**FEI Python**

Ultra-efficient memory management

**Introduction**

Python is a scripting language that has become increasingly popular among scientists in recent years. Besides being an intuitive language, it is also surrounded with an entire eco-system of freely available tools. The decision to integrate Python into Amira was motivated by the desire to offer Amira customers a modern scripting language to create their own Amira solutions, and enable integration of the vast landscape of algorithms in the Python eco-system into such solutions.

The goal of the integration was to offer FEI customers the most performant Python integration available in the market today while, at the same time, maintaining compatibility with the existing Python eco-system. This led to the creation of the FEI Python distribution that enables customers to use Python tools with the same performance and similar memory consumption inside of Amira as would be expected from any other modern standalone Python release. With its novel memory sharing technology, Amira and FEI Python are tightly integrated, as the following examples show.

**Methods**

In order to compare the performance of FEI Python integrated into Amira, we executed two different Python scripts on data of various sizes using Enthought’s Canopy 1.7.2 (Enthought, Austin, TX, USA), the Open Source project Icy 1.8.3.2 (Institute Pasteur, Paris, France), and Amira 6.3 (FEI, Hillsboro, OR, USA). Icy was chosen as one of the few Open Source projects that offers a classic Python bridge integration with full access to the Python eco-system through the execnet mechanism.

The first Python script was designed to execute a computationally intensive task. For this, the absolute value of a shifted Fourier transform for a relatively small data set of 350 MB of unsigned bytes was computed. The second Python script was designed to test the efficiency of the shared memory technology developed during this integration. Here, two 1 GB data sets of unsigned bytes were subtracted from each other. All three tools shared the same initial conditions, e.g. the data was already loaded into memory prior to the start of the script. During the execution of the scripts in Canopy, only the computation itself was executed directly on the loaded data. In Amira, the scripts first had to exchange the pointers to the shared memory, where in Icy, data had to be sent to the Python process and back to Icy to be able to directly visualize the images in the application. The individual scripts can be viewed upon request. All tests were executed on an HP Z820 with 64 GB of main memory and 2 Intel Xeon E5-2620 CPUs.
Results
During the execution of the scripts, peak memory usage of the script itself was measured, which did not include the memory used by the application or the preloaded data. After completion of the script, the residual memory usage of the entire application was also measured, including all loaded data sets. In addition, the duration from start to finish of the script was measured.

![Figure 1](image1)
![Figure 2](image2)
![Figure 3](image3)

Figure 1 shows the memory usage of the script computing the Fourier transform. Here, peak memory usage of Amira and Canopy are almost identical at approximately 22 GB, while Icy uses up to 32 GB. The residual memory usage of the three different applications shows Amira occupying less than half the memory of both Canopy and Icy.

Figure 2 shows the memory usage during the subtraction task. Here, Amira uses twice the memory during computation of the script than Canopy, while Icy uses eight times as much. However, the residual memory consumption of Amira and Canopy is almost identical, while Icy occupies four times as much.

Figure 3 illustrates the amount of time the script needs for computing the same results on either platform. For the subtraction and Fourier transform task, Amira and Canopy take about the same amount of time, while Icy is approximately 40% slower in the computationally intensive task and about 100 times slower for the memory-sharing intensive task.

Discussion
The peak memory usage results show that Amira more efficiently uses the system memory, with memory usage that is very much comparable to Canopy. During the memory-sharing intensive task, a doubling in peak memory consumption is observed. After closer investigation, it was observed that this increase in peak memory consumption is for only a brief duration. This is attributed to the fact that Python generates the result and has ownership of the allocated memory. The only option to improve this situation would require for Python to accept pointers to allocated memory for result storage. This would require an extension of the Python interface that, for compatibility reasons, we did not perform in our distribution. The residual memory usage shows that Amira’s Python integration not only uses memory efficiently, but also assists the user in deallocating unused memory. In this category, both Icy and Canopy require additional considerations by the script developer. This is especially apparent in Figure 1. The much slower execution performance of Icy is due to the overhead of the data management. Here, data must be copied and duplicated in both directions, e.g. when sending from Icy to Python and when returning the results from Python to Icy.
Conclusion
This comparison of Python integrations shows that FEI Python integrated into Amira offers a very competitive and memory efficient alternative to a standalone Python distribution such as Canopy. It offers the convenience of immediately visualizing the results computed with Python using the high-quality rendering tools in Amira. In addition, all Amira tools can be used in these Python scripts, providing any Python developer more than what a Python distribution alone can offer. Once a Python script is converted into an Amira Python Script Object, it can easily be accessed through Amira’s proven user interface and treated as a common Amira module. The memory sharing technology developed for Amira’s Python integration allows for a much more fluent programming experience. Handling the data transfer via the execnet bridge, as was done for Icy, is not only time consuming, but also prone to error and difficult to debug.