Zones Convolver Documentation

LEON PATERSON-STEPHENS[1]
leon@leonps.com
MICAH STRANGE[1]
micahstrange16@gmail.com

0 Introduction

Uniform partitioned convolvers (UPC) are able to evenly distribute load as all operations needed for the entire convolution including their required transforms, are completed in a single process call. For larger filter sizes, it can be beneficial to employ non-uniform partitioned convolution (NUPC). In these schemes, larger partitions are required to deliver the results of convolution at an interval of the partition's size, not the block size/latency that the real-time callback is being run at. This can be problematic regarding load-distribution, particularly in the case of real-time audio plugins which are block based and conventionally limited to a single thread.

0.1 Schemes for NUPC

A simple approach to implementing NUPC may be to process entire partitions in a single block at their respective intervals. However this can be problematic when processing at lower latencies. This is because, the entire computation of the forward and inverse transform along with the complex multiplications of the entire partition are completed in a single frame. This can yield dropouts in audio. This is often compounded in schedules with multiple partitions as other partitions deadlines may also land at the same time and now the computation of that entire partition's FDL, forward and inverse transform is also computed in that same frame. This is along with the computation of the uniform convolver running on every block to achieve minimal latency.

Optimal partitioning schemes commonly include upper partition sizes of 65,536 samples for larger filters. These partitions use FFTs of twice the partition's size. The computation of the different partitions in the NUPC is not equal, each have their own FFT and FDL size along with deadlines for when each partition should deliver it's convolved samples.

Selecting schemes with longer clearances gives more time for computation. Whilst this may not be as optimal as schemes closer to Canonical partitioning, it allows distribution across frames. Alternative popular schemes, such as Gardener and Garcia partitioning can account for this.

0.2 Approaches to Distributing Load

One potential solution to realising NUPC in real-time is to ensure clearances of at least the partition size allowing the effective size of each transform in each block to be equal. The operating systems preemptive multi-threading can then be used to distribute the load of transforms and complex multiplication over time while still running the UPC in the real-time thread. Using the operating systems preemptive threading divides arbitrary schemes, without specific implementation effort. In addition, the use of highly optimised FFT libraries such as FFTW is possible. This scheme also has challenges to its implementation. For example, scheduling threads and their respective results safely in real-time and ensuring the results of each partition are available at the right time.

In some contexts (such as web audio/mobile/embedded) the use of multiple threads may be problematic. Furthermore, it is contentious as to whether it is appropriate in audio plugins.

An alternative approach to distribute the work is through time-distributed transforms and splitting the complex multiplications. This is the approach used by this library. The premise of this is that transforms can be decomposed into smaller independent sub-transforms that can then be computed independently across multiple process calls.

Decomposing to achieve a well balanced work load is dependant on a number of factors, which is discussed below.

1 Implementation Detail

This section will present this libraries approach to a time-distributed NUPC. Each time-distributed NUPC is composed of a single UPC processing at the block size and multiple time-distributed UPCs (TDUPCs). Each TDUPC can be thought of as an independent UPC operating at it's own block size, processing a given segment of the filter. The top level of the convolver is responsible for sequencing results of all the TDUPCs.

TDUPCs are given data as it becomes available, at the rate of process calls. However are internally processed in

three stages. Stages and phases are used to help simplify decomposition and transform scheduling. This is a similar approach to that taken by the RTConvolve library, however is able to accommodate varying partition sizes.

1.1 Stages / Phases

Each stage is responsible for different types of work. Stages are promoted when a partition size worth of samples are collected. Each stage operates over a number of phases directly corresponding to process calls. Stages contain $partitionsize/blocksize$ number of phases. For example, a TDUPC operating at 16 times the block size (16B) has 16 phases per stage. During Stage A, samples are collected and decomposed immediately as they become available. Next, in Stage B, the sub-transforms and complex multiplications of the frequency-domain delay line (FDL) are performed. Finally, in Stage C, the inverse decompositions are performed before delivering samples. Intuitively, all phases must be completed before a stage is promoted. However, all three stages are occurring concurrently every phase, acting on different buffers.

The number of phases is fixed at the partition size due to a number of factors. For small clearances it is required to wait until a partition size worth of samples is collected. Though the decompositions can be performed across all phases, stage B still delivers at intervals of the partition size, however can't distribute it's work across all phases as it's result is required in stage C at an earlier point in time and is then left waiting for samples. Whilst large clearances can allow distribution of work across more phases, convolved samples are still delivered at an interval of the partition size therefore within one TDUPC there will occur overlapping work. In theory this could also be balanced however the same result is achieved applying a delay to the output. Considering this, only schemes with clearances greater than or equal to the partition size are used.

1.2 Stage A/C

In stage A, incoming samples are immediately placed into the stage A buffer where forward decompositions can be started immediately, in place. This is only possible using decimation in frequency (DIF) decompositions as the first stage of the butterfly corresponds to the first sample and the $(N/2)^{th}$ sample where the value is known to be zero, due to the zero padding and 2B transform size. Half way through the decompositions, when the first level decomposition is completed, there are sufficient samples to process the next level of the butterfly subdividing the transform further. This decomposition scheme can continue until the transform is completed.

Inverse decomposition's are performed in Stage C, and are a mirrored version of the forward decomposition's. The cost of decomposition's in the first phase is the highest and decreases toward the end, this has a benefit of balancing the uneven load of the forward decomposition's. Similarly to the forward decomposition's, the inverse schedule also delivers samples in a just in time basis.

Increasing the depth of decomposition begins to yield larger spikes of load at the ends of the decomposition schedule. This can be seen in Figure 1.1. Decomposition work is not free as it involves computing stages of the butterfly just as the FFT does. Furthermore, it can't make certain optimisations that fully formed FFTs can and therefore should be limited to certain extents as will be discussed.

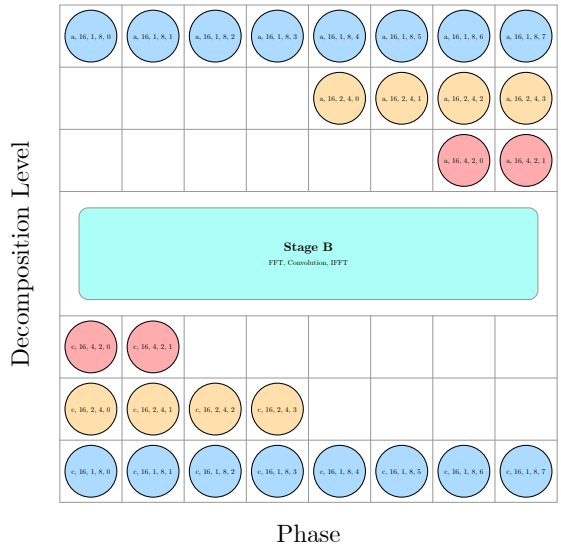

Fig. 1.1. Maximally decomposed 8 Phase Schedule

1.3 Decomposition Depth

Figure 1.1 depicts a maximally decomposed 8 phase decomposition schedule. The size of the transforms are 16B and are decomposed three times across 8 phases. This yields 8 sub transforms of 2B. In this case, decomposing this far is sub-optimal as the number of resulting transforms (forward and inverse) is double the number of phases. It would be more optimal to decompose twice leaving 4 sub transforms of 4B allowing 4 forward transforms in the first 4 phases followed by the remaining 4 inverse transforms in the last 4 phases.

Decomposition is continued until the number of sub-FFTs equals half the number of phases in the schedule. This is to allow for the cost of forward and inverse transforms to be equal across all phases.

1.4 Stage B

Once stage B is reached, the initial large transforms will be composed of several smaller sub-transforms that can now be distributed across the available phases of stage B. The number of transforms is determined by the number of decompositions performed in the previous stage. However the number of complex multiplications is always constant for a given partition size and FDL length $(2 * partition\_size * fdl\_size)$.

The aim of FFT scheduling is to distribute the total work as evenly as possible across the total phases. The number of decompositions is chosen such that ideally there is a sin-

gle transform in each phase. The number of channels effects the depth of decomposition and where transforms are placed in the schedule.

1.4.1 Mono distribution

For a single channel, the sub FFTs and IFFTs are performed on alternate phases, and the complex multiplications are spread evenly across every phase.

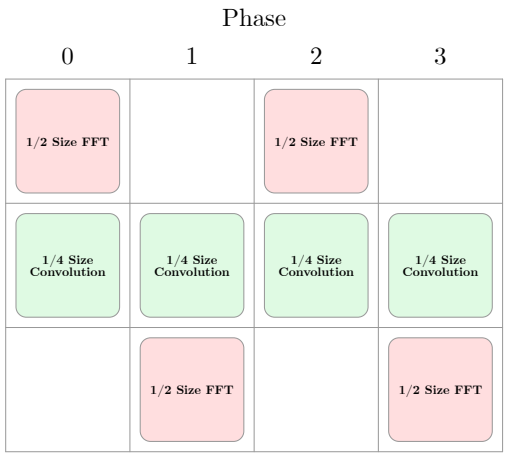

Fig. 1.2. 1 channel distribution for a 4B partition

This arrangement allows a perfect distribution and is simple to realise. For a mono distribution the decomposition depth is as follows $decomposition\_depth = log_2(num\_phases/2)$ This results in a number of sub transforms equal to the number of phases.

1.4.2 Multi-channel distribution

For multiple channels, savings can be made on the amount of work by reducing the number of decompositions, and interleaving work between channels. For a power of 2 channel count each channel equally occupies less phases. This can be expanded for other channel counts by finding the highest power of 2 recursively.

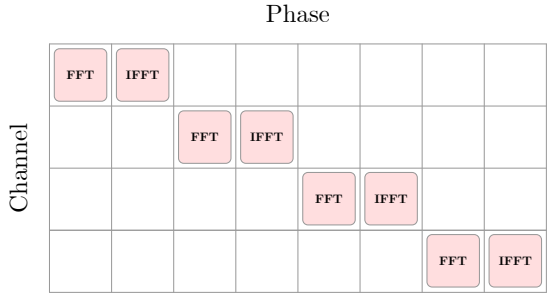

Fig. 1.3. 4 channel interleaving for an 8B partition, where the number of transforms per channel is reduced from 8 to 2

Channels are able to share the frequency domain filter even with different FFT schedules as the same frequency

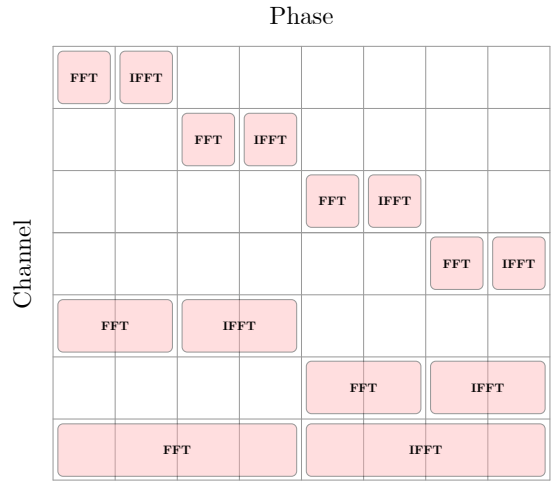

Fig. 1.4. 7 channel interleaving for an 8B partition. The channels are grouped in 4,2,1

domain representation is achieved irrespective of the sub-transform size. An area for further research may be applying a similar scheme to interleave multiple TDUPCs.

1.5 Handling Block Sizes

Each convolver must be initially prepared with a block size. This is used for finding the optimal partition scheme and breaking up the filter into different sized partitions.

The handling of block sizes across DAWs varies greatly. In the model of audio plugins a maximum block size is provided, which is often the block size that is processed. However, smaller block sizes can be presented and this case should be expected. This presents a problem for a convolution engine set up to run at a fixed block size. One approach would be to add latency to allow input samples to be buffered up to the maximum block size. However, this means that DAWs not processing at the maximum block size do not achieve a minimum latency. The approach taken in this library solves this issue. A similar approach is also found in JUCE's convolver. In this implementation, the convolution engine is able to run at a fixed internal block size and can handle both larger and smaller processing block sizes.

1.5.1 Implementation

The incoming block of data is immediately convolved with the first block within the FDL of the UPC regardless of it's processing block size. This is achieved through zero padding the input up to the internal block size of the UPC. The inputs are saved, and the results of convolution are added directly to the output. Subsequent process calls continue to fill the input buffer and outputs are read with an offset in order to align in time. When the internal block size of the UPC is reached a complete transform is added to the FDL. The entire FDL is processed at intervals of the block size and then stored. When block sizes are pre-

sented that are greater than the internal block size, the UPC is effectively processed using multiple subdivisions of the incoming block. A similar approach is used by the NUPC to allow the TDUPCs to function as normal. The TDUPCs are only processed at intervals of the internal block size.

1.5.2 Limitations

This approach can increase workload and decrease distribution of the load of the convolver. For the UPC, a transform of two times the internal block size is computed every process call, regardless of the processing block size. For example, if the incoming data is half the internal block size, the total FFT work doubles compared to preparing the convolution engine to match the incoming block size. For the TDUPCs, the workload may not be distributed evenly every process call. When the incoming data is half the internal block size, a previously well distributed TDUPC will now only do work every other call and has less time to perform it's computation.

1.6 Partition Schemes

A modified Garcia partitioning algorithm is used to find optimum layouts for each block size and filter length. These schemes have been pre-computed and embedded in the library. This is necessary as computing Garcia schemes when presented with specific configurations would be impractical, especially for large filters. This allows for an approximation of the Garcia schemes to be found. This approach provides flexibility to modify schemes in the future with developments in partitioning algorithms.

Embedded schemes are first searched to find the nearest available scheme. Where the filter is longer than the closest pre-computed scheme, additional partitions are added to the largest partition size. If the filter is shorter, partitions are removed, including removing entire TDUPCs if necessary. A future improvement for filter lengths above the largest saved scheme would add additional TDUPCs following Gardener partitioning rules.

The modified Garcia partition scheme has been implemented in python along with a cpp generator used to embed the partitioning results.