodytrack	1.000	0.998	1.013	0.989	0.993	0.997
Canneal	1.000	1.000	1.000	1.000	1.000	1.000
Facesim	1.000	1.000	1.000	1.000	1.000	1.000
Ferrret	1.000	1.004	1.022	1.207	1.287	0.948
Fluidanimate	1.000	1.005	0.995	1.003	1.000	1.000
Freqmine	1.000	0.999	0.995	0.963	0.944	0.979
Raytrace	1.000	0.999	1.001	1.000	0.995	0.995
Streamcluster	1.000	1.000	1.000	1.000	1.000	1.000
Swaptions	1.000	0.974	1.042	0.962	0.965	0.972
Vips	1.000	1.002	1.140	1.032	1.032	1.029
X264	1.000	1.000	1.000	1.000	1.000	1.000

X. COMPARISON OF AMOUNT OF CPU UTILIZATIONS OF WORKLOADS FOR DIFFERENT COMPILER OPTIMIZATIONS.

Table: 11 CPU Utilization for different optimizations normalized to the gcc-O0(native – 8 threads)

Program	gcc-00	gcc-O2	gcc-O3	icc-O0	icc-O2	icc-O3
Blackscholes	1.000	0.854	0.782	0.977	0.771	0.465
Bodytrack	1.000	0.907	0.909	1.000	0.936	0.936
Canneal	1.000	1.000	1.015	0.995	0.995	0.988
Facesim	1.000	0.999	0.999	0.999	1.000	1.000
Ferrret	1.000	0.998	0.998	0.882	0.999	0.999
Fluidanimate	1.000	1.000	1.008	1.000	1.000	1.003
Freqmine	1.000	0.991	0.988	1.000	0.994	0.991

Raytrace	1.000	1.000	1.592	0.941	0.452	0.452
Streamcluster	1.000	0.996	0.986	1.000	1.004	1.004
Swaptions	1.000	1.000	1.002	1.001	1.002	1.000
Vips	1.000	0.980	1.000	0.947	0.898	0.957
X264	1.000	0.997	0.995	0.991	0.960	0.979

As per our analysis, CPU utilization is maximum for gccO0 and minimum for icc-O3. This strengthens the fact that, the best optimization will have the less amount of CPU utilization.

XI. COMPARISON OF COMPILE TIMES OF GCC AND ICC.

We also observed the time taken to compile each workload in gcc and icc and calculated the overall compile time for all the workloads using both. We have observed that the overall compile time is less for gcc and high for icc.

Table: 12 Compilation Time of gcc v/s icc

Program	gcc	Icc
Blackscholes	0.759	1.160
Bodytrack	35.198	49.970
Canneal	1.721	1.126
Dedup	0.574	0.973
Faceim	125.177	97.144
Ferrret	208.902	313.661
Fluidanimate	1.960	2.864
Freqmine	3.545	4.594
Raytrace	464.526	560.357
Treamcluter	1.7103s	2.202
Swaptions	2.317	3.419
Vips	232.099	400.576
X264	41.739	73.485
Total Compilation Time in seconds	**1120.215**	**1511.531**

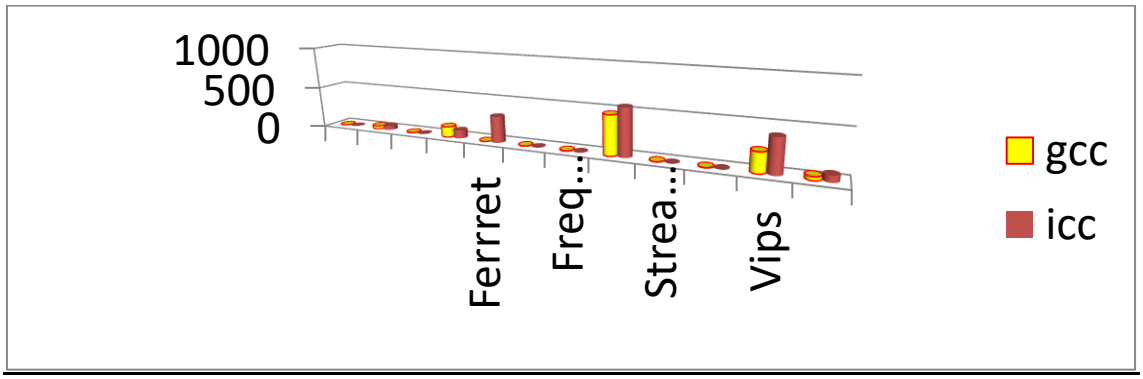

Figure: 1 Compilation Time of gcc v/s icc

XII. CONCLUSION

In this project we analyzed the impact of various compiler optimizations on program performance using two widely used state-of-the-art compiler suites: GNU C Compiler and Intel's C/C++ Compiler using PARSEC benchmark suites. The results indicate that Intel's compiler with optimization (O2, O3) in general outperforms GNU C compiler in almost all the parameters we observed. Speed up is the factor we primarily taken into account. We have observed that the overall speed up is best for icc-02 for 1, 2 4 and 8 threads. The overall speed up with icc-O2 is almost 3 times compared to the speed up of gcc without any optimization.

Table: 13 Total Speed-Ups all the programs normalized to the gcc-O0 (native).

Number of Threads (native)	Best for /Speed Up	Worst for /Speed UP
1	icc-O2 / 3.01	icc-O0/ . 95
2	icc-O2 / 3.27	icc-O0/ .974
4	icc-O2 / 2.8	icc-O0/ .95
8	icc-O2 / 2.72	icc-O0/ .94

Also, the experimental results help to identify the best and worse optimizations for each parameter we have observed. Table: 14 shows the best and worse optimizations for different parameters for the overall PARSEC benchmark suite

.

Table:14 Best and worse optimization for different parameters

Parameters	Best	Worse
Over-all Speed up	icc -O2	icc -O0
Over-all Cache Misses	icc-O3	gcc-O3
Over-all Number of Cycles	icc-O2	icc-O0
Over-all page faults	icc-O3	gcc-O3
Total Compilation Time	gcc	icc

ACKNOWLEDGEMENT

We would like to thank Dr. Aleksandar Milenkovic in University of Alabama in Huntsville, USA for giving complete support and insightful comments about the study.

REFERENCES

1. Compiler Optimizations in Multi-core Era: A Performance Study Using Intel C++/Fortran and GNU C++/Gfortran , Aleksandar Milenkovic LaCASA Laboratory, Electrical and Computer Engineering The University of Alabama in Huntsville.
2. Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh and Kai Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, October 2008.
3. Christian Bienia, Sanjeev Kumar and Kai Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. In Proceedings of the 2008 Annual IEEE International Symposium on Workload Characterization, September 2008.
4. http://parsec.cs.princeton.edu/download/tutorial/3.0/parsec-tutorial.pdf
5. J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach. San Francisco: Morgan Kaufmann, 2007.

AUTHORS PROFILE

Abedalmuhdi Almomany, A PHD candidate in the University of Alabama in Huntsville ,he has a Master Degree in Embedded Systems from Yarmouk University - Jordan. Has a published paper “Real time Radio Frequency Identification Vehicles Data logger Traffic Management System”. Current working focus on parallel programming using CUDA and OpenCL and parallel processing.

Afnan Alquraan, has a bachelor degree in computer engineering from the Yarmouk University- Jordan, her working focus on programming languages and networking.

Lakshmy Balachandran, MSE in Computer Engineering, University of Alabama in Huntsville, USA. Major research work includes Data Aggregation in Wireless Sensor Network (WSN) using STDCA method, Application of Digital Image Processing in Drug Industry, MIL-STD 1553B ENCODER AND DECODER..