Masterpraktikum Scientific Computing

High Performance Computing Tutorial 4

Session 4: Profiler

Authors
Mantosh Kumar (Matriculation number : 03662915)
João Trindade (Matriculation number : 03673251)

Course of Study: Master of Science Informatics

Question 1 - Gauss elimination - GNU gprof

Part 1: Set number of system of equation = 300000000

Comparison between runtime of application with and without profiling before change in code:

	gcc (seconds)	icc (seconds)
profile_enable	117.328	118.477
profile_disable	103.604	103.777

For the Intel compiler, the default optimization is -O2 that can trigger Interprocedural optimization for our code and thus empowering the compiler to perform inline function expansion for calls to functions defined within the current source file.

For gcc compiler, the default optimization is -O0.

Solution: Disable default optimization of intel compiler

```
icc gauss.c timer.c -o gauss_icc_enable_profile.exe -pg -O0
```

Question 1 - Gauss elimination - GNU gprof

Part 2: Comparison between GNU and intel compiler profile results before change in code:

	gcc (seconds)	icc (seconds)
gauss_elimination	69.25	71.05
init	21.86	18.71

gauss_elimination should be optimized first

Question 1 - Gauss elimination - GNU gprof

Part 3: Comparison between GNU and intel compiler profile results after change in code:

	gcc(seconds)	icc(seconds)
gauss_elimination	1.11	3.31
gauss_convert_to_echelon	52.29	56.28
gauss_back_substituting	14.84	12.38
init	21.71	18.63

gauss_convert_to_echelon should be optimized first (the first part of gauss algorithm)

Question 1 - Gauss elimination - GNU gprof

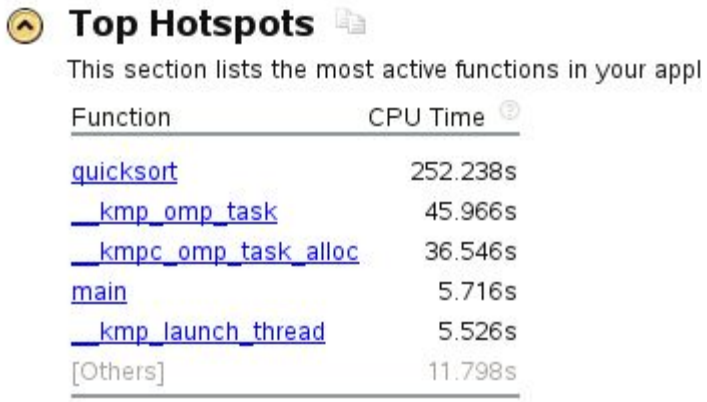
Comparison between runtime of application with and without profiling after change in code:

	gcc (seconds)	icc (seconds)
profile_enable	141.300	180.545
profile_disable	107.796	108.593

Question 2 - Quicksort Analysis

Basic Hotspots

Ran tests for different array sizes, thread numbers and threshold values. No definite conclusions.



Top Hotspots

This section lists the most active functions in your appl

Function	CPU Time
quicksort	252.238s
__kmp_omp_task	45.966s
__kmpc_omp_task_alloc	36.546s
main	5.716s
__kmp_launch_thread	5.526s
[Others]	11.798s

Question 2 - Quicksort Analysis

Basic Hotspots

Ran tests for different array sizes, thread numbers and threshold values. No definite conclusions.

Question 2 - Quicksort Analysis

Concurrency
Analyze how your application is using available logical CPUs, discover where parallelism is incurring synchronization overhead, and identify potential candidates for parallelization. This analysis type uses user-mode sampling and tracing collection.

Locks and Waits
Identify where your application is waiting on synchronization objects or I/O operations and discover how these waits affect your application performance. This analysis type uses user-mode sampling and tracing collection.

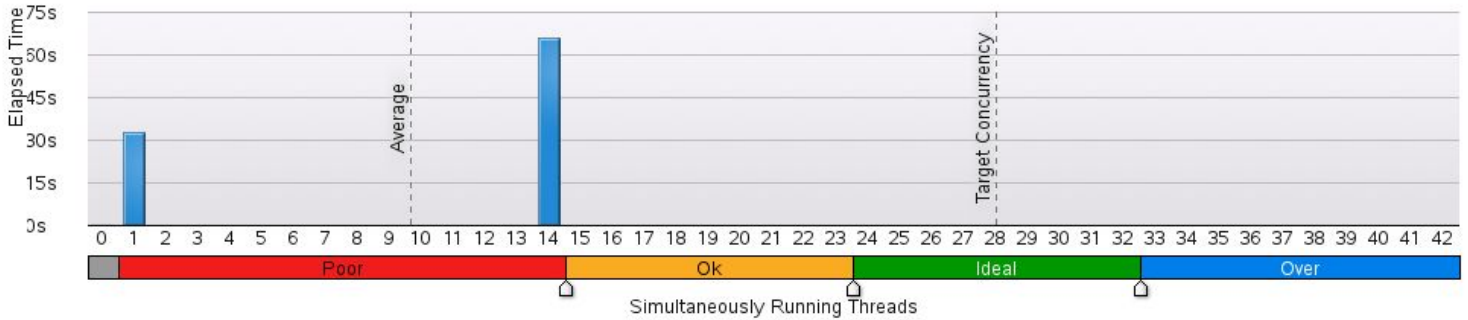
Source: Intel V-Tune

Question 2 - Quicksort Analysis

Concurrency - Example of 14 Thread Run

Thread Concurrency Histogram

This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were running simultaneously. Threads are considered running if they are either actually running on a CPU or are in the runnable state in the OS scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that were not waiting. Thread Concurrency may be higher than CPU usage if threads are in the runnable state and not consuming.



Question 2 - Locks & Waits

Top Waiting Objects

This section lists the objects that spent the most time waiting in your app synchronizations. A significant amount of Wait time associated with a sy

Sync Object	Wait Time	Wait Count
Condition Variable 0x3877509c	346.880s	77
Condition Variable 0x435fe938	8.349s	1
Stream 0x0669dc06	0.000s	1
Stream /proc/meminfo 0x93189e00	0.000s	1
Stream /proc/meminfo 0xa74985cf	0.000s	1

Grouping: Sync Object / Function / Call Stack

Sync Object / Function / Call Stack	Wait Time by Thread Concurrency (Idle, Poor, Ok, Ideal, Over)	Wait Cou..	Mod..	Object Type	Object Creation Module and Function
Condition Variable 0x3877509c	346.880s	77		Condition Variable	libiomp5.so!__kmp_launch_thread
__kmp_launch_thread	346.880s	77	libi...	Condition Variable	libiomp5.so!__kmp_launch_thread
[OpenMP worker]	346.880s	77	libi...	Condition Variable	libiomp5.so![OpenMP worker]
start_thread	346.880s	77	libp...	Condition Variable	libiomp5.so!start_thread
__clone	346.880s	77	libc...	Condition Variable	libiomp5.so!__clone
Condition Variable 0x435fe938	8.349s	1		Condition Variable	libiomp5.so!__kmp_join_call
Stream 0x0669dc06	0.000s	1		Stream	libiomp5.so!__kmp_api_omp_set_num_threads

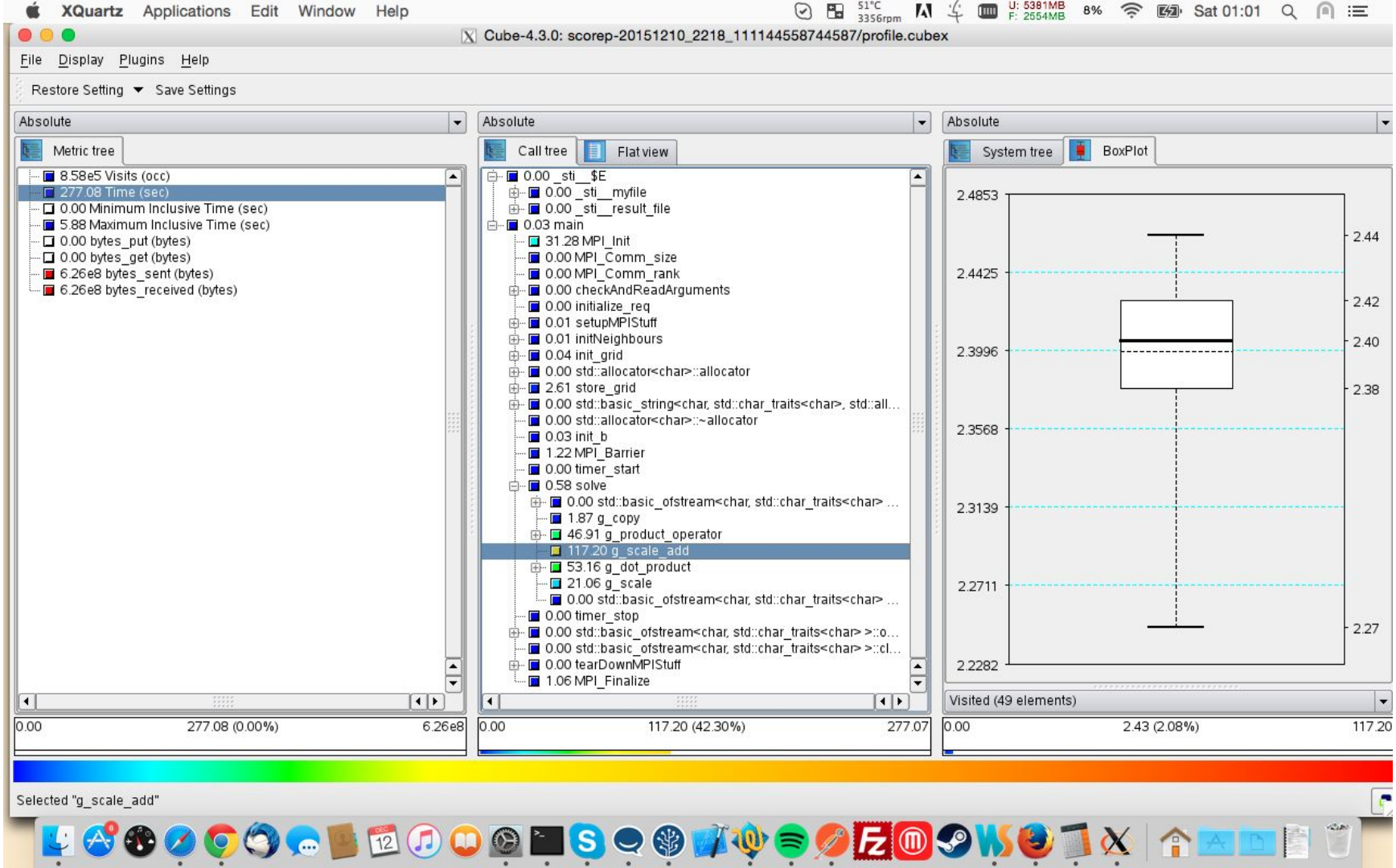
Question 3 - CG Analysis

Load Imbalances ?

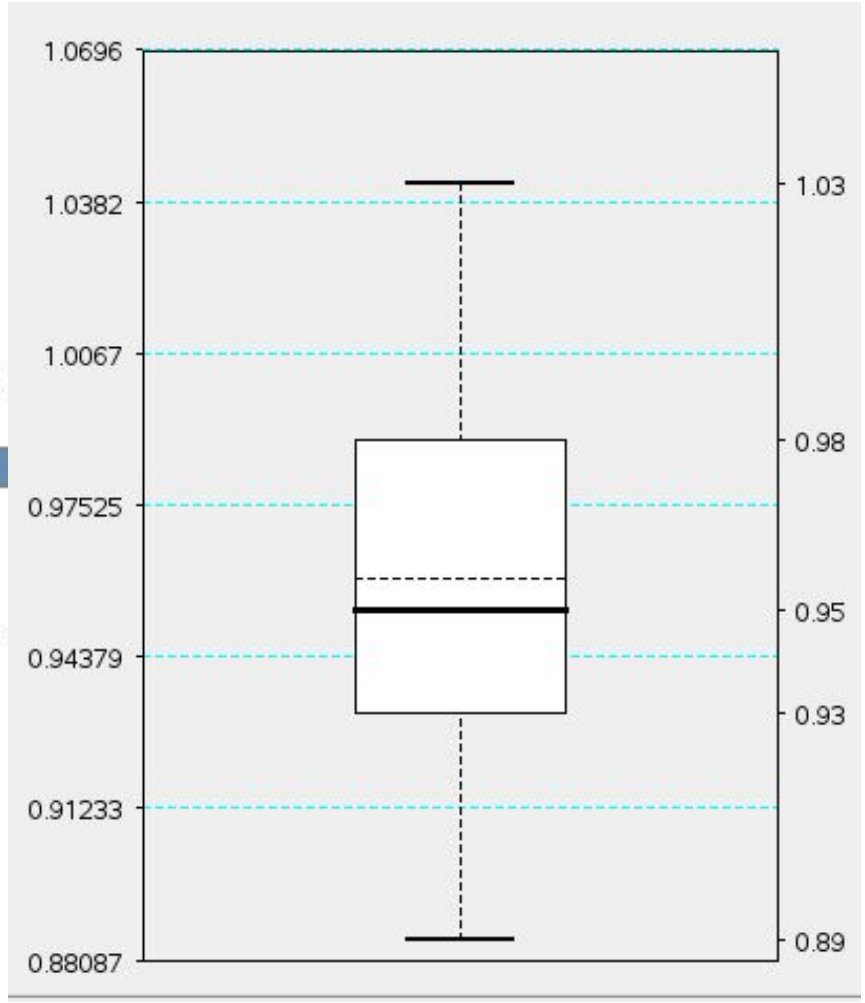
What is the boxplot ?

Manual Instrumentation

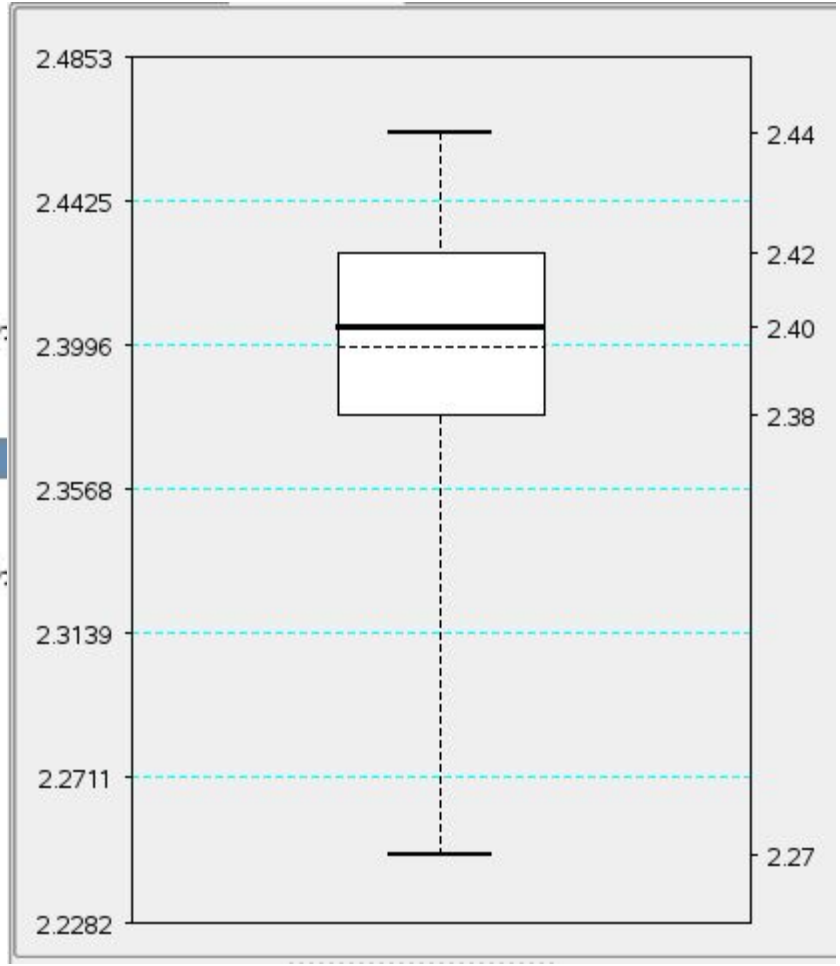
Hybrid MPI / OpenMP parallelization. Is it worth it ?


XQuartz Applications Edit Window Help
Sat 01:01
Cube-4.3.0: scorep-20151210_2218_111144558744587/profile.cubex
File Display Plugins Help
Restore Setting
Save Settings
Absolute
Metric tree
8.58e5 Visits (occ)
277.08 Time (sec)
0.00 Minimum Inclusive Time (sec)
5.88 Maximum Inclusive Time (sec)
0.00 bytes_put (bytes)
0.00 bytes_get (bytes)
6.26e8 bytes_sent (bytes)
6.26e8 bytes_received (bytes)
0.00 277.08 (0.00%) 6.26e8
Absolute
Call tree
Flat view
0.00 _sti__$E
0.00 _sti__myfile
0.00 _sti__result_file
0.03 main
31.28 MPI_Init
0.00 MPI_Comm_size
0.00 MPI_Comm_rank
0.00 checkAndReadArguments
0.00 initialize_req
0.01 setupMPIStuff
0.01 initNeighbours
0.04 init_grid
0.00 std::allocator<char>::allocator
2.61 store_grid
0.00 std::basic_string<char, std::char_traits<char>, std::all...
0.00 std::allocator<char>::~allocator
0.03 init_b
1.22 MPI_Barrier
0.00 timer_start
0.58 solve
0.00 std::basic_ofstream<char, std::char_traits<char> ...
1.87 g_copy
46.91 g_product_operator
117.20 g_scale_add
53.16 g_dot_product
21.06 g_scale
0.00 std::basic_ofstream<char, std::char_traits<char> ...
0.00 timer_stop
0.00 std::basic_ofstream<char, std::char_traits<char> >::o...
0.00 std::basic_ofstream<char, std::char_traits<char> >::cl...
0.00 tearDownMPIStuff
1.06 MPI_Finalize
0.00 117.20 (42.30%) 277.07
Absolute
System tree
BoxPlot
2.4853
2.4425
2.3996
2.3568
2.3139
2.2711
2.2282
2.44
2.42
2.40
2.38
2.27
Visited (49 elements)
0.00 2.43 (2.08%) 117.20
Selected "g_scale_add"

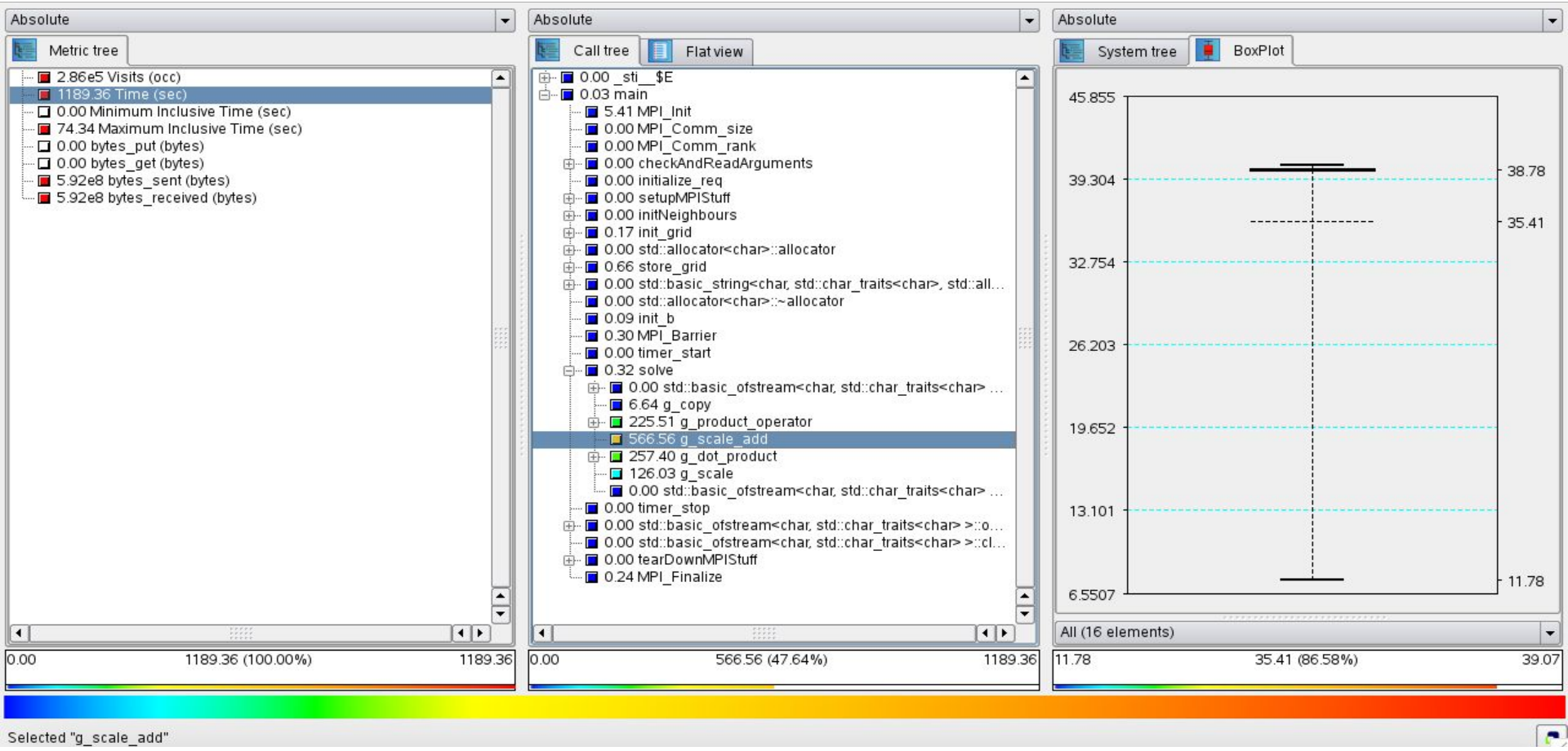
- 0.58 solve
  - 0.00 std::basic_ofstream<char, std::char_traits<char>
  - 1.87 g_copy
  - 46.91 g_product_operator
  - 117.20 g_scale_add
  - 53.16 g_dot_product
  - 21.06 g_scale
  - 0.00 std::basic_ofstream<char, std::char_traits<char>



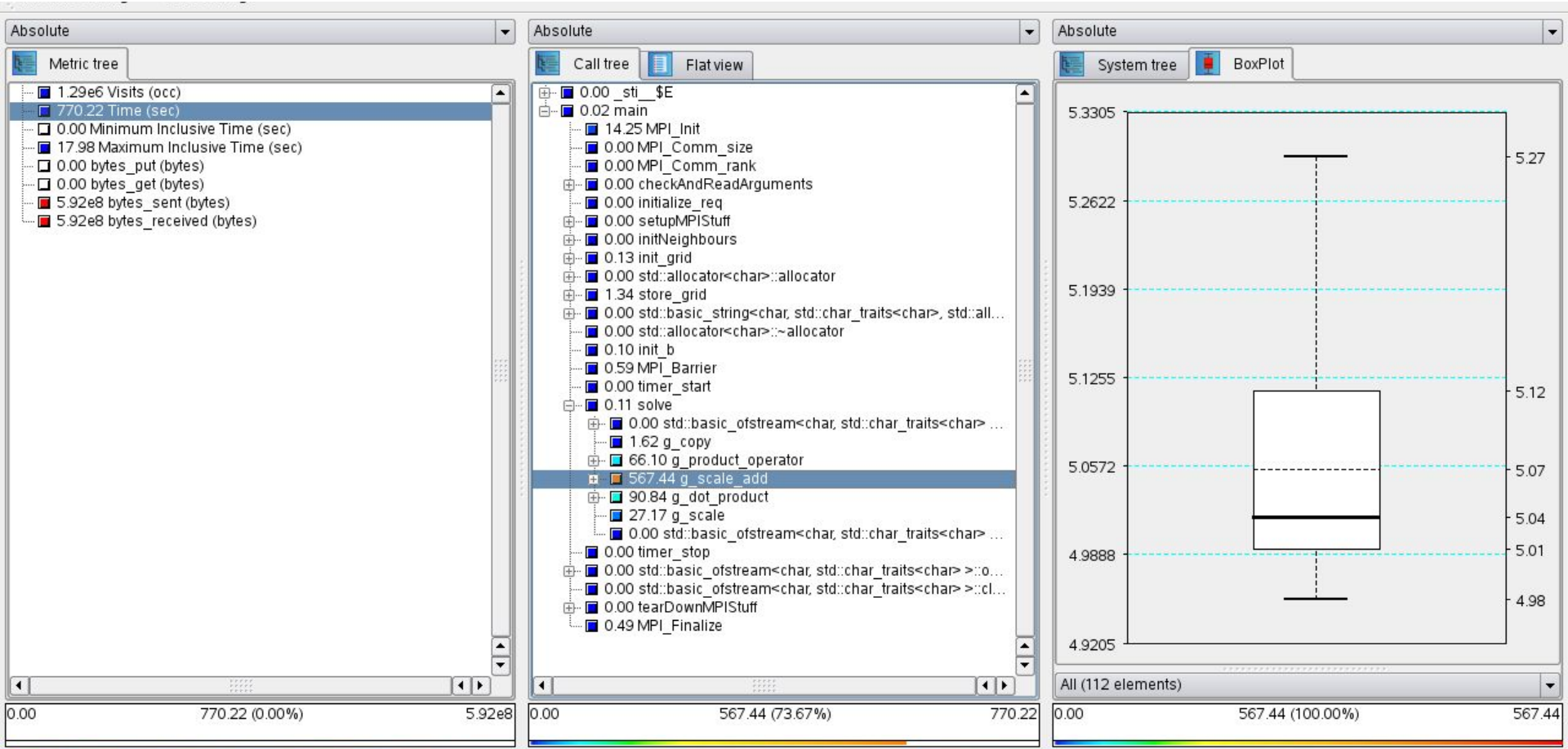
- 0.58 solve
  - 0.00 std::basic_ofstream<char, std::char_traits<char
  - 1.87 g_copy
  - 46.91 g_product_operator
  - 117.20 g_scale_add
  - 53.16 g_dot_product
  - 21.06 g_scale
  - 0.00 std::basic_ofstream<char, std::char_traits<char



MPI Only Run (6000 1000 0.0001 4 4) -> 73.9055s



MPI + OpenMP Run (6000 1000 0.0001 4 4) -> 16.7258s

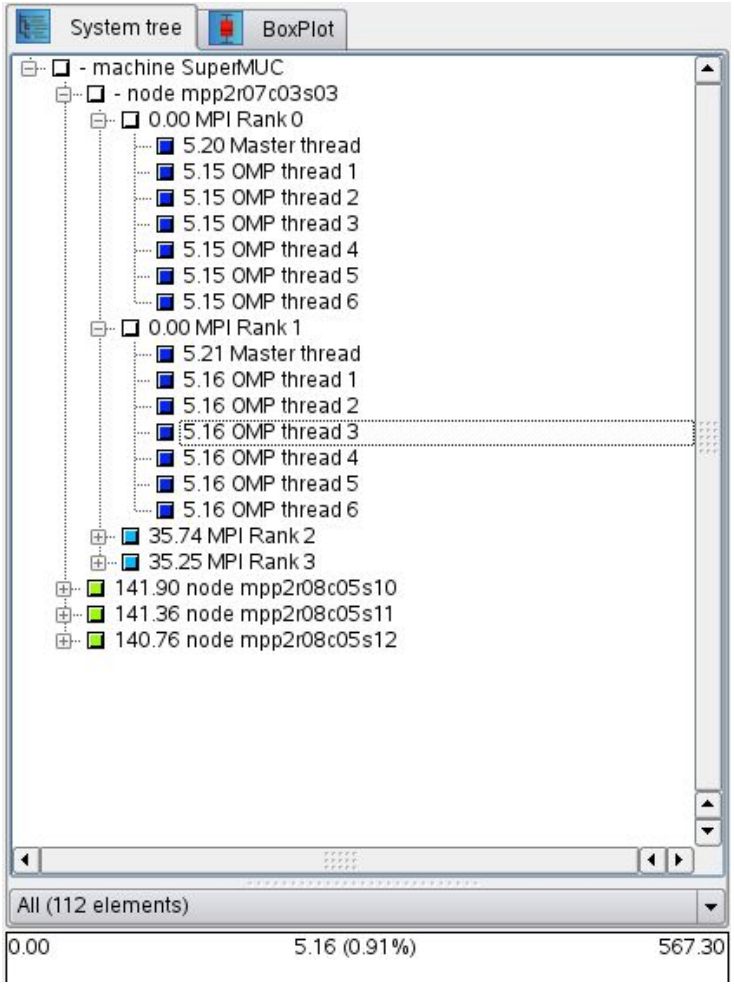


Hybrid Parallelization

omp instrumentation added

Serial Time: 380 s
MPI Only: 73.90 s
MPI + OpenMP: 16.65 s

MPI Speedup: 5.14
MPI + OpenMP Speedup: 22.8



Question 4 - DGEMM - Perf

Useful Events for DGEMM optimization:

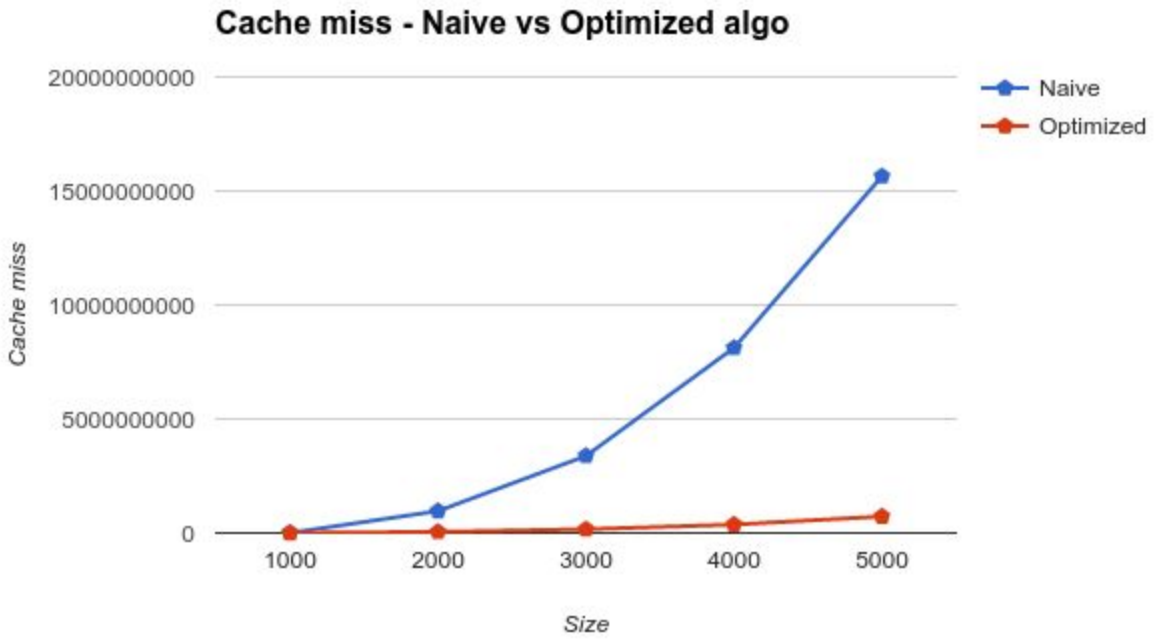
Hardware Events: cache-misses

Hardware Cache Events: LLC-loads, LLC-load-misses, dTLB-load-misses, L1-dcache-load-misses

Perf -Cache misses comparision between naive and optimized dgemm algorithm (mac-login-intel.tum-mac.cos.lrz.de)

Size	Algo type	Cache-miss	L1-Load-miss	LLC-Load-miss	dTLB-Load-miss	Elapsed time(s)
1000	Naive	3,566,302	1,125,136,829	3	161,792	12.22061025
	Optimized	542,849	130,695,095	0	52,304	9.506367515
2000	Naive	961,856,464	8,607,359,880	0	1,860,122	175.4595018
	Optimized	48,757,427	1,084,882,864	0	79,883	75.97345791
3000	Naive	3,381,020,403	30,435,798,814	0	176,026,132	444.3726011
	Optimized	162,964,704	6,764,459,595	0	586,565	258.318864
4000	Naive	8,117,926,925	113,772,013,198	0	1,032,509,707	1347.328946
	Optimized	370,145,135	16,038,854,049	0	648,043	606.4898687
5000	Naive	15,646,301,418	141,092,147,565	0	2,684,191,273	2659.116061
	Optimized	724,838,943	31,321,819,449	6	1,970,185	1188.22697

Question 4 - DGEMM - Perf

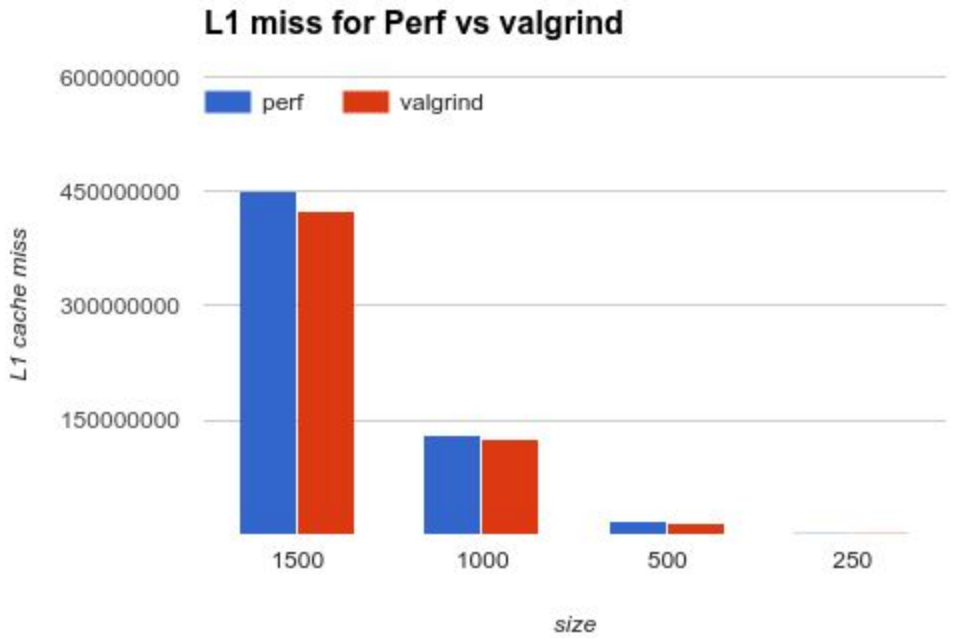


Question 4 - DGEMM - Perf

Perf vs valgrind with optimized dgemm algorithm (mac-login-intel.tum-mac.cos.lrz.de)						
Size	Perf			Valgrind		
	Elapsed Time(s)	L1-Load-miss	LLC-Load-miss	Elapsed time(s)	D1 misses	LLd misses
1500	31.91366122	450,783,987	0	1877.01	423,285,980	1,690,018
1000	9.506367515	130,695,095	0	556.944	125,629,232	575,107
500	1.210139646	16,440,649	0	68.4107	15,784,982	96,228
250	0.148300976	2,014,632	0	9.11408	1,995,540	25,914

Reason of difference: Cachegrind (**valgrind --tool=cachegrind**) is a cache simulator. Even though it tries to mimic some of hardware's characteristics (cache size, associativity, etc), it does not model every single feature and behavior of the system. Therefore there might be some differences in result in some cases.

Question 4 - DGEMM - Perf

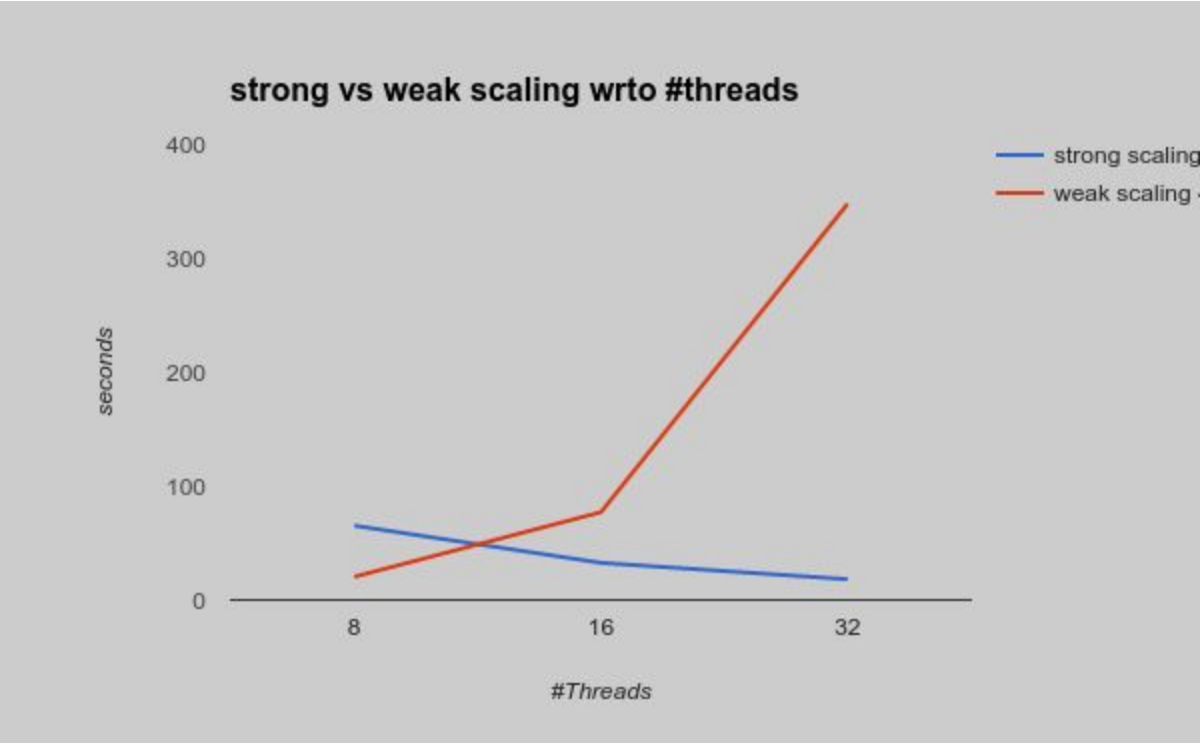


Question 4 - DGEMM - Perf

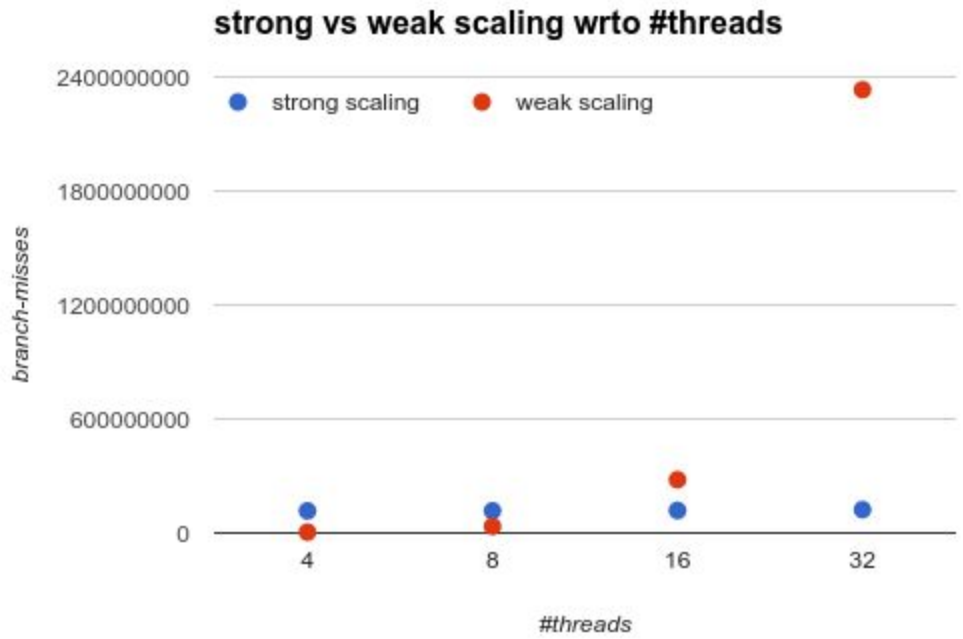
Performance counters for OpenMP DGEMM:

instructions: Number of instructions per cycle and stalled cycles per instructions.
cache-misses: When data is not found in any of the cache.
page-faults: When a program needs part of its virtual memory's content to be copied in the physical memory.
cpu-migrations: On multiprocessor systems Linux tries to keep the workload balanced among the available CPUs.
stalled-cycles-frontend: waste because that means that the CPU does not feed the Back End with instructions.
stalled-cycles-backend: Waste because the CPU has to wait for resources (usually memory) or to finish long latency instructions
L1-dcache-load-misses: When data is not available for loading in L1 cache.
L1-dcache-prefetch-misses: prefetching unused cache lines in L1 data cache
LLC-prefetch-misses: prefetching unused cache lines in LLC
dTLB-load-misses: When data is not found in TLB (TLB miss)
branch-misses: When branch predictor fails.
branch-load-misses: Branch load mispredictions

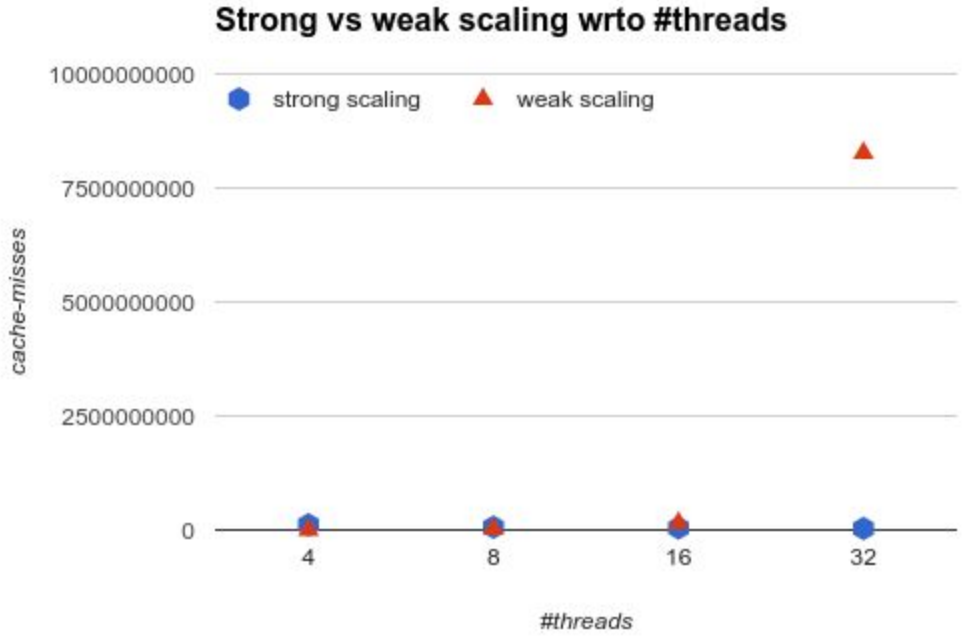
Question 4 - DGEMM - Perf



Question 4 - DGEMM - Perf



Question 4 - DGEMM - Perf



Thank you!