

Enhancements to the pd developer branch initiated by the vibrez project

Thomas Grill

vibrez.net /

University of Music and Performing
Arts, Vienna

thomas@vibrez.net

Hannes Köcher

vibrez.net

hannes@vibrez.net

Tim Blechmann

tim@klingt.org

ABSTRACT

vibrez is a *pure-data*-based multimedia system which focuses on user-friendliness, high performance and integration of rich media and networking technologies. For its kernel it uses the *devel_0_39* branch of the *pure data* CVS repository as this contains key functionality which has never made it into the *pure data* main CVS branch or the *pd-extended* distribution. The *vibrez* project has initiated a number of enhancements to the CVS branch, including SIMD-accelerated DSP operation, a new low-latency *portaudio*-based audio interface, message-based configuration of the audio and MIDI system, basic multi-threading functionality, loader hooks, and many bug fixes. The successful use of *vibrez* in a number of media projects has proven the underlying kernel to be readily usable and stable but documentation is still sparse on its new features. This paper shall give an overview about those improvements, along with the particular substructure and objectives.

Keywords

Pure Data, API, Maintenance, Optimization, Low Latency.

1. INTRODUCTION

Since the *pure data* [9] system has settled on the *sourceforge* CVS server [15] in July 2002 in its version 0.35, in the meantime it has progressed to version 0.41 and several CVS branches have developed apart from *MAIN*. The most important of these are various branches called *stable_0_xx* and *devel_0_xx*, with *xx* denoting version numbers of the *pd MAIN* (aka *vanilla*) branch they roughly correspond to. In contrast to other CVS-based open-source projects changes in the development branch of *pure data* are not regularly propagated into the stable or *MAIN* branch but have been run in parallel for a long time.

While the *MAIN* branch is maintained solely by *pure data*'s inventor and main architect Miller Puckette, the developer branch has been maintained by several people over time, lately by Tim Blechmann, Mathieu Bouchard and Thomas Grill.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Pd Convention 2007. Montréal, Québec, Canada.
Copyright 2007. Copyright remains with the author(s).

After the *desiredata* project [6] by Mathieu Bouchard and Chun Lee has forked off from *devel_0_39* and Tim Blechmann has started his *nova* project [5], the commercially funded *vibrez* project [10] initiated by Thomas Grill and Hannes Köcher seems to be the only remaining immediate user of the developer branch.

2. VIBREZ-INITIATED FEATURES

vibrez has used the developer branch from its beginning and over time, further improvements have been introduced, supported by business funds [7] generated by the commercial nature of the *vibrez* project. Although the larger part of *vibrez* is not open-source all changes to the underlying kernel have been made public for the *pure data* community and packaged for download as the *vibrez_pure* application [11].

The lack of exchange of improvements between the developer branch and the *MAIN* branch over time has led to a strong divergence and has made ongoing maintenance of the developer branch tiresome and next to useless to the *pure data* community as a whole. We strongly feel on the other hand that the enhancements are beneficial for *pure data* in general and the energy already put into the developer branch should not be wasted. That's why we'd like to give an overview of the most important improvements to make them more accessible for discussion (and/or adoption).

2.1 SIMD-accelerated DSP

SIMD (single instruction, multiple data) operations [18] for 32-bit floating point numbers have been available for desktop computing since the Pentium III (as *SSE*) or PowerPC G4 (as *AltiVec*) processors. These instruction sets allow considerable speedups for many kinds of DSP operations because the CPU is able to operate on typically four samples in parallel. In order to exploit the full potential of SIMD manual coding using assembly or special *compiler intrinsics* for SSE [2] or AltiVec [1] is indispensable. Although *auto-vectorization* [8] has become available with recent compilers (starting with GCC version 4.0), potentially transforming standard C/C++ code into SIMD assembly, this technique currently has several limitations depending on the type of code, like linear, non-linear, accumulative or special functions [Blechmann, personal communication].

For efficient use, SIMD-processable data typically has to come in aligned contiguous chunks, that is four 32-bit floats on 128-bit alignment for processing of DSP code in *pure data*. Therefore the SIMD framework in the developer branch (introduced into *devel_0_37* in December 2003) uses new API functions (*getbytes_aligned* and *freebytes_aligned*) that are

used for memory allocation of array data for signal vectors and sample tables.

Many DSP functions in *pure data* already had two incarnations: One version for sample counts that are multiples of 8, used for manually loop-unrolled codelets, and one version for other sizes (e.g. for very small signal vectors or table processing). The SIMD framework introduces a third version using 16-sample chunks and proper alignment. Most major DSP functions have been implemented in this format using SIMD instructions and respective switching code to choose one of the three incarnations. Although the implementation has only been carried out straightforward with little fine-tuning the gains are considerable, as seen in Table 1.

Table 1. Comparison of devel_0_39 kernel builds with and without SIMD-accelerated DSP

System / CPU type	Load	% CPU mean±standard dev.
PC Desktop WinXP Intel Core2Duo 2.4 GHz	3x	non-SIMD: 59.9±3.2 SIMD: 40.0±2.4
PC Desktop WinXP Intel Pentium4 2.53 GHz	2x	non-SIMD: 84.8±2.9 SIMD: 62.9±4.0
Apple MacBookPro OSX.4.10 Intel Core2Duo 2.16 GHz	3x	non-SIMD: 69.2±0.7 SIMD: 55.7±0.5
Apple MiniMac OSX.4.10 PowerPC G4 1.66 GHz	1x	non-SIMD: 58.5±0.8 SIMD: 38.1±0.5

The comparison of CPU load percentages in Table 1 was measured using a mixture of the *pd* audio examples as usually distributed with the *pd* application: two instances of *I08.pvoc.reverb*, featuring an FFT-based reverberator, one instance of *J08.classicsynth*, featuring subtractive synthesis and ten instances of *B14.sampler.rockafella*, featuring looped sampling. Depending on the available CPU power, this mixture was loaded one, two or three times. All builds were done with the GCC compiler with full speed-favoring optimization (-O3 -funroll-loops). The gains in speed are in the range of 20 to 50%, heavily depending on CPU cache load and memory throughput. As only simple DSP functions have been SIMD-coded so far, further optimizations on sampling, waveform and FFT functions which dominate the above benchmark would accelerate DSP even more.

Although the SIMD framework has already been discussed at the PD conference in 2004 and the desirability of integration into *MAIN* has been expressed, it hasn't happened yet. It seems that the existence of a large amount of inline assembly is scary. Therefore it might be fruitful to transform the code into the respective C compiler intrinsics where possible, probably sacrificing a bit of performance but gaining a certain amount of portability.

2.2 Loader Hooks

In *pure data*, binary plug-ins (*externals*) can be loaded when a previously unseen object is instantiated in a patcher. The *sys_load_lib* API function is then used for translating from the object name to the name of the respective shared library, to find it in the file system using the path list and to dynamically load it into the running system.

It is conceivable that the default strategy *pd* uses to find these externals is not always what is desired. One could want to use a different path resolution or to even load plug-ins that are not

usual shared libraries but moreover other text or binary files, e.g. written in a script language. In order to achieve that, *loader hook* functionality has been implemented into the developer branch in November 2005.

Using the corresponding *sys_register_loader* API function external libraries can implement their own strategy of how to deal with the load request emitted by *pd*. This has originally been used for the CLR external [12] which is able to load mono [14].NET assemblies, but has also been adopted by the *MAIN* branch soon after. Later it has been extended with the idea of the *libdir loader* [17] for the widely used *pd-extended* distribution [16].

2.3 Unified Low-latency Audio Support

As *vibrez* is targeted for Windows and MacOS audio performance on these platforms is decisive. The unsatisfying experience with ASIO under Windows was that even with the fastest audio devices it was hardly possible to achieve latencies below 20 ms, making *pd* unusable for many applications of live electronics. This was due to the ringbuffer-based implementation of the audio IO. Another issue was that WDM kernel streaming, the audio IO standard that all newer consumer soundcards under Windows are targeted for, was not supported at all. Furthermore, the fact that the supported IO standards in *pd* all had individual implementation code made it difficult to maintain. We decided to rather completely rely on the cross-platform *portaudio* [3] library (in version V19), which was previously only used for the MacOS part (in the older V18 version) and to implement callback-based IO for lower latencies.

Depending on the latency setting in *pd*, *portaudio* chooses an appropriate buffer size and calls the callback in regular intervals. DSP is in turn run directly from this callback function. Because accurate timing in *pure data* relies on messages and DSP blocks to be processed alternately, scheduled clock messages also have to be run from this callback. The *portaudio* callback is normally called from a secondary thread, hence the *sys_lock* functionality is used to avoid collisions of the scheduler loop with the work done in the *portaudio* callback.

Additionally, notification functionality for buffer under-runs and timeouts has been integrated, using the message selectors *xrun* and *sys_lock_timeout* which are distributed via the *pd* sender.

Audio loop-back measurements (Table 2, [13]) show that the audio latency has considerably improved for pro-quality audio devices (especially ASIO-driven cards) while it has not substantially changed for consumer-grade audio cards which presumably still require a ringbuffer (which is now relocated into *portaudio*) for MMIO- and DirectSound (DS)-based IO. The measurements were performed with standard *vanilla* kernels (*pd-extended* 0.39-2 or Miller Puckette's 0.40-2, which yield equal test results) and with the *vibrez_pure* package using the *devel_0_39* kernel. The audio system was running at 44.1 kHz sample rate using a DAC block size of 64 samples with stable audio output and no other patch loaded apart from the latency test. The relatively high standard deviation of about half a millisecond results from the jittery message-based timing measurement.

It's worth noting that the *portaudio* WDM implementation and/or the WDM device drivers are error-prone, often resulting in crashes or silence. MMIO and ASIO are generally stable,

probably because these are the IO standards having been available for a longer time.

Table 2. Measurements of audio latencies across an analog loop-back cable

OS / Audio interface	Latency in ms mean±standard dev.
WinXP RME Fireface 400	vanilla ASIO: 19.6±0.4 devel_0_39 ASIO: 6.5±0.4
WinXP RME HDSP Multiface	vanilla ASIO: 18.2±0.4 devel_0_39 ASIO: 5.0±0.4
WinXP M-audio FW 410	vanilla MMIO: 87.4±0.6 vanilla ASIO: 147.3±0.7 devel_0_39 MMIO: 73.3±0.4 devel_0_39 ASIO: 6.5±0.4
OSX.4.10 RME Fireface 400	vanilla CoreAudio: 21.0±0.4 devel_0_39 CoreAudio: 13.8±0.4
OSX.4.10 built-in audio	vanilla CoreAudio: 31.2±0.4 devel_0_39 CoreAudio: 32.6±0.4
WinXP SoundMAX “digital audio”	vanilla MMIO: 79.2±0.5 devel_0_39 MMIO: 112.0±0.6 devel_0_39 DS: 116.0±0.4
WinXP Realtek HD-Audio	vanilla MMIO: 102.4±0.8 devel_0_39 MMIO: 116.7±0.5 devel_0_39 DS: 125.5±0.4 devel_0_39 WDM: 122.6±0.4

2.4 Unified MIDI Support

Similar to the restructuring of the audio interface we chose to unify and largely source out MIDI interfacing by adopting the cross-platform *portmidi* [4] library instead of the previously separate implementations of the various supported platform-specific MIDI APIs.

An important bug fix was provided for MIDI input. Previously it has not been working at all for system exclusive, system common and real-time messages which have varying byte counts. These messages are important for many professional applications like expander module or digital mixer programming respectively state saving, as well as for syncing to MIDI sequencers using MIDI time-code (MTC) or song position pointer (SPP) messages.

2.5 Message-based Configuration of the Audio and MIDI System

The setup of audio and MIDI devices and system parameters in *pure data* can traditionally be conducted either by specifying command-line arguments which are not convenient for inexperienced users or by using TCL/TK-based dialog boxes which are not accessible in `-nogui` mode. *Vibrez* provides its own user interface and is supposed to run without TCL/TK for actual deployment, so another way had to be found to list and configure audio and MIDI devices.

On the one hand, the *pure data* C API doesn't disclose all necessary functionality to write an appropriate configuration external. On the other hand the traditional configuration using the TCL/TK dialog boxes already uses messaging via the *pd* symbol. Hence we chose to extend the kernel and add new messages for configuration and querying of audio and MIDI devices.

Those new messages understood by the *pd* receiver are *getaudioindev* and *getaudiooutdev*. When used with an integer device index deliver the associated device name via a *pd* sender object. When used without an index they deliver a list of devices. *getaudiodevice* returns the currently used input and output devices as a pair of indices. MIDI-related messages are named *getmidiindev* etc., accordingly. Additionally, *getaudioininfo* and *getaudiooutinfo* return audio device specific information, like channel count, sample rate and input and output latencies as acquired by the underlying *portaudio* library. All returned answers from a *pd* sender come with the given message selector preceding the result data.

2.6 Basic Multi-threading Support

In a real-time system like *pd* it is often useful or even necessary to use secondary lower-priority threads for computation-heavy or blocking tasks, like e.g. manipulation of large data structures or networking activity. *Pure data* is not designed to be thread-safe itself, hence some measures have to be taken to ensure undisturbed operation of the kernel and to enable rudimentary communication between the kernel and secondary threads. Although the *sys_lock* functionality is readily existing to constrain execution to one thread at a time, lock-free operation is preferable in order not to endanger low audio latencies by thread-lock timeouts.

Basic often-used functionality which should work without thread-locks involves the generation of symbols (cached string constants) and the passing of messages from secondary threads to the primary kernel thread, e.g. from/to inlets/outlets or senders/receivers.

Consequently, the *gensym* API function was modified so that at least the retrieval of already existing symbols avoids thread locking. The generation of new symbols on the other hand is still blocking, hence a secondary thread should use cached symbol pointers when possible.

The passing of messages from secondary threads to the kernel is supported by the new idle callback functionality. It can be used to install callback functions that are called from the main thread whenever the system has time to service them. Since installation of a callback function using the *sys_callback* API function is thread-safe and can take atom argument lists, idle callbacks can be used to transfer messages from a secondary to the kernel thread. Under the hood this is realized by the use of lock-free FIFO queues which are accessible by a few dedicated API functions (*fifo_init*, *fifo_destroy*, *fifo_put*, *fifo_get*) and can be used for other inter-thread translating jobs as well. The idle callback mechanism is also generally useful for all tasks that can defer their activities to low priority in order to avoid CPU overload.

3. CONCLUSION

This paper gives an overview of the features introduced into the developer branch of the *pd* kernel in the course of *vibrez* development. The improvements presented considerably enhance the performance and use experience of the *pure data* real-time system. Therefore, source code patches against the *MAIN* branch should be provided, despite the amount of work involved, so that the new features easily find their way into popular distributions like *pd-extended*.

Since *vibrez* is currently only developed for Windows and MacOS, additional testing of the presented features might be necessary for Linux, BSD, etc.. Future enhancements to the

kernel should originally be made in the form of patches against *MAIN* as the public acceptance of the developer branch is negligible and it is not actively maintained any more.

The *vibrez* project will certainly be happy to initiate and host further development on the *pure data* kernel, given that tasks are appropriate and funding is available.

4. ACKNOWLEDGMENTS

Thanks go to Miller Puckette for creating and open-sourcing the *pure data* system as a whole, to Hans-Christoph Steiner for maintaining the successful *pd-extended* distribution which also integrates many fresh enhancements to the kernel, as well as especially to Tim Blechmann who implemented many of the features needed by the *vibrez* project and discussed here.

5. REFERENCES

- [1] Apple Inc., *AltiVec Instruction Cross-Reference*. http://developer.apple.com/hardware/drivers/ve/instruction_crossref.html, July 2007
- [2] Apple Inc., *SSE Performance Programming*. <http://developer.apple.com/hardware/drivers/ve/sse.html>, July 2007
- [3] Bencina R., Burk P., *portaudio - portable cross-platform Audio API*. <http://www.portaudio.com>, July 2007
- [4] Bencina R., Burk P., Dannenberg R., *portmidi - Platform Independent Library for MIDI*. <http://www.cs.cmu.edu/~music/portmusic/portmidi>, July 2007
- [5] Blechmann, T., *nova*. <http://tim.klingt.org/nova>, July 2007
- [6] Bouchard M., Lee C., *desiredata*. <http://artengine.ca/desiredata>, July 2007
- [7] City of Vienna, *the departure funding program*. <http://www.departure.at>, July 2007
- [8] Free Software Foundation, *Auto-vectorization in GCC*, <http://gcc.gnu.org/projects/tree-ssa/vectorization.html>, July 2007
- [9] Puckette, M., *Pure Data*. Proceedings of International Computer Music Conference, San Francisco 1996.
- [10] Grill, T., Köcher, H., *vibrez*. <http://vibrez.net>, July 2007
- [11] Grill, T., Köcher, H., *vibrez_pure*. http://vibrez.net/vibrez_pure, July 2007
- [12] Grill T., Morelli D., *CLR external*. <http://pure-data.cvs.sourceforge.net/pure-data/externals/clr>, July 2007
- [13] MacMillan K., Droettboom M., Fujinaga I., Audio Latency Measurements of Desktop Operating Systems. Proceedings of International Computer Music Conference, Havana 2001.
- [14] Novell Inc., *mono*. <http://www.mono-project.com>, July 2007
- [15] Sourceforge Inc., <http://sourceforge.net>, July 2007
- [16] Steiner, H.C., *pd-extended distribution*. <http://at.or.at/hans/pd/installers.html>, July 2007
- [17] Steiner, H.C., *Libdir loader*. <http://puredata.org/docs/developer/Libdir>, July 2007
- [18] Stokes, J., *SIMD architectures*, ars technica. <http://arstechnica.com/articles/paedia/cpu/simd.ars>, July 2007