Enhancing User Experience: Krita Application utilizing Multiple Cores on Intel® Architecture Platforms

By Fredrick Odhiambo and Dmitry Kazakov

Krita is a professional, FREE, and open source painting program. It is made by artists that want to see affordable art tools for everyone.

Executive Summary

"Now I can paint with twice and even three times bigger brushes! Which actually means performance is about 5–10 times better." Rad (Igor Wect), Krita* artist.

Computer processing demands have grown and become more complex and intensive. Software applications demand more compute resources. This has led to software performance bottlenecks and unreliable software services, which lead to a poor user experience, and may eventually turn into complete rejection of an application by its users. Multicore refers to a single physical processor packaged into a single integrated circuit, which contains two or more processing units attached for simultaneous processing of multiple tasks, resulting in enhanced performance and reduced power consumption. Intel® Hyper-Threading Technology (Intel® HT Technology), which is available on the latest Intel Core™ vPro™ processors, the Intel Core™ processor family, the Intel Core™ m processor family, and the Intel® Xeon® processor family is a technology that allows a single microprocessor to act like two separate processors to the operating system and to the application program. This white paper highlights how the Krita application combines multicore and Intel HT Technology to improve the application's performance and the overall user experience of Krita. Krita software developers used Intel VTune™ Amplifier to analyze Krita use cases that had been previously identified by Krita users as use cases that required improvement. The VTune Amplifier enabled quick identification of concurrency. These issues were resolved mostly through multithreading and thread synchronization techniques. An improved version of Krita was built and the aforementioned use cases were re-tested for performance gain. The results collected show a measurable performance gain of upto 2X on the different system configurations. System configuration information is included in a later section of this paper.

Software Developers Value Proposition

Multicore and Intel HT Technology afford software developers the ability to write software applications that take advantage of these technologies, and competitively positions the applications to the end user by providing an outstanding user experience. Besides that, software developers through their applications can now:

- Run demanding applications simultaneously while maintaining system responsiveness.
- Keep systems protected, efficient, and manageable while minimizing impact on productivity.
- Provide headroom for future business growth and new solution capabilities.

- Reduce application energy consumption, since multiple cores run on a lower frequency than a single core, which significantly reduces heat dissipation.
- Increase the number of transactions that can be processed simultaneously.
- Utilize existing 32-bit application technologies while maintaining 64-bit future readiness.
- Multimedia applications can create, render, edit, and encode graphically intensive files while running background functions, without compromising overall performance of the application.

Krita* Case Study

Methodology
Krita gathered application usage complaints and issues using Bugzilla on https://bugs.kde.org/. On this online forum, artists and other users report their challenges and pain points from their real-world experiences of using Krita application as far as user experience is concerned. Among the issues raised, slow brush speeds featured prominently and Krita asked issue authors additional questions i.e. system settings to enable them define and de-limit the issue more concretely. From multiple issues raised, two use cases were prioritized: Brushes and Animation rendering.

Using the VTune Amplifier, Concurrency and Hotspot Analysis on affected use cases were done for establishing an in-depth view of performance issues. These issues were resolved through rewriting code, changing libraries, incorporating Intel® Advanced Vector Extensions (Intel® AVX) for Fused Multiply Add instructions, and multithreading the use cases as much as possible. This resulted in development of Krita version 4.0.0, which takes advantage of multicore and Intel HT Technology to improve application performance.

Krita Situation
The VTune Amplifier was used to profile Krita application performance. In-depth Concurrency and Basic Hotspot Analysis were reviewed to establish performance issues in the two use cases of importance (brush painting and animation rendering). The following issues were established**.**

Rendering Use Case

Each frame was being rendered sequentially in almost single thread. After initial profiling by the VTune Amplifier (Concurrency and Waits benchmarks), two issues both related to usage of mutexes were located:

Issue 1: Tiles hash table. All the image data in Krita is represented in a form of 256 by 256 pixel tiles, which are stored in hash tables. Each pixel layer has a hash table containing all the tiles dedicated to the layer, and each hash table is guarded by its own read-write lock (QReadWriteLock). According to the VTune Amplifier profiling results, up to 40 percent of the time was spent waiting on these locks.

Issue 2: The Krita application internal scheduler also had a bottleneck. The tasks in the scheduler are represented as QRunnable objects running on a QThreadPool pool. It was

established that QRunnable-based tasks are too small, and there is high probability of the thread going to sleep just after one task is completed and before the next one is started, so the tasks were aggregated in bunches. Now each QRunnable-based task can *pull* extra tasks from the scheduler, therefore avoiding any sleeps.

These two fixes significantly improved the Krita rendering engine and it showed almost linear speed scaling, up to six physical cores. When adding more cores, the mutexes start to show up again. To ensure Krita scales well on higher CPU core numbers, there were two options—either rewrite the hash table in a lock-free manner or use more high-level methods of multithreading. The second approach was chosen.

Solution
Tiles Hash table issue was resolved by the usage of two-phase locking inside the hash table (KisTileHashTableTraits<T>::getTileLazy()). At first an attempt to fetch the tile **without any lock** held (KisTileHashTableTraits<T>::getTileMinefieldWalk()), and only if this process fails, the lock is taken and the process is repeated. Such trick is thread-safe because of two reasons: thread-safe shared pointers are used inside the hash table (so in the worst case the thread will just read outdated data) and the high level scheduler guarantees that two threads will never try to read +write to the same pixel of the image (it means that the tile cannot be deleted while some other thread is still accessing it). Such two-phase locking significantly reduced the contention over the lock.

When saving animation video clip, all frames need to be rendered and saved into separate files. Technically, each frame represents a different version of the same image, but with its own pixel data. Therefore different frames cannot be rendered on the same image at the same time. But copies of the image can be made! If each copy is passed to its own thread, the rendering processes will be fully independent!

A possibility of making shallow copies of the image was implemented. When creating a shallow copy, most of the pixel data becomes shared between the two images. No extra memory is needed except for the temporary “projections” used for rendering. A tests on a huge animation file was done by a professional animator, which occupies about 7.2 GB of RAM when fully loaded. Making a shallow copy of such image takes only about 200MB extra RAM. Regular user’s files demand even less extra memory. Even though shallow copies don’t occupy too much extra memory, it still takes a bit of time to create them. A trade off was considered: not creating a dedicated copy for each core, but create a few copies and assign each copy to several cores. E.g. in a scenario whereby we have 6+6 cores, create 4 shallow copies and assign each copy to 5 threads. In the result, CPU is utilized almost to 100% by making 12 threads and run almost without interdependencies.

Caution: when doing such memory/efficiency tradeoffs, one should pay great attention to the amount of memory the user has. After a few tests on real users it was established that even though the user can adjust the number of “clones” in the settings, none of the users would like to think about it. Therefore an automatic clones limit was implemented. Before cloning the image, calculation of the actual amount of memory the projections **will** take is done and compared to the amount of memory the user allowed Krita to use.

Painting Use Case

The brushes' painting was inherently a sequential algorithm. The brush itself is just a set of small, round images (called *dabs*), which are painted one by one along the stroke line with small offsets. The situation is even more complicated, because some of the dabs could have dependencies between each other: two dabs can use the same *base* dab, but apply different post-processing to it.

Solutions

The solution for the problem consists of two parts:

1. The *dabs* themselves are prepared in parallel, asynchronously, from the main rendering thread. Here is the tricky part—when *dabs* are dispatched for parallel execution, they might become reordered, and some *newer dabs* will become ready before the *older* ones are done. A special queue was implemented (KisDabRenderingQueue) that handles all these cases, gathers the *dabs* into *sequential* bunches, and passes them further down the pipeline.
2. Each 20–100ms the *dab* rendering process is stopped, all prepared *dabs* taken then rendered onto a canvas. This rendering is also done in parallel: the working area is split into multiple smaller areas which are assigned to different threads.

Together with rendering core optimizations, this work guarantees that the brushes now scale nicely, up to 6 physical cores.

An interesting observation was made on the most responsive brush (Basic_tip_default), which utilizes Intel AVX technology for vectorized calculations. After all the optimizations were done, it became so fast that it almost hit the ceiling of the memory throughput (according to the VTune Amplifier, when painting with a 1000 px brush on a 5k canvas).

Future Krita Optimization Goals

- It was established that the brush, optimized with Intel AVX technology, renders 4–6 times faster than non-optimized ones. The next step in brushes optimization is to implement vectorized versions of other brushes.
- Utilizing specific integer instructions introduced in Intel® Advanced Vector Extensions 2 (Intel® AVX2) for composition can also give up to two times better rendering speed.
- Implementing a lock-free hash table for tiles can also give some benefit for systems with over 6 physical cores.

Results

After implementing the solutions above, results were collected after benchmarking tests were done on two systems with system configurations below.

System 1:
Processor: Intel® Core™ i7-8550U @ 1.80GHz, 4Cores
Physical Memory: 8GB DDR4 2400MHz
Operating System: Windows 10® Pro (64 bit)

System 2
Processor: Intel® Core™ i7-8550U @ 3.2GHz, 6Cores

Physical Memory: 32GB DDR4 2400MHz
Operating System: Windows 10® Pro (64 bit)

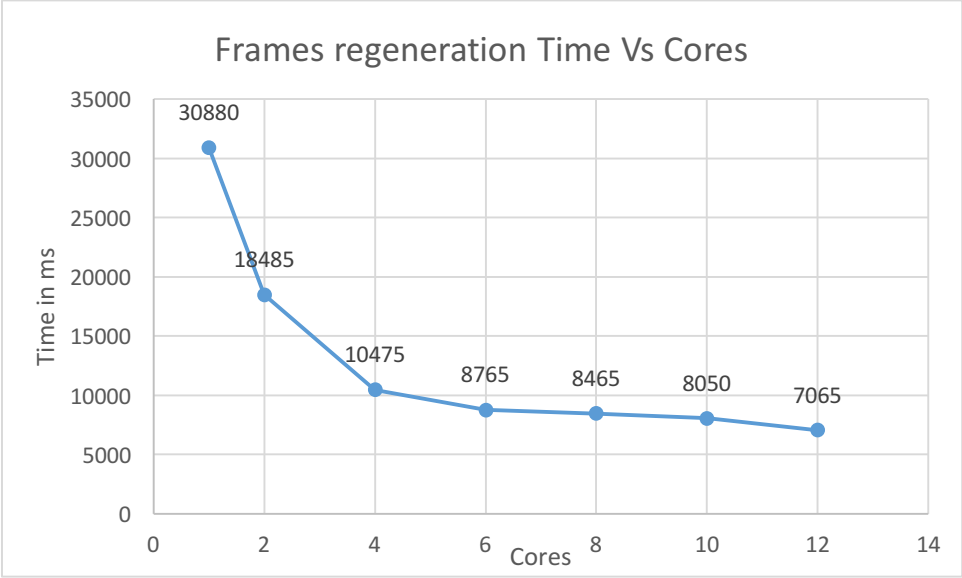
Use Case 1: Animation Rendering

The following data depicts time taken in milliseconds for frame generation (animation rendering pre-step) on different cores. This was achieved by enabling cores on the machine in an incremental fashion. Frames generation duration significantly determines animation rendering time in Krita.

Note: The animation file used for the test is from Wolthera Van Hovell (A top Krita artist), https://yadi.sk/d/n9a6YRrP3Lc3Ts.

Figure 1: Test animation screenshot

Figure 2: Animation results from System 2

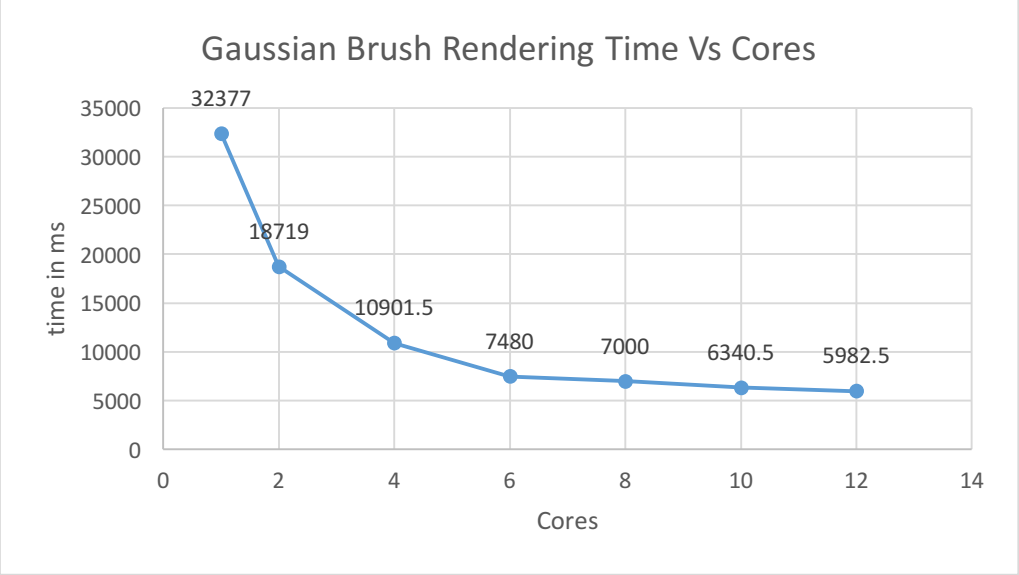


NB: Cores in the graph above refers to physical cores + logical cores (i.e. 12 cores = 6physical cores + 6 logical cores)

Use case 2. Brushes

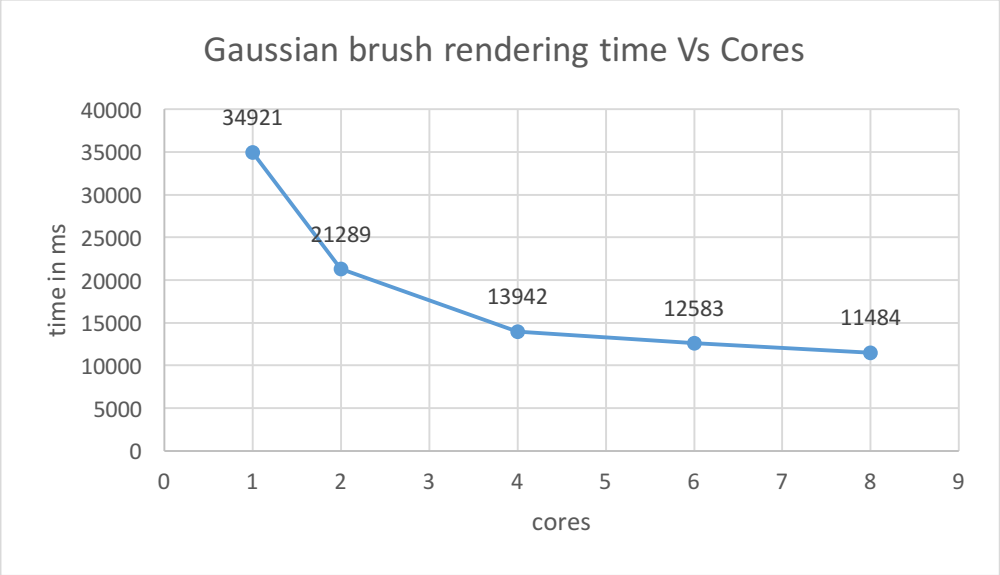
For reliability, an automated script was developed and executed on different core numbers in the two afore-mentioned systems, the script logged the brush speeds. The table below shows time spent to render 16 5000-px long strokes with the Gaussian brush of 1000px diameter. As the core number increases, the time spent on rendering reduces (brush speed increases).

Figure 3: Gaussian Brush speed results from System 2



NB: Cores in the graph above refers to physical cores + logical cores (i.e. 12 cores = 6 physical cores + 6 logical cores)

Figure 4: Gaussian Brush speed results from system 1



NB: Cores in the graph above refers to physical cores + logical cores (i.e. 8 cores = 4 physical cores + 4 logical cores)

Conclusion and Recommendations

From the two use cases above, multiple cores and Intel HT Technology significantly improve the performance of processor-intensive computations. A performance gain of 4.37X is achieved when an animation is rendered using all cores in a 6Core system with Intel HT Technology turned on, compared to rendering the animation on one core. This visible and measurable performance gain translates to reduced waiting times by Krita application users, thus increasing their overall productivity.

Intel provides technologies and hardware platforms that meet the ever-growing computer processing demands by software applications. Independent software developers can take advantage of these capabilities by writing multithreaded software which, in turn, offers an excellent user experience to the application users. Different tools are provided by Intel to software developers that could be used to determine software application performance bottlenecks, hotspots, and concurrency issues. An example of such a tool, and which was used in this study, is the Intel VTune Amplifier.

Notices

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

System Configurations used for testing

System 1:
Processor: Intel® Core™ i7-8550U @ 1.80GHz, 4Cores
Physical Memory: 8GB DDR4 2400MHz
Operating System: Windows 10® Pro (64 bit)

System 2
Processor: Intel® Core™ i7-8550U @ 3.2GHz, 6Cores
Physical Memory: 32GB DDR4 2400MHz
Operating System: Windows 10® Pro (64 bit)

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, Intel Core, Xeon, and VTune are trademarks of Intel Corporation in the U.S. and/or other countries.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.