60.9. Stockfish Benchmarks

Subtitle: Stockfish 14.1 *and* NEON_opt *Stockfish benchmarks on six different CPUs*

This is a summary of the benchmark results for Stockfish on six different Apple computers dating from 2013 through 2021.

The Stockfish chess engine[96] does not achieve high nodes per second search results on Apple computers, probably because it has been highly optimized for x86_64 CPU architectures[97]. So you may be wondering why run Stockfish benchmarks on Apple computers[**112**], and this is a very good question. The chief reason is because Stockfish benchmarks can be employed to compare the speeds of various Apple computers performing typical, unoptimized, computational tasks. These benchmarks are not meant to compare Apple computers with other types of personal computers, but only to make comparisons internally for the Apple product line.

In addition, Stockfish is an available open-source engine that can be compiled on multiple different computers including iOS and Android devices under different operating systems, CPU architectures, and external GPUs. There has been a recent attempt at some optimization of Stockfish for Apple computers, and this code is also available. Thus in the following benchmarks I employ the Homebrew^{TM} port[**112**] of Stockfish as well as the 'NEON_opt' optimized version by Sopel97 (Tomasz Sobczyk)[**108**].

We begin by listing the specifications for the six computers tested in these Stockfish benchmarks:

(I) MBP2021-16: 2021 MBP 16″ M1 Max (**M1 Max**)
  (a) MacBookPro18,2
  (b) OS: macOS Monterey 12.0.1:
  Darwin Kernel Version 21.1.0: Wed Oct 13 17:33:24 PDT 2021;
  root:xnu-8019.41.5 1/RELEASE_ARM64_T8101 arm64
  (c) CPU: 10-core Apple M1 Max (8 performance, 2 efficiency)
  (d) GPU: 32-core Apple M1 Max
  (e) NE: 16-core Apple M1 Max
  (f) Memory: 64 GB LPDDR5 UMA (integrated on SOC)
  400 GB/s memory bandwidth
  (g) SSD: APPLE SSD AP8192R, Apple Fabric
  7.4 GB/s R/W bandwidth

(II) MBP2020-13: 2020 MBP 13″ M1 (**M1**)
  (a) MacBookPro17,1
  (b) OS: macOS Monterey 12.0.1:
  Darwin Kernel Version 21.1.0: Wed Oct 13 17:33:24 PDT 2021;
  root:xnu-8019.41.5 1/RELEASE_ARM64_T8101 arm64
  (c) CPU: 8-core Apple M1 (4 performance, 4 efficiency)
  (d) GPU: 8-core Apple M1
  (e) NE: 16-core Apple M1
  (f) Memory: 16 GB LPDDR4 UMA (integrated on SOC)
  (g) SSD: APPLE SSD AP2048Q, Apple Fabric

[96] Various versions of Stockfish[**110**] have won the 'Top Chess Engine Championship' 11 times, more than any other chess engine[**111**].

[97] Stockfish also runs fastest on specialized equipment, such as those having $64K$ cores, 33 TB of DRAM memory, and even the Syzygy 7-piece endgame tablebase files require more than 18.4 TB of SSD storage[**24**]. Initially, before looking into NNUE, I naively thought that the neural network evaluation (NNUE) ran at least partially on the GPU but this appears not to be the case. There is, however, a neural network trainer that does run on GPUs.

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

(III) MBP2019-15: 2019 MBP 15″ i9 9980HK (**9980HK**)
  (a) MacBookPro15,3
  (b) OS: macOS Monterey 12.0.1:
  Darwin Kernel Version 21.1.0: Wed Oct 13 17:33:23 PDT 2021;
  root:xnu-8019.41.5 1/RELEASE_X86_64 x86_64
  (c) CPU: 8-core Intel Core i9 9980HK 2.4 GHz
  (d) GPU: Intel UHD Graphics 630; Radeon Pro Vega 20 4GB VRAM
  (e) Memory: 32 GB DDR4 2400 MHz
  (f) SSD: APPLE SSD AP4096M, PCI-Express

(IV) MBP2016-15: 2016 MBP 15″ i7 6920HQ (**6920HQ**)
  (a) MacBookPro13,3
  (b) OS: macOS Monterey 12.1:
  Darwin Kernel Version 21.2.0: Sun Nov 28 20:28:54 PST 2021;
  root:xnu-8019.61.5 1/RELEASE_X86_64 x86_64
  (c) CPU: 4-core Intel Core i7 6920HQ 2.9 GHz
  (d) GPU: Intel HD Graphics 530; Radeon Pro 460 4GB VRAM
  (e) Memory: 16 GB DDR3 2133 MHz
  (f) SSD: APPLE SSD SM2048L, PCI-Express

(V) iMac2013-27: 2013 iMac 27″ i7 4771 (**4771**)
  (a) iMac14,2
  (b) OS: macOS Catalina 10.14.7:
  Darwin Kernel Version 19.6.0: Sun Nov 14 19:58:51 PST 2021;
  root:xnu-6153.141.50 1/RELEASE_X86_64 x86_64
  (c) CPU: 4-core Intel Core i7 4771 3.5 GHz
  (d) GPU: NVIDIA GeForce GTX 780M 4GB VRAM, PCLe
  (e) Memory: 32 GB DDR3 1600 MHz
  (f) SSD: APPLE SSD SM1024F, PCI

(VI) MBP2013-15: 2013 MBP 15″ i7 4960HQ (**4960HQ**)
  (a) MacBookPro11,3
  (b) OS: macOS Big Sur 11.6.2:
  Darwin Kernel Version 20.6.0: Wed Nov 10 22:23:07 PST 2021;
  root:xnu-7195.141.14 1/RELEASE_X86_64 x86_64
  (c) CPU: 4-core Intel Core i7 4960HQ 2.6 GHz
  (d) GPU: Intel Iris Pro; NVIDIA GeForce GT 750M 2GB VRAM, PCLe
  (e) Memory: 16 GB DDR3 1600 MHz
  (f) SSD: APPLE SSD SM1024F, PCI

Each of the above six Apple computers were benchmarked using four Stockfish programs, listed below.

(i) Stockfish 14.1: Homebrew (**SF:14.1**)[**112**]
  There are several ways of obtaining Stockfish, all the way from its github site for the source code[**110**] to already compiled programs such as that supplied by Homebrew™[**112**]:
  https://formulae.brew.sh/formula/stockfish#default
  The Homebrew version may be downloaded and installed *via* the Terminal command-line (after installing Homebrew, of course):
  brew install stockfish
  and Homebrew may be installed by executing the Terminal command:
  /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
(ii) Stockfish 14.1+nnue: (**SF:14.1:nnue**)[**112**]
(iii) Stockfish NEON_opt: (**SF:NEON_opt**)[**108**]

Optimization: NEON_opt and compiled with PGO: Stockfish 081221
https://github.com/Sopel97/Stockfish/tree/neon_opt
For Apple Silicon: make -j profile-build ARCH=apple-silicon
For Intel Silicon: make -j profile-build ARCH=x86-64-modern

(iv) Stockfish NEON_opt+nnue: (**SF:NEON_opt:nnue**)[**108**]
Optimization: NEON_opt and compiled with PGO: Stockfish 091221

The benchmarks were run using a hash (transposition) table size of 8192 MB, a depth (level) of 28, and three alternative thread counts: a maximum thread count allowed for each CPU, a performance thread count for each CPU, and a minimum (single) thread count.

The benchmark statistics were collected through execution of bash scripts using the following commands. *(The scripts executing Stockfish and collecting the results for plotting are 'caffeinated' so that the computers do not enter sleep mode during the benchmark tests.)*

(1) . caffeinate-script.sh stockfish-benchmark-runs.sh MBP16-M1Max $\Longrightarrow$ execute various stockfish benchmarks on the 2021 MBP 16″ M1 Max
(2) . caffeinate-script.sh stockfish-benchmark-runs.sh MBP13-M1 $\Longrightarrow$ execute various stockfish benchmarks on the 2020 MBP 13″ M1
(3) . caffeinate-script.sh stockfish-benchmark-runs.sh MBP15-i9-9980HK $\Longrightarrow$ execute various stockfish benchmarks on the 2019 MBP 15″ i9 9980HK
(4) . caffeinate-script.sh stockfish-benchmark-runs.sh MBP15-i7-6920HQ $\Longrightarrow$ execute various stockfish benchmarks on the 2016 MBP 15″ i7 6920HQ
(5) . caffeinate-script.sh stockfish-benchmark-runs.sh iMac27-i7-4771 $\Longrightarrow$ execute various stockfish benchmarks on the 2013 iMac 27″ i7 4771
(6) . caffeinate-script.sh stockfish-benchmark-runs.sh MBP15-i7-4960HQ $\Longrightarrow$ execute various stockfish benchmarks on the 2013 MBP 15″ i7 4960HQ

The actual Stockfish executions were run *via* the script having three parameters passed to it as arguments:

```
. stockfish-bench.sh ${max_TT} ${max_threads} ${max_limit}
. stockfish-bench.sh ${max_TT} ${performance_threads} ${max_limit}
. stockfish-bench.sh ${max_TT} ${min_threads} ${max_limit}
```

and the actual command-line executing the four varieties of Stockfish is:

```
${Which_Stockfish_opt} bench ${Stockfish_TT} ${Stockfish_Threads} ${Stockfish_Limit} \
default depth ${Stockfish_NNUE} 2 > &1 | tail -n 4 | tee -a ${STOCKFISHLOG}
```

where the bash script configures all variables *[the ${max_TT}, ${max_threads}, ${max_limit}, ${Which_Stockfish_opt}, ${Stockfish_TT}, ${Stockfish_Threads}, ${Stockfish_Limit}, ${Stockfish_NNUE}, and ${STOCKFISHLOG} variables]* for the benchmark runs and collects and outputs the results and timing measurements to a log file and to a data file for plotting. *(The data file is further automatically edited using 'cat', 'grep', 'sed', and 'awk' so that it may be plotted using* gnuplot *to generate the 3-dimensional candlestick plots shown in Figures 60.9.1–60.9.2. See the Stockfish-Benchmark-Results.gnu command file that plots the data from the Stockfish-Benchmark-Results.dat data file to produce these plots.)*

Figure 60.9.1 plots the relative percentages for NPS (Nodes Per Second) for these Stockfish benchmarks employing the maximum number of threads allowed for each CPU. I found

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

little difference in the NPS results when varying the hash table sizes between 1024 MB and 8192 MB, thus all of the benchmarks were run using hash table sizes of 8192 MB. In order for these benchmarks to test the equilibrium thermals for each CPU, the default depths were all set to 28 levels. This produced individual runs ranging from 15 min to 52 min in length, long enough times for the CPU temperatures and fan speeds to reach equilibrium values.

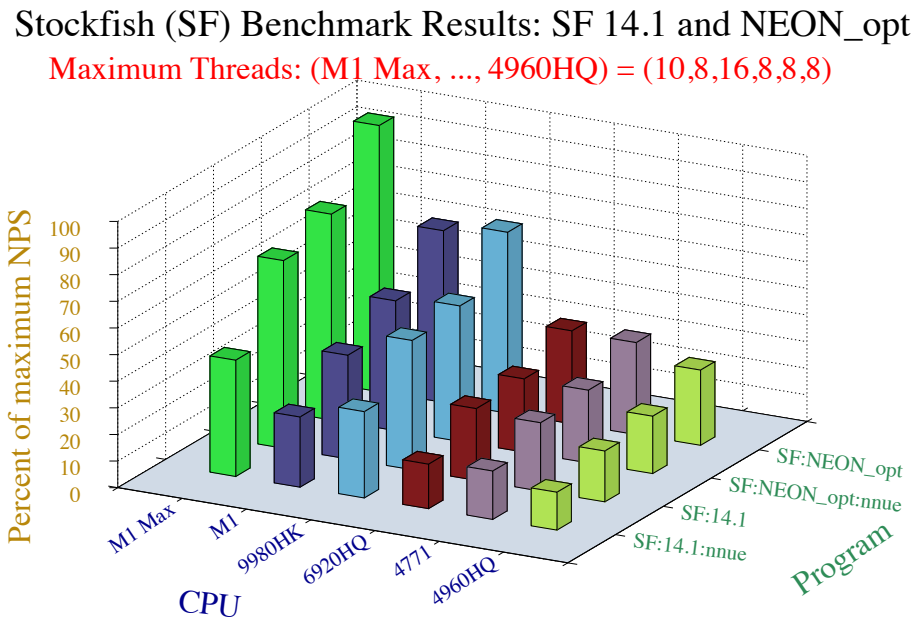

FIGURE 60.9.1. StockfishNews: This Figure plots the relative benchmark results for Stockfish and its optimizations for six different Apple computers employing the maximum number of threads available for the specific CPU.

Figure 60.9.2 includes three plots: (1) the left-hand plot is for the maximum number of threads allowed for each CPU, (2) the middle plot employs the thread number just for the performance cores, and (3) the right-hand plot uses just a single thread.

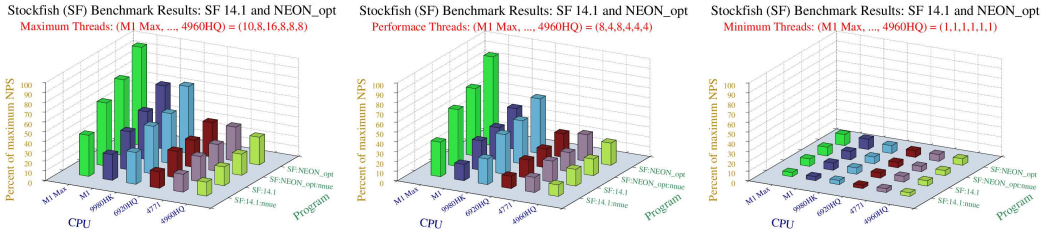

FIGURE 60.9.2. StockfishNews: This Figure plots the relative benchmark results for Stockfish and its optimizations for six different Apple computers employing the maximum number of threads available for the specific CPU, the number of performance threads available, and a single thread.

I executed these benchmark runs a number of times each after fresh reboots of the computers and no other user applications running, and always found results within 5% of each other, thus I plot just a single run's results in order that the plots be more clear.

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

In addition, I also ran these benchmarks separately with the Activity Monitor and the TG Pro applications running in order to record the various temperatures and fan speeds during the runs.

FIGURE 60.9.3. StockfishNews: This Figure includes the Activity Monitor's CPU History plots for the Stockfish benchmarks running on the 2021 MBP 16″ M1 Max using the maximum number of threads allowed for the CPU (left-hand plot), the thread count just for the performance cores (middle plot), and the minimum (single) thread count (right-hand plot).

Figure 60.9.3 shows the Activity Monitor's CPU History plots. Notice that when the maximum number of threads, *i.e.*, 10, are employed, the CPU History plot shows all cores are fully active. When the number of threads equals the count of performance cores, 8, then just the performance cores are fully active. And when a single thread is used, then just the $P0$ cluster of cores are sporadically active (the $P1$ cluster stays inactive).

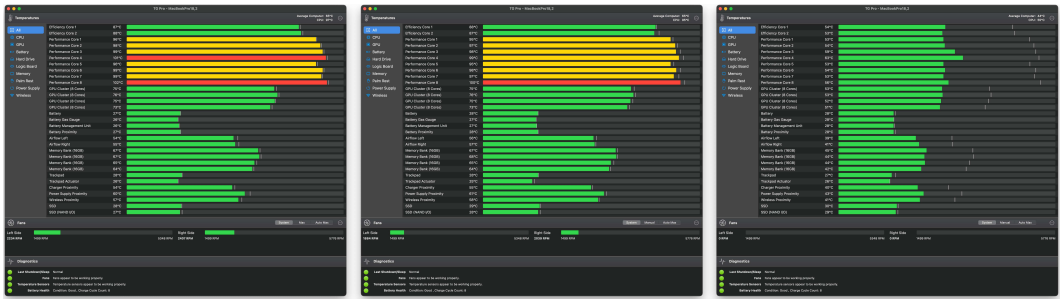

FIGURE 60.9.4. StockfishNews: This Figure includes the TG Pro's CPU temperature and fan plots for the Stockfish benchmarks running on the 2021 MBP 16″ M1 Max using the maximum number of threads allowed for the CPU (left-hand plot), the thread count just for the performance cores (middle plot), and the minimum (single) thread count (right-hand plot).

Figure 60.9.4 shows how the CPU temperatures and fans operate. Notice that even when the thread count is the maximum allowed, the fans are still just at around 40% of their full RPM speeds (the left-hand diagram). The CPU temperatures, on the other hand, hover somewhere just under 100 °C. Thus Apple's algorithm for adjusting the fans allows the CPU temperatures to rise to their maximums and increases the fan RPMs to keep the CPU temperatures near their maximum allowed values. The middle diagram once again shows the CPU temperatures of the performance cores near their maximum values, but now the fan speeds are even lower since the efficiency cores are not being employed. The right-hand diagram shows that when only a single thread is being utilized, the fans can remain off.

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

In addition, notice that the two efficiency cores never warm up to the maximum allowed temperatures of 100 °C.

Since the fans never spun up to their maximum RPMs, I decided to see what would happen if the fans were set to their maximum speeds.

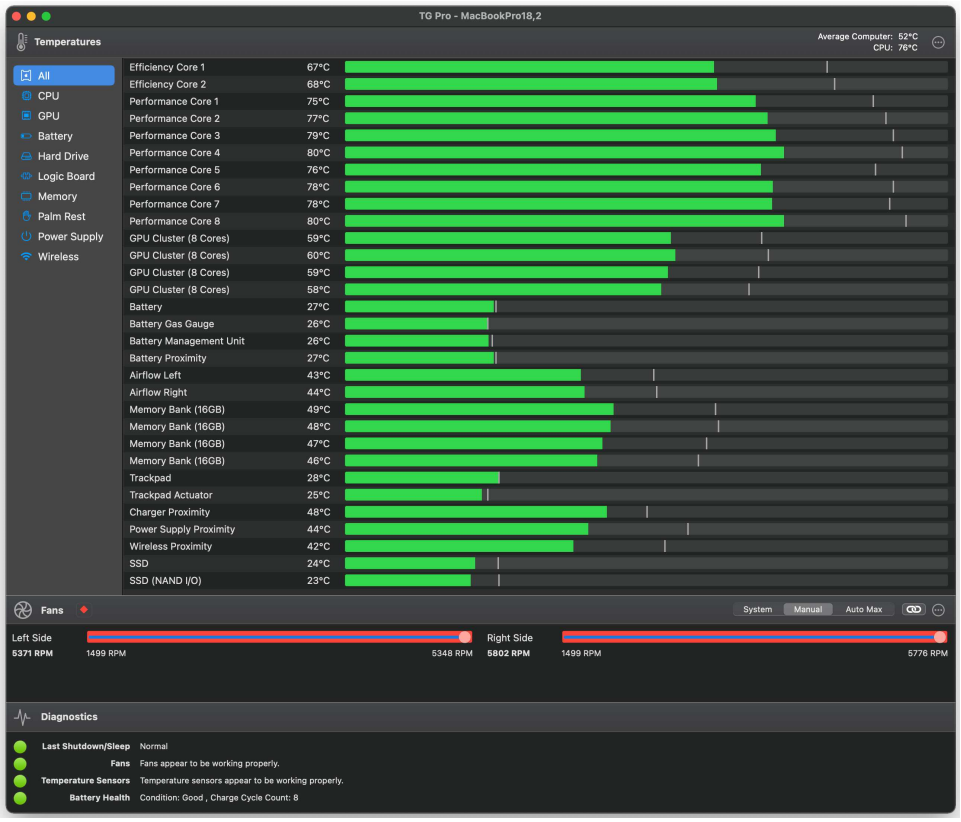

FIGURE 60.9.5. StockfishNews: This Figure shows the TG Pro's CPU temperature and fan plots for the maximum thread Stockfish benchmark when the fans are manually set to their maximum RPM speeds.

Figure 60.9.5 illustrates what occurs when the fans are manually set to their maximum RPM speeds during the maximum thread Stockfish benchmark run. When the fan speeds are at maximum, the CPU temperatures never reach their maximum values of 100 °C, rather the CPU temperatures remain at or below 80 °C.

For the 2020 M1 MacBook Pro 13″, the Activity Monitor's CPU History plots and the TG Pro's temperature and fan plots are analogous to those for the 2021 M1 Max MacBook Pro 16″ diagrams. In particular, these CPU histories and thermal diagrams for the M1 MacBook Pro are shown in Figures 60.9.6–60.9.7. Once again, the fan speed never reaches its maximum RPM value even for the maximum number of threads. Apple's fan speed algorithm allows the CPU temperatures to rise up to their maximum values, 100 °C, and then the fan RPM is increased to keep the CPU temperatures from going any higher.

Figure 60.9.8 shows the corresponding Activity Monitor's CPU History plots for the 2019 MBP 15″ i9 9980HK. These plots are what we expect.

On the other hand, Figure 60.9.9 shows the TG Pro's temperature and fan graphs for the 2019 MBP 15″ i9 9980HK laptop. Now we notice that for both the maximum number of

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

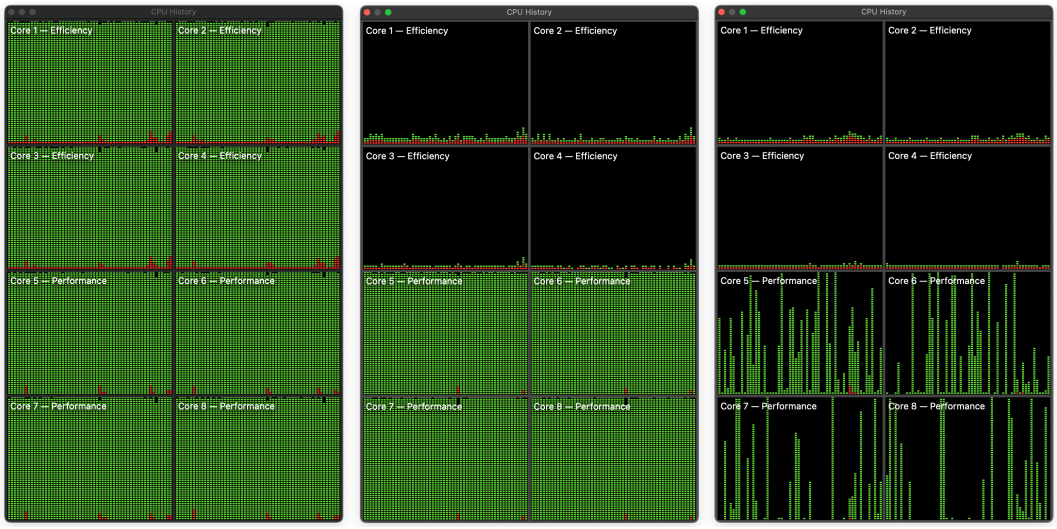

FIGURE 60.9.6. StockfishNews: This Figure includes the Activity Monitor's CPU History plots for the Stockfish benchmarks running on the 2020 MBP 13″ M1 using the maximum number of threads allowed for the CPU (left-hand plot), the thread count just for the performance cores (middle plot), and the minimum (single) thread count (right-hand plot).

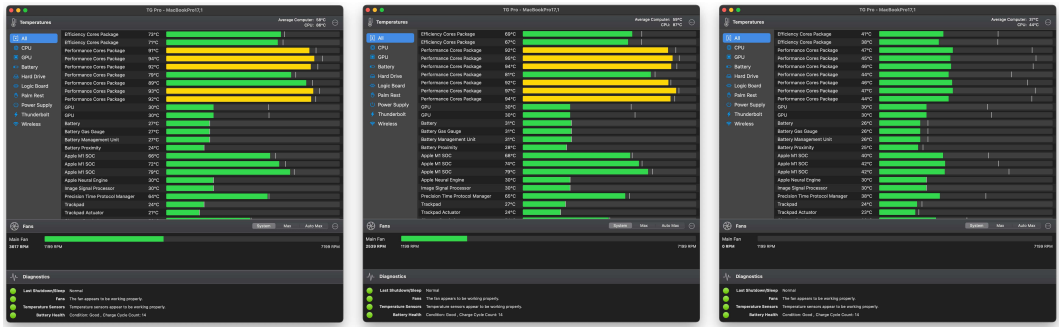

FIGURE 60.9.7. StockfishNews: This Figure includes the TG Pro's CPU temperature and fan plots for the Stockfish benchmarks running on the 2020 MBP 13″ M1 using the maximum number of threads allowed for the CPU (left-hand plot), the thread count just for the performance cores (middle plot), and the minimum (single) thread count (right-hand plot).

threads, 16, as well as for the performance number of threads, 8, the two fan speeds are at their maximum RPM values while the CPU temperatures are also at their maximum values of 100 °C. Even a single thread pushes the fans to roughly 85% of their maximum values, although now the majority of CPU temperatures are below their maximums of 100 °C.

The follow table, Table 60.9.1 on page 16313, lists the actual values of the Stockfish benchmark results[98].

[98] Also, the *Stockfish-Benchmark-Results.txt* includes even more data and measurements results, and the individual, time-dated, log files incorporate all information and data about individual runs of the *stockfish-benchmark-runs.sh* bash script.

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

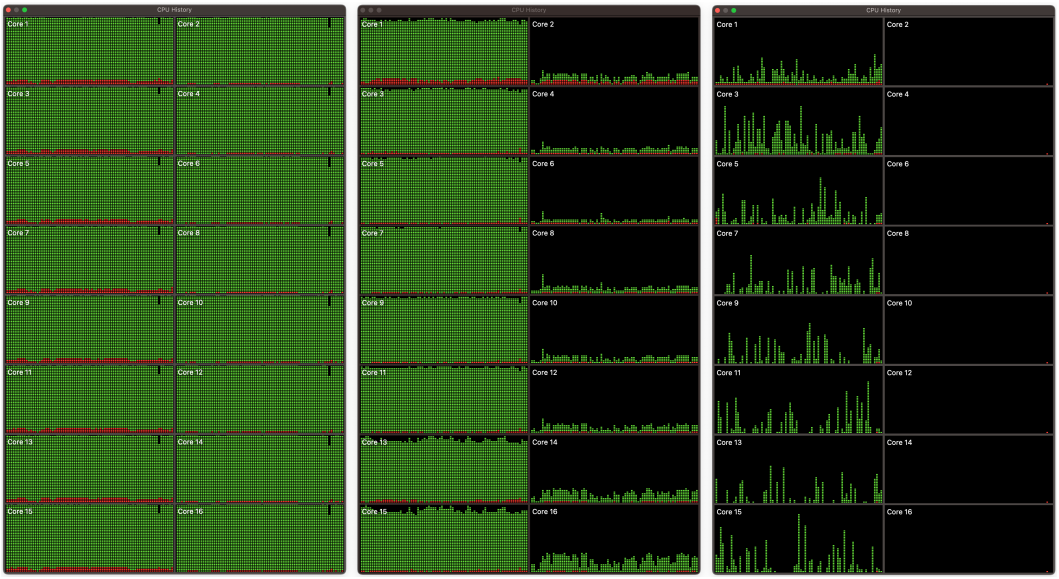

FIGURE 60.9.8. StockfishNews: This Figure includes the Activity Monitor's CPU History plots for the Stockfish benchmarks running on the 2019 MBP 15″ i9 9980HK using the maximum number of threads allowed for the CPU (left-hand plot), the thread count just for the performance cores (middle plot), and the minimum (single) thread count (right-hand plot).

FIGURE 60.9.9. StockfishNews: This Figure includes the TG Pro's CPU temperature and fan plots for the Stockfish benchmarks running on the 2019 MBP 15″ i9 9980HK using the maximum number of threads allowed for the CPU (left-hand plot), the thread count just for the performance cores (middle plot), and the minimum (single) thread count (right-hand plot).

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

Table 60.9.1: These are the Stockfish benchmark comparisons for different Apple computers. The computers are listed by their CPUs: **M1 Max** $\Longrightarrow$ 2021 MacBook Pro 16″, Apple M1 Max 10-core (8 performance, 2 efficiency) CPU, 32-core GPU, 16-core NE, 64 GB LPDDR5 UMA, 8 TB APPLE SSD AP8192R Apple Fabric; **M1** $\Longrightarrow$ 2020 MacBook Pro 13″, Apple M1 8-core (4 performance, 4 efficiency) CPU, 8-core GPU, 16-core NE, 16 GB LPDDR4 UMA, 2 TB APPLE SSD AP2048Q Apple Fabric; **9980HK** $\Longrightarrow$ 2019 MacBook Pro 15″, 2.4 GHz Intel Core i9 9980HK 8-core CPU, Radian Pro Vega 20 4 GB VRAM, 32 GB DDR4 2400 MHz, 4 TB APPLE SSD AP4096M PCI-Express; **6920HQ** $\Longrightarrow$ 2016 Touch Bar MacBook Pro 15″, 2.9 GHz Intel Core i7 6920HQ 4-core CPU, Intel HD Graphics 530, Radeon Pro 460 4GB VRAM 16 GB DDR3 2133 MHz, 2 TB APPLE SSD SM2048L PCLe/NVMe; **4771** $\Longrightarrow$ 2013 iMac 27″, 3.5 GHz Intel Core i7 4771 4-core CPU, NVIDIA GeForce GTX 780M 4 GB VRAM PCLe, 32 GB DDR3 1600 MHz, 1 TB APPLE SSD SM1024F PCI; and **4960HQ** $\Longrightarrow$ 2013 MacBook Pro 15″, 2.6 GHz Intel Core i7 4960HQ 4-core CPU, Intel Iris Pro, NVIDIA GeForce GT 750M 2 GB VRAM PCLe, 16 GB DDR3 1600 MHz, 1 TB APPLE SSD SM1024F PCI. These benchmark results are plotted in Figures 60.9.1 and 60.9.2.

List of Stockfish Benchmarks for 6 CPUs

Program	M1 Max	M1	9980HK	6920HQ	4771	4960HQ
Maximum Threads †						
Threads (count)	10	8	16	8	8	8
SF:14.1:nnue◀ (nps)	8282651	5208318	6404171	3301192	3587174	2809834
SF:14.1▶ (nps)	12715326	7695581	9588313	5326952	5072134	3773080
SF:neon_opt:nnue♣ (nps)	15286013	9646585	10063766	5434614	5394550	4254725
SF:neon_opt♠ (nps)	19764873	12755117	13435124	6900557	6829769	5582630
Wall Clock Time (s)	901	1262	1670	2040	1997	2600
CPU Power⊙ (mW)	26488	8628	–	–	–	–
DRAM Power⊗ (mW)	1944	656	–	–	–	–
GPU Power⊕ (mW)	25	31	–	–	–	–
Package Power⊙ (mW)	33165	9515	56670	53370	–	49420
Performance Threads ‡						
Threads (count)	8	4	8	4	4	4
SF:14.1:nnue (nps)	7151583	3274797	5165538	2466946	3046691	2320471
SF:14.1 (nps)	11748814	5922342	7979932	3655715	4223783	3355015
SF:neon_opt:nnue (nps)	13742389	6468545	8677821	3755498	3834088	3325916
SF:neon_opt (nps)	17899003	8244322	11016059	4736316	5349962	4464028
Wall Clock Time (s)	908	1203	1412	1742	1610	2056
Minimum Threads ♯						
Threads (count)	1	1	1	1	1	1
SF:14.1:nnue (nps)	826088	777518	832918	663978	758772	720312
SF:14.1 (nps)	1483765	1446866	1265651	1036491	1144263	1050987
SF:neon_opt:nnue (nps)	1753129	1680562	1365801	1083851	1073158	1010327
SF:neon_opt (nps)	2209682	2056524	1563329	1320621	1320621	1295761
Wall Clock Time (s)	2234	2344	2558	3152	2899	3088

NOTES:

† $\Longrightarrow$ The **Maximum Threads** for a particular CPU is equal to the number of CPU cores, with hyperthreading on Intel CPUs, available in the CPU. Thus the Intel i9 CPU with 8-cores, being of x86_64 architecture and thus utilizing hyperthreading, may actually run 16 threads. The Apple Silicon M1 Max CPU, being of arm64e architecture and thus not having hyperthreading but rather having higher power 'performance' cores and lower power 'efficiency' cores, may run 10 threads on its 8 performance cores and 2 efficiency cores.

‡ $\Longrightarrow$ The **Performance Threads** for a particular Apple Silicon CPU depends upon its design with the Apple M1 Max having 8 performance cores and 2 efficiency cores. For an Intel CPU, the count of performance CPUs is equal to the number of cores, thereby not utilizing hyperthreading. For the Intel i9 8-core CPU the performance thread count is thus 8.

♯ $\Longrightarrow$ The **Minimum Thread** count for all CPUs is 1.

◀ $\Longrightarrow$ The **SF:14.1:nnue** program is Stockfish 14.1 (Homebrew) running with the 'nnue' evaluation. Stockfish 14.1 is highly optimized for the Intel x86_64 architecture.

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

▶ $\Longrightarrow$ The **SF:14.1** program is Stockfish 14.1 (Homebrew) running with the 'classical' evaluation. Stockfish 14.1 is highly optimized for the Intel x86_64 architecture.

♣ $\Longrightarrow$ The **SF:neon_opt:nnue** program is Stockfish 14.1 with NEON_opt optimizations for the arm64 architecture running with the 'nnue' evaluation. Stockfish 14.1 with NEON_opt is an early optimization for the Apple M1 arm64e architecture.

♠ $\Longrightarrow$ The **SF:neon_opt** program is Stockfish 14.1 with NEON_opt optimizations for the arm64 architecture running with the 'classical' evaluation. Stockfish 14.1 with NEON_opt is an early optimization for the Apple M1 arm64e architecture.

⊙ $\Longrightarrow$ The CPU Power is measured by the 'powermetrics' utility for Apple Silicon. For Intel silicon, the power is measured by an Intel simulation and thus only the Intel Package Power is available which does not include DRAM power nor GPU power.

⊗ $\Longrightarrow$ The DRAM Power is measured by the 'powermetrics' utility for Apple Silicon. For Intel silicon, the power is measured by an Intel simulation and thus only the Intel Package Power is available which does not include DRAM power nor GPU power.

⊕ $\Longrightarrow$ The GPU Power is measured by the 'powermetrics' utility for Apple Silicon. For Intel silicon, the power is measured by an Intel simulation and thus only the Intel Package Power is available which does not include DRAM power nor GPU power.

⊙ $\Longrightarrow$ The Package Power is measured by the 'powermetrics' utility for Apple Silicon; it includes the CPU, DRAM, GPU, and NE powers. For Intel silicon, the Package Power is measured by an Intel simulation and thus does not include DRAM, GPU, or NE powers. Rather the Intel Package Power is described as for CPUs+GT+SA.

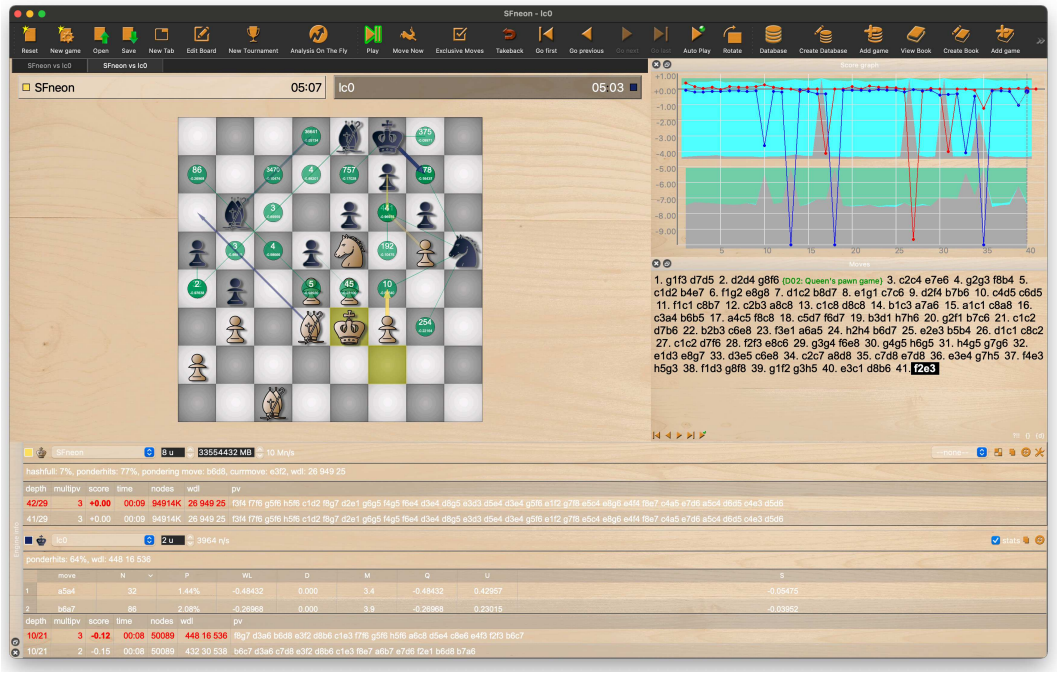

FIGURE 60.9.10. StockfishNews: This Figure shows the Banksia GUI window for a chess game between Stockfish NEON_opt and lc0.

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

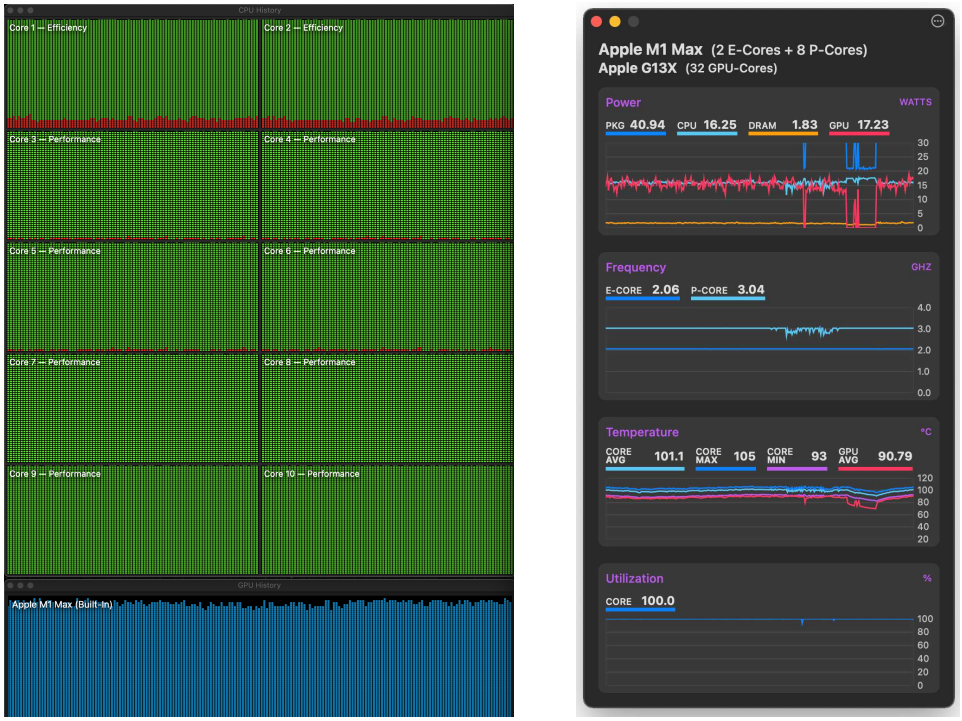

FIGURE 60.9.11. StockfishNews: On the left-hand side, this Figure shows the CPU and GPU Histories from the Activity Monitor while Stockfish NEON_opt is playing a chess game against lc0. On the right-hand side is the Mx Power Gadget window providing the Package, CPU, DRAM, and GPU powers.

While Stockfish by itself is not able to push the Apple Silicon SOCs to their thermal limits, it is interesting that one can employ Stockfish to maximize CPU utilization and, at the same time, employ Leela Chess Zero (lc0) to maximize the GPU utilization. I achieved this result by having Stockfish NEON_opt play against lc0 in a chess GUI.

Figure 60.9.10 provides the Banksia GUI's window for the chess game between Stockfish NEON_opt and lc0. This GUI is a fairly nice one allowing the user to download opening books, endgame databases, and historical games as well as using any chess engine, including humans. In addition, Banksia is open source software that allows the user to design the interface to suit his/her needs. And the Banksia GUI provides many tools for deep analyses of games, including analyses on the fly and plots of WDL, nodes, depth, elapse, speed and tablebase hits. During this particular machine-against-machine game, Stockfish ranged from searching a minimum of about 10 Mnps to a maximun of over 24 Mnps.

While Stockfish NEON_opt was roughly fully utilizing the CPUs of Apple's M1 Max SOC, lc0 was taxing the 32-core GPU of the M1 Max. As you can see on the left-hand side of Figure 60.9.11, the Activity Monitor's CPU History and GPU History panes show that all of the 10 CPU cores as well as all of the GPU cores are being utilized more-or-less fully[99]. The right-hand side of this Figure shows the Mx Power Gadget app that provides graphs of the Package, CPU, DRAM, and GPU powers.

[99] While playing Stockfish against lc0 in a machine-on-machine chess game appears to fully utilize the CPUs and GPUs of Apple's M1 Max, it does not spin up the fans to their maximum RPMs. On the other hand, running some codes do maximize the 2021 MBP 16″'s fans's speeds, thus this is possible; it is just not easy to do with chess engines.

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb

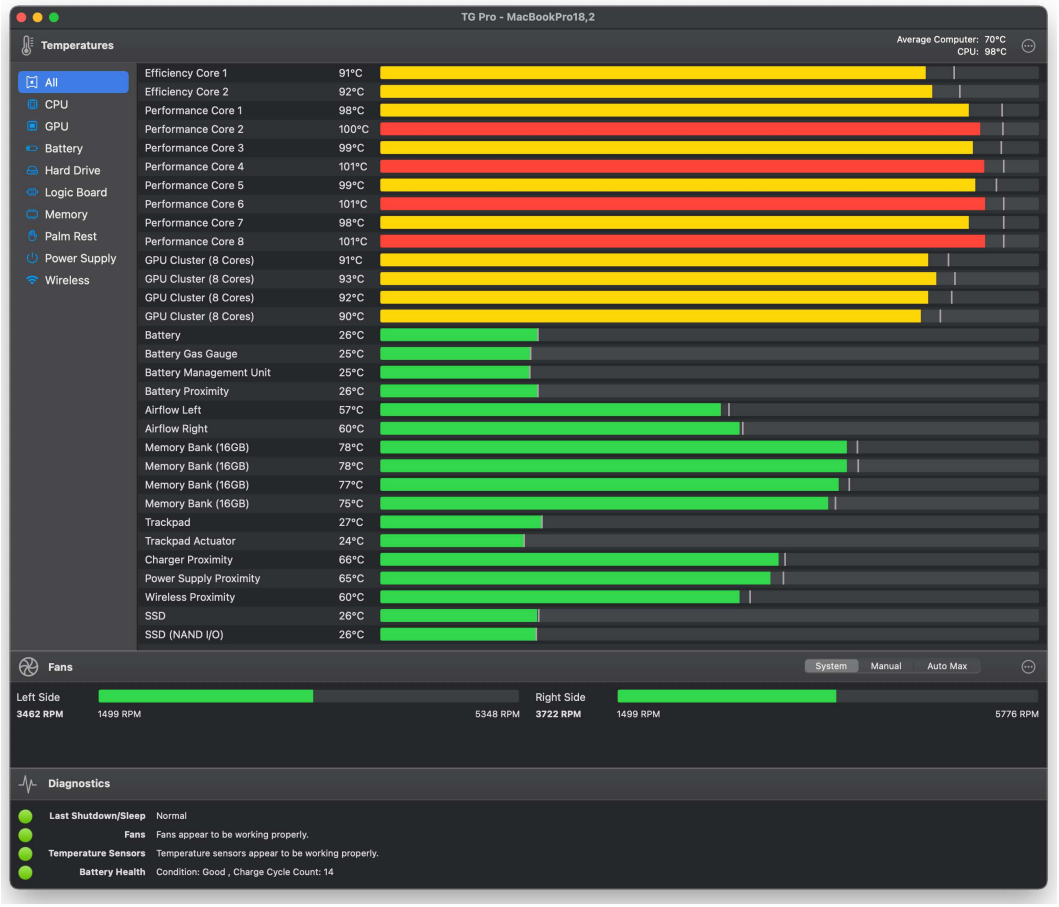

FIGURE 60.9.12. StockfishNews: This Figure shows the TG Pro's main window providing the CPU and GPU temperatures as well as the fan speeds during the Banksia GUI mediated chess game between Stockfish NEON_opt and lc0.

And, lastly, Figure 60.9.12 provides the TG Pro's window giving the CPU and GPU temperatures as well as the fan speeds during this chess game between Stockfish and lc0. Notice that even under these conditions utilizing all of the CPUs and GPUs of the M1 Max SOC, the fans still do not ramp up to their maximum RPM speeds.

IMPORTANT NOTE: (ADDED JANUARY 15, 2022) In order to employ an easily available Stockfish engine, I picked the Homebrew Stockfish 14.1 which supplied a precompiled executable. This is the SF:14.1 engine employed in the benchmarks listed in Table 60.9.1 and plotted in Figures 60.9.1–60.9.2. I also compiled Stockfish 14.1 from source code using the Apple clang 13.0.0 (clang-1300.0.29.30) compiler with PGO. This PGO executable was roughly 5% faster (in terms of NPS) than the Homebrew compiled executable running the NNUE evaluator and roughly 11% faster than the Hombrew compiled executable without NNUE.

UID: 960b1f334d356d2f635e4d704e4aaf42612b92fb