Benchmarking ROOT

Comparison of ICC compiled version vs GCC compiled one.
ROOT on different architectures.
PROOF – CPU benchmark, I/O benchmark.

Mirela-Madalina Botezatu
Supervisor: Andrzej Nowak
Revision 3 : 03/08/2012
Originally published : 05/03/2012

1. Comparison of ICC compiled version vs GCC compiled one.

The compiler versions used to compile ROOT and the version of ROOT itself are shown in the following table.

Root 5.32
GCC 4.6.2
ICC 12.1.2

Optimization levels:

- **GCC - O2 (Full optimization; generates highly optimized code and has the slowest compilation time)**
- **ICC - O2 (Optimizes for code speed and it is the generally recommended optimization level)**

The tests we ran are classical root tests used in benchmarking root's performance.

- **./bench -b –q**
- **./stress -b –q** *well known root test (mixture of I/O and CPU)*
- **./stressShapes -b –q**
- **./stressLinear** *linear algebra test*
- **./stressSpectrum -b –q** *peak search (typical pattern recognition)*
- **./stressFit** *random number generation and fitting with TMinuit*

As performance measures we took into account the outputted values of Rootmarks and CPU time. These tests were ran on different platforms as it is also relevant to see on which we obtain the best results.

The different architectures are:

Code Name	Model	Frequency	Cores	Sockets	Hyper-Threading	Cache	RAM
Westmere	Intel(R) Xeon(R) CPU X5650	2931 MHz	24	2	ON	12288KB	48GB
Sandy Bridge	Intel(R) Xeon(R) CPU E5-2680	2713 MHz	32	2	ON	20480KB	64GB
Magny Cours	AMD Opteron(tm) Processor 6164 HE	1679 MHz	48	4	N.A.	12288KB	96GB

In order to compare the results we had to scale the values according to the frequency of the CPU-s we were running on.

We introduce a scaling factor, a value by which we will multiply our results for Rootmarks and divide the CPU time, in order to make a fair comparative analysis. This scaling factor is computed with respect to the frequency we have on Westmere machine.

We have three different machines with different micro-architectures. Frequency is just one of the differentiating features but it tells us how much work can be done by the CPU in a certain time.
As in our paper we present the values for CPU time and for Rootmarks (a performance measure that is also based on CPU time), we can have some insight on the behavior of these benchmarks on different architectures if we scale the results relative to a fixed frequency.

Consequently:

- **Westmere 2931 MHz**
- **Sandy Bridge 2713 MHz -> scaling factor = 1.09**
- **Magny-Cours 1679 MHz -> scaling factor = 1.72**

Turbo mode was enabled on all the machines. We did not set CPU affinity

Our *frequency scaled* results are the following:

Westmere

GCC 4.6.2

Test	Rootmarks	CPU Time (seconds)
stressFit	2588	3.7
stress	1647	22.6
stressShapes	3269	1.4
stressLinear	2049	10.3
stressSpectrum	2754	5.5
bench	2571	60.9

ICC 12.1.2

Test	Rootmarks	CPU Time (seconds)	Gain (%)
stressFit	3017	3.1	16.5
stress	1727	21.5	4.8
stressShapes	3880	1.2	18.6
stressLinear	2202	9.8	7.4
stressSpectrum	3064	4.9	11.2
bench	2505	62.3	-2.5

Gain* represents the increase in performance we obtain in ICC compiled version vs the GCC compiled one.

Sandy Bridge

GCC 4.6.2

Test	Rootmarks	CPU Time (seconds)
stressFit	3023	3.15
stress	1795	27.5
stressShapes	3515	1.35
stressLinear	2405	8.96
stressSpectrum	3198	4.7
bench	3168	49.1

ICC 12.1.2

Test	Rootmarks	CPU Time (seconds)	Gain (%)
stressFit	3472	2.7	14.8
stress	1910	19.2	6.4
stressShapes	4229	1.12	20.3
stressLinear	2551	8.4	6.0
stressSpectrum	3488	4.3	9.0
bench	3026	51.3	-4.4

Gain* represents the increase in performance we obtain in ICC compiled version vs the GCC compiled one.

AMD

GCC 4.6.2

Test	Rootmarks	CPU Time (seconds)
stressFit	2201	4.4
stress	1613	22.9
stressShapes	2879	1.4
stressLinear	1788	12
stressSpectrum	2067	7.2
bench	2782	56.3

ICC 12.1.2

Test	Rootmarks	CPU Time (seconds)	Gain (%)
stressFit	2509	3.89	13.9
stress	1680	22.2	4.1
stressShapes	3309	1.3	4.9
stressLinear	1911	11.2	6.8
stressSpectrum	2350	6.5	13.6
bench	2614	60	-6.0

Gain* represents the increase in performance we obtain in ICC compiled version vs the GCC compiled one.

Except for the bench test, we see that we obtain better results on ROOT compiled with ICC than on the version compiled with GCC.

Comparing the Rootmarks, we see that on average ICC is :

- 8.7% better than GCC on the Intel Sandy Bridge machine.
- 7.9% better than GCC on the AMD machine
- 9.3% better than GCC on the Westmere machine

For the ICC compiled versions in terms of CPU time we have an average speedup of:

- 1.3 on Sandy Bridge vs AMD and
- 1.13 on Sandy Bridge vs Westmere.

For the ICC compiled versions in terms of Rootmarks we see that we have :

- 29.6% better results on Sandy Bridge than on AMD
- 14.2 % better results on Sandy Bridge than on Westmere

2. ROOT on different architectures

ICC

Machines / Tests	Westmere	Sandy-Bridge	Magny-Cours
	Rootmarks	Rootmarks	Rootmarks
stressFit	100%	115%	83.1%
stress	100%	110.5%	97.2%
stressShapes	100%	108.9%	85.2%
stressLinear	100%	115.8%	86.7%
stressSpectrum	100%	113.8%	76.6%
bench	100%	120.7%	104.3%

GCC

Machines / Tests	Westmere	Sandy-Bridge	Magny-Cours
	Rootmarks	Rootmarks	Rootmarks
stressFit	100%	116.8%	85%
stress	100%	108.9%	97.9%
stressShapes	100%	107.5%	88%
stressLinear	100%	117.3%	87.2%
stressSpectrum	100%	116.1%	75%
bench	100%	123.2%	108.2%

3. PROOF – CPU benchmark, I/O benchmark

We used the functionality of TProofBench class which steers the running of two type of benchmarks: *cycle-driven* (aka CPU-intensive) and *data-driven* (aka IO-intensive).

CPU:
We start a new session where we set the number of workers equal to the nr of cores we have on the machine (24).
TProof:Open("workers=24")
We called the function:
RunCPU(Long64_t ncycles=-1, Int_t start=-1, Int_t stop=-1, Int_t step=-1)
For ncycles: we left it to default value: 1000000 and also for the other parameters (default values), as it starts with one worker and it stops with the number of workers given as parameter to the session initiated.

I/O:
We called the function:
RunDataSet(const char *dset = "BenchDataSet", Int_t start = 1, Int_t stop = -1, Int_t step = 1);
One has to create a dataset first: for example :MakeDataSet("ssdSATA"), and we also ran it with the default parameters.

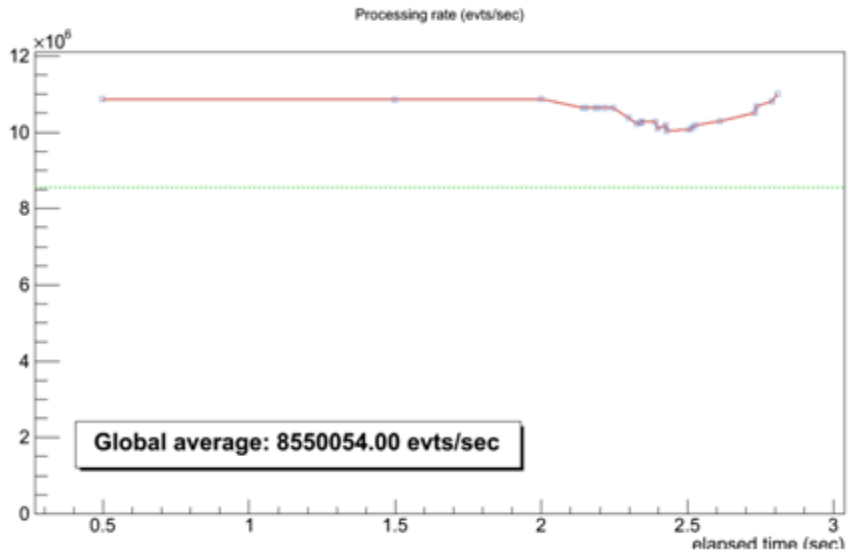
Machine: Westmere [Intel(R) Xeon(R) CPU X5650 2713 MHz, 24 cores , 2 sockets, Hyper-Threading on, Cache size: 12288KB, RAM size: 47 GB]

Results:

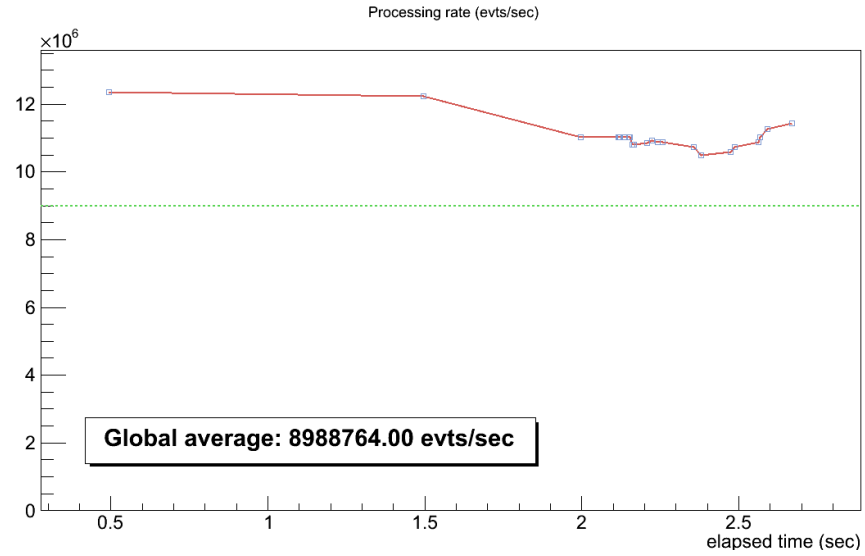
CPU-Intensive

After running the test we obtained the performance plot. Here we have the graph for the last query of the last worker but also we have the global average number of events per second.

- GCC compiled version



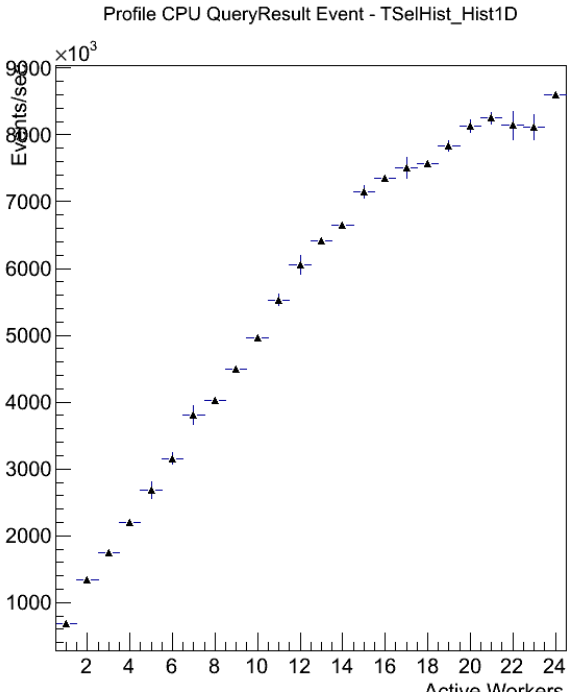
- **ICC compiled version**

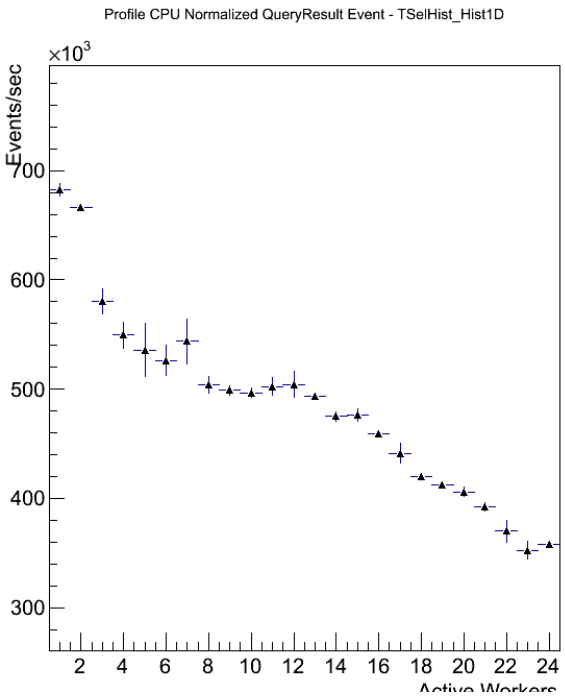


Now if we compare the two figures we see we have better performance for the ICC compiled version than on the GCC compiled version, as the global average number of events per second is 5.1% higher on ICC compiled version.

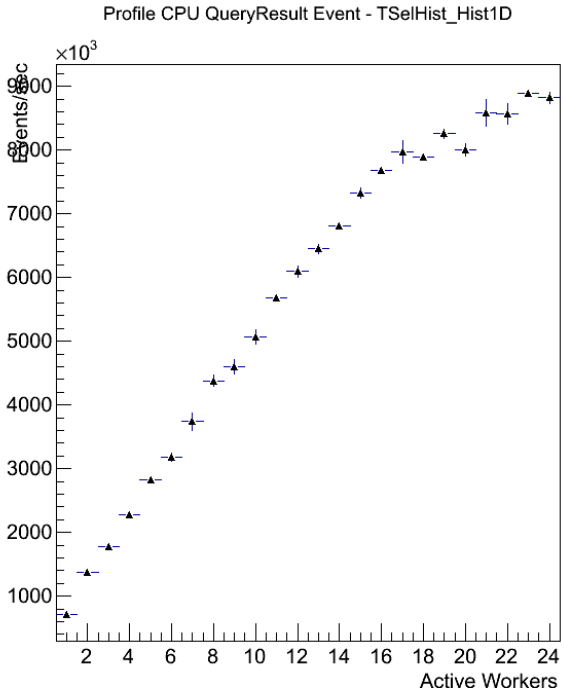
SCALING:

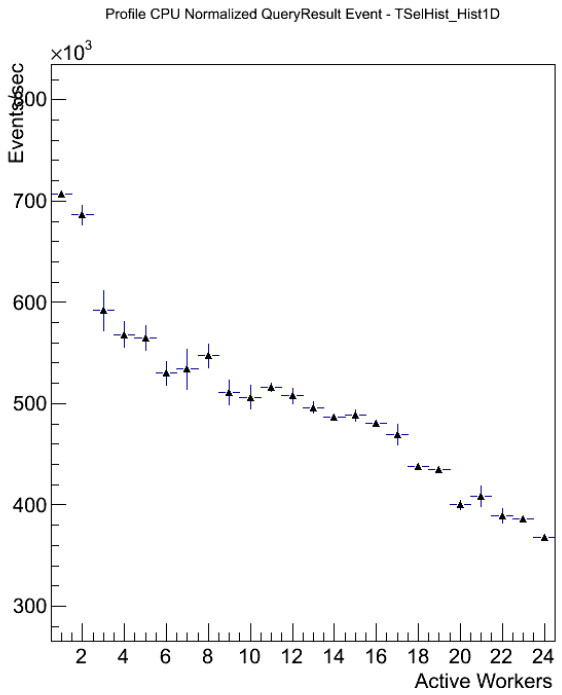
- **GCC compiled version:**





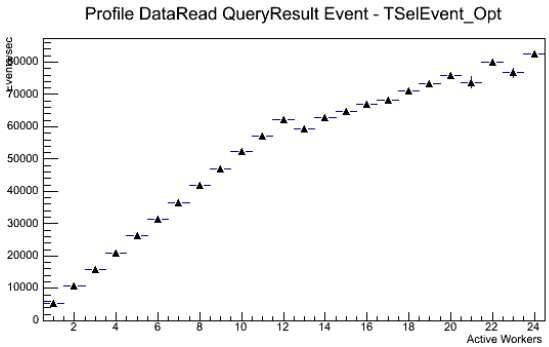
- **ICC compiled version:**

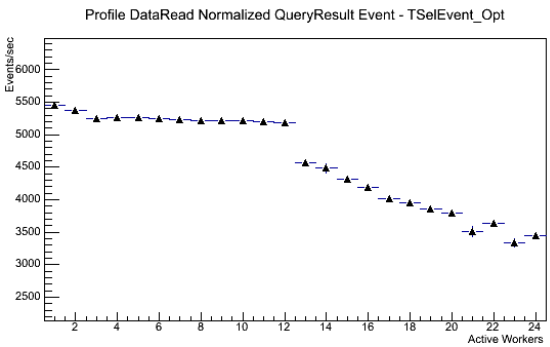


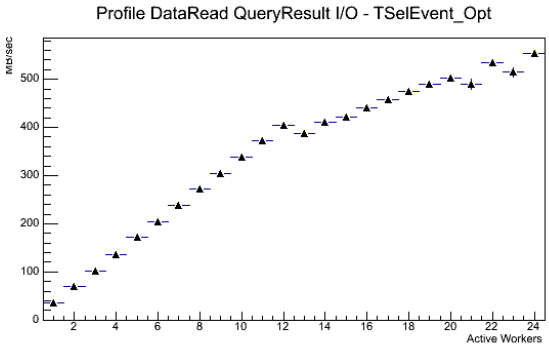


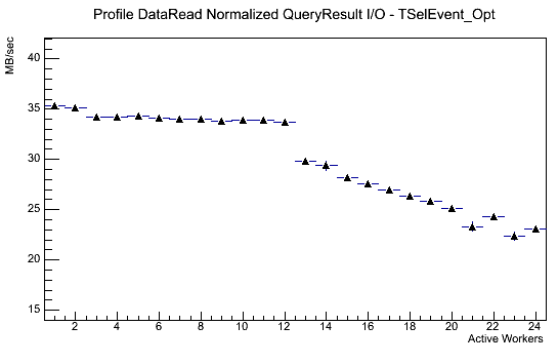
I/O-Intensive

- **GCC compiled version:**

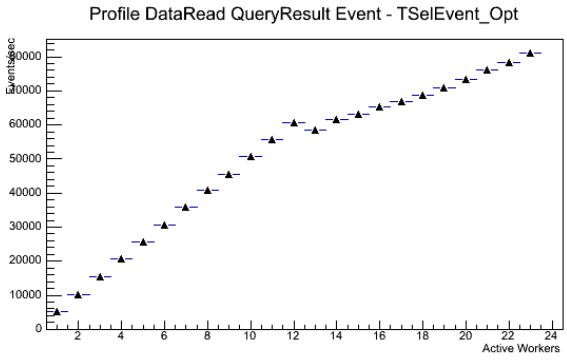


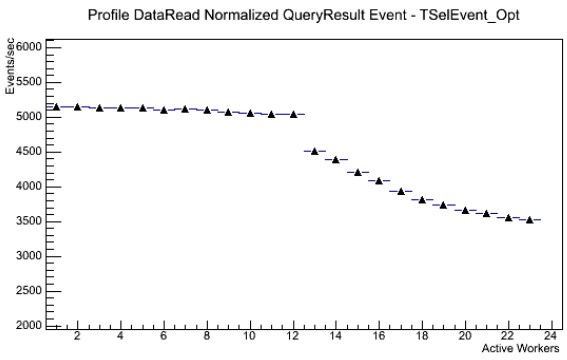


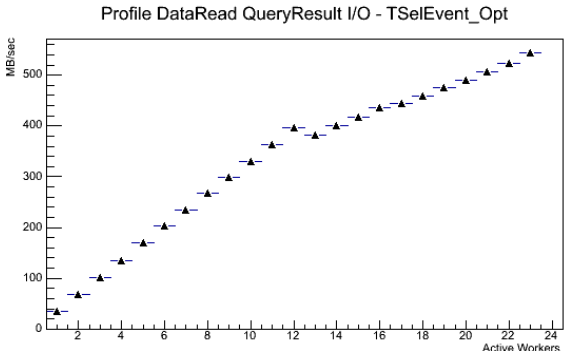


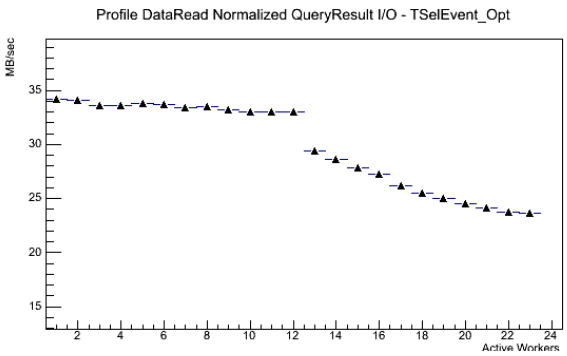


- **ICC compiled version:**









For the I/O intensive tests we see that there is no big difference between GCC compiled version and ICC compiled version both versions managing to reach a performance of ~560MB/sec with 24 workers.

Also we see that due to the fact that Hyper-Threading is on we have a big jump when we go from 12 workers to 13 workers. We have 24 logical cores but starting from the 13th worker two threads will be running on the same core competing for the same resources. As we are running I/O intensive tests, the memory subsystem is the limiting factor that produces this behavior.

Also the I/O intensive test is very relevant to see how different storage hardware influences the performance.

Machine: Arrandale [Intel(R) Xeon(R) CPU E7- 4870 2400MHz, 80 cores , 4 sockets, Hyper-Threading on, Cache size: 30720 KB, RAM size: 125GB]

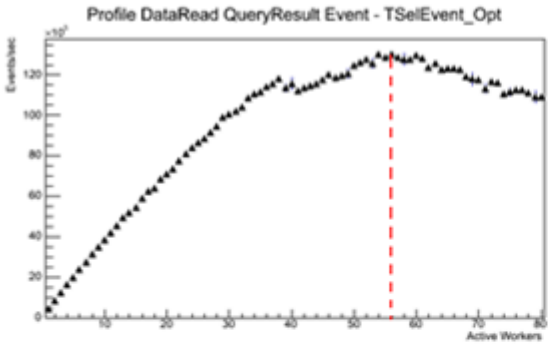
Disks:

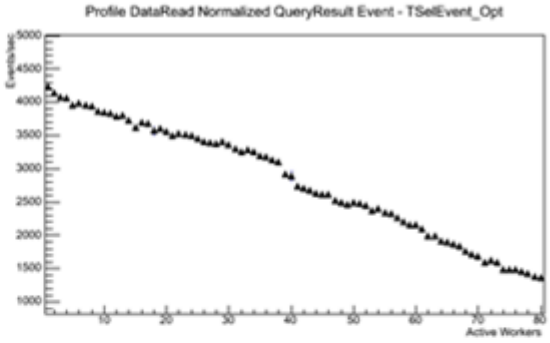
- **Hard Drive**
- **SSD**
- **SSD (Ramsdale)**
- **RAM**

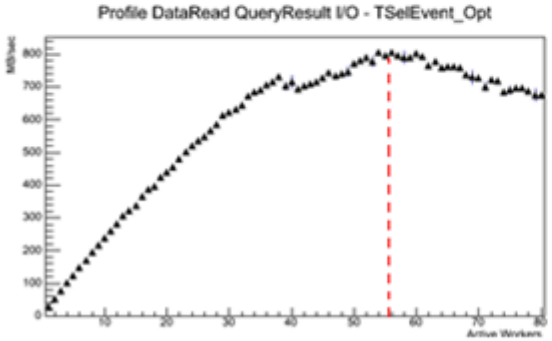
As expected the best performance was reached on the RAM partition - /dev/shm/ where we obtained a peak performance of 1042 GB/sec with 79 workers. The results obtained after running on SSD were not far from what we obtained on the RAM partition, more precisely peak -> 1023 GB/sec with 79 workers.

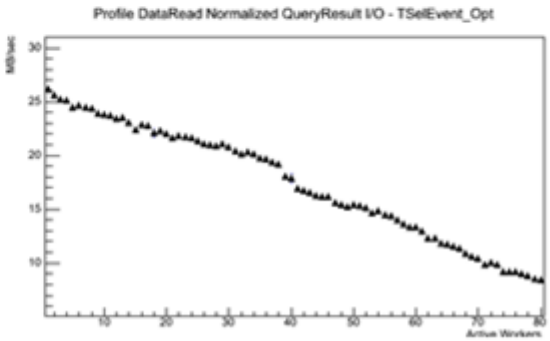
On the hard drive we had 819 GB/sec peak performance reached with 57 workers. (With 79-80 workers the performance was worse ~ 677GB/sec).
These results are also illustrated in the pictures bellow:

Hard Drive

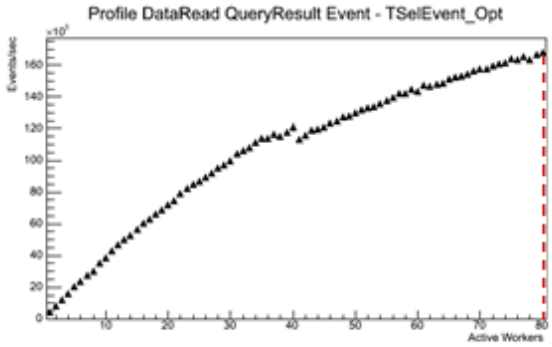


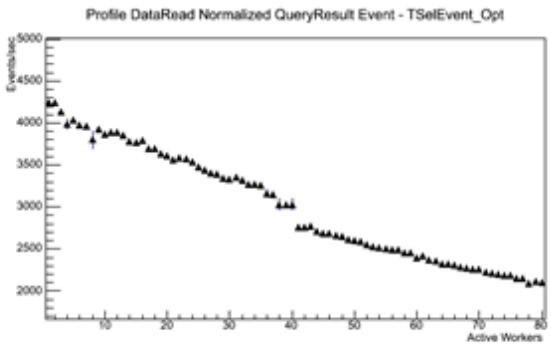


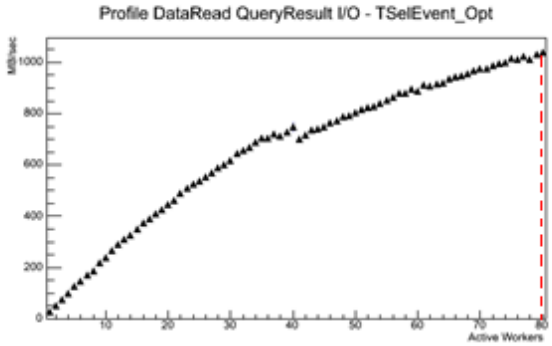


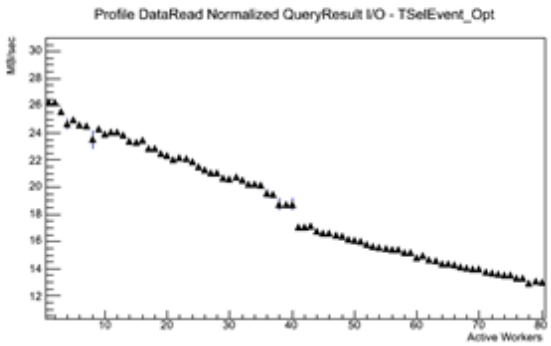


SSD (/RMD_data)

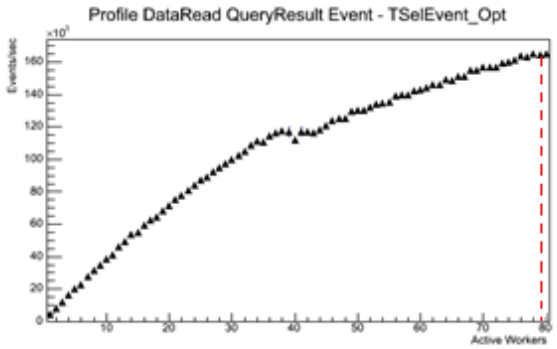


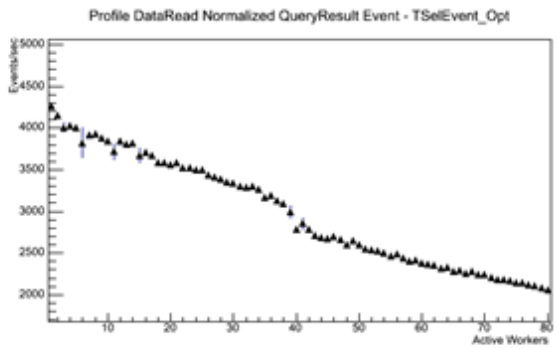


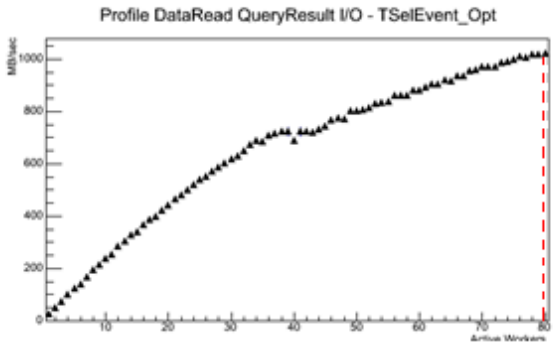


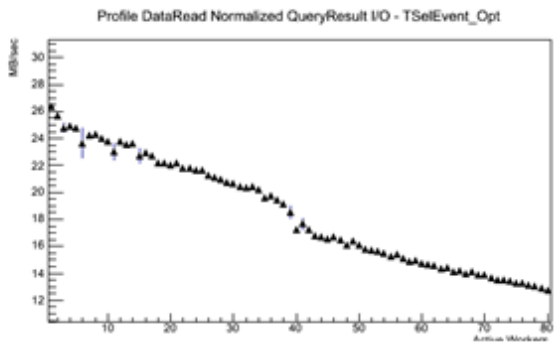


SSD

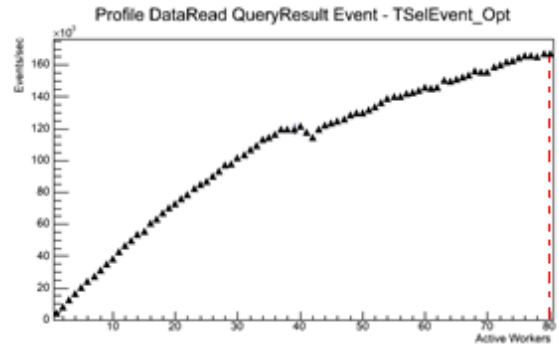

Profile DataRead QueryResult Event - TSelEvent_Opt
Events/sec
Active Workers

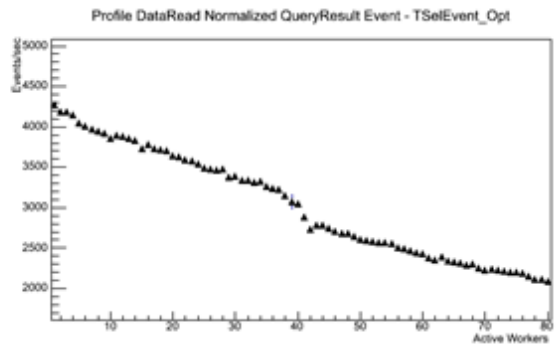

Profile DataRead Normalized QueryResult Event - TSelEvent_Opt
Events/sec
Active Workers

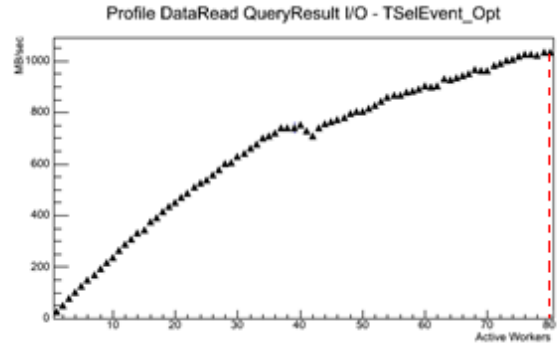

Profile DataRead QueryResult I/O - TSelEvent_Opt
MB/sec

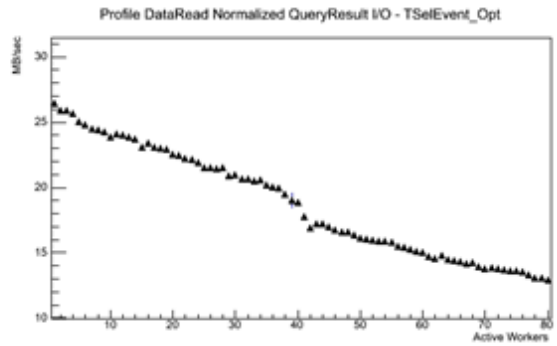

Profile DataRead Normalized QueryResult I/O - TSelEvent_Opt
MB/sec

RAM


Profile DataRead QueryResult Event - TSelEvent_Opt
Events/sec
Active Workers


Profile DataRead Normalized QueryResult Event - TSelEvent_Opt
Events/sec
Active Workers


Profile DataRead QueryResult I/O - TSelEvent_Opt
MB/sec
Active Workers


Profile DataRead Normalized QueryResult I/O - TSelEvent_Opt
MB/sec
Active Workers