CS 280: Assignment 1

Emaad Ahmed Manzoor, 129110

Task 1

`papi_mem_info` provides us information about the processor cache sizes:

	L1 Data Cache	L1 Instruction Cache	L2 Unified Cache
Total Size (KB)	32	32	6144
Line Size (B)	64	64	64
Number of Lines	512	512	98304
Associativity	8	8	24

`papi_avail` provides us further information about the CPU make:

Model Code	Intel Xeon CPU E5420 2.50GHz	**NUMA nodes**	1
CPU Revision	10	**CPUs per node**	8
CPUID Info	Family: 6 Model: 23 Stepping: 10	**Total CPUs**	8
CPU Max Mhz	2490	**Running in a VM**	no
CPU Min Mhz	1992	**Number of hardware counters**	5
Hardware threads per core	1	**Max multiplex counters**	64
Cores per socket	4		

Task 2

Instrumentation Process

I used the PAPI low level API so I could check for unsupported/conflicting events and handle them gracefully. All the code described here is present in `driver.c`.

The following steps were taken to instrument the code:

1. Define the events we would like to monitor in the `allEvents` array.

2. For each of the requested events, attempt adding them to a PAPI *eventSet* by calling `PAPI_add_event`. If an event cannot be added due to a conflict or being unavailable, report it and continue with the remaining requested events.
3. Keep track of the events successfully added.
4. Start the hardware counters with `PAPI_start`.
5. Stop and store the hardware counters with `PAPI_stop`.
6. Output counter values.

The following events were attempted to be tracked:

PAPI_L1_TCA	PAPI_L1_TCM
PAPI_L1_ICA	PAPI_L1_ICM
PAPI_L1_DCA	PAPI_L1_DCM
PAPI_L2_TCM	PAPI_L2_TCA
PAPI_L2_ICM	PAPI_L2_ICA
PAPI_L2_DCM	PAPI_L2_DCA
PAPI_FP_OPS	PAPI_BR_INS

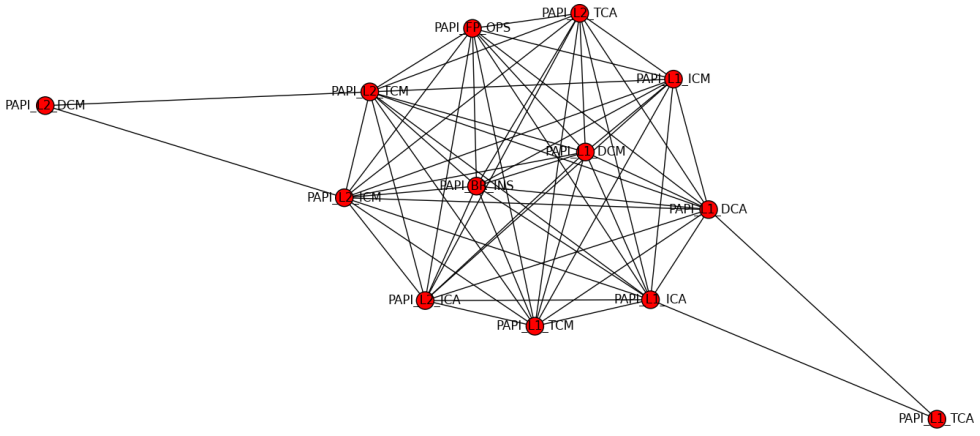


Figure: PAPI event compatibility graph.
Nodes are events. Compatible events are connected by an edge.

Many event pairs were not compatible, as reported by the `papi_event_chooser` tool. Hence the events needed to be separated into groups. Minimizing the number of such

groups of mutually non-conflicting events is equivalent to finding maximal cliques in the event compatibility graph (see figure). In this case, there are 3 maximal cliques.

However, the conflicting events reported by papi_event_chooser changed when combinations of events were queried, and hence the maximal clique formulation is not applicable directly as presented above.

I finally arrived at the following grouping of events:

Group 1	PAPI_L1_TCA	PAPI_L1_ICA	PAPI_L1_DCA
Group 2	PAPI_L2_TCM	PAPI_L2_ICM	PAPI_L2_DCM
Group 3	PAPI_L2_DCA		
Group 4	PAPI_L1_TCM		
Group 5	PAPI_L1_ICM	PAPI_L1_DCM	
Group 6	PAPI_L2_TCA	PAPI_L2_ICA	
Group 7	PAPI_BR_CN	PAPI_FP_OPS	

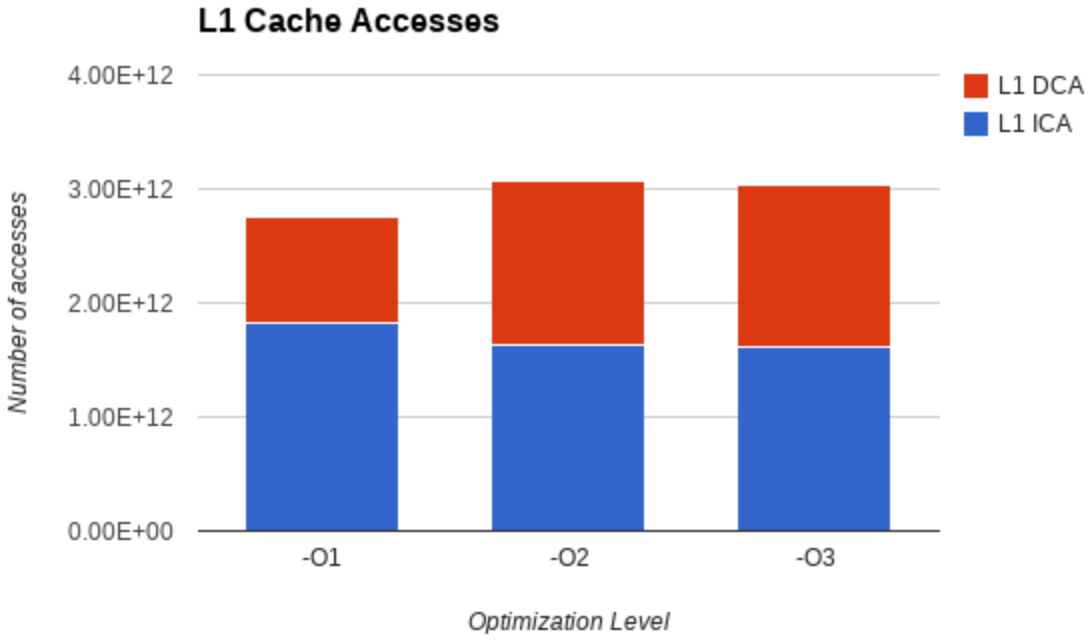
I created a separate binary for each group by commenting out the lines of code corresponding to events not in that group. Each group binary was then run 5 times and the results were accumulated and averaged.

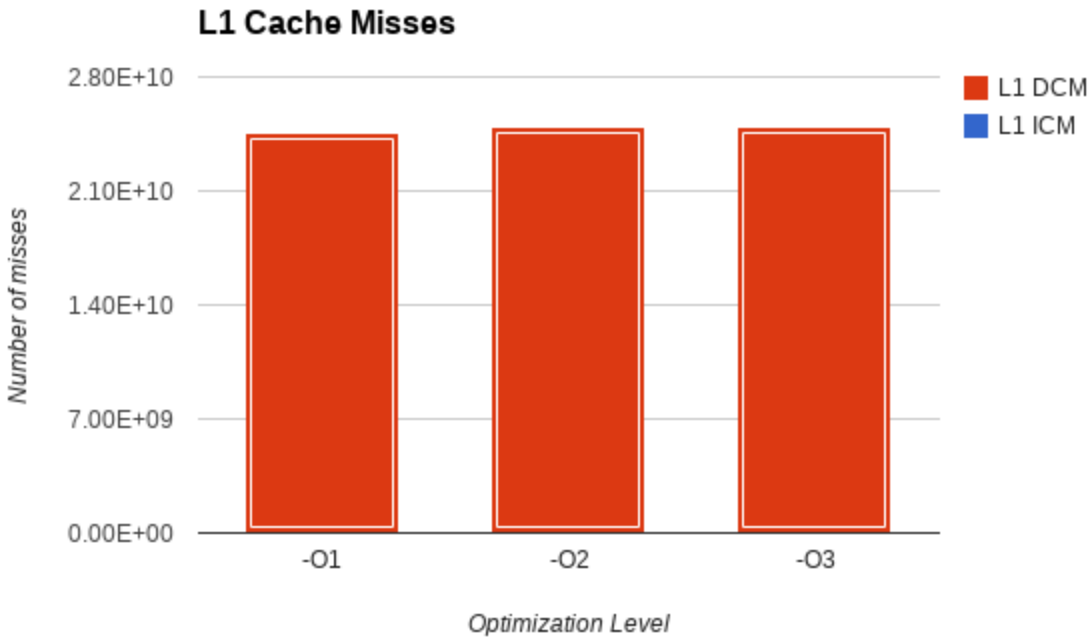
Instrumentation Results: Instruction and Data Caches

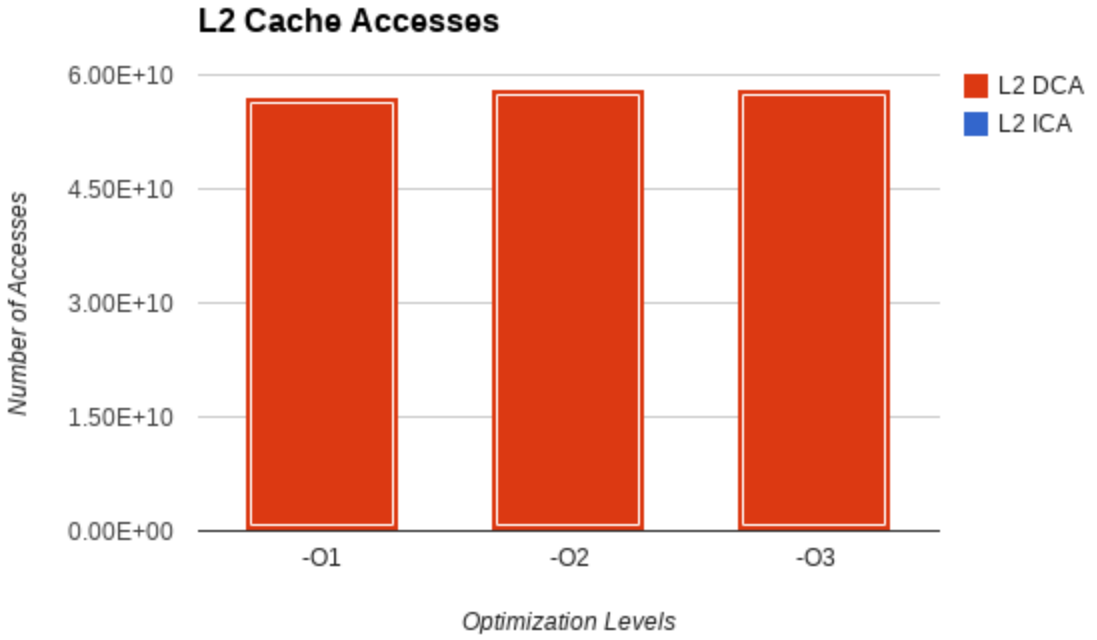
We first observe the distribution between the instruction and data cache accesses and misses for different optimization levels.

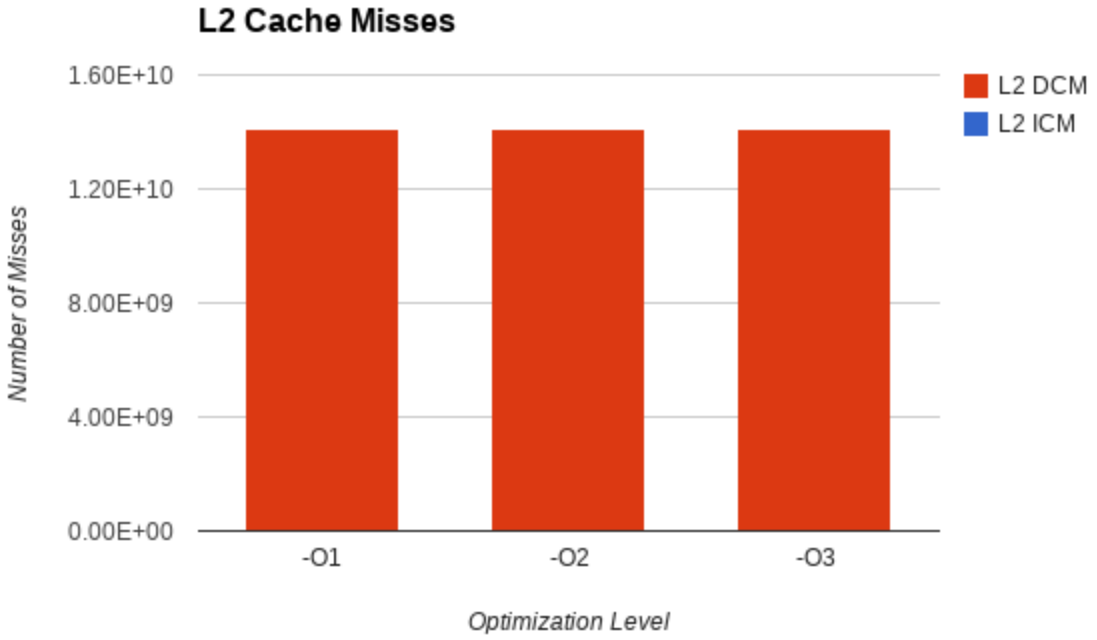
For the L1 cache, we can see that accesses are split almost evenly between instructions and data for all optimization levels. However, misses are almost exclusively for data.

Though the relative volumes of instruction cache misses are much smaller, we shall see that as we increase the optimization level, sometimes the increase in instruction cache misses is much larger than that of data cache misses. This is due to the compiler reorganizing code that may result in worse instruction cache efficiency (eg. due to increased program size).


L1 Cache Accesses
4.00E+12
3.00E+12
2.00E+12
1.00E+12
0.00E+00
Number of accesses
-O1
-O2
-O3
Optimization Level
L1 DCA
L1 ICA


L1 Cache Misses
2.80E+10
2.10E+10
1.40E+10
7.00E+09
0.00E+00
Number of misses
-O1
-O2
-O3
Optimization Level
L1 DCM
L1 ICM


L2 Cache Accesses
6.00E+10
4.50E+10
3.00E+10
1.50E+10
0.00E+00
Number of Accesses
-O1
-O2
-O3
Optimization Levels
L2 DCA
L2 ICA


L2 Cache Misses
1.60E+10
1.20E+10
8.00E+09
4.00E+09
0.00E+00
Number of Misses
-O1
-O2
-O3
Optimization Level
L2 DCM
L2 ICM

Instrumentation Results: Total Cache Access and Miss Behaviour

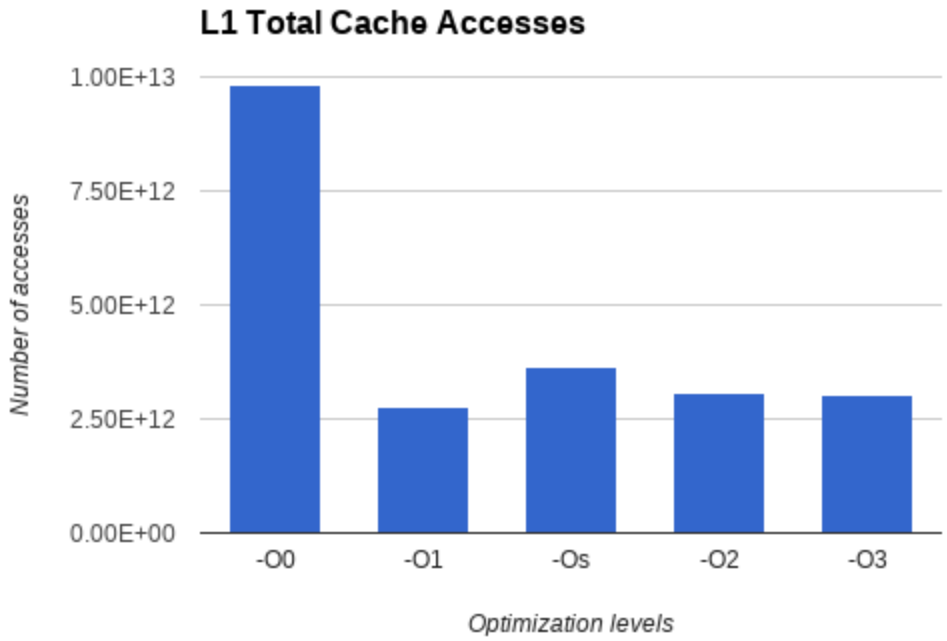
This section will consider the cache accesses and missed in total and highlight the differences between different optimization levels.

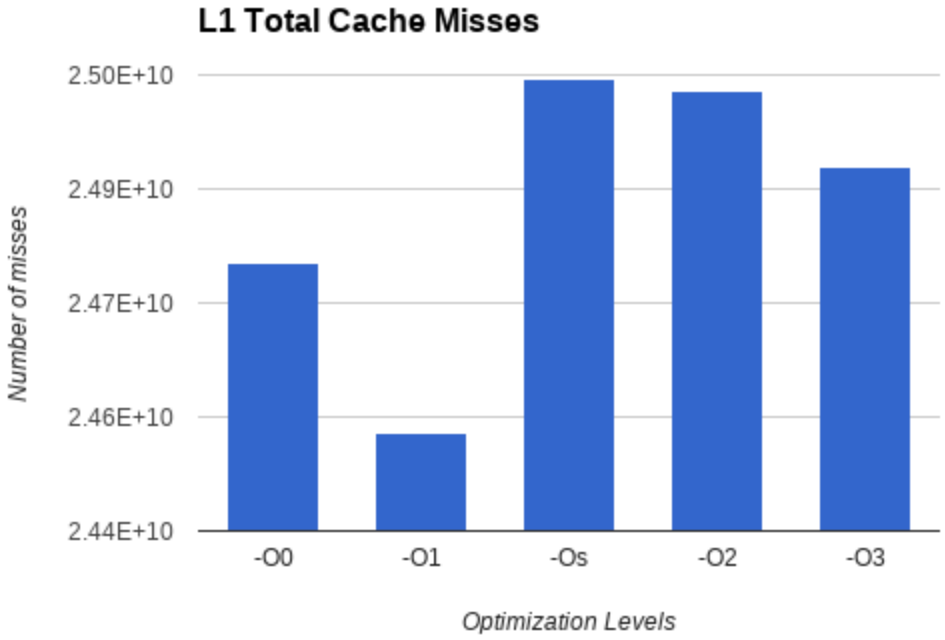
For the L1 cache, there is a dramatic drop in cache accesses from no optimizations to the remaining optimization levels, and the numbers remain stable henceforth.

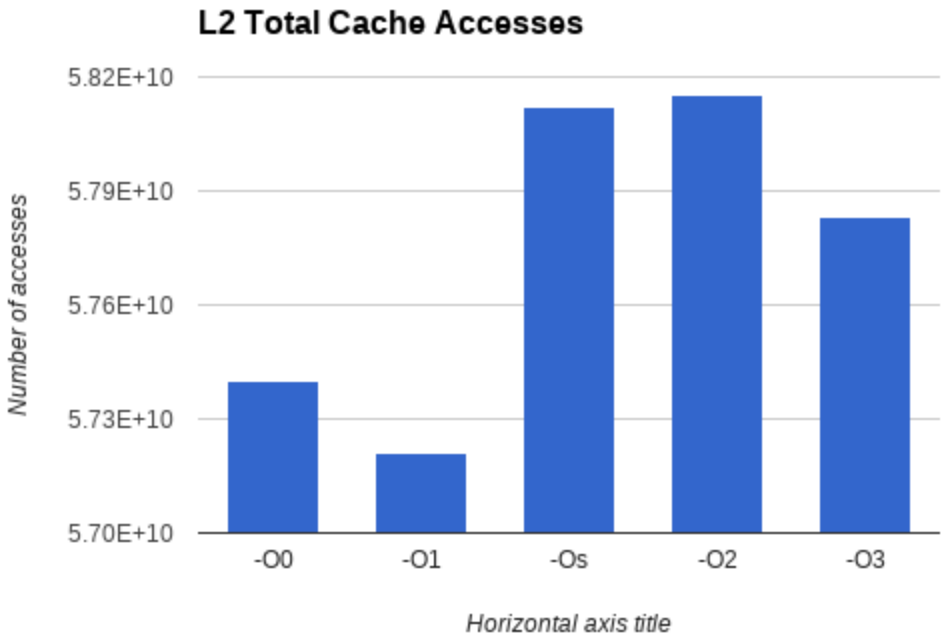
However, for cache misses, the number drops from none to the first optimization level, and then increases greatly from `-O1` to -O2. The reason for this is that the number of instruction cache misses more than doubles from `-O1` to `-O2`, while the data cache misses remain almost the same.

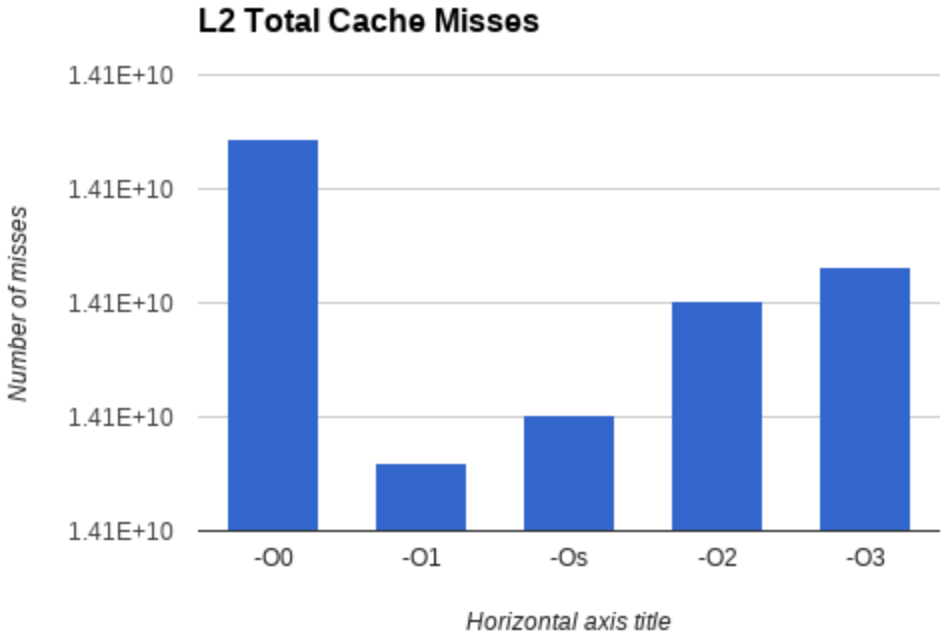
For the L2 cache, the pattern is explained by the behaviour of the L1 cache. Because of the increase in L1 cache misses from `-O1` to `-O2`, there is a corresponding increase in the accesses to the L2 cache.

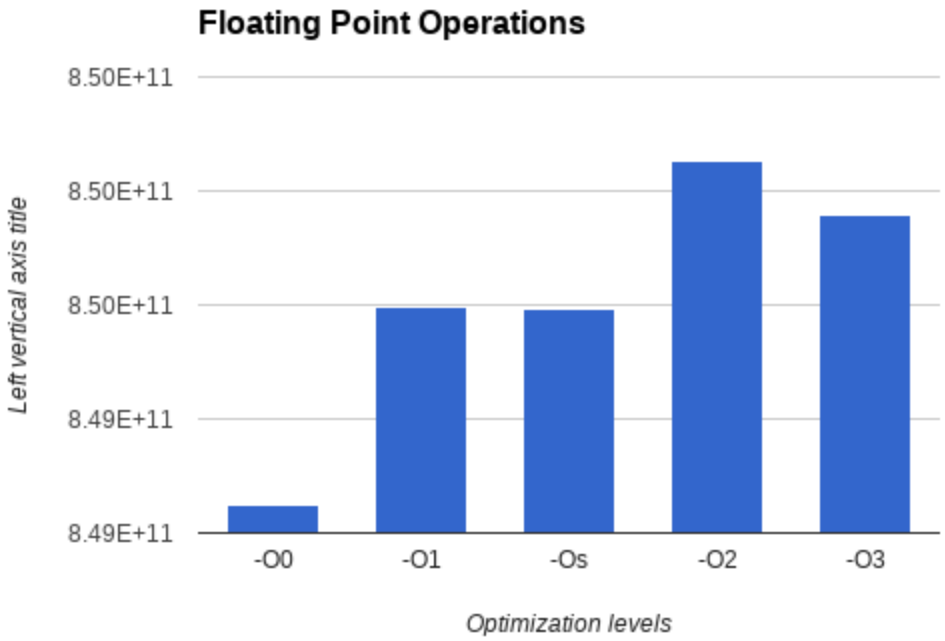
The total number of floating point operations increases with the increase in the optimization level. However, there is a decrease from `-O2` to `-O3`, indicating some optimization having been applied that reduced the number of floating point operations.

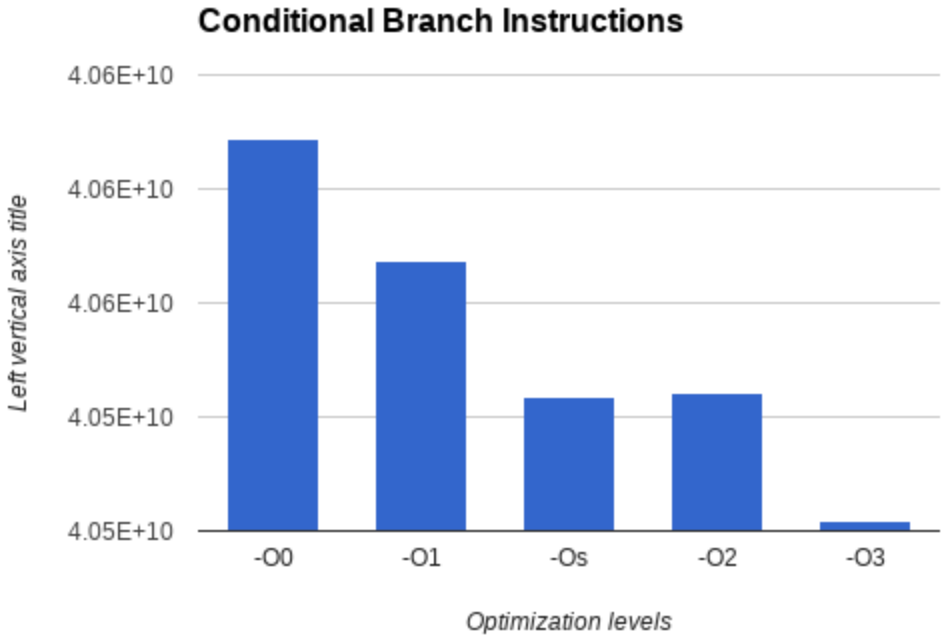



L1 Total Cache Misses
2.50E+10
2.49E+10
2.47E+10
2.46E+10
2.44E+10
Number of misses
-O0
-O1
-Os
-O2
-O3
Optimization Levels


L2 Total Cache Accesses
5.82E+10
5.79E+10
5.76E+10
5.73E+10
5.70E+10
Number of accesses
-O0
-O1
-Os
-O2
-O3
Horizontal axis title


L2 Total Cache Misses
1.41E+10
1.41E+10
1.41E+10
1.41E+10
1.41E+10
Number of misses
-O0
-O1
-Os
-O2
-O3
Horizontal axis title

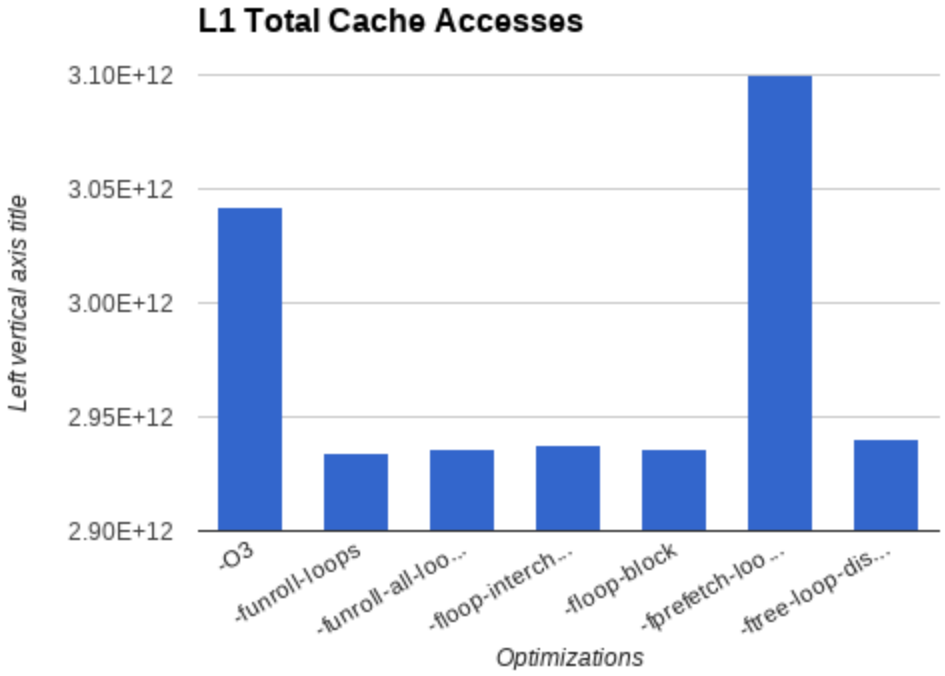

Floating Point Operations
8.50E+11
8.50E+11
8.50E+11
8.49E+11
8.49E+11
Left vertical axis title
-O0
-O1
-Os
-O2
-O3
Optimization levels

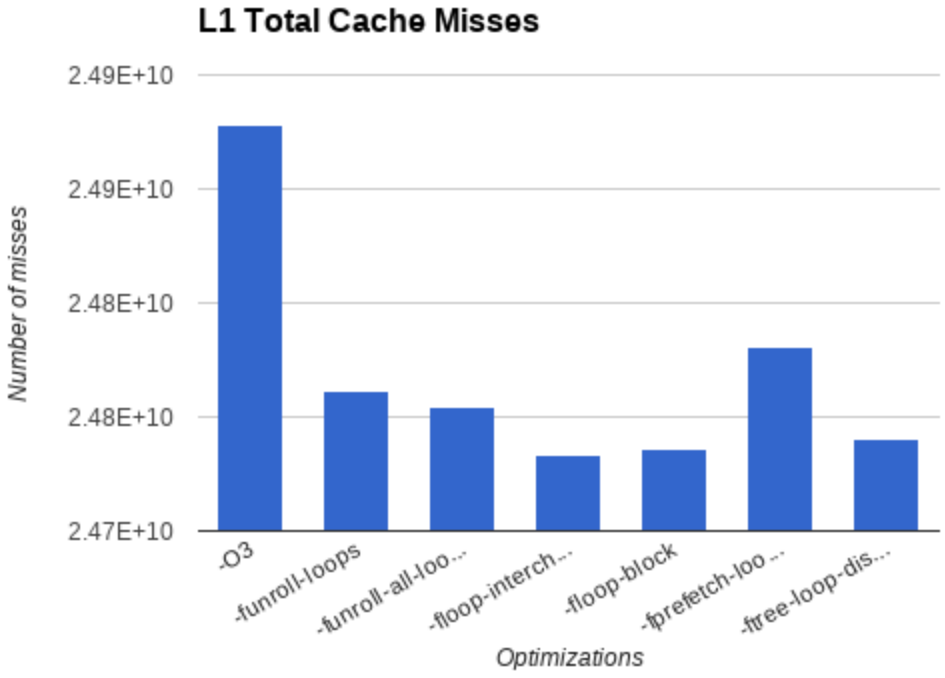


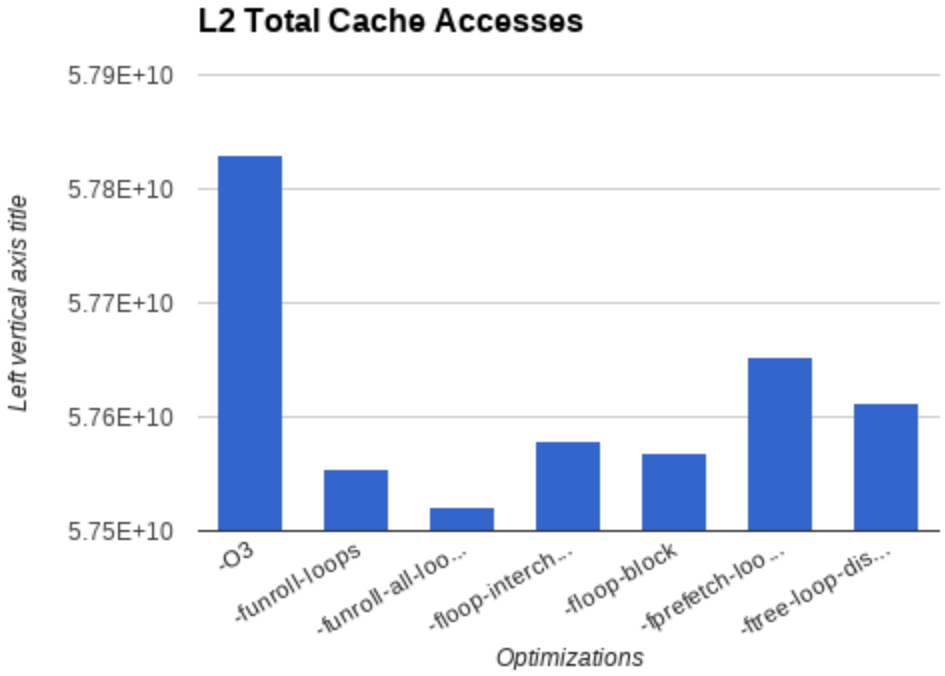
Instrumentation Results: Additional Optimizations

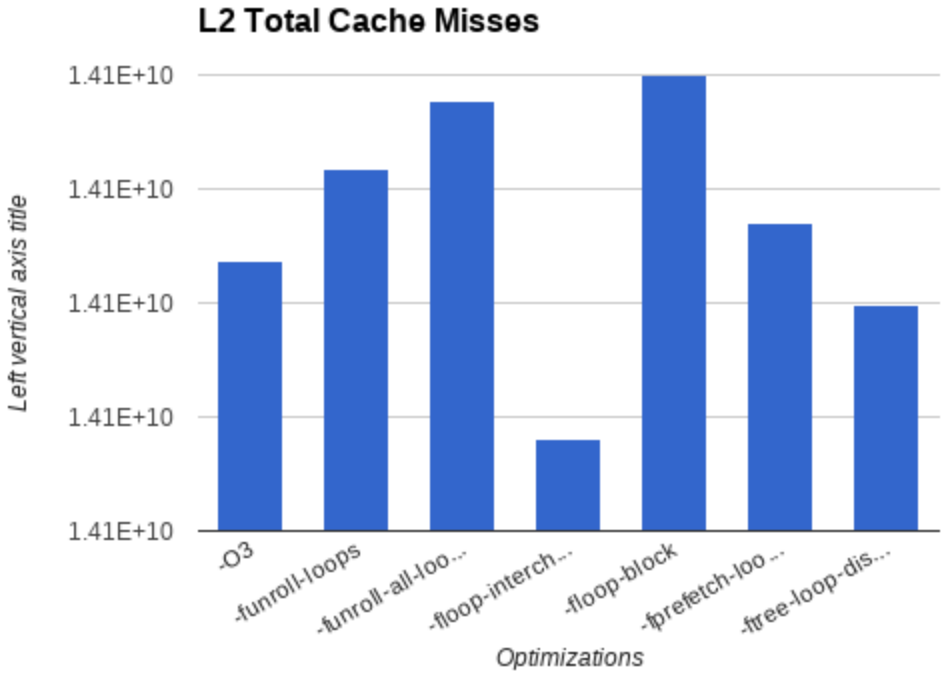
The additional optimizations are all applied on top of the `-O3` optimization level, and are hence plotted against this level as a baseline.

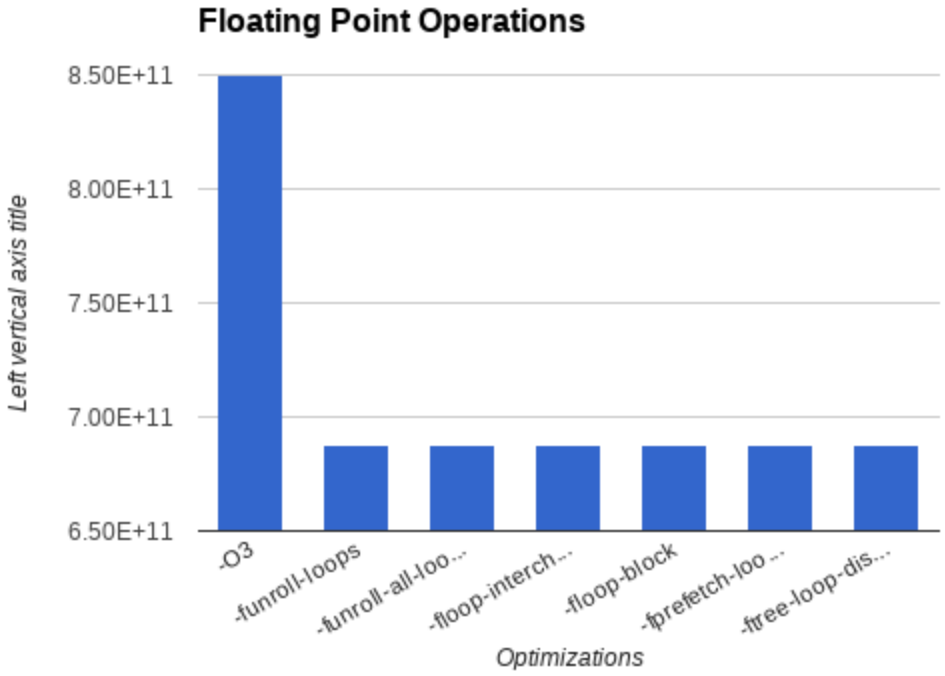
The additional optimizations were selected keeping in mind that the profiled section of code is mainly loops.

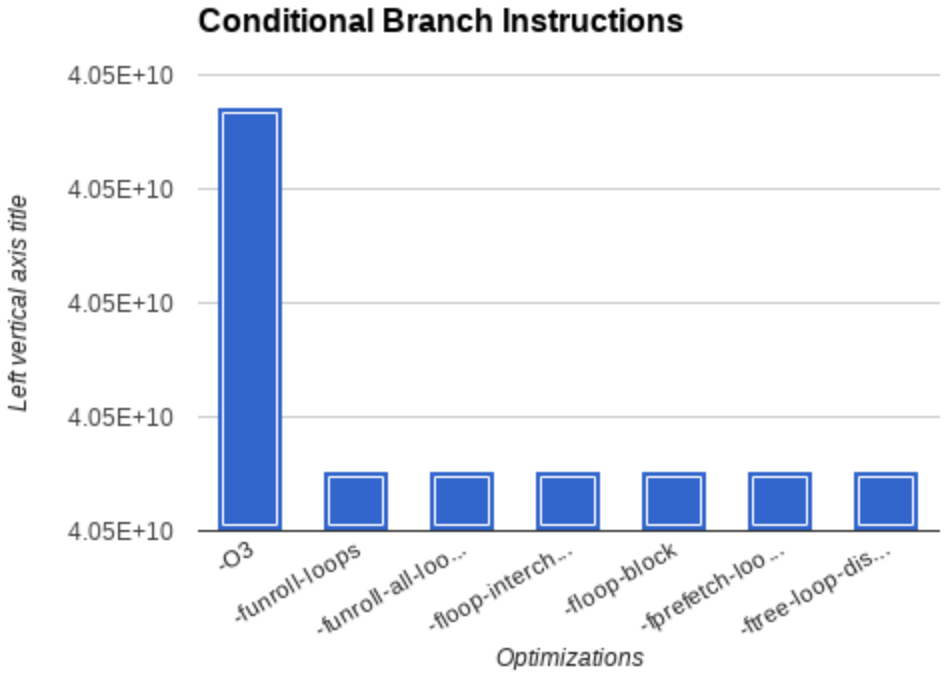

L1 Total Cache Accesses
3.10E+12
3.05E+12
3.00E+12
2.95E+12
2.90E+12
Left vertical axis title
-O3
-funroll-loops
-funroll-all-loo...
-floop-interch...
-floop-block
-fprefetch-loo...
-ftree-loop-dis...
Optimizations


L1 Total Cache Misses
2.49E+10
2.49E+10
2.48E+10
2.48E+10
2.47E+10
Number of misses
-O3
-funroll-loops
-funroll-all-loo...
-floop-interch...
-floop-block
-fprefetch-loo...
-ftree-loop-dis...
Optimizations


L2 Total Cache Accesses
5.79E+10
5.78E+10
5.77E+10
5.76E+10
5.75E+10
Left vertical axis title
-O3
-funroll-loops
-funroll-all-loo...
-floop-interch...
-floop-block
-fprefetch-loo...
-ftree-loop-dis...
Optimizations


L2 Total Cache Misses
1.41E+10
1.41E+10
1.41E+10
1.41E+10
1.41E+10
Left vertical axis title
-O3
-funroll-loops
-funroll-all-loo...
-floop-interch...
-floop-block
-fprefetch-loo...
-ftree-loop-dis...
Optimizations


Floating Point Operations
8.50E+11
8.00E+11
7.50E+11
7.00E+11
6.50E+11
Left vertical axis title
-O3
-funroll-loops
-funroll-all-loo...
-floop-interch...
-floop-block
-fprefetch-loo...
-ftree-loop-dis...
Optimizations


Conditional Branch Instructions
4.05E+10
4.05E+10
4.05E+10
4.05E+10
Left vertical axis title
-O3
-funroll-loops
-funroll-all-loo...
-floop-interch...
-floop-block
-fprefetch-loo...
-ftree-loop-dis...
Optimizations

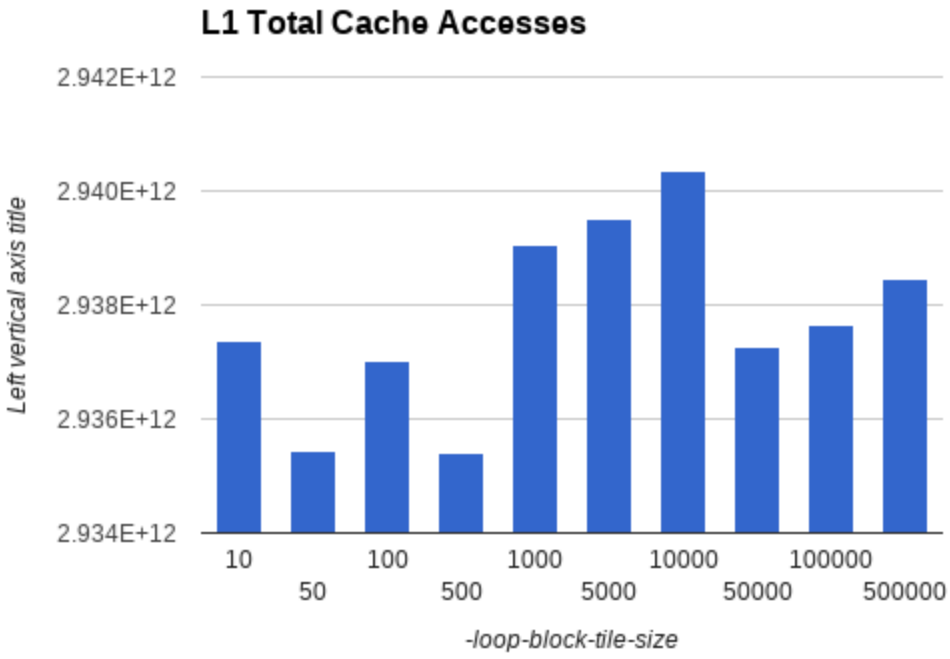
Instrumentation Results: Parameterizable Optimizations

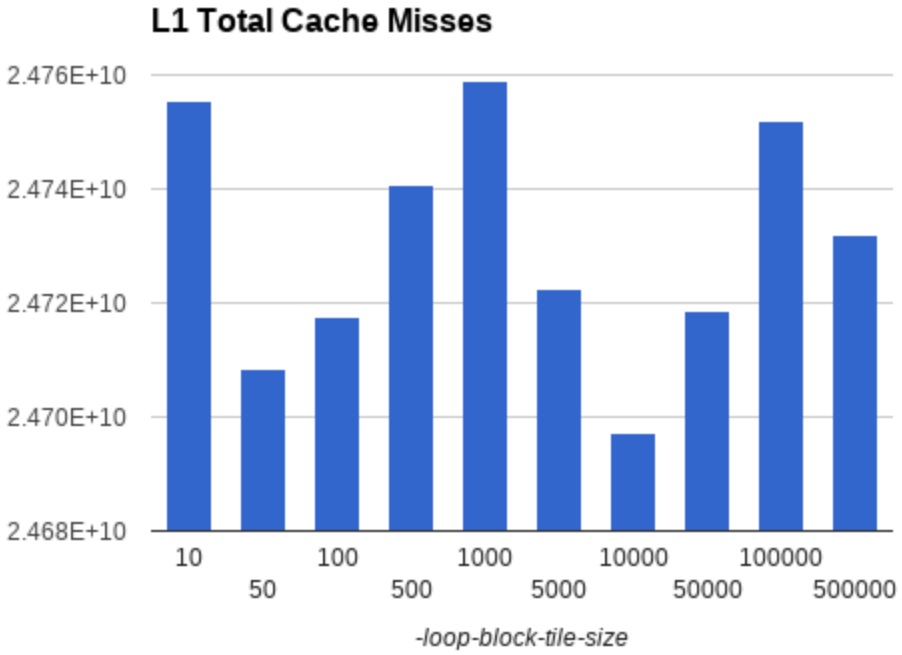
To observe the effect of parameterizing an optimization, we must take into account that the instrumentation is for a loop section of the code. Hence, selecting a parameterizable optimization that concerns branches or other kinds of instructions will not offer any significant insight.

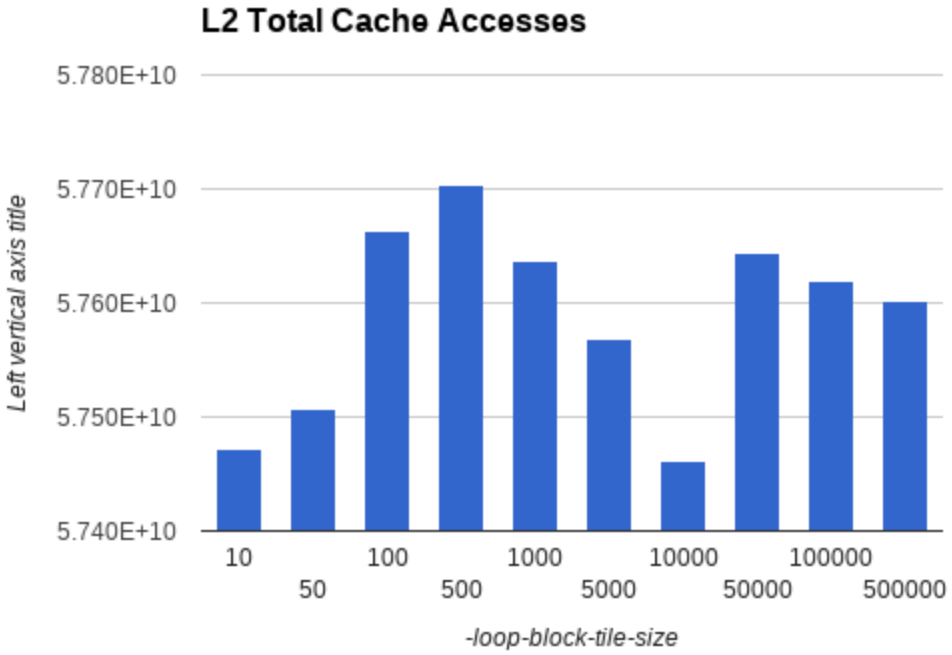
In this section, we observe the effect of the `-floop-block` optimization applied on top of the -O3 flag, with varying values of the `loop-block-tile-size` parameter.

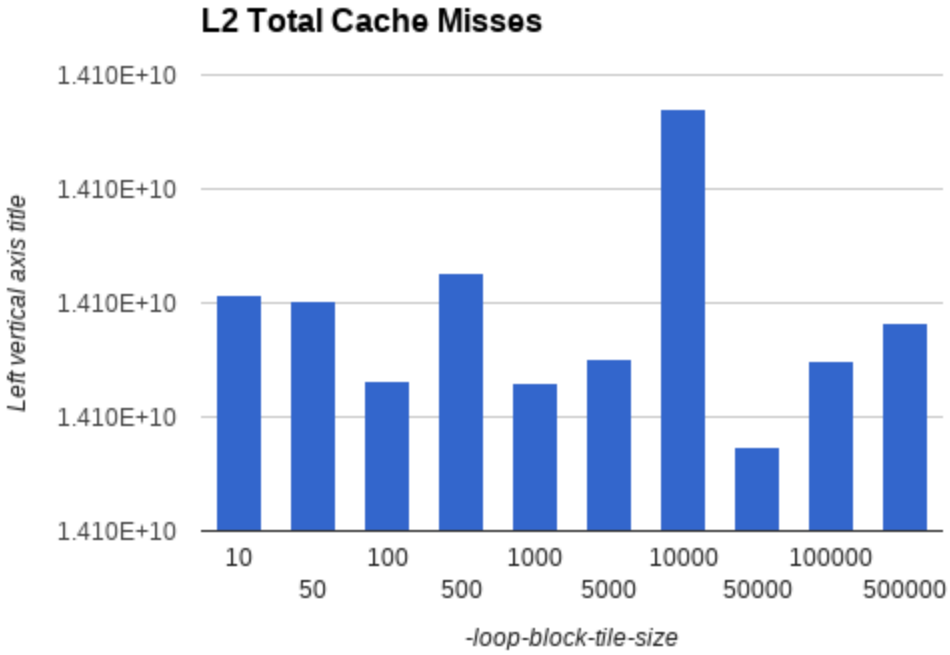
This optimization's goal is to improve cache efficiency in loops, such that each the data accessed by each inner loop fits inside the cache. The parameter controls the length of each loop strip. This causes the cache miss and access rates to fluctuate since the strip length may not be efficient for the given fixed cache line.

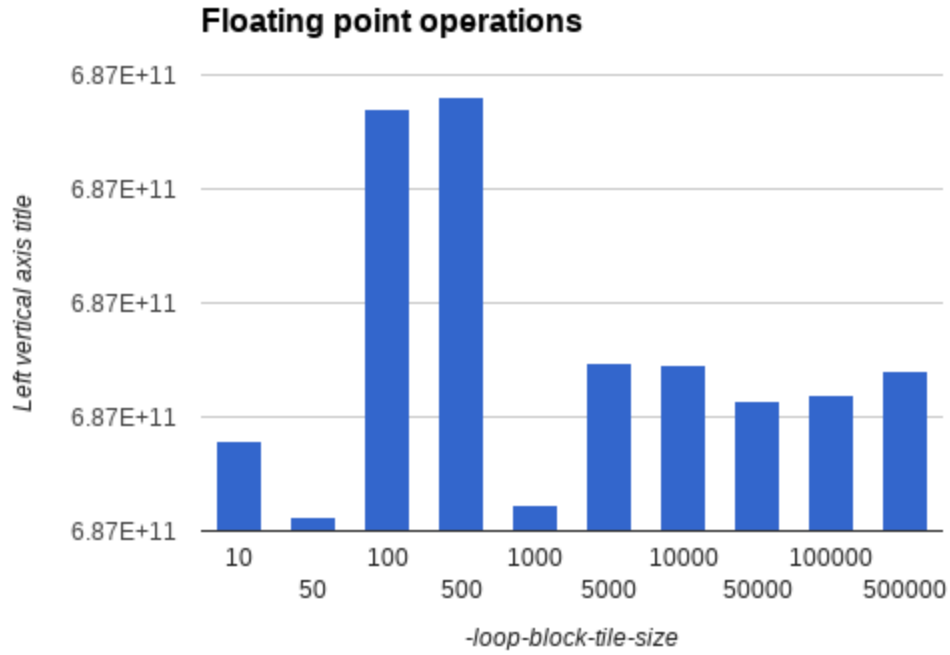
The conditional branch instructions and floating point operations will not be affected by this parameter. Though the graphs indicate some fluctuations, these are only in a few hundreds or thousands and may be attributed to noise.

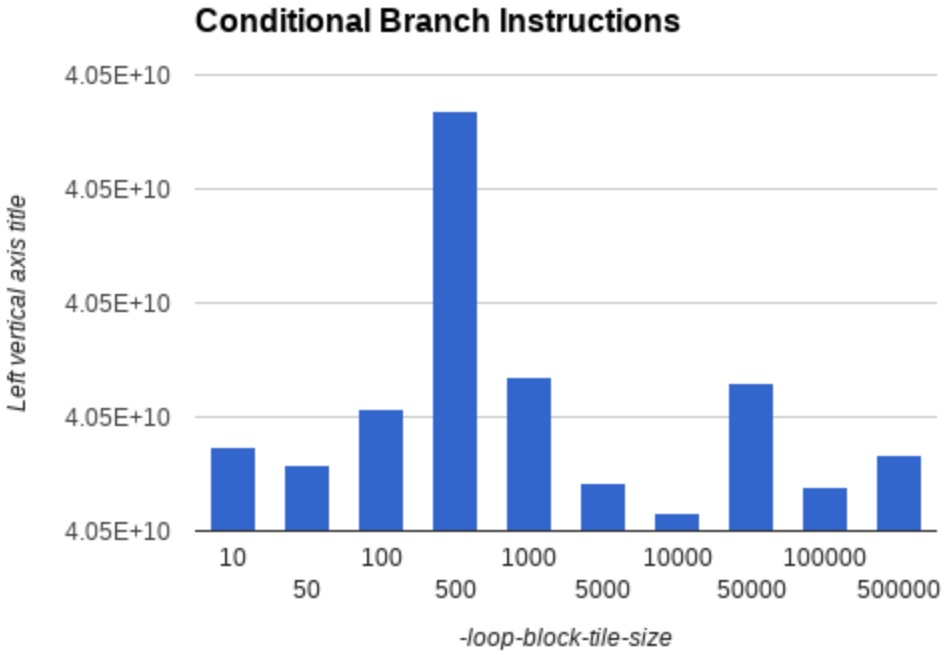



L1 Total Cache Misses
2.476E+10
2.474E+10
2.472E+10
2.470E+10
2.468E+10
10
50
100
500
1000
5000
10000
50000
100000
500000
-loop-block-tile-size


L2 Total Cache Accesses
Left vertical axis title
5.780E+10
5.770E+10
5.760E+10
5.750E+10
5.740E+10
10
50
100
500
1000
5000
10000
50000
100000
500000
-loop-block-tile-size

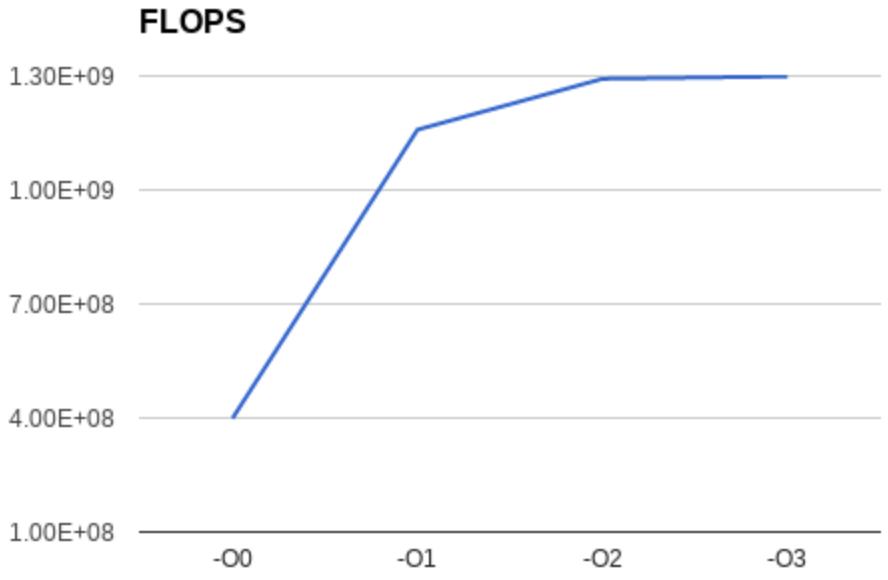

L2 Total Cache Misses
1.410E+10
1.410E+10
1.410E+10
1.410E+10
1.410E+10
Left vertical axis title
10
50
100
500
1000
5000
10000
50000
100000
500000
-loop-block-tile-size


Floating point operations
6.87E+11
6.87E+11
6.87E+11
6.87E+11
6.87E+11
Left vertical axis title
10
50
100
500
1000
5000
10000
50000
100000
500000
-loop-block-tile-size



Instrumentation Results: Overall Performance

The overall performance in floating point operations per second was derived from the running time and number of floating point operations over 5 runs of the code. With the `-O3` and `fast-math` optimizations the performance was 1300 megaflops.



Task 4

Peter Ungaro, CEO of Cray Inc., presented his views on the fusion of supercomputing and data analysis. The focus of the talk was on the problems faced by data analysts who traditionally work on systems optimized for large data, and not for low-latency exploratory analysis. Though supercomputers were designed for highly-parallel tasks that are primarily compute-bound, Peter believes that they could also fill the gap between the data analyst and a low-latency system to process fast data.

Peter discussed a new Cray division called YarcData, set up to deal with solving data analysis problems on supercomputers. He mentioned a case study of Urika, a YarcData product, and how it enabled scientists to quickly prototype and test approaches for drug repurposing, bringing down the entire successful cycle time from years to just three months. He highlighted the interplay between the strengths of a supercomputer in rapid querying and explorations, and that of traditional data warehouses in storage, organization and reliability.

Considering the future, Peter spoke about building a single turnkey solution by partnering with Hadoop vendors (Intel) and providing the systems and support in a single package. He spoke about the importance of fast interconnect and tightly integrated software that makes it easy to work with both the data management and supercomputing aspects of the system.