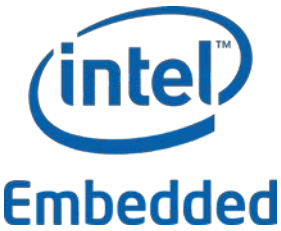


White Paper

Vinodh Gopal
Jim Guilford
Chengda Yang
Wajdi Feghali
Erdinc Ozturk
Gil Wolrich
Kirk Yap
Martin Dixon
IA Architects

Intel Corporation

High Performance DEFLATE Decompression on Intel® Architecture Processors

November 2010

324704

Executive Summary

There is a critical need for lossless data compression in enterprise storage and applications such as Databases, which process huge amounts of data. DEFLATE is a widely used standard to perform lossless compression, and forms the basis of utilities such as gzip and libraries such as Zlib. In these applications, decompression imposes a large computational burden on the servers, and they could benefit from a highly optimized implementation. This paper describes the performance characteristics of an optimized implementation of DEFLATE decompression, on Intel® processors based on the 32nm microarchitecture. As the performance of decompression is data dependent, we report the performance on various industry standard corpora data sets.

This paper describes the performance characteristics of an optimized prototype implementation of DEFLATE decompression. In terms of throughput, we are able to perform DEFLATE decompression at the aggregate rate of ~**4.5 Gigabits/sec** on the Calgary Corpus data-set, on a **single core** of an Intel® Core™ i5 650 processor, with Intel® Hyper-Threading Technology[1].

[1] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to the Performance Section on page 7. For more information go to http://www.intel.com/performance

*Our optimized decompression implementation is ~**1.75 times** as fast as the best open source version of Zlib decompression, on the Intel® Core™ i5 650 processor.* [2]

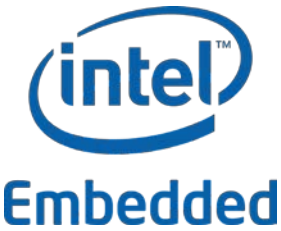
The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. www.intel.com/embedded/edc.

---

[2] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 6. For more information go to http://www.intel.com/performance.

Contents



Overview

There is a critical need for lossless data compression in enterprise storage and applications such as databases, which process huge amounts of data. DEFLATE [1] is a widely used standard to perform lossless compression, and forms the basis of higher level specifications such as gzip [5] and Zlib [6]. In these applications, decompression imposes a large computational burden on the servers, and they could benefit from a highly optimized implementation.

This paper describes the performance characteristics of an optimized implementation of DEFLATE decompression, on Intel® processors based on the 32nm microarchitecture. As the performance of decompression is data dependent, we report the performance on various industry standard corpora data sets such as the Calgary [7], Canterbury [8] and Silesia Corpus [9].

DEFLATE Decompression

Background

The DEFLATE compressed data format consists of a series of blocks, corresponding to successive blocks of input data. Each block is compressed using a combination of the LZ77 [3] algorithm and Huffman coding [2]. The LZ77 algorithm finds repeated substrings and replaces them with backward references (relative distance offsets). The Huffman trees for each block are independent; the LZ77 algorithm can use a reference to a duplicated string occurring in the same or previous blocks, up to 32K input bytes back.

A compressed block can have either static (fixed codes defined in the standard) or dynamic Huffman codes. Each dynamic block consists of two parts: a pair of Huffman code trees that describe the representation of the compressed data part, and the compressed payload. The compressed data consists of a series of elements of two types: literal bytes and pointers to replicated strings, where a pointer is represented as a pair <length, backward distance>. The DEFLATE format limits distances to 32K bytes and lengths to 258 bytes, with a minimum length of 3 bytes.

Each type of token or value (literals, distances, and lengths) in the compressed data is represented using a Huffman code, using one code tree for literals and lengths (LL-tree) and a separate code tree for distances (D-tree).

The decompression procedure

The decompression procedure needs to first process the tree information at the start of the block (for dynamic Huffman codes) and create efficient data structures such as lookup tables, to perform the decoding of symbols in the block.

Once the lookup tables have been constructed, the procedure to decompress a block consists of iterating the following basic steps:

1. Read some number of bits from the input-stream to create an index for lookup into the LL-Table structure
2. From the information returned, we find the length of the variable symbol and advance the bit stream accordingly. We determine whether the symbol is a literal, length or end-of-block symbol. If we find a literal symbol, we output that literal to the output stream and repeat step1.
3. If the symbol is a length, we need to compute the effective length based on an optional number of variable extra bits from the input stream.
4. After length L has been processed, read some number of bits from the input-stream to create an index for lookup into the D-Table structure
5. From the information returned, we find the length of the variable symbol and advance the bit stream accordingly. We need to compute the effective distance based on an optional number of variable extra bits from the input stream.
6. After distance D has been processed, we copy a substring of length L from a location in the history buffer, which is D positions earlier in the byte-stream already decompressed so far, to the current output buffer. Repeat step1.

Our Implementation

We developed a highly optimized prototype implementation of DEFLATE decompression to work efficiently on data buffers of all types and sizes. We refer to this implementation as "**igunzip**". It is possible to get further performance gains for specific usages that target very small data buffers, work only on fixed sized buffers or have very different compression ratios. Such optimizations have not been attempted and are beyond the scope of the current paper.

We optimized the main loop described in the above section. Most of our optimizations were in the structure of the copy code, and the Huffman decode code flow. Although the flow is sequential in nature, there are different ways to structure the lookup table data structures (compared to the Zlib method) in terms of the size of the index and the layout/encoding of the fields in each entry. We can also manage the history buffer in various ways for efficient copying of substrings, given the maximum size of 32 Kbytes of history, the

size of the first level data cache, the size of other data structures and managing the input/output stream buffers.

In the gzip format, we also need to compute a 32-bit CRC of the uncompressed buffer. We do this efficiently using the PCLMULQDQ instruction as described in [4]. The CRC computation is a small proportion of the overall decompression compute time.

Performance

The performance results provided in this section were measured on an Intel® Core™ i5 650 processor at a frequency of 3.20 GHz, supporting Intel® PCLMULQDQ instruction. The tests were run with Intel® Turbo Boost Technology off, and represent the performance with and without Intel® Hyper-Threading Technology (Intel® HT Technology) on a **single core**. We present the results with data in memory, excluding file I/O, to reflect a variety of usage models.

Since the decompression performance depends on the type and size of the compressed data, we study the performance on a set of industry-standard corpus data sets. We perform 2 tests for each data file, where we compress it with the lowest compression setting (Zlib -1) and then with the highest (Zlib -9). This method allows us to capture the sensitivities of the decompression speed to the actual method used for compressing a given file.

For a fair comparison of the performance of our implementation "**igunzip**" to Zlib, we compiled the Zlib decompressor with 2 different compilers, ICC and Microsoft* Compiler with aggressive optimizations, and picked the better performing version for Zlib. The ICC Compiler generated the faster code for Zlib. Our implementation is largely in optimized assembly code and thus less sensitive to compilers. We used version 1.2.5 of Zlib and generated 64-bit code. We built Zlib to generate 64-bit code with assembly support (i.e. we built it in such a way as to include the asm modules that are present in the zlib distribution).

The performance results of the decompressors thus represent all compute elements such as table build-up, the actual process of decompressing the symbols, CRC, excluding File IO time.

Methodology

To measure the performance of a decompressor on a given compressed file, we read the file into a large memory buffer, and then run a large warm-up sequence of unrelated computations to ensure that the processor core clock frequency has reached the rated frequency and is not low due to power-states. At this point, we repeatedly call the decompression function to decompress this memory buffer and generate the original content in another memory output buffer. We measure the cycles consumed by each function

call, sort them, discard the top and bottom quartiles and average the rest. This average gives a very stable performance number for the given compressed file. This procedure is then iterated over all the compressed files in the corpora. This gives us the performance of the decompressor over all files. The same methodology is followed for the other decompressor.

The files in the data-set range from a few Kbytes to several Mbytes in uncompressed size. As the amount of memory required to process the larger files is significantly larger than the size of the last-level Cache, we expect to see cache misses to memory. A large number of the files have sizes that will fit in the last-level cache, but not entirely in the lower levels. As a result, we expect to see a mix of warm and cold data in these performance tests.

The timing is measured using the **rdtsc**() function which returns the processor time stamp counter (TSC). The TSC is the number of clock cycles since the last reset. The 'TSC_initial' is the TSC recorded before the function is called. After the function is complete, the **rdtsc**() is called again to record the new cycle count 'TSC_final'. The effective cycle count for the called routine is computed using

# of cycles = (TSC_final-TSC_initial).

A large number of such measurements are made for the current file and then averaged as described above.

When two such identical threads are run simultaneously, the number of cycles measured for the function represents the cycles consumed in the core for 2 data buffers (from each thread) and we calculate the cycles/byte accordingly. Thus, if each thread was decompressing buffers of size 1KB, then the net cycles/byte = # of cycles / 2*1KB

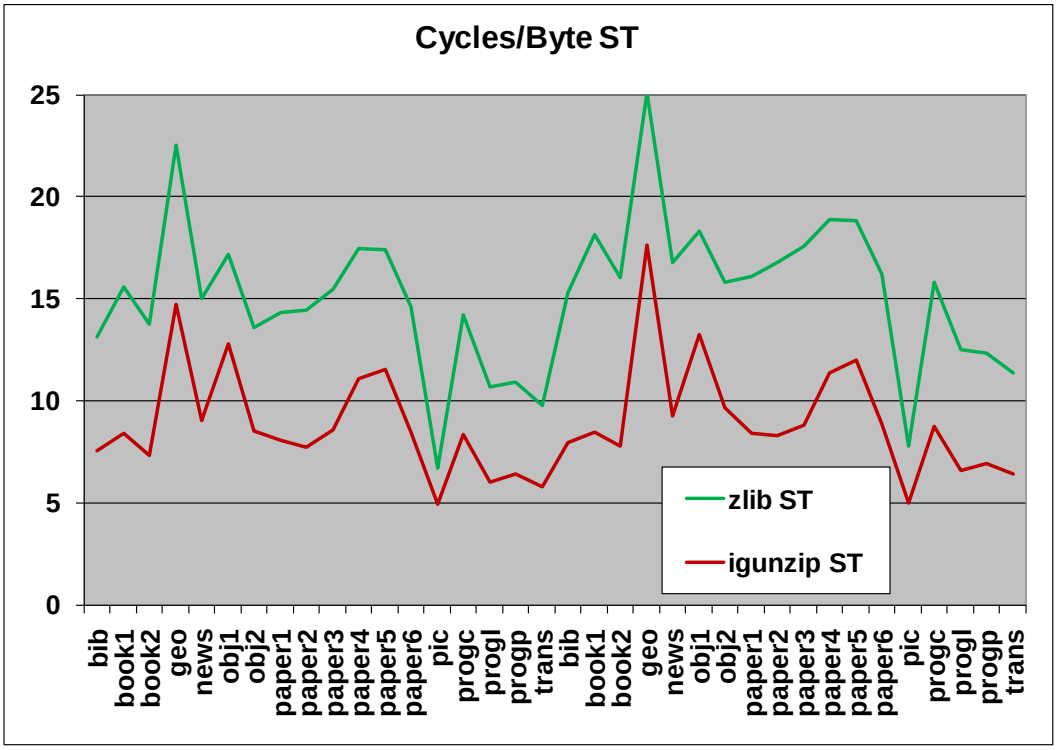
Note: Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 7. For more information go to http://www.intel.com/performance.

Results

When we compared the decompression performance across the various corpora, we found that they were close on the average, for a given decompressor. We therefore discuss the results on the Calgary Corpus in detail here, and provide the extended performance data in the Appendix. We show performance with a single thread (ST) as well as with Intel® Hyper-Threading Technology (Intel® HT Technology).

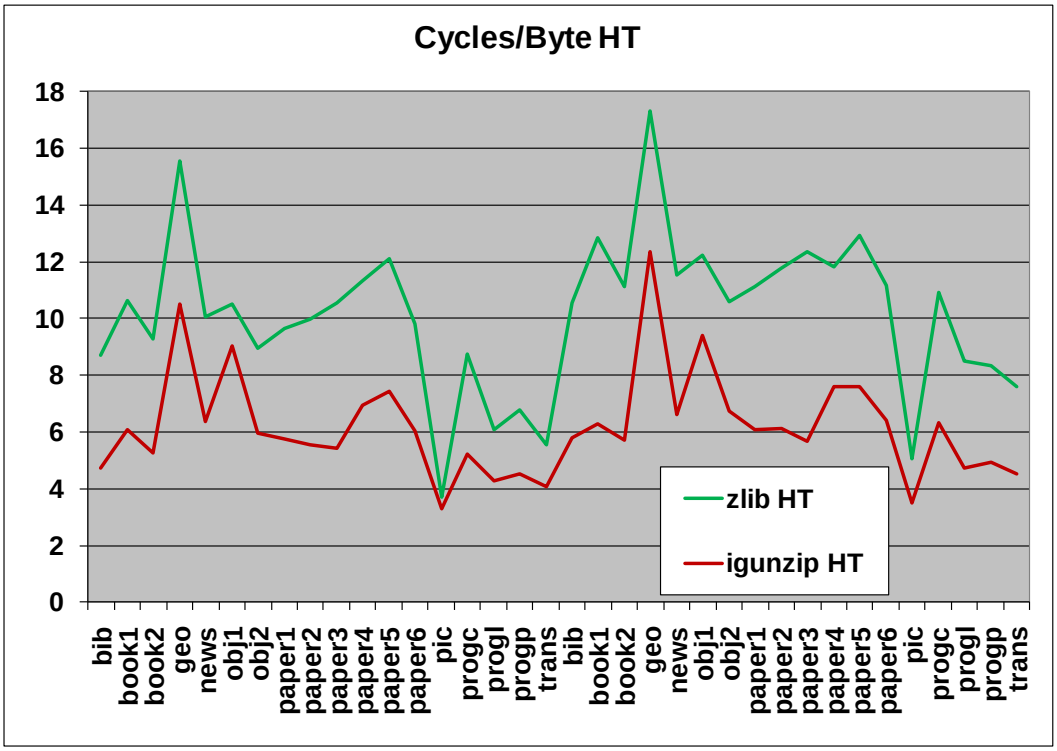
Figure 1: Performance (Cycles/byte) of igunzip/Zlib decompression with Single Thread[3]



[3] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 6. For more information go to http://www.intel.com/performance

Figure 2: Performance (Cycles/byte) of igunzip/Zlib decompression with Intel® HT Technology[4]



The figures show the decompression performance in Cycles/byte as a function of the data files in the Calgary Corpus. The file set appears twice; the set of points on the left comprises compressed files that were generated with "Zlib -9", and the set on the right generated with "Zlib -1". The decompression speeds for a given file are a little slower when that file has been compressed with the "-1" option instead of "-9".

The **igunzip** performance averages to ~**5.7 cycles/byte** on 1 core with Intel® HT Technology on the Calgary Corpus.[4]

We determine the averages based on summing the aggregate cycles and the aggregate bytes of the uncompressed files and computing the ratio. This method weights the average more towards bigger files. An alternate method

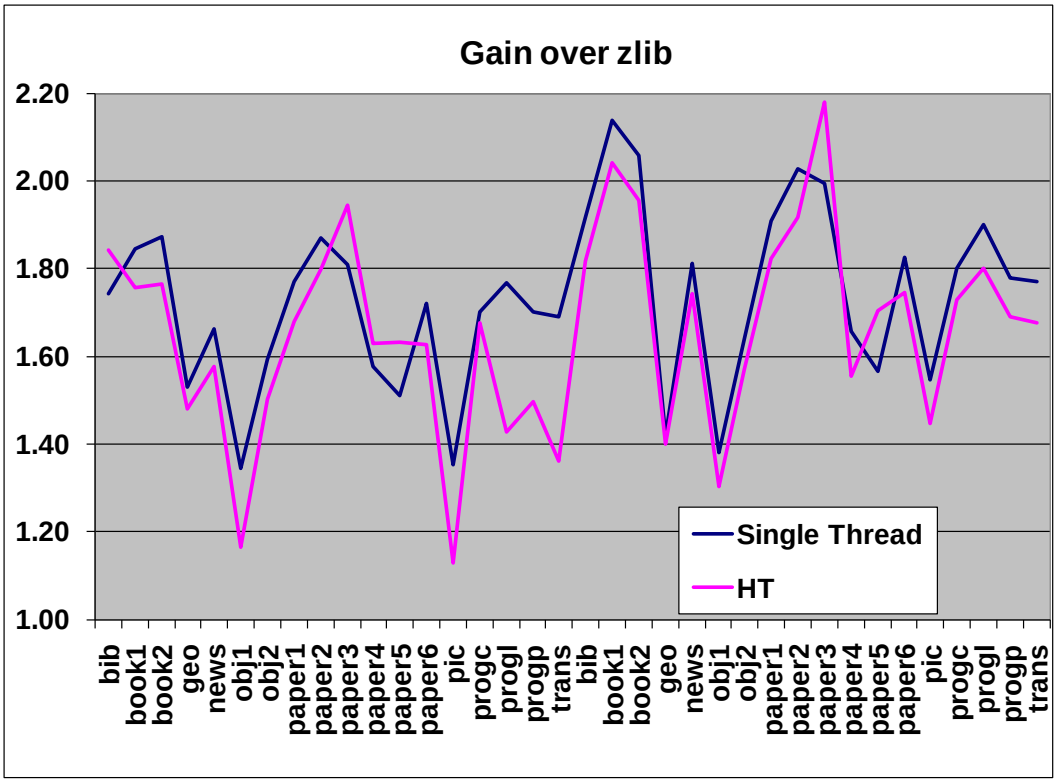
---

[4] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 7. For more information go to http://www.intel.com/performance.

would be to average the cycles/byte of each file, which weights all files evenly. However as both methods give close performance when we break up the performance per Corpus, we chose the former method.

Figure 3: Performance Gain of igunzip over Zlib decompression ~1.75X[5]



Our optimized **igunzip** decompression is **1.7-1.8X** faster than Zlib decompression on the average with and without Intel® HT Technology respectively.[5]

---

[5] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 7. For more information go to http://www.intel.com/performance.

Figure 4: Performance Scaling of decompression with Intel® HT Technology ~1.4X[6]

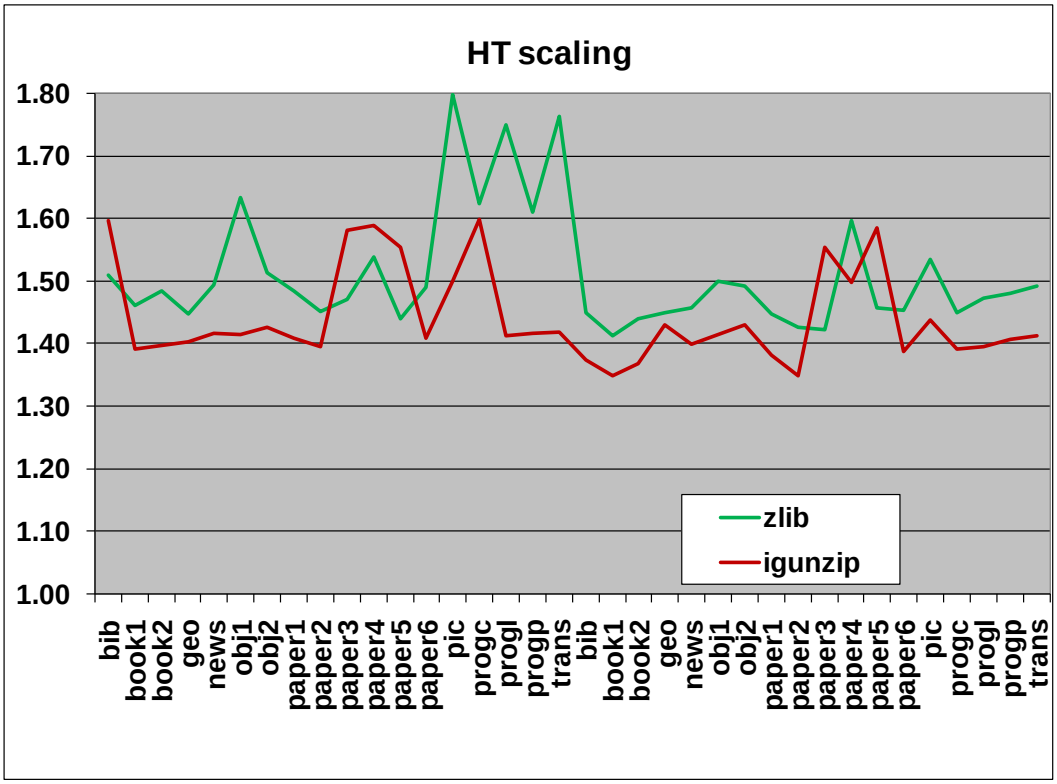


Figure 4 shows the excellent performance scaling that we can achieve with Intel® HT Technology, **~1.41-1.48 X** for **igunzip** and Zlib decompression respectively.[6] By performance scaling, we refer to how much additional performance is gained by using Intel® HT Technology over a single thread for a given decompressor on a single core.

---

[6] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 7. For more information go to http://www.intel.com/performance.

Figure 5: Detailed Performance of Decompression on Calgary Corpus[7]

					igunzip		zlib		Gain	
	File	Original size	Comp size	Ratio	ST Cycles	HT Cycles	ST Cycles	HT Cycles	ST	HT
zlib -9	bib	111,261	35,083	32%	839,693	526,008	1,462,473	968,530	1.74	1.84
	book1	768,771	312,498	41%	6,484,746	4,661,356	11,958,506	8,186,428	1.84	1.76
	book2	610,856	206,147	34%	4,493,683	3,215,207	8,418,433	5,671,479	1.87	1.76
	geo	102,400	68,373	67%	1,508,260	1,075,873	2,306,391	1,592,709	1.53	1.48
	news	377,109	144,492	38%	3,409,412	2,407,616	5,669,449	3,796,041	1.66	1.58
	obj1	21,504	10,329	48%	274,550	194,046	368,759	225,728	1.34	1.16
	obj2	246,814	81,027	33%	2,103,304	1,474,304	3,351,676	2,213,895	1.59	1.50
	paper1	53,161	18,536	35%	429,413	304,944	760,354	512,395	1.77	1.68
	paper2	82,199	29,677	36%	635,930	456,022	1,188,885	819,465	1.87	1.80
	paper3	46,526	18,067	39%	398,845	252,143	721,003	490,044	1.81	1.94
	paper4	13,286	5,527	42%	146,956	92,446	231,857	150,670	1.58	1.63
	paper5	11,954	4,988	42%	137,771	88,618	208,247	144,580	1.51	1.63
	paper6	38,105	13,292	35%	323,488	229,702	556,349	373,386	1.72	1.63
	pic	513,216	52,233	10%	2,540,483	1,693,230	3,436,359	1,910,590	1.35	1.13
	progc	39,611	13,342	34%	330,533	206,901	562,603	346,554	1.70	1.67
	progl	71,646	16,158	23%	432,080	306,091	763,821	436,684	1.77	1.43
	progp	49,379	11,180	23%	317,046	223,690	539,171	335,004	1.70	1.50
	trans	93,695	18,928	20%	540,946	381,449	914,467	518,792	1.69	1.36
zlib -1	bib	111,261	43,894	39%	887,299	645,838	1,701,895	1,174,525	1.92	1.82
	book1	768,771	365,268	48%	6,515,413	4,834,024	13,929,980	9,864,763	2.14	2.04
	book2	610,856	248,929	41%	4,756,706	3,478,520	9,786,768	6,799,517	2.06	1.95
	geo	102,400	69,878	68%	1,808,012	1,264,164	2,567,143	1,771,067	1.42	1.40
	news	377,109	164,300	44%	3,487,262	2,492,239	6,322,699	4,341,697	1.81	1.74
	obj1	21,504	10,702	50%	285,506	201,786	394,079	262,885	1.38	1.30
	obj2	246,814	93,481	38%	2,379,037	1,663,727	3,899,164	2,614,502	1.64	1.57
	paper1	53,161	21,605	41%	448,110	324,223	854,749	590,863	1.91	1.82
	paper2	82,199	35,092	43%	680,807	504,467	1,379,717	966,864	2.03	1.92
	paper3	46,526	20,812	45%	409,608	263,590	817,284	574,850	2.00	2.18
	paper4	13,286	6,066	46%	151,226	101,024	250,586	157,029	1.66	1.55
	paper5	11,954	5,417	45%	143,587	90,646	224,847	154,349	1.57	1.70
	paper6	38,105	15,275	40%	338,044	243,558	617,241	424,852	1.83	1.74
	pic	513,216	65,571	13%	2,578,836	1,793,153	3,984,758	2,596,472	1.55	1.45
	progc	39,611	15,449	39%	347,609	249,917	625,825	431,771	1.80	1.73
	progl	71,646	20,032	28%	471,649	337,953	896,401	608,521	1.90	1.80
	progp	49,379	13,376	27%	341,899	243,186	608,120	410,862	1.78	1.69
	trans	93,695	23,960	26%	600,045	424,981	1,063,009	712,886	1.77	1.68

[7] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 7. For more information go to http://www.intel.com/performance.

Figure 5 shows the compression ratios, total cycles to decompress and the relative gain of **igunzip** over Zlib decompression with and without Intel® HT Technology.

Conclusion

The paper describes the performance of the best-known implementation of DEFLATE Decompression, **igunzip**, on Intel® processors based on the 32-nm micro-architecture, specifically an Intel® Core™ i5 650 processor at a frequency of 3.20 GHz. We are able to perform decompression at ~ **5.7 cycles/byte** on a single core with Intel® HT Technology, on the Calgary Corpus[8]. In terms of throughput, a **single core** can perform decompression at the aggregate rate of ~**4.5 Gigabits/sec** on the Calgary Corpus[8].

Our optimized **igunzip** decompression is **1.7-1.8X** faster than Zlib decompression on the average, with and without Intel® HT Technology respectively[8].

It is possible to get further performance gains for specific usages that target very small data buffers, work only on fixed sized buffers or have very different compression ratios. Such optimizations have not been attempted in the current paper.

We also show the excellent performance scaling that we can achieve with Intel® HT Technology, **~1.41-1.48 X** for **igunzip** and Zlib decompression respectively[8].

[8] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 7. For more information go to http://www.intel.com/performance.

References

[1] DEFLATE Compressed Data Format Specification - RFC1951: http://www.faqs.org/rfcs/rfc1951.html

[2] Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers, September 1952, Volume 40, Number 9, pp. 1098-1101.

[3] Ziv J., Lempel A., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.

[4] Fast CRC Computation for Generic Polynomials using PCLMULQDQ Instruction http://download.intel.com/design/intarch/papers/323102.pdf

[5] RFC1952 - GZIP file format specification version 4.3 http://www.faqs.org/rfcs/rfc1952.html

[6] RFC1950 - ZLIB Compressed Data Format Specification version 3.3 http://www.faqs.org/rfcs/rfc1950.html

[7] Calgary Corpus http://www.data-compression.info/Corpora/CalgaryCorpus/index.htm

[8] Canterbury Corpus http://corpus.canterbury.ac.nz/descriptions/

[9] Silesia Corpus http://www.data-compression.info/Corpora/SilesiaCorpus/index.htm

Appendix – Detailed Performance Results

This section contains all the detailed performance data for the rest of the corpora.

Figure 6: Detailed Performance of Decompression on Canterbury Corpus[9]

					igunzip		zlib		Gain	
	File	Original size	Comp size	Ratio	ST Cycles	HT Cycles	ST Cycles	HT Cycles	ST	HT
zlib -9	alice29.txt	152,089	54,182	36%	1,139,528	819,617	2,153,143	1,463,249	1.89	1.79
	asyoulik.txt	125,179	48,790	39%	1,041,550	716,816	1,898,700	1,296,495	1.82	1.81
	cp.html	24,603	7,952	32%	224,171	159,048	345,077	229,753	1.54	1.44
	fields.c	11,150	3,127	28%	96,303	60,818	149,701	92,640	1.55	1.52
	grammar.lsp	3,721	1,234	33%	44,616	28,305	65,063	40,426	1.46	1.43
	kennedy.xls	1,029,744	207,041	20%	6,050,584	4,550,196	9,678,539	6,787,960	1.60	1.49
	lcet10.txt	426,754	144,451	34%	3,033,725	2,184,940	5,822,890	3,941,915	1.92	1.80
	plrabn12.txt	481,861	194,344	40%	4,030,884	2,899,162	7,427,729	5,091,831	1.84	1.76
	ptt5	513,216	52,233	10%	2,540,058	1,692,983	3,434,006	2,164,561	1.35	1.28
	sum	38,240	12,850	34%	372,795	257,776	549,544	363,722	1.47	1.41
	xargs.1	4,227	1,748	41%	53,506	34,120	79,467	49,691	1.49	1.46
zlib -1	alice29.txt	152,089	65,148	43%	1,230,537	908,036	2,545,250	1,790,720	2.07	1.97
	asyoulik.txt	125,179	56,809	45%	1,061,940	784,003	2,204,716	1,557,147	2.08	1.99
	cp.html	24,603	9,046	37%	234,036	166,979	375,178	251,713	1.60	1.51
	fields.c	11,150	3,665	33%	102,619	65,993	167,362	114,834	1.63	1.74
	grammar.lsp	3,721	1,344	36%	47,098	29,801	68,281	42,250	1.45	1.42
	kennedy.xls	1,029,744	242,311	24%	7,315,391	5,385,078	10,698,356	7,363,902	1.46	1.37
	lcet10.txt	426,754	174,142	41%	3,254,371	2,401,501	6,839,031	4,772,891	2.10	1.99
	plrabn12.txt	481,861	228,901	48%	4,144,981	3,077,216	8,785,968	6,242,514	2.12	2.03
	ptt5	513,216	65,571	13%	2,579,003	1,793,489	3,983,589	2,559,684	1.54	1.43
	sum	38,240	14,130	37%	442,139	305,740	632,465	424,717	1.43	1.39
	xargs.1	4,227	1,864	44%	55,223	35,318	85,184	52,919	1.54	1.50

The **igunzip** performance averages to ~**5.04 cycles/byte** on 1 core with Intel® HT Technology on the Canterbury Corpus[9].

[9] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 7. For more information go to http://www.intel.com/performance.

Figure 7: Detailed Performance of Decompression on Silesia Corpus[10]

					igunzip		zlib		Gain	
	File	Original size	Comp size	Ratio	ST Cycles	HT Cycles	ST Cycles	HT Cycles	ST	HT
zlib -9	dickens	10,192,446	3,854,747	38%	80,367,393	57,629,664	151,394,500	103,447,608	1.88	1.80
	mozilla	51,220,480	19,044,408	37%	445,747,709	322,417,258	681,939,419	452,486,108	1.53	1.40
	mr	9,970,564	3,660,806	37%	85,697,117	60,416,451	133,283,594	91,193,350	1.56	1.51
	nci	33,553,445	2,988,013	9%	118,691,681	83,543,837	211,264,543	135,662,962	1.78	1.62
	ooffice	6,152,192	3,092,938	50%	74,458,463	52,181,682	113,798,691	75,344,244	1.53	1.44
	osdb	10,085,684	3,673,189	36%	85,719,822	61,446,494	130,794,724	85,429,731	1.53	1.39
	reymont	6,627,202	1,823,208	28%	43,371,512	30,717,380	80,738,136	53,416,937	1.86	1.74
	samba	21,606,400	5,402,722	25%	128,298,651	93,600,863	214,797,950	141,659,937	1.67	1.51
	sao	7,251,944	5,325,932	73%	100,574,616	73,265,561	124,151,663	84,443,165	1.23	1.15
	webster	41,458,703	12,073,487	29%	293,640,583	207,010,312	525,895,921	349,691,575	1.79	1.69
	x-ray	8,474,240	6,045,050	71%	127,641,744	91,501,629	185,826,286	127,745,002	1.46	1.40
	xml	5,345,280	658,917	12%	21,826,895	15,520,420	38,788,210	24,978,269	1.78	1.61
zlib -1	dickens	10,192,446	4,585,630	45%	83,492,791	61,573,890	179,167,678	126,815,604	2.15	2.06
	mozilla	51,220,480	20,577,238	40%	490,617,745	353,107,565	767,655,805	516,144,221	1.56	1.46
	mr	9,970,564	3,828,378	38%	89,595,784	63,471,140	149,411,296	104,495,861	1.67	1.65
	nci	33,553,445	4,624,609	14%	136,229,974	98,071,334	269,281,529	177,988,013	1.98	1.81
	ooffice	6,152,192	3,290,544	53%	83,232,841	57,706,707	128,092,594	86,135,690	1.54	1.49
	osdb	10,085,684	4,076,403	40%	95,950,606	68,143,544	146,586,744	96,692,538	1.53	1.42
	reymont	6,627,202	2,376,442	36%	48,772,602	35,414,802	99,867,070	69,348,066	2.05	1.96
	samba	21,606,400	6,329,467	29%	137,804,662	101,243,420	245,846,356	165,723,575	1.78	1.64
	sao	7,251,944	5,567,786	77%	110,559,599	80,345,670	140,586,067	96,414,835	1.27	1.20
	webster	41,458,703	14,991,254	36%	301,423,712	218,723,082	614,007,741	421,323,092	2.04	1.93
	x-ray	8,474,240	6,033,944	71%	117,096,131	84,825,783	197,770,115	139,804,633	1.69	1.65
	xml	5,345,280	965,260	18%	25,384,050	18,241,770	49,054,002	32,653,055	1.93	1.79

The **igunzip** performance averages to ~**5.64 cycles/byte** on 1 core with Intel® HT Technology on the Silesia Corpus[10].

---

[10] Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Configurations, testing: Refer to Section Performance on page 7. For more information go to http://www.intel.com/performance.

Authors

Vinodh Gopal, Jim Guilford, Chengda Yang, Erdinc Ozturk, Gil Wolrich, Wajdi Feghali, Kirk Yap and **Martin Dixon** are IA Architects with the IAG Group at Intel Corporation.

Acronyms

IA	Intel® Architecture
ILP	Instruction level parallelism
SIMD	Single Instruction Multiple Data
SSE	Streaming SIMD Extensions