A few thoughts about boosting the TEXT search by using compressed Building-Blocks.

The testbed is OSHO.TXT, it plays the role (being a 32++MB) of a 197MB chunk. My 26GB text corpus is comprised of 500,000+ text files which are to be merged into, say, 256MB chunks. The goal is to perform a string pattern search into these chunks as quick as possible.

One approach is to compress them with 'qpress' derivative, that is *Lasse*'s PTHREADED **QuickLZ** super library. Another is to read (at burst speed with one 'fread') the BB (it stands for Building-Block) pool from the .SS files made with Simplicius_Simplicissimus_Septupleton from afore mentioned text chunks.

For example the BB pool of OSHO.TXT.SS is only (17% or one sixth) 32MB (highly LZ[MA] compressible not-by-the-way). Sizes of OSHO.TXT BB order 7 pool: 5,508,090 BB007.7z; 12,155,765 BB007.zip; 33,622,064 BB007.txt.

This BB pool allows to probe whether a pattern can be found with FULL-text search (in that case decompression will be needed). Here we have some 4+ million **SORTED** 7bytes chunks (Building-Blocks). Instead of executing FULL-text traversal first, it is enough to probe (by using binary-search into the BB pool) our pattern.

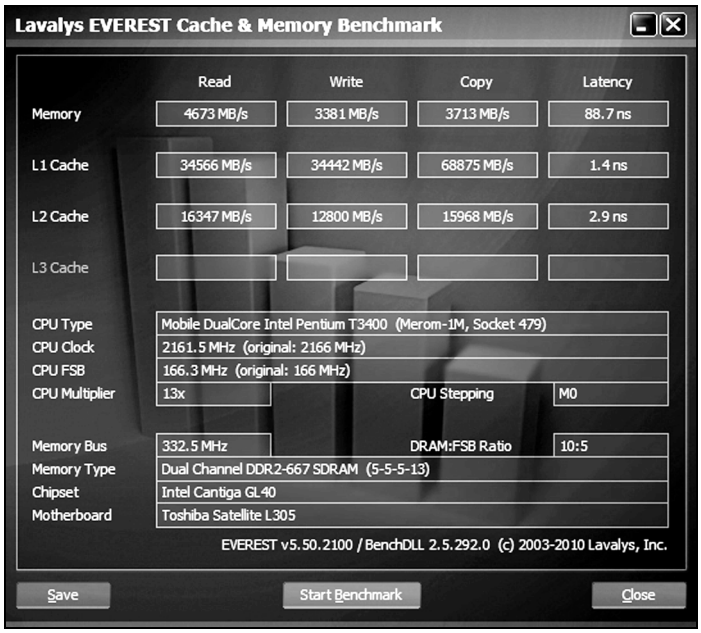
If it is less than 7bytes then we compare only the first 7-bytes (by using binary-search) and if it fails then perform the brute-force to ensure its absence, also the literals (7-bytes long) at the end of .SS file should not be forgotten – these literals RUIN the whole search process (because of a possible pattern spanning across BB-pool and literals-pool), grmbl. The solution is easy but UGLY: depending on pattern length the last (or more) triad must be decompressed in order to be prefixed to the literals-pool.

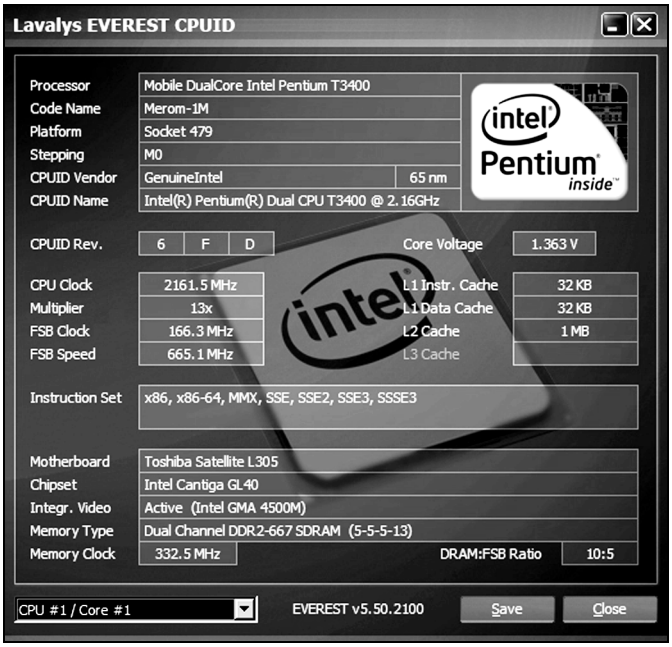
If the pattern is 7[+]bytes long then we probe BBs of the pattern one-by-one versus BBs of the file being searched namely (or rather numberly) our 32MB, (here the spanning problem remains unsolved too).

For example if the pattern is 'get down' i.e. 8bytes long then we search firstly for 'get dow' and (only if it is successful) secondly for 'et down'.

If there is at least one unsuccessful search then this pattern cannot be found using FULL-text search i.e. there is no need of decompression. PROBABLY the pattern exists in the decompressed text when ALL BBs of the pattern match respective BBs of the file.

CPU	CPU Clock	Motherboard	Chipset	Memory	CL-RCD-RP-RAS	Copy Speed
Core i7 Extreme 965	3333 MHz	Asus P6T Deluxe	X58	Triple DDR3-1333	9-9-9-24 CR1	14727 MB/s
Xeon X3430	2400 MHz	Supermicro X8SIL-F	i3420	Dual DDR3-1333	9-9-9-24 CR1	12034 MB/s
Core i5 650	3466 MHz	Supermicro C7SIM-Q	Q57 Int.	Dual DDR3-1333	9-9-9-24 CR1	11041 MB/s
Xeon X5550	2666 MHz	Supermicro X8DTN+	i5520	Triple DDR3-1333	9-9-9-24 CR1	9489 MB/s
Sempron 140	2700 MHz	Gigabyte GA-790FXTA-UD5	AMD790FX	Unganged Dual DDR3-1066	8-8-8-20 CR1	9391 MB/s
Phenom II X4 Black 940	3000 MHz	Asus M3N78-EM	GeForce8300 Int.	Ganged Dual DDR2-800	5-5-5-18 CR2	8400 MB/s
Athlon64 X2 Black 6400+	3200 MHz	MSI K9N SLI Platinum	nForce570SLI	Dual DDR2-800	4-4-4-11 CR1	7222 MB/s
Opteron 2378	2400 MHz	Tyan Thunder n3600R	nForcePro-3600	Unganged Dual DDR2-800R	6-6-6-18 CR1	7052 MB/s
Athlon64 X2 4000+	2100 MHz	ASRock ALiveNF7G-HDready	nForce7050-630a Int.	Dual DDR2-700	5-5-5-18 CR2	6858 MB/s
12x Opteron 2431	2400 MHz	Supermicro H8DI3+-F	SR5690	Unganged Dual DDR2-800R	6-6-6-18 CR1	6747 MB/s
Core 2 Extreme QX9650	3000 MHz	Gigabyte GA-EP35C-DS3R	P35	Dual DDR3-1066	8-8-8-20 CR2	6305 MB/s
Pentium EE 955	3466 MHz	Intel D955XBK	i955X	Dual DDR2-667	4-4-4-11	6218 MB/s
P4EE	3733 MHz	Intel SE7230NH1LX	iE7230	Dual DDR2-667	5-5-5-15	6170 MB/s
Phenom X4 9500	2200 MHz	Asus M3A	AMD770	Ganged Dual DDR2-800	5-5-5-18 CR2	5479 MB/s
Xeon E5462	2800 MHz	Intel S5400SF	i5400	Quad DDR2-640FB	5-5-5-15	5441 MB/s
Core 2 Extreme X6800	2933 MHz	Abit AB9	P965	Dual DDR2-800	5-5-5-18 CR2	5433 MB/s
Core 2 Duo P8400	2266 MHz	MSI MegaBook PR201	GM45 Int.	Dual DDR2-667	5-5-5-15	5110 MB/s
Pentium D 820	2800 MHz	Abit Fatal1ty F-I90HD	RS600 Int.	Dual DDR2-800	5-5-5-18 CR2	4843 MB/s
Athlon64 3200+	2000 MHz	ASRock 939S56-M	SiS756	Dual DDR400	2.5-3-3-8 CR2	4632 MB/s
Core 2 Extreme QX6700	2666 MHz	Intel D975XBX2	i975X	Dual DDR2-667	5-5-5-15	4606 MB/s
Xeon	3400 MHz	Intel SE7320SP2	iE7320	Dual DDR333R	2.5-3-3-7	4165 MB/s
Celeron 420	1600 MHz	Intel DQ965CO	Q965 Int.	Dual DDR2-667	5-5-5-15	4139 MB/s
Opteron HE 2344	1700 MHz	Supermicro H8DME-2	nForcePro-3600	Unganged Dual DDR2-667R	5-5-5-15 CR1	4022 MB/s
Opteron HE 2210	1800 MHz	Tyan Thunder h2000M	BCM5785	Dual DDR2-600R	5-5-5-15 CR1	3945 MB/s
Pentium T3400	**2166 MHz**	**Toshiba Satellite L305**	**GL40 Int.**	**Dual DDR2-667**	**5-5-5-13**	**3718 MB/s**





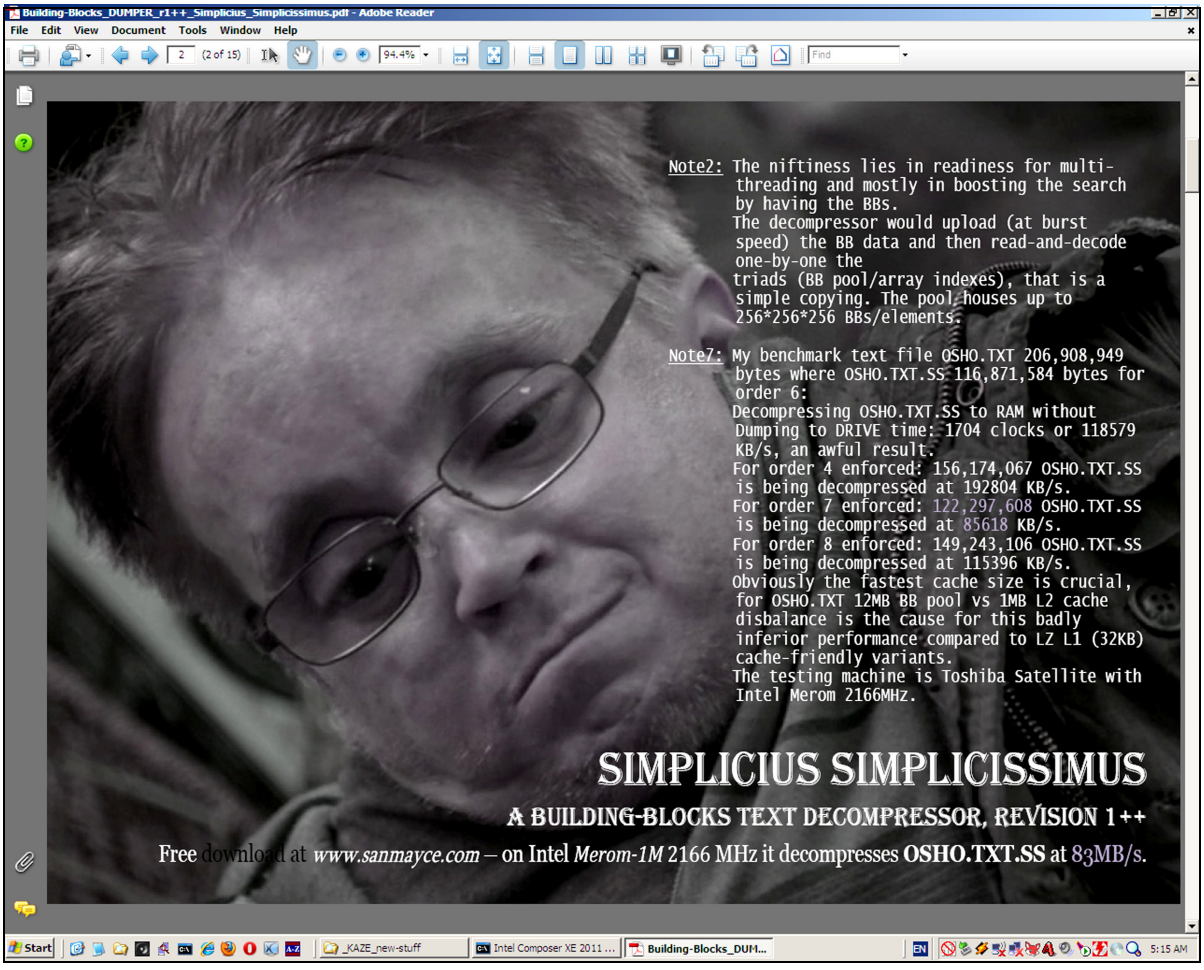
Intel Merom 2166MHz			
Functions	32bit Compiler: Microsoft 16.00.30319.01 Revision 6	32bit Compiler: Microsoft 16.00.30319.01 Revision 6+	32bit Compiler: Intel 11.1 Revision 6
strstr_Microsoft	'an': 172KB/clock	'an': 168KB/clock	'an': 147KB/clock
strstr_GNU_C_Library	'an': 279KB/clock	'an': 268KB/clock	'an': 268KB/clock
Railgun	'an': 262KB/clock	'an': 248KB/clock	'an': 257KB/clock
Railgun_Quadruplet	'an': 265KB/clock	'an': **315KB**/clock	'an': 279KB/clock
strstr_Microsoft	'to': 179KB/clock	'to': 175KB/clock	'to': 152KB/clock
strstr_GNU_C_Library	'to': 297KB/clock	'to': 288KB/clock	'to': 285KB/clock
Railgun	'to': 294KB/clock	'to': 279KB/clock	'to': 292KB/clock
Railgun_Quadruplet	'to': 297KB/clock	'to': **349KB**/clock	'to': 310KB/clock
strstr_Microsoft	'TDK': 214KB/clock	'TDK': 210KB/clock	'TDK': 178KB/clock
strstr_GNU_C_Library	'TDK': 411KB/clock	'TDK': 398KB/clock	'TDK': 375KB/clock
Railgun	'TDK': 339KB/clock	'TDK': 330KB/clock	'TDK': 340KB/clock
Railgun_Quadruplet	'TDK': 341KB/clock	'TDK': **424KB**/clock	'TDK': 350KB/clock
strstr_Microsoft	'the': 164KB/clock	'the': 160KB/clock	'the': 141KB/clock
strstr_GNU_C_Library	'the': 256KB/clock	'the': 247KB/clock	'the': 244KB/clock
Railgun	'the': 262KB/clock	'the': 253KB/clock	'the': 261KB/clock
Railgun_Quadruplet	'the': 263KB/clock	'the': **285KB**/clock	'the': 271KB/clock
strstr_Microsoft	'fast': 210KB/clock	'fast': 206KB/clock	'fast': 175KB/clock
strstr_GNU_C_Library	'fast': 396KB/clock	'fast': 386KB/clock	'fast': 375KB/clock
Railgun	'fast': 339KB/clock	'fast': 330KB/clock	'fast': 339KB/clock
Railgun_Quadruplet	'fast': 395KB/clock	'fast': **398KB**/clock	'fast': 382KB/clock
strstr_Microsoft	'easy': 177KB/clock	'easy': 174KB/clock	'easy': 152KB/clock
strstr_GNU_C_Library	'easy': 296KB/clock	'easy': 289KB/clock	'easy': 281KB/clock
Railgun	'easy': 337KB/clock	'easy': 329KB/clock	'easy': **339KB**/clock
Railgun_Quadruplet	'easy': 302KB/clock	'easy': 300KB/clock	'easy': 298KB/clock
strstr_Microsoft	'grmbl': 209KB/clock	'grmbl': 206KB/clock	'grmbl': 175KB/clock
strstr_GNU_C_Library	'grmbl': 393KB/clock	'grmbl': 382KB/clock	'grmbl': 375KB/clock
Railgun	'grmbl': 341KB/clock	'grmbl': 333KB/clock	'grmbl': 344KB/clock
Railgun_Quadruplet	'grmbl': 395KB/clock	'grmbl': **397KB**/clock	'grmbl': 383KB/clock
strstr_Microsoft	'email': 178KB/clock	'email': 175KB/clock	'email': 152KB/clock
strstr_GNU_C_Library	'email': 301KB/clock	'email': 293KB/clock	'email': 285KB/clock
Railgun	'email': 338KB/clock	'email': 329KB/clock	'email': **340KB**/clock
Railgun_Quadruplet	'email': 303KB/clock	'email': 301KB/clock	'email': 299KB/clock
strstr_Microsoft	'pasting': 210KB/clock	'pasting': 207KB/clock	'pasting': 175KB/clock
strstr_GNU_C_Library	'pasting': 396KB/clock	'pasting': 386KB/clock	'pasting': 374KB/clock
Railgun	'pasting': 346KB/clock	'pasting': 338KB/clock	'pasting': 347KB/clock
Railgun_Quadruplet	'pasting': 401KB/clock	'pasting': **404KB**/clock	'pasting': 389KB/clock
strstr_Microsoft	'amazing': 191KB/clock	'amazing': 188KB/clock	'amazing': 162KB/clock
strstr_GNU_C_Library	'amazing': 338KB/clock	'amazing': 329KB/clock	'amazing': 327KB/clock
Railgun	'amazing': 345KB/clock	'amazing': 337KB/clock	'amazing': 347KB/clock
Railgun_Quadruplet	'amazing': 347KB/clock	'amazing': **347KB**/clock	'amazing': 340KB/clock
strstr_Microsoft	'underdog': 203KB/clock	'underdog': 200KB/clock	'underdog': 170KB/clock
strstr_GNU_C_Library	'underdog': 376KB/clock	'underdog': 366KB/clock	'underdog': 363KB/clock
Railgun	'underdog': 348KB/clock	'underdog': 339KB/clock	'underdog': 350KB/clock
Railgun_Quadruplet	'underdog': 385KB/clock	'underdog': **389KB**/clock	'underdog': 376KB/clock
strstr_Microsoft	'superdog': 195KB/clock	'superdog': 191KB/clock	'superdog': 164KB/clock
strstr_GNU_C_Library	'superdog': 347KB/clock	'superdog': 337KB/clock	'superdog': 335KB/clock
Railgun	'superdog': 347KB/clock	'superdog': 338KB/clock	'superdog': 348KB/clock
Railgun_Quadruplet	'superdog': 355KB/clock	'superdog': **356KB**/clock	'superdog': 348KB/clock
strstr_Microsoft	'participants': 210KB/clock	'participants': 206KB/clock	'participants': 175KB/clock
strstr_GNU_C_Library	'participants': 396KB/clock	'participants': 386KB/clock	'participants': 373KB/clock
Railgun	'participants': 355KB/clock	'participants': 346KB/clock	'participants': 357KB/clock
Railgun_Quadruplet	'participants': 411KB/clock	'participants': **415KB**/clock	'participants': 400KB/clock
strstr_Microsoft	'skillessness': 195KB/clock	'skillessness': 192KB/clock	'skillessness': 164KB/clock
strstr_GNU_C_Library	'skillessness': 348KB/clock	'skillessness': 338KB/clock	'skillessness': 336KB/clock
Railgun	'skillessness': 352KB/clock	'skillessness': 340KB/clock	'skillessness': 353KB/clock
Railgun_Quadruplet	'skillessness': 363KB/clock	'skillessness': **363KB**/clock	'skillessness': 357KB/clock
strstr_Microsoft	'I should have known': 213KB/clock	'I should have known': 208KB/clock	'I should have known': 177KB/clock
strstr_GNU_C_Library	'I should have known': 407KB/clock	'I should have known': 393KB/clock	'I should have known': 373KB/clock
Railgun	'I should have known': 369KB/clock	'I should have known': 359KB/clock	'I should have known': 371KB/clock
Railgun_Quadruplet	'I should have known': 437KB/clock	'I should have known': **438KB**/clock	'I should have known': 424KB/clock
strstr_Microsoft	'human consciousness': 197KB/clock	'human consciousness': 193KB/clock	'human consciousness': 166KB/clock
strstr_GNU_C_Library	'human consciousness': 356KB/clock	'human consciousness': 344KB/clock	'human consciousness': 346KB/clock
Railgun	'human consciousness': 366KB/clock	'human consciousness': 355KB/clock	'human consciousness': 367KB/clock
Railgun_Quadruplet	'human consciousness': 389KB/clock	'human consciousness': **389KB**/clock	'human consciousness': 382KB/clock

Dataset: 2,459,508 lines; File: OSHO.TXT 206,908,949 bytes; Shortest/Longest line: 0/161 chars
Note #1: The benchmark program is compiled as 32bit code:
compile line: cl /Ox strstr_SHORT-SHOWDOWN.c
compile line: icl /O3 /Qparallel strstr_SHORT-SHOWDOWN.c
Note #2: The fastest result for a given pattern is in bold.

For reference: www.sanmayce.com/Railgun/index.html#Heavy-Search-Hustle

The following test is done on Toshiba Satellite with Intel Merom 2166MHz from **_KAZE_Simplicius_Simplicissimus_Septupleton_r1++_strstr_SHORT-SHOWDOWN_r6+.7z** 52,874,058 bytes package:

```
D:\_KAZE_new-stuff\_KAZE_Simplicius_Simplicissimus_Septupleton_r1++_strstr_SHORT-SHOWDOWN_r6+>dir/og

09/01/2011  05:04 AM    <DIR>          _Simplicissimus_SCANNED+HTML
09/01/2011  05:04 AM    <DIR>          Printer-friendly
08/31/2011  03:50 PM             1,074 Compile_Intel_Septupleton.bat
08/31/2011  03:50 PM               207 Compile_Microsoft_Septupleton.bat
08/31/2011  03:50 PM           203,469 cpuid_Merom_2166Mhz.png
08/31/2011  03:50 PM                13 frustration
08/31/2011  03:50 PM       206,908,949 OSHO.TXT
08/31/2011  03:50 PM             1,606 Command prompt.lnk
08/31/2011  03:50 PM           116,235 Report_Merom_2166Mhz.htm
08/31/2011  03:50 PM           108,464 Results_Merom_2166Mhz.txt
08/31/2011  03:50 PM             1,569 RUNME_IT_TAKES_AN_HOUR.BAT
08/31/2011  03:50 PM            86,328 Simplicius_Simplicissimus_Septupleton.c
08/31/2011  03:50 PM           100,864 Simplicius_Simplicissimus_Septupleton_Intel_v12_fast.exe
08/31/2011  03:50 PM            98,816 Simplicius_Simplicissimus_Septupleton_Intel_v12_O3.exe
08/31/2011  03:50 PM           129,024 Simplicius_Simplicissimus_Septupleton_Intel_v12_O3_Qparallel.exe
08/31/2011  03:50 PM           208,672 cachemem_Merom_2166Mhz.png
08/31/2011  03:50 PM           104,448 Simplicius_Simplicissimus_Septupleton_Intel_v12_Ox.exe
08/31/2011  03:50 PM           108,544 Simplicius_Simplicissimus_Septupleton_Intel_v12_Ox_Qparallel.exe
08/31/2011  03:50 PM           223,158 Simplicius_Simplicissimus_Septupleton_Microsoft_v16_Ox.cod
08/31/2011  03:50 PM            87,552 Simplicius_Simplicissimus_Septupleton_Microsoft_v16_Ox.exe
08/31/2011  03:50 PM               754 strstr_COMPILE_Intel.bat
08/31/2011  03:50 PM               143 strstr_COMPILE_Microsoft.bat
08/31/2011  03:50 PM           116,873 strstr_SHORT-SHOWDOWN.c
08/31/2011  03:50 PM            94,208 strstr_SHORT-SHOWDOWN_Intel_v12_fast.exe
08/31/2011  03:50 PM            90,624 strstr_SHORT-SHOWDOWN_Intel_v12_O3.exe
08/31/2011  03:50 PM           118,272 strstr_SHORT-SHOWDOWN_Intel_v12_O3_Qparallel.exe
08/31/2011  03:50 PM           657,780 strstr_SHORT-SHOWDOWN_Intel_v12_Ox.cod
08/31/2011  03:50 PM            90,112 strstr_SHORT-SHOWDOWN_Intel_v12_Ox.exe
08/31/2011  03:50 PM            91,136 strstr_SHORT-SHOWDOWN_Intel_v12_Ox_Qparallel.exe
08/31/2011  03:50 PM           224,469 strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.cod
08/31/2011  03:50 PM            86,528 strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.exe
08/31/2011  03:50 PM         1,060,513 Simplicius_Simplicissimus_Septupleton_Intel_v12_Ox.cod

D:\_KAZE_new-stuff\_KAZE_Simplicius_Simplicissimus_Septupleton_r1++_strstr_SHORT-SHOWDOWN_r6+>
```




```
Simplicius_Simplicissimus_Septupleton_Intel_v12_fast.exe OSHO.TXT.SS
                                    Decompression to RAM without Dumping to DRIVE performance: 93676 KB/s
```

```
Simplicius_Simplicissimus_Septupleton_Intel_v12_O3.exe OSHO.TXT.SS
                                                    Decompression to RAM without Dumping to DRIVE performance: 81311 KB/s
```

```
Simplicius_Simplicissimus_Septupleton_Intel_v12_O3_Qparallel.exe OSHO.TXT.SS
                                                    Decompression to RAM without Dumping to DRIVE performance: 83946 KB/s
```

```
Simplicius_Simplicissimus_Septupleton_Intel_v12_Ox.exe OSHO.TXT.SS
                                                    Decompression to RAM without Dumping to DRIVE performance: 80791 KB/s
```

```
Simplicius_Simplicissimus_Septupleton_Intel_v12_Ox_Qparallel.exe OSHO.TXT.SS
                                                    Decompression to RAM without Dumping to DRIVE performance: 84508 KB/s
```

```
Simplicius_Simplicissimus_Septupleton_Microsoft_v16_Ox.exe OSHO.TXT.SS
                                                    Decompression to RAM without Dumping to DRIVE performance: 79802 KB/s
```

```
D:\_KAZE_new-stuff\>Simplicius_Simplicissimus_Septupleton_Intel_v12_fast.exe OSHO.TXT
. . .
Sorting 206908943 Pointers to Building-Blocks 7 chars in size ...
Allocated memory for pointers-to-words in MB: 790
Writing Sorted Building-Blocks to BB007.txt ...
Patternlen:7|206908949-7+1|linecounterRL:4803152|202105791
Gain(reduced-size) or Strnglen-(Patternlen*linecounterRL+3*(Strnglen/Patternlen)+(Strnglen%Patternlen)): 84611620

Maximum compression (or minimum expansion) for order: 7

Creating OSHO.TXT.SS ...

Building-Blocks_DUMPER .SS dumping time: 43095 clocks
Building-Blocks_DUMPER total time: 269516 clocks
```

```
D:\_KAZE_new-stuff\>Simplicius_Simplicissimus_Septupleton_Intel_v12_fast.exe OSHO.TXT.SS
Simplicius_Simplicissimus_Septupleton rev.1++, written by Kaze.
Note1: This is the precursor of Simplicius Simplicissimus - a superfast-low-performance TEXT decompressor.
Note2: The niftiness lies in readiness for multi-threading and mostly in boosting the search by having the BBs.
       The decompressor would upload (at burst speed) the BB data and then read-and-decode one-by-one the
       triads (BB pool/array indexes), that is a simple copying. The pool houses up to 256*256*256 BBs/elements.
Note3: Compiler used: Intel(R) C++ Compiler XE for applications running on IA-32, Version 12.0.4.196 Build 20110427.
Note4: Compile line: icl /Ox /TcBuilding-Blocks_DUMPER.c /FaBuilding-Blocks_DUMPER /w /FAcs
Note5: Maximum size of input text file is arbitrary - 384MB, in fact 384MB+4*384MB must be less than 19??MB.
Note6: The file format is given by typing the compressed file e.g. D:\>type OSHO.TXT.SS.
       The header is 270 bytes long followed by BBs and triads/indexes and the eventual remainder/literals.
Note7: My benchmark text file OSHO.TXT 206,908,949 bytes where OSHO.TXT.SS 116,871,584 bytes for order 6:
       Decompressing OSHO.TXT.SS to RAM without Dumping to DRIVE time: 1704 clocks or 118579 KB/s, an awful result.
       For order 4 enforced: 156,174,067 OSHO.TXT.SS is being decompressed at 192804 KB/s.
       For order 7 enforced: 122,297,608 OSHO.TXT.SS is being decompressed at 85618 KB/s.
       For order 8 enforced: 149,243,106 OSHO.TXT.SS is being decompressed at 115396 KB/s.
       Obviously the fastest cache size is crucial, for OSHO.TXT 12MB BB pool vs 1MB L2 cache disbalance is the
       cause for this badly inferior performance compared to LZ L1 (32KB) cache-friendly variants.
       The test machine is Toshiba Satellite with Intel Merom 2166MHz, with rough 'memcpy' performance: 1646 MB/s.
Note8: For more fast decompressing idea(s) you may contact me at sanmayce@sanmayce.com freely.
Note9: Got it? The eighth note has two meanings: 1] for faster 2] for additional/other fast.
NoteA: Got it? The ninth note has two meanings: 1] for quicker/tighter 2] for additional/other quick/tight.

Allocating 384MB ... OK

Simplicius Simplicissimus is decompressing OSHO.TXT ...
Allocating 256MB for BBs pool + 384MB for RAM-to-RAM decompression ... OK
BBs order: 7
Original file size: 206908949
BBs pool size: 33622064

Decompression to RAM without Dumping to DRIVE time: 2095 clocks
Decompression to RAM without Dumping to DRIVE performance: 96448 KB/s
Total time: 3907 clocks

Benchmarking 'memcpy' by copying 197MB (OSHO.TXT size) ten times ...
Simplicius says for 'memcpy' performance: 1636 MB/s
Simplicius says for Decompression Ratio: 5%

D:\_KAZE_new-stuff\>
```

The two lines only fragment which does all the decoding.
(Listing generated by Microsoft (R) Optimizing Compiler Version 16.00.30319.01):

```
; 1745 :       for( jREGISTER = 0; jREGISTER < iREGISTER; jREGISTER++ )

  01721   85 db              test    ebx, ebx
  01723   7e 4e              jle     SHORT $LN72@main
  01725   8b 44 24 24        mov     eax, DWORD PTR _Strng$[esp+376]
```

```
  01729   8b 4c 24 1c          mov       ecx, DWORD PTR tv1280[esp+376]
  0172d   33 db                xor       ebx, ebx
  0172f   89 44 24 14          mov       DWORD PTR tv1970[esp+376], eax
  01733   89 4c 24 28          mov       DWORD PTR tv930[esp+376], ecx
  01737   eb 07 8d a4 24
     00 00 00 00     npad      9
$LL74@main:

; 1746 :    {
; 1747 : //ThunderwithL=0; // Zeroing the highest byte i.e. the fourth
; 1748 : //memcpy (&ThunderwithL, BB+Patternlen+jREGISTER*3, 3);
; 1749 : memcpy (Strng+GainOrderREGISTER*jREGISTER, BB+GainOrderREGISTER*( 0x00FFFFFF & *(unsigned long *)(BB+Patternlen+jREGISTER*3) ), GainOrderREGISTER);

  01740   8b 15 00 00 00
     00             mov       edx, DWORD PTR _Patternlen
  01746   8d 04 13   lea       eax, DWORD PTR [ebx+edx]
  01749   8b 0c 30   mov       ecx, DWORD PTR [eax+esi]
  0174c   8b 54 24 14          mov       edx, DWORD PTR tv1970[esp+376]
  01750   81 e1 ff ff ff
     00             and       ecx, 16777215                ; 00ffffffH
  01756   0f af cf   imul      ecx, edi
  01759   57                   push      edi
  0175a   03 ce                add       ecx, esi
  0175c   51                   push      ecx
  0175d   52                   push      edx
  0175e   e8 00 00 00 00       call      _memcpy
  01763   01 7c 24 20          add       DWORD PTR tv1970[esp+388], edi
  01767   83 c4 0c   add       esp, 12                      ; 0000000cH
  0176a   83 c3 03   add       ebx, 3
  0176d   ff 4c 24 28          dec       DWORD PTR tv930[esp+376]
  01771   75 cd                jne       SHORT $LL74@main
$LN72@main:

; 1750 :    }
```

Doing Search for Pattern(11bytes i.e. **frustration**) into String(206,908,949bytes i.e. **OSHO.TXT**) line-by-line ...
Shortest/Longest line: **0**/**161** chars
StrnglenTRAVERSED: **204,383,508** bytes
LinesEncountered: **2,459,508**

strstr_SHORT-SHOWDOWN_Intel_v12_fast.exe:

```
strstr_Microsoft_hits/strstr_Microsoft_clocks: 1269/814
strstr_Microsoft performance: 243KB/clock

strstr_GNU_C_Library_hits/strstr_GNU_C_Library_clocks: 1269/527
strstr_GNU_C_Library performance: 383KB/clock

Railgun_Quadruplet_hits/Railgun_Quadruplet_clocks: 1269/613
Railgun_Quadruplet performance: 324KB/clock

Boyer-Moore-Horspool_hits/Boyer-Moore-Horspool_clocks: 1269/539
Boyer-Moore-Horspool performance: 368KB/clock
```

strstr_SHORT-SHOWDOWN_Intel_v12_O3.exe:

```
strstr_Microsoft_hits/strstr_Microsoft_clocks: 1269/755
strstr_Microsoft performance: 279KB/clock

strstr_GNU_C_Library_hits/strstr_GNU_C_Library_clocks: 1269/525
strstr_GNU_C_Library performance: 374KB/clock

Railgun_Quadruplet_hits/Railgun_Quadruplet_clocks: 1269/591
Railgun_Quadruplet performance: 334KB/clock

Boyer-Moore-Horspool_hits/Boyer-Moore-Horspool_clocks: 1269/512
Boyer-Moore-Horspool performance: 392KB/clock
```

strstr_SHORT-SHOWDOWN_Intel_v12_O3_Qparallel.exe:

```
strstr_Microsoft_hits/strstr_Microsoft_clocks: 1269/708
strstr_Microsoft performance: 279KB/clock

strstr_GNU_C_Library_hits/strstr_GNU_C_Library_clocks: 1269/535
strstr_GNU_C_Library performance: 368KB/clock

Railgun_Quadruplet_hits/Railgun_Quadruplet_clocks: 1269/600
Railgun_Quadruplet performance: 313KB/clock

Boyer-Moore-Horspool_hits/Boyer-Moore-Horspool_clocks: 1269/500
SECOND-BEST! Boyer-Moore-Horspool performance: 395KB/clock
```

strstr_SHORT-SHOWDOWN_Intel_v12_Ox.exe:

```
strstr_Microsoft_hits/strstr_Microsoft_clocks: 1269/731
strstr_Microsoft performance: 271KB/clock

strstr_GNU_C_Library_hits/strstr_GNU_C_Library_clocks: 1269/538
strstr_GNU_C_Library performance: 368KB/clock

Railgun_Quadruplet_hits/Railgun_Quadruplet_clocks: 1269/597
Railgun_Quadruplet performance: 337KB/clock

Boyer-Moore-Horspool_hits/Boyer-Moore-Horspool_clocks: 1269/521
Boyer-Moore-Horspool performance: 385KB/clock
```

strstr_SHORT-SHOWDOWN_Intel_v12_Ox_Qparallel.exe:

```
strstr_Microsoft_hits/strstr_Microsoft_clocks: 1269/741
strstr_Microsoft performance: 255KB/clock

strstr_GNU_C_Library_hits/strstr_GNU_C_Library_clocks: 1269/553
strstr_GNU_C_Library performance: 366KB/clock

Railgun_Quadruplet_hits/Railgun_Quadruplet_clocks: 1269/597
Railgun_Quadruplet performance: 338KB/clock

Boyer-Moore-Horspool_hits/Boyer-Moore-Horspool_clocks: 1269/521
Boyer-Moore-Horspool performance: 351KB/clock
```

strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.exe:

```
strstr_Microsoft_hits/strstr_Microsoft_clocks: 1269/887
strstr_Microsoft performance: 207KB/clock

strstr_GNU_C_Library_hits/strstr_GNU_C_Library_clocks: 1269/516
strstr_GNU_C_Library performance: 386KB/clock

Railgun_Quadruplet_hits/Railgun_Quadruplet_clocks: 1269/504
BEST! Railgun_Quadruplet performance: 417KB/clock

Boyer-Moore-Horspool_hits/Boyer-Moore-Horspool_clocks: 1269/625
Boyer-Moore-Horspool performance: 311KB/clock
```

The test results are taken from:
http://www.sanmayce.com/Downloads/_KAZE_Simplicius_Simplicissimus_Septupleton_r1++_strstr_SHORT-SHOWDOWN_r6+_Results_Merom_2166Mhz.txt

According to my superficial knowledge of (and experience with) Intel v12 and Microsoft v16 compilers my conclusion:
In two words: A MESS-MIX, I am still puzzled how small changes degrade or boost the performance across different options and compilers.
In some cases one outspeeds the other significantly but in other cases the TRADING-PLACES occurs!
The tests show that Microsoft compiler boosts RADICALLY the strstr-like function Railgun_Quadruplet while for Boyer-Moore-Horspool the compiler falls far behind Intel's one!
Of course when it comes to strstring in files-not-lines Boyer-Moore-Horspool rules monstrously, no doubt.

By the way (because currently I am not interested even a bit in speeding up the compression of Simplicius_Simplicissimus_Septupleton) it is very easy and mostly superfast to create .SS files by using the modified Leprechaun_quadrupleton_r14_minus (an external B-trees technique will be used instead of slow-and-memory-greedy Quicksort).
Instead of ripping the distinct 4-grams up to 31 (by default, but easily changeable through DEFINE in the source) the target will be all distinct 7bytes BBs.

Add-on from 2011-Sep-05:

```
strstr_SHORT-SHOWDOWN_Intel_v12_fast.exe:
strstr_SHORT-SHOWDOWN, revision 6++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
BM_HORSPOOL 8x2 i.e. average performance: 955KB/clock

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
Railgun_6pp 8x2 i.e. average performance: 924KB/clock
```

```
strstr_SHORT-SHOWDOWN_Intel_v12_O3.exe:
strstr_SHORT-SHOWDOWN, revision 6++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
BM_HORSPOOL 8x2 i.e. average performance: 955KB/clock
```

```
Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
Railgun_6pp 8x2 i.e. average performance: 978KB/clock
```

```
strstr_SHORT-SHOWDOWN_Intel_v12_O3_Qparallel.exe:
strstr_SHORT-SHOWDOWN, revision 6++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
BM_HORSPOOL 8x2 i.e. average performance: 953KB/clock

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
Railgun_6pp 8x2 i.e. average performance: 1033KB/clock
```

```
strstr_SHORT-SHOWDOWN_Intel_v12_Ox.exe:
strstr_SHORT-SHOWDOWN, revision 6++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
BM_HORSPOOL 8x2 i.e. average performance: 938KB/clock

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
Railgun_6pp 8x2 i.e. average performance: 978KB/clock
```

```
strstr_SHORT-SHOWDOWN_Intel_v12_Ox_Qparallel.exe:
strstr_SHORT-SHOWDOWN, revision 6++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
BM_HORSPOOL 8x2 i.e. average performance: 956KB/clock

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
...
Railgun_6pp 8x2 i.e. average performance: 978KB/clock
```

```
strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.exe:
strstr_SHORT-SHOWDOWN, revision 6++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), BM_HORSPOOL performance: 287KB/clock
Found ('to') 1076629 time(s), BM_HORSPOOL performance: 293KB/clock
Found ('TDK') 0 time(s), BM_HORSPOOL performance: 515KB/clock
Found ('the') 2114180 time(s), BM_HORSPOOL performance: 379KB/clock
Found ('fast') 5945 time(s), BM_HORSPOOL performance: 559KB/clock
Found ('easy') 5191 time(s), BM_HORSPOOL performance: 614KB/clock
Found ('grmbl') 0 time(s), BM_HORSPOOL performance: 759KB/clock
Found ('email') 1 time(s), BM_HORSPOOL performance: 713KB/clock
Found ('pasting') 2 time(s), BM_HORSPOOL performance: 990KB/clock
Found ('amazing') 323 time(s), BM_HORSPOOL performance: 990KB/clock
Found ('underdog') 4 time(s), BM_HORSPOOL performance: 1074KB/clock
Found ('superdog') 0 time(s), BM_HORSPOOL performance: 1069KB/clock
Found ('participants') 147 time(s), BM_HORSPOOL performance: 1603KB/clock
Found ('skillessness') 0 time(s), BM_HORSPOOL performance: 1433KB/clock
Found ('I should have known') 1 time(s), BM_HORSPOOL performance: 1422KB/clock
Found ('human consciousness') 519 time(s), BM_HORSPOOL performance: 1603KB/clock
BM_HORSPOOL 8x2 i.e. average performance: 666KB/clock

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), Railgun_6pp performance: 678KB/clock
Found ('to') 1076629 time(s), Railgun_6pp performance: 680KB/clock
Found ('TDK') 0 time(s), Railgun_6pp performance: 1167KB/clock
Found ('the') 2114180 time(s), Railgun_6pp performance: 678KB/clock
Found ('fast') 5945 time(s), Railgun_6pp performance: 918KB/clock
Found ('easy') 5191 time(s), Railgun_6pp performance: 990KB/clock
Found ('grmbl') 0 time(s), Railgun_6pp performance: 1287KB/clock
Found ('email') 1 time(s), Railgun_6pp performance: 1167KB/clock
Found ('pasting') 2 time(s), Railgun_6pp performance: 1603KB/clock
Found ('amazing') 323 time(s), Railgun_6pp performance: 1603KB/clock
Found ('underdog') 4 time(s), Railgun_6pp performance: 1820KB/clock
Found ('superdog') 0 time(s), Railgun_6pp performance: 1603KB/clock
Found ('participants') 147 time(s), Railgun_6pp performance: 2149KB/clock
Found ('skillessness') 0 time(s), Railgun_6pp performance: 2557KB/clock
Found ('I should have known') 1 time(s), Railgun_6pp performance: 2126KB/clock
Found ('human consciousness') 519 time(s), Railgun_6pp performance: 2557KB/clock
Railgun_6pp 8x2 i.e. average performance: 1219KB/clock
```

I didn't find how to squeeze more from the Intel v12 compiler, anyway again I am surprised how strong is the Microsoft v16 compiler. The two bottom-lines follow:

Microsoft compiled **Railgun_6pp** is ((1219-1033)/1033)*100% = **18+%** faster than Intel compiled **Railgun_6pp** best result.
Microsoft compiled **Railgun_6pp** is ((1219-956)/956)*100% = **27%** faster than Intel & Microsoft compiled **BM_HORSPOOL** best result.

As for the slightly tweaked new Simplicius_Simplicissimus_Septupleton:

```
Simplicius_Simplicissimus_Septupleton rev.1+++, written by Kaze.
Note1: This is the precursor of Simplicius Simplicissimus - a superfast-low-performance TEXT decompressor.
Note2: The niftiness lies in readiness for multi-threading and mostly in boosting the search by having the BBs.
       The decompressor would upload (at burst speed) the BB data and then read-and-decode one-by-one the
       triads (BB pool/array indexes), that is a simple copying. The pool houses up to 256*256*256 BBs/elements.
Note3: Compiler used: Intel(R) C++ Compiler XE for applications running on IA-32, Version 12.0.4.196 Build 20110427.
Note4: Compile line: icl /Ox /TcBuilding-Blocks_DUMPER.c /FaBuilding-Blocks_DUMPER /w /FAcs
Note5: Maximum size of input text file is arbitrary - 384MB, in fact 384MB+4*384MB must be less than 19??MB.
Note6: The file format is given by typing the compressed file e.g. D:\>type OSHO.TXT.SS.
       The header is 270 bytes long followed by BBs and triads/indexes and the eventual remainder/literals.
Note7: My benchmark text file OSHO.TXT 206,908,949 bytes where OSHO.TXT.SS 116,871,584 bytes for order 6:
       Decompressing OSHO.TXT.SS to RAM without Dumping to DRIVE time: 1704 clocks or 118579 KB/s, an awful result.
       For order 4 enforced: 156,174,067 OSHO.TXT.SS is being decompressed at 192804 KB/s.
       For order 7 enforced: 122,297,608 OSHO.TXT.SS is being decompressed at 85618 KB/s.
       For order 8 enforced: 149,243,106 OSHO.TXT.SS is being decompressed at 115396 KB/s.
       Obviously the fastest cache size is crucial, for OSHO.TXT 12MB BB pool vs 1MB L2 cache disbalance is the
       cause for this badly inferior performance compared to LZ L1 (32KB) cache-friendly variants.
       The test machine is Toshiba Satellite with Intel Merom 2166MHz, with rough 'memcpy' performance: 1646 MB/s.
Note8: Compile line: cl /Ox /TcBuilding-Blocks_DUMPER.c /FaBuilding-Blocks_DUMPER /w /FAcs
Note9: Major (but still inferior) decompressing tweak since r.1+++, this time Microsoft v16 excels:
       For order 7 enforced: 122,297,608 OSHO.TXT.SS is being decompressed at 170083 KB/s.
NoteA: Simplicius_Simplicissimus stands up for his name, the decompressing fragment is 13 instructions small:
       $LL74@main:
       a1 00 00 00 00   mov     eax, DWORD PTR _Patternlen
       03 c2            add     eax, edx
       8b 04 30         mov     eax, DWORD PTR [eax+esi]
       25 ff ff ff 00   and     eax, 16777215
       0f af c7         imul    eax, edi
       8b 1c 30         mov     ebx, DWORD PTR [eax+esi]
       89 19            mov     DWORD PTR [ecx], ebx
       8b 44 30 04      mov     eax, DWORD PTR [eax+esi+4]
       89 41 04         mov     DWORD PTR [ecx+4], eax
       83 c2 03         add     edx, 3
       03 cf            add     ecx, edi
       ff 4c 24 1c      dec     DWORD PTR tv977[esp+376]
       75 d7            jne     SHORT $LL74@main
NoteB: For more fast decompressing idea(s) you may contact me at sanmayce@sanmayce.com freely.
NoteC: Got it? The eighth note has two meanings: 1] for faster 2] for additional/other fast.
NoteD: Got it? The ninth note has two meanings: 1] for quicker/tighter 2] for additional/other quick/tight.

Allocating 384MB ... OK

Simplicius Simplicissimus is decompressing OSHO.TXT ...
Allocating 256MB for BBs pool + 384MB for RAM-to-RAM decompression ... OK
BBs order: 7
Original file size: 206908949
BBs pool size: 33622064

Decompression to RAM without Dumping to DRIVE time: 1235 clocks
Decompression to RAM without Dumping to DRIVE performance: 163610 KB/s or 159 MB/s
Total time: 3876 clocks

Benchmarking 'memcpy' by copying 197MB (OSHO.TXT size) ten times ...
Simplicius says for 'memcpy' performance: 1235 MB/s
Simplicius says for Decompression Ratio: 12%
```

Add-on from 2011-Sep-07:

How embarrassing: a bug appeared in (only) r.6++, now fixed.

```
// strstr_SHORT-SHOWDOWN.c, revision 6+++, 2011 Sep 07.
// homepage: www.sanmayce.com/Railgun/index.html

// 1] A BUG (which appeared in r.6++) crushed:
//    Fix #1: The actual fix for r.6+++:
//        //countSTATIC = cbPattern-2-3;
//        countSTATIC = cbPattern-2-2; // r.6+++ I suppose that the awful degradation comes from 2bytes more (from either 'if (countSTATIC<0) countSTATIC=0;' or 'count >0' fixes) which make the function unfitable in code cache lines?!
//    Fix #2: One possible fix for r.6+++:
//        // Next line fixes the BUG from r.6++: but with awful speed degradation! So the bug is fixed in the definitions by setting 'countSTATIC = cbPattern-2-2;', bug appears only for patterns with lengths of 4, The setback is one unnecessary comparison for 5bytes patterns, stupidly such setback exists (from before) for 4bytes as well.
//        //if (countSTATIC<0) countSTATIC=0;
//    Fix #3: Another possible fix for r.6+++:
//        // A BUG (in next line) crushed from r.6++: 'count !=0' becomes 'count >0' in r.6+++ but with awful speed degradation! So the bug is fixed outside the cycles by setting 'countSTATIC' from -1 to 0, bug appears only for patterns with lengths of 4.
```

```
//        while ( count !=0 && *(char *)(pbPattern+1+3+(countSTATIC-count)) == *(char *)(&pbTarget[i]+1+3+(countSTATIC-count)) ) {
    // if pattern length is 4 or 5 we have count=-1 and count=0 respectively i.e. no need of comparing in-between chars.

/*
The bug (appearing in BMH section only with pattern lengths of 4) which I overlooked comes in handy, it shows something important
    (as far as I understand), namely the significance of one instruction (2bytes of code)!
Disastrous degradation (1209KB/clock down to 910KB/clock) just from 'count !=0' replacement with 'count >0', how is this possible,
    maybe it is related to some microcode cache limits?!
I should surely like to hear (at sanmayce@sanmayce.com) from an expert what causes this 300MB difference.

For r.6++ (buggy) the 'while' code is 11 bytes:
$LL3@Railgun_Qu@2:
  009d2  8a 16              mov     dl, BYTE PTR [esi]
  009d4  3a 11              cmp     dl, BYTE PTR [ecx]
  009d6  75 05              jne     SHORT $LN68@Railgun_Qu@2
  009d8  41                 inc     ecx
  009d9  46                 inc     esi
  009da  48                 dec     eax
  009db  75 f5              jne     SHORT $LL3@Railgun_Qu@2
$LN68@Railgun_Qu@2:

For r.6+++ (fixed) the 'while' code is 11+2 bytes:
$LL3@Railgun_Qu@2:
  009d2  8a 16              mov     dl, BYTE PTR [esi]
  009d4  3a 11              cmp     dl, BYTE PTR [ecx]
  009d6  75 0f              jne     SHORT $LN2@Railgun_Qu@2
  009d8  48                 dec     eax
  009d9  41                 inc     ecx
  009da  46                 inc     esi
  009db  85 c0              test    eax, eax
  009dd  7f f3              jg      SHORT $LL3@Railgun_Qu@2
$LN59@Railgun_Qu@2:

Fragment r.6++ (buggy) [
strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.exe fly Dumbo fly
strstr_SHORT-SHOWDOWN, revision 6++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), Railgun_6pp performance: 645KB/clock
Found ('to') 1076629 time(s), Railgun_6pp performance: 678KB/clock
Found ('TDK') 0 time(s), Railgun_6pp performance: 1167KB/clock
Found ('the') 2114180 time(s), Railgun_6pp performance: 678KB/clock
Found ('fast') 5945 time(s), Railgun_6pp performance: 918KB/clock
Found ('easy') 5191 time(s), Railgun_6pp performance: 990KB/clock
Found ('grmbl') 0 time(s), Railgun_6pp performance: 1287KB/clock
Found ('email') 1 time(s), Railgun_6pp performance: 1167KB/clock
Found ('pasting') 2 time(s), Railgun_6pp performance: 1422KB/clock
Found ('amazing') 323 time(s), Railgun_6pp performance: 1603KB/clock
Found ('underdog') 4 time(s), Railgun_6pp performance: 1836KB/clock
Found ('superdog') 0 time(s), Railgun_6pp performance: 1603KB/clock
Found ('participants') 147 time(s), Railgun_6pp performance: 2557KB/clock
Found ('skillessness') 0 time(s), Railgun_6pp performance: 2126KB/clock
Found ('I should have known') 1 time(s), Railgun_6pp performance: 2126KB/clock
Found ('human consciousness') 519 time(s), Railgun_6pp performance: 2149KB/clock
Railgun_6pp 8x2 i.e. average performance: 1209KB/clock
Fragment r.6++ (buggy) ]

Fragment r.6+++ (fixed) with either Fix #2 or Fix #3 [
strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.exe fly Dumbo fly
strstr_SHORT-SHOWDOWN, revision 6+++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), Railgun_6pp performance: 678KB/clock
Found ('to') 1076629 time(s), Railgun_6pp performance: 678KB/clock
Found ('TDK') 0 time(s), Railgun_6pp performance: 1167KB/clock
Found ('the') 2114180 time(s), Railgun_6pp performance: 643KB/clock
Found ('fast') 5945 time(s), Railgun_6pp performance: 587KB/clock
Found ('easy') 5191 time(s), Railgun_6pp performance: 614KB/clock
Found ('grmbl') 0 time(s), Railgun_6pp performance: 805KB/clock
Found ('email') 1 time(s), Railgun_6pp performance: 713KB/clock
Found ('pasting') 2 time(s), Railgun_6pp performance: 990KB/clock
Found ('amazing') 323 time(s), Railgun_6pp performance: 990KB/clock
Found ('underdog') 4 time(s), Railgun_6pp performance: 1167KB/clock
Found ('superdog') 0 time(s), Railgun_6pp performance: 1074KB/clock
Found ('participants') 147 time(s), Railgun_6pp performance: 1603KB/clock
Found ('skillessness') 0 time(s), Railgun_6pp performance: 1422KB/clock
Found ('I should have known') 1 time(s), Railgun_6pp performance: 1603KB/clock
Found ('human consciousness') 519 time(s), Railgun_6pp performance: 1603KB/clock
Railgun_6pp 8x2 i.e. average performance: 910KB/clock
Fragment r.6+++ (fixed) with either Fix #2 or Fix #3 ]
```

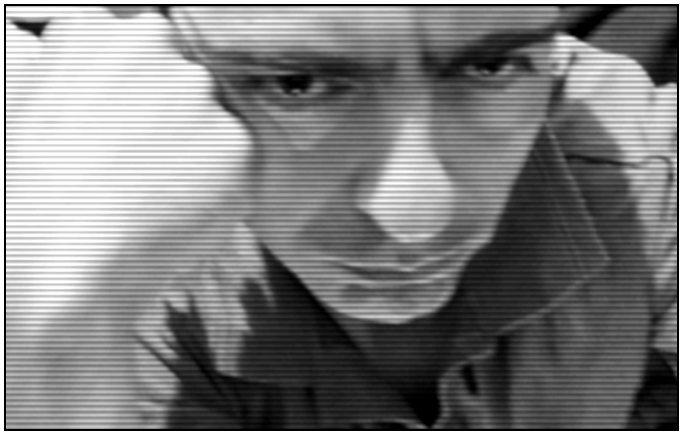
```
Fragment r.6+++ (fixed) with Fix #1 on Microsoft v16 [
strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.exe fly Dumbo fly
strstr_SHORT-SHOWDOWN, revision 6+++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), Railgun_6pp performance: 678KB/clock
Found ('to') 1076629 time(s), Railgun_6pp performance: 678KB/clock
Found ('TDK') 0 time(s), Railgun_6pp performance: 1167KB/clock
Found ('the') 2114180 time(s), Railgun_6pp performance: 716KB/clock
Found ('fast') 5945 time(s), Railgun_6pp performance: 918KB/clock
Found ('easy') 5191 time(s), Railgun_6pp performance: 990KB/clock
Found ('grmbl') 0 time(s), Railgun_6pp performance: 1287KB/clock
Found ('email') 1 time(s), Railgun_6pp performance: 1167KB/clock
Found ('pasting') 2 time(s), Railgun_6pp performance: 1433KB/clock
Found ('amazing') 323 time(s), Railgun_6pp performance: 1603KB/clock
Found ('underdog') 4 time(s), Railgun_6pp performance: 1820KB/clock
Found ('superdog') 0 time(s), Railgun_6pp performance: 1836KB/clock
Found ('participants') 147 time(s), Railgun_6pp performance: 2557KB/clock
Found ('skillessness') 0 time(s), Railgun_6pp performance: 2126KB/clock
Found ('I should have known') 1 time(s), Railgun_6pp performance: 2149KB/clock
Found ('human consciousness') 519 time(s), Railgun_6pp performance: 2525KB/clock
Railgun_6pp 8x2 i.e. average performance: 1213KB/clock
Fragment r.6+++ (fixed) with Fix #1 on Microsoft v16 ]

Fragment r.6+++ (fixed) with Fix #1 on Intel v12 fastest result [
strstr_SHORT-SHOWDOWN_Intel_v12_O3_Qparallel.exe fly Dumbo fly
strstr_SHORT-SHOWDOWN, revision 6+++, written by Kaze.

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), Railgun_6pp performance: 537KB/clock
Found ('to') 1076629 time(s), Railgun_6pp performance: 537KB/clock
Found ('TDK') 0 time(s), Railgun_6pp performance: 585KB/clock
Found ('the') 2114180 time(s), Railgun_6pp performance: 445KB/clock
Found ('fast') 5945 time(s), Railgun_6pp performance: 918KB/clock
Found ('easy') 5191 time(s), Railgun_6pp performance: 990KB/clock
Found ('grmbl') 0 time(s), Railgun_6pp performance: 1287KB/clock
Found ('email') 1 time(s), Railgun_6pp performance: 1167KB/clock
Found ('pasting') 2 time(s), Railgun_6pp performance: 1603KB/clock
Found ('amazing') 323 time(s), Railgun_6pp performance: 1603KB/clock
Found ('underdog') 4 time(s), Railgun_6pp performance: 1820KB/clock
Found ('superdog') 0 time(s), Railgun_6pp performance: 1836KB/clock
Found ('participants') 147 time(s), Railgun_6pp performance: 2126KB/clock
Found ('skillessness') 0 time(s), Railgun_6pp performance: 2126KB/clock
Found ('I should have known') 1 time(s), Railgun_6pp performance: 2149KB/clock
Found ('human consciousness') 519 time(s), Railgun_6pp performance: 2525KB/clock
Railgun_6pp 8x2 i.e. average performance: 1029KB/clock
Fragment r.6+++ (fixed) with Fix #1 on Intel v12 fastest result ]
*/
```

My pre-final conclusion:

It is hard to say which compiler excels even for small etude like this, I rather would stick to heavy tests than to draw quick conclusions. All-in-all Intel v12 with its **1029KB**/clock outspeeds Microsoft V16 with its **910KB**/clock, but only when ignorance of what-really-happens exists. I am still shocked how one line OUTSIDE the cycles (or 2bytes of code INSIDE) can inflict such an ugly damage: **910KB**/clock vs **1213KB**/clock.

These 300MB will haunt me until an expert explains the mystery.

The new package is downloadable at:
http://www.sanmayce.com/Downloads/_KAZE_Simplicius_Simplicissimus_Septupleton_r1+++_strstr_SHORT-SHOWDOWN_r6+++.7z



Georgi 'Kaze',
2011 Aug 31

Add-on from 2011-Sep-27:

Stomp stomp I arrived, here comes the Railgun_Quadruplet revision 7:

```
// Caution: For better speed the case 'if (cbPattern==1)' was removed, so Pattern must be longer than 1 char.
char * Railgun_Quadruplet_7 (char * pbTarget, char * pbPattern, unsigned long cbTarget, unsigned long cbPattern)
{
        char * pbTargetMax = pbTarget + cbTarget;
        register unsigned long ulHashPattern;
        unsigned long ulHashTarget;
        signed long count;
        signed long countSTATIC;

        unsigned char SINGLET;
        unsigned long Quadruplet2nd;
        unsigned long Quadruplet3rd;
        unsigned long Quadruplet4th;
        unsigned long AdvanceHopperGrass;

        long i;
        int a, j, bm_bc[ASIZE];
        unsigned char ch;

        if (cbPattern > cbTarget) return(NULL);

        if (cbPattern<4) {
                pbTarget = pbTarget+cbPattern;
                ulHashPattern = ( (*(char *)(pbPattern))<<8 ) + *(pbPattern+(cbPattern-1));
                if (cbPattern==3) {
                        for ( ;; ) {
                                if ( ulHashPattern == ( (*(char *)(pbTarget-3))<<8 ) + *(pbTarget-1) ) {
                                        if ( *(char *)(pbPattern+1) == *(char *)(pbTarget-2) ) return((pbTarget-3));
                                }
                                if ( (char)(ulHashPattern>>8) != *(pbTarget-2) ) pbTarget++;
                                pbTarget++;
                        if (pbTarget > pbTargetMax) return(NULL);
                        }
                } else {
                }
                        for ( ;; ) {
                                if ( ulHashPattern == ( (*(char *)(pbTarget-2))<<8 ) + *(pbTarget-1) ) return((pbTarget-2));
                                if ( (char)(ulHashPattern>>8) != *(pbTarget-1) ) pbTarget++;
                                pbTarget++;
                                if (pbTarget > pbTargetMax) return(NULL);
                        }
        } else {
                if (cbTarget<961) {
                        pbTarget = pbTarget+cbPattern;
                        ulHashPattern = *(unsigned long *)(pbPattern);
                        SINGLET = ulHashPattern & 0xFF;
                        Quadruplet2nd = SINGLET<<8;
                        Quadruplet3rd = SINGLET<<16;
                        Quadruplet4th = SINGLET<<24;
                        for ( ;; ) {
                                AdvanceHopperGrass = 0;
                                ulHashTarget = *(unsigned long *)(pbTarget-cbPattern);
                                if ( ulHashPattern == ulHashTarget ) {
                                        count = cbPattern-1;
                                        while ( count && *(char *)(pbPattern+(cbPattern-count)) == *(char *)(pbTarget-count) ) {
                                                if ( cbPattern-1==AdvanceHopperGrass+count && SINGLET != *(char *)(pbTarget-
count) ) AdvanceHopperGrass++;
                                                count--;
                                        }
                                        if ( count == 0) return((pbTarget-cbPattern));
                                } else {
                                        if ( Quadruplet2nd != (ulHashTarget & 0x0000FF00) ) {
                                                AdvanceHopperGrass++;
                                                if ( Quadruplet3rd != (ulHashTarget & 0x00FF0000) ) {
                                                        AdvanceHopperGrass++;
                                                        if ( Quadruplet4th != (ulHashTarget & 0xFF000000) )
AdvanceHopperGrass++;
                                                }
                                        }
                                }
                                AdvanceHopperGrass++;
                                pbTarget = pbTarget + AdvanceHopperGrass;
                                if (pbTarget > pbTargetMax) return(NULL);
                        }
                } else {
                        countSTATIC = cbPattern-2-2;
```

```
                        ulHashPattern = *(unsigned long *)(pbPattern);
                        for (a=0; a < ASIZE; a++) bm_bc[a]=cbPattern;
                        for (j=0; j < cbPattern-1; j++) bm_bc[pbPattern[j]]=cbPattern-j-1;
                        i=0;
                        while (i <= cbTarget-cbPattern) {
                                if ( *(unsigned long *)&pbTarget[i] == ulHashPattern ) {
                                        count = countSTATIC;
                                        while ( count !=0 && *(char *)(pbPattern+1+3+(countSTATIC-count)) == *(char
*)(&pbTarget[i]+1+3+(countSTATIC-count)) ) count--;
                                        if ( count == 0) return(pbTarget+i);
                                }
                                i= i + bm_bc[(unsigned char)pbTarget[i+cbPattern-1]];
                        }
                        return(NULL);
                }
        }
}
```

[strstr_SHORT-SHOWDOWN_Intel_v12_fast.exe:]

Note: Executing the next two tests 256 times i.e. the search is for 256x8x2 patterns!

```
Railgun_Quadruplet_6pp 8x2 i.e. average performance: 1475KB/clock
ReallyTraversed: 310,477,846,528 bytes

Railgun_Quadruplet_7 8x2 i.e. average performance: 1394KB/clock
ReallyTraversed: 310,477,846,528 bytes
```

Note: Executing the next two tests 256 times i.e. the search is for 256x52 patterns, all utilizing BMH!

```
Railgun_Quadruplet_6pp 52 i.e. average performance: 1751KB/clock
TotalRoughSearchTime: 1,083,626 clocks
ReallyTraversed: 1,943,883,635,456 bytes

Railgun_Quadruplet_7 52 i.e. average performance: 1701KB/clock
TotalRoughSearchTime: 1,115,611 clocks
ReallyTraversed: 1,943,883,635,456 bytes
```

```
BM_HORSPOOL 8x2 i.e. average performance: 971KB/clock
Railgun_6pp 8x2 i.e. average performance: 933KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 908KB/clock
```

[strstr_SHORT-SHOWDOWN_Intel_v12_O3.exe:]

Note: Executing the next two tests 256 times i.e. the search is for 256x8x2 patterns!

```
Railgun_Quadruplet_6pp 8x2 i.e. average performance: 1407KB/clock
ReallyTraversed: 310,477,846,528 bytes

Railgun_Quadruplet_7 8x2 i.e. average performance: 1319KB/clock
ReallyTraversed: 310,477,846,528 bytes
```

Note: Executing the next two tests 256 times i.e. the search is for 256x52 patterns, all utilizing BMH!

```
Railgun_Quadruplet_6pp 52 i.e. average performance: 1773KB/clock
TotalRoughSearchTime: 1,070,220 clocks
ReallyTraversed: 1,943,883,635,456 bytes

Railgun_Quadruplet_7 52 i.e. average performance: 1685KB/clock
TotalRoughSearchTime: 1,126,142 clocks
ReallyTraversed: 1,943,883,635,456 bytes
```

```
BM_HORSPOOL 8x2 i.e. average performance: 970KB/clock
Railgun_6pp 8x2 i.e. average performance: 1054KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 1076KB/clock
```

[strstr_SHORT-SHOWDOWN_Intel_v12_O3_Qparallel.exe:]

Note: Executing the next two tests 256 times i.e. the search is for 256x8x2 patterns!

```
Railgun_Quadruplet_6pp 8x2 i.e. average performance: 1414KB/clock
ReallyTraversed: 310,477,846,528 bytes

Railgun_Quadruplet_7 8x2 i.e. average performance: 1319KB/clock
ReallyTraversed: 310,477,846,528 bytes
```

Note: Executing the next two tests 256 times i.e. the search is for 256x52 patterns, all utilizing BMH!

```
Railgun_Quadruplet_6pp 52 i.e. average performance: 1761KB/clock
TotalRoughSearchTime: 1,077,767 clocks
ReallyTraversed: 1,943,883,635,456 bytes

Railgun_Quadruplet_7 52 i.e. average performance: 1664KB/clock
TotalRoughSearchTime: 1,140,188 clocks
ReallyTraversed: 1,943,883,635,456 bytes
```

```
BM_HORSPOOL 8x2 i.e. average performance: 961KB/clock
Railgun_6pp 8x2 i.e. average performance: 1046KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 1068KB/clock
```

[strstr_SHORT-SHOWDOWN_Intel_v12_Ox.exe:]

```
Note: Executing the next two tests 256 times i.e. the search is for 256x8x2 patterns!
Railgun_Quadruplet_6pp 8x2 i.e. average performance: 1393KB/clock
ReallyTraversed: 310,477,846,528 bytes

Railgun_Quadruplet_7 8x2 i.e. average performance: 1305KB/clock
ReallyTraversed: 310,477,846,528 bytes

Note: Executing the next two tests 256 times i.e. the search is for 256x52 patterns, all utilizing BMH!
Railgun_Quadruplet_6pp 52 i.e. average performance: 1744KB/clock
TotalRoughSearchTime: 1,088,048 clocks
ReallyTraversed: 1,943,883,635,456 bytes

Railgun_Quadruplet_7 52 i.e. average performance: 1680KB/clock
TotalRoughSearchTime: 1,129,720 clocks
ReallyTraversed: 1,943,883,635,456 bytes

BM_HORSPOOL 8x2 i.e. average performance: 967KB/clock
Railgun_6pp 8x2 i.e. average performance: 995KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 1075KB/clock
```

[strstr_SHORT-SHOWDOWN_Intel_v12_Ox_Qparallel.exe:]

```
Note: Executing the next two tests 256 times i.e. the search is for 256x8x2 patterns!
Railgun_Quadruplet_6pp 8x2 i.e. average performance: 1395KB/clock
ReallyTraversed: 310,477,846,528 bytes

Railgun_Quadruplet_7 8x2 i.e. average performance: 1307KB/clock
ReallyTraversed: 310,477,846,528 bytes

Note: Executing the next two tests 256 times i.e. the search is for 256x52 patterns, all utilizing BMH!
Railgun_Quadruplet_6pp 52 i.e. average performance: 1749KB/clock
TotalRoughSearchTime: 1,085,361 clocks
ReallyTraversed: 1,943,883,635,456 bytes

Railgun_Quadruplet_7 52 i.e. average performance: 1686KB/clock
TotalRoughSearchTime: 1,125,704 clocks
ReallyTraversed: 1,943,883,635,456 bytes

BM_HORSPOOL 8x2 i.e. average performance: 968KB/clock
Railgun_6pp 8x2 i.e. average performance: 1047KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 1008KB/clock
```

[strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.exe:]

```
Note: Executing the next two tests 256 times i.e. the search is for 256x8x2 patterns!

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ... Not-Counting-Hits-Just-Returns-First-One
Found ('an') at 157 position, Railgun_Quadruplet_6pp performance: 0KB/clock
Found ('to') at 853 position, Railgun_Quadruplet_6pp performance: 0KB/clock
Found ('TDK') at -4325408 position, Railgun_Quadruplet_6pp performance: 918KB/clock
Found ('the') at 511 position, Railgun_Quadruplet_6pp performance: 0KB/clock
Found ('fast') at 42381 position, Railgun_Quadruplet_6pp performance: 41KB/clock
Found ('easy') at 30163 position, Railgun_Quadruplet_6pp performance: 29KB/clock
Found ('grmbl') at -4325408 position, Railgun_Quadruplet_6pp performance: 1167KB/clock
Found ('email') at 69587982 position, Railgun_Quadruplet_6pp performance: 1061KB/clock
Found ('pasting') at 134600126 position, Railgun_Quadruplet_6pp performance: 1398KB/clock
Found ('amazing') at 1629806 position, Railgun_Quadruplet_6pp performance: 1591KB/clock
Found ('underdog') at 137424498 position, Railgun_Quadruplet_6pp performance: 1698KB/clock
Found ('superdog') at -4325408 position, Railgun_Quadruplet_6pp performance: 1820KB/clock
Found ('participants') at 16060 position, Railgun_Quadruplet_6pp performance: 15KB/clock
Found ('skillessness') at -4325408 position, Railgun_Quadruplet_6pp performance: 2126KB/clock
Found ('I should have known') at 41831880 position, Railgun_Quadruplet_6pp performance: 2553KB/clock
Found ('human consciousness') at 3863 position, Railgun_Quadruplet_6pp performance: 3KB/clock
Railgun_Quadruplet_6pp 8x2 i.e. average performance: 1410KB/clock
ReallyTraversed: 310,477,846,528 bytes

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ... Not-Counting-Hits-Just-Returns-First-One
Found ('an') at 157 position, Railgun_Quadruplet_7 performance: 0KB/clock
Found ('to') at 853 position, Railgun_Quadruplet_7 performance: 0KB/clock
Found ('TDK') at -4325408 position, Railgun_Quadruplet_7 performance: 990KB/clock
Found ('the') at 511 position, Railgun_Quadruplet_7 performance: 0KB/clock
Found ('fast') at 42381 position, Railgun_Quadruplet_7 performance: 41KB/clock
Found ('easy') at 30163 position, Railgun_Quadruplet_7 performance: 29KB/clock
Found ('grmbl') at -4325408 position, Railgun_Quadruplet_7 performance: 1167KB/clock
Found ('email') at 69587982 position, Railgun_Quadruplet_7 performance: 1078KB/clock
Found ('pasting') at 134600126 position, Railgun_Quadruplet_7 performance: 1383KB/clock
Found ('amazing') at 1629806 position, Railgun_Quadruplet_7 performance: 1591KB/clock
Found ('underdog') at 137424498 position, Railgun_Quadruplet_7 performance: 1412KB/clock
Found ('superdog') at -4325408 position, Railgun_Quadruplet_7 performance: 1603KB/clock
Found ('participants') at 16060 position, Railgun_Quadruplet_7 performance: 15KB/clock
Found ('skillessness') at -4325408 position, Railgun_Quadruplet_7 performance: 2126KB/clock
Found ('I should have known') at 41831880 position, Railgun_Quadruplet_7 performance: 2553KB/clock
```

```
Found ('human consciousness') at 3863 position, Railgun_Quadruplet_7 performance: 3KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 1377KB/clock
ReallyTraversed: 310,477,846,528 bytes

Note: Executing the next two tests 256 times i.e. the search is for 256x52 patterns, all utilizing BMH!

Doing Search for 52 Patterns into String(206908949bytes) as-one-line ... Not-Counting-Hits-Just-Returns-First-One
Found ('fast') at 42381 position, Railgun_Quadruplet_6pp performance: 41KB/clock
Found ('from') at 996 position, Railgun_Quadruplet_6pp performance: 0KB/clock
Found ('SR71') at -4325408 position, Railgun_Quadruplet_6pp performance: 990KB/clock
Found ('SONY') at -4325408 position, Railgun_Quadruplet_6pp performance: 990KB/clock
Found ('Lexx') at -4325408 position, Railgun_Quadruplet_6pp performance: 990KB/clock
Found ('Rabea') at -4325408 position, Railgun_Quadruplet_6pp performance: 990KB/clock
Found ('makes') at 15096 position, Railgun_Quadruplet_6pp performance: 14KB/clock
Found ('punny') at -4325408 position, Railgun_Quadruplet_6pp performance: 1167KB/clock
Found ('funny') at 1110233 position, Railgun_Quadruplet_6pp performance: 1084KB/clock
Found ('Orion') at 180456170 position, Railgun_Quadruplet_6pp performance: 1018KB/clock
Found ('lemon') at 4043605 position, Railgun_Quadruplet_6pp performance: 3948KB/clock
Found ('Hi-Fi') at -4325408 position, Railgun_Quadruplet_6pp performance: 1069KB/clock
Found ('Tesla') at -4325408 position, Railgun_Quadruplet_6pp performance: 990KB/clock
Found ('Apache') at 79960763 position, Railgun_Quadruplet_6pp performance: 1239KB/clock
Found ('monkey') at 481771 position, Railgun_Quadruplet_6pp performance: 470KB/clock
Found ('ramjet') at -4325408 position, Railgun_Quadruplet_6pp performance: 1167KB/clock
Found ('fallen') at 31696 position, Railgun_Quadruplet_6pp performance: 30KB/clock
Found ('Albert') at 1156344 position, Railgun_Quadruplet_6pp performance: 1129KB/clock
Found ('Toshiba') at -4325408 position, Railgun_Quadruplet_6pp performance: 1422KB/clock
Found ('grammar') at 1759745 position, Railgun_Quadruplet_6pp performance: 1718KB/clock
Found ('pharaoh') at 31632134 position, Railgun_Quadruplet_6pp performance: 965KB/clock
Found ('decades') at 7374902 position, Railgun_Quadruplet_6pp performance: 720KB/clock
Found ('#define') at -4325408 position, Railgun_Quadruplet_6pp performance: 1287KB/clock
Found ('Delerium') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('numberly') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('stellify') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('elephant') at 1495572 position, Railgun_Quadruplet_6pp performance: 1460KB/clock
Found ('Layalina') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('Fibonacci') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('profanism') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('butterfly') at 85068 position, Railgun_Quadruplet_6pp performance: 83KB/clock
Found ('Beth Ditto') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('Got Nuffin') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('fly fly fly') at -4325408 position, Railgun_Quadruplet_6pp performance: 1820KB/clock
Found ('quintillion') at -4325408 position, Railgun_Quadruplet_6pp performance: 1836KB/clock
Found ('Sick Of Love') at -4325408 position, Railgun_Quadruplet_6pp performance: 1603KB/clock
Found ('Love Lockdown') at -4325408 position, Railgun_Quadruplet_6pp performance: 2126KB/clock
Found ('Dannii Minogue') at -4325408 position, Railgun_Quadruplet_6pp performance: 1836KB/clock
Found ('299,792,458 m/s') at -4325408 position, Railgun_Quadruplet_6pp performance: 2126KB/clock
Found ('Extreme-Fidelity') at -4325408 position, Railgun_Quadruplet_6pp performance: 2557KB/clock
Found ('I should have known') at 41831880 position, Railgun_Quadruplet_6pp performance: 2403KB/clock
Found ('human consciousness') at 3863 position, Railgun_Quadruplet_6pp performance: 3KB/clock
Found ('Truth's your only fear') at -4325408 position, Railgun_Quadruplet_6pp performance: 2149KB/clock
Found ('Never thought I'd do it') at -4325408 position, Railgun_Quadruplet_6pp performance: 2126KB/clock
Found ('innocence over fading fast') at -4325408 position, Railgun_Quadruplet_6pp performance: 2126KB/clock
Found ('I hit the ground runnin' free') at -4325408 position, Railgun_Quadruplet_6pp performance: 2126KB/clock
Found ('I colour out the darkest clouds') at -4325408 position, Railgun_Quadruplet_6pp performance: 2557KB/clock
Found ('But he's looking right through me') at -4325408 position, Railgun_Quadruplet_6pp performance: 2126KB/clock
Found ('I'm living in an age that calls darkness light') at -4325408 position, Railgun_Quadruplet_6pp performance: 2557KB/clock
Found ('Following the wanderings of a dream - a dream that keeps my soul alive') at -4325408 position, Railgun_Quadruplet_6pp
performance: 2557KB/clock
Found ('I notice what matters and I got nothing to lose but darkness and shadows') at -4325408 position, Railgun_Quadruplet_6pp
performance: 3207KB/clock
Found ('Then, singing among the savage branches, it impales itself upon the longest, sharpest spine. And, dying, it rises above
its own agony to outcarol the lark and the nightingale.') at -4325408 position, Railgun_Quadruplet_6pp performance:
2557KB/clock
Railgun_Quadruplet_6pp 52 i.e. average performance: 1584KB/clock
TotalRoughSearchTime: 1,198,329 clocks
ReallyTraversed: 1,943,883,635,456 bytes

Doing Search for 52 Patterns into String(206908949bytes) as-one-line ... Not-Counting-Hits-Just-Returns-First-One
Found ('fast') at 42381 position, Railgun_Quadruplet_7 performance: 41KB/clock
Found ('from') at 996 position, Railgun_Quadruplet_7 performance: 0KB/clock
Found ('SR71') at -4325408 position, Railgun_Quadruplet_7 performance: 922KB/clock
Found ('SONY') at -4325408 position, Railgun_Quadruplet_7 performance: 985KB/clock
Found ('Lexx') at -4325408 position, Railgun_Quadruplet_7 performance: 922KB/clock
Found ('Rabea') at -4325408 position, Railgun_Quadruplet_7 performance: 1069KB/clock
Found ('makes') at 15096 position, Railgun_Quadruplet_7 performance: 14KB/clock
Found ('punny') at -4325408 position, Railgun_Quadruplet_7 performance: 1167KB/clock
Found ('funny') at 1110233 position, Railgun_Quadruplet_7 performance: 1084KB/clock
Found ('Orion') at 180456170 position, Railgun_Quadruplet_7 performance: 1122KB/clock
Found ('lemon') at 4043605 position, Railgun_Quadruplet_7 performance: 232KB/clock
Found ('Hi-Fi') at -4325408 position, Railgun_Quadruplet_7 performance: 1174KB/clock
Found ('Tesla') at -4325408 position, Railgun_Quadruplet_7 performance: 1167KB/clock
Found ('Apache') at 79960763 position, Railgun_Quadruplet_7 performance: 1220KB/clock
Found ('monkey') at 481771 position, Railgun_Quadruplet_7 performance: 470KB/clock
Found ('ramjet') at -4325408 position, Railgun_Quadruplet_7 performance: 1287KB/clock
Found ('fallen') at 31696 position, Railgun_Quadruplet_7 performance: 30KB/clock
Found ('Albert') at 1156344 position, Railgun_Quadruplet_7 performance: 1129KB/clock
Found ('Toshiba') at -4325408 position, Railgun_Quadruplet_7 performance: 1422KB/clock
Found ('grammar') at 1759745 position, Railgun_Quadruplet_7 performance: 1718KB/clock
Found ('pharaoh') at 31632134 position, Railgun_Quadruplet_7 performance: 1930KB/clock
```

```
Found ('decades') at 7374902 position, Railgun_Quadruplet_7 performance: 7202KB/clock
Found ('#define') at -4325408 position, Railgun_Quadruplet_7 performance: 1422KB/clock
Found ('Delerium') at -4325408 position, Railgun_Quadruplet_7 performance: 1603KB/clock
Found ('numberly') at -4325408 position, Railgun_Quadruplet_7 performance: 1603KB/clock
Found ('stellify') at -4325408 position, Railgun_Quadruplet_7 performance: 1603KB/clock
Found ('elephant') at 1495572 position, Railgun_Quadruplet_7 performance: 1460KB/clock
Found ('Layalina') at -4325408 position, Railgun_Quadruplet_7 performance: 1603KB/clock
Found ('Fibonacci') at -4325408 position, Railgun_Quadruplet_7 performance: 1836KB/clock
Found ('profanism') at -4325408 position, Railgun_Quadruplet_7 performance: 1820KB/clock
Found ('butterfly') at 85068 position, Railgun_Quadruplet_7 performance: 83KB/clock
Found ('Beth Ditto') at -4325408 position, Railgun_Quadruplet_7 performance: 1603KB/clock
Found ('Got Nuffin') at -4325408 position, Railgun_Quadruplet_7 performance: 1836KB/clock
Found ('fly fly fly') at -4325408 position, Railgun_Quadruplet_7 performance: 1836KB/clock
Found ('quintillion') at -4325408 position, Railgun_Quadruplet_7 performance: 2126KB/clock
Found ('Sick Of Love') at -4325408 position, Railgun_Quadruplet_7 performance: 1820KB/clock
Found ('Love Lockdown') at -4325408 position, Railgun_Quadruplet_7 performance: 2557KB/clock
Found ('Dannii Minogue') at -4325408 position, Railgun_Quadruplet_7 performance: 2149KB/clock
Found ('299,792,458 m/s') at -4325408 position, Railgun_Quadruplet_7 performance: 2126KB/clock
Found ('Extreme-Fidelity') at -4325408 position, Railgun_Quadruplet_7 performance: 2557KB/clock
Found ('I should have known') at 41831880 position, Railgun_Quadruplet_7 performance: 2403KB/clock
Found ('human consciousness') at 3863 position, Railgun_Quadruplet_7 performance: 3KB/clock
Found ('Truth's your only fear') at -4325408 position, Railgun_Quadruplet_7 performance: 2126KB/clock
Found ('Never thought I'd do it') at -4325408 position, Railgun_Quadruplet_7 performance: 2557KB/clock
Found ('innocence over fading fast') at -4325408 position, Railgun_Quadruplet_7 performance: 2126KB/clock
Found ('I hit the ground runnin' free') at -4325408 position, Railgun_Quadruplet_7 performance: 3207KB/clock
Found ('I colour out the darkest clouds') at -4325408 position, Railgun_Quadruplet_7 performance: 2557KB/clock
Found ('But he's looking right through me') at -4325408 position, Railgun_Quadruplet_7 performance: 2557KB/clock
Found ('I'm living in an age that calls darkness light') at -4325408 position, Railgun_Quadruplet_7 performance: 2557KB/clock
Found ('Following the wanderings of a dream - a dream that keeps my soul alive') at -4325408 position, Railgun_Quadruplet_7
performance: 3157KB/clock
Found ('I notice what matters and I got nothing to lose but darkness and shadows') at -4325408 position, Railgun_Quadruplet_7
performance: 3207KB/clock
Found ('Then, singing among the savage branches, it impales itself upon the longest, sharpest spine. And, dying, it rises above
its own agony to outcarol the lark and the nightingale.') at -4325408 position, Railgun_Quadruplet_7 performance: 2525KB/clock
Railgun_Quadruplet_7 52 i.e. average performance: 1659KB/clock
TotalRoughSearchTime: 1,143,580 clocks
ReallyTraversed: 1,943,883,635,456 bytes

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), BM_HORSPOOL performance: 287KB/clock
Found ('to') 1076629 time(s), BM_HORSPOOL performance: 293KB/clock
Found ('TDK') 0 time(s), BM_HORSPOOL performance: 495KB/clock
Found ('the') 2114180 time(s), BM_HORSPOOL performance: 379KB/clock
Found ('fast') 5945 time(s), BM_HORSPOOL performance: 561KB/clock
Found ('easy') 5191 time(s), BM_HORSPOOL performance: 614KB/clock
Found ('grmbl') 0 time(s), BM_HORSPOOL performance: 805KB/clock
Found ('email') 1 time(s), BM_HORSPOOL performance: 713KB/clock
Found ('pasting') 2 time(s), BM_HORSPOOL performance: 990KB/clock
Found ('amazing') 323 time(s), BM_HORSPOOL performance: 990KB/clock
Found ('underdog') 4 time(s), BM_HORSPOOL performance: 1167KB/clock
Found ('superdog') 0 time(s), BM_HORSPOOL performance: 1074KB/clock
Found ('participants') 147 time(s), BM_HORSPOOL performance: 1422KB/clock
Found ('skillessness') 0 time(s), BM_HORSPOOL performance: 1603KB/clock
Found ('I should have known') 1 time(s), BM_HORSPOOL performance: 1433KB/clock
Found ('human consciousness') 519 time(s), BM_HORSPOOL performance: 1820KB/clock
BM_HORSPOOL 8x2 i.e. average performance: 670KB/clock

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), Railgun_6pp performance: 716KB/clock
Found ('to') 1076629 time(s), Railgun_6pp performance: 645KB/clock
Found ('TDK') 0 time(s), Railgun_6pp performance: 1278KB/clock
Found ('the') 2114180 time(s), Railgun_6pp performance: 678KB/clock
Found ('fast') 5945 time(s), Railgun_6pp performance: 990KB/clock
Found ('easy') 5191 time(s), Railgun_6pp performance: 990KB/clock
Found ('grmbl') 0 time(s), Railgun_6pp performance: 1287KB/clock
Found ('email') 1 time(s), Railgun_6pp performance: 1167KB/clock
Found ('pasting') 2 time(s), Railgun_6pp performance: 1603KB/clock
Found ('amazing') 323 time(s), Railgun_6pp performance: 1603KB/clock
Found ('underdog') 4 time(s), Railgun_6pp performance: 1836KB/clock
Found ('superdog') 0 time(s), Railgun_6pp performance: 1820KB/clock
Found ('participants') 147 time(s), Railgun_6pp performance: 2557KB/clock
Found ('skillessness') 0 time(s), Railgun_6pp performance: 2126KB/clock
Found ('I should have known') 1 time(s), Railgun_6pp performance: 2149KB/clock
Found ('human consciousness') 519 time(s), Railgun_6pp performance: 2557KB/clock
Railgun_6pp 8x2 i.e. average performance: 1246KB/clock

Doing Search for 8x2 Patterns into String(206908949bytes) as-one-line ...
Found ('an') 1987797 time(s), Railgun_Quadruplet_7 performance: 678KB/clock
Found ('to') 1076629 time(s), Railgun_Quadruplet_7 performance: 678KB/clock
Found ('TDK') 0 time(s), Railgun_Quadruplet_7 performance: 1167KB/clock
Found ('the') 2114180 time(s), Railgun_Quadruplet_7 performance: 645KB/clock
Found ('fast') 5945 time(s), Railgun_Quadruplet_7 performance: 1069KB/clock
Found ('easy') 5191 time(s), Railgun_Quadruplet_7 performance: 990KB/clock
Found ('grmbl') 0 time(s), Railgun_Quadruplet_7 performance: 1433KB/clock
Found ('email') 1 time(s), Railgun_Quadruplet_7 performance: 1278KB/clock
Found ('pasting') 2 time(s), Railgun_Quadruplet_7 performance: 1603KB/clock
Found ('amazing') 323 time(s), Railgun_Quadruplet_7 performance: 1603KB/clock
Found ('underdog') 4 time(s), Railgun_Quadruplet_7 performance: 1836KB/clock
Found ('superdog') 0 time(s), Railgun_Quadruplet_7 performance: 1836KB/clock
```

Found ('participants') 147 time(s), Railgun_Quadruplet_7 performance: 2557KB/clock
Found ('skillessness') 0 time(s), Railgun_Quadruplet_7 performance: 2525KB/clock
Found ('I should have known') 1 time(s), Railgun_Quadruplet_7 performance: 2149KB/clock
Found ('human consciousness') 519 time(s), Railgun_Quadruplet_7 performance: 3157KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 1266KB/clock

Note1: The code is 32bit both for Intel and Microsoft, the machine has Intel Merom 2166MHz CPU with DDR2 667MHz, OS: Windows XP 32bit.
Note2: The inconsistency in time is a mystery for me, for example the heaviest and most precise test (52 patterns and 1000 seconds in length) gives:

[strstr_SHORT-SHOWDOWN_Microsoft_v16_Ox.exe:]
The variants returning the offset of first hit only:
Railgun_Quadruplet_6pp 52 i.e. average performance: 1584KB/clock
Railgun_Quadruplet_7 52 i.e. average performance: 1659KB/clock

The variants counting all hits:
Railgun_6pp 8x2 i.e. average performance: 1246KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 1266KB/clock

[strstr_SHORT-SHOWDOWN_Intel_v12_O3.exe:]
The variants returning the offset of first hit only:
Railgun_Quadruplet_6pp 52 i.e. average performance: 1773KB/clock
Railgun_Quadruplet_7 52 i.e. average performance: 1685KB/clock

The variants counting all hits:
Railgun_6pp 8x2 i.e. average performance: 1054KB/clock
Railgun_Quadruplet_7 8x2 i.e. average performance: 1076KB/clock

Note3: For Railgun_Quadruplet_7 on CPUs with fast DWORD (compared to BYTE) memory fetching I expect significant boost, AFAIK the Intel i7 is one of those, unfortunately I lack the opportunity to play with it.

Or the two bottom-lines:
The FASTEST variant counting all hits: Railgun_Quadruplet_7 8x2 with average performance: **1266KB**/clock (best performance achieved with **Microsoft_v16_Ox**)!
The FASTEST variant returning the offset of first hit only: Railgun_Quadruplet_6pp 52 with average performance: **1773KB**/clock (1,943,883,635,456 bytes / 1,070,220 clocks = 1732 MB/s best performance achieved with **Intel_v12_O3**)!

I am a bit confused, again, these trading places are stopless.