New FFTW developments

Matteo Frigo

June 2, 2003

FFTW is a library for computing discrete Fourier transforms. Unlike most other libraries that compute the DFT by means of a fixed algorithm, FFTW changes its operation dynamically in order to maximize performance on a given hardware.

Recently, Steven G. Johnson and I released FFTW3. This new version introduces many improvements over the previous FFTW2 release of four years ago.

Improved semantics FFTW3 computes the complex DFT, the DFT of real data, the DFT of real-symmetric data (also known as the discrete cosine or sine transform), and the discrete Hartley transform. There are no restrictions on the size and dimensionality of the transform. FFTW2 does not compute the symmetric transforms, which are useful for the solution of differential equations with symmetric boundary conditions.

Complex arrays can now be stored as arrays of complex numbers or as separate real and imaginary arrays, the latter format being the one used by `matlab`. In FFTW2, only the former storage scheme was allowed, making the `matlab` interface a little clumsy.

Improved speed FFTW3 is significantly faster than FFTW2 for out-of-cache transforms, for two reasons. First, FFTW3 can either load trigonometric constants ("twiddle factors") from memory, or recompute them on the fly. FFTW3 selects the fastest alternative automatically. Second, FFTW3 optionally rearranges the data in memory so as to exploit spatial locality. This rearrangement is internal to FFTW and it does not change the output order.

Independently of the memory hierarchy, FFTW3 is faster than FFTW2 on processors that support certain special instructions. Specifically, we support the SSE, SSE2, Altivec, and 3DNow! "SIMD" instruction sets. FFTW3 can exploit SIMD instructions for most transform sizes, not just for powers of 2. This property is especially important for 3D problems that arise in physical simulations, where rounding up to the next power of 2 increases the size of the problem by a factor of up

to 8. We also have special code for processors, such as the PowerPC, that feature fused multiply/add instructions.

Transforms of real data whose length is a prime number are now much faster. We use a new algorithm, which is a variant of Rader's algorithm for complex transforms of prime length.

In this talk/poster, I discuss the implementation of FFTW3. Specifically, I discuss:

1. The new structure of the FFTW runtime system. In this new design, one can easily add new FFT algorithms and/or new transforms. I have exploited this flexibility to compute convolutions within the FFTW framework.

2. The automatic "vectorizer" that produces 3DNow! code. This program automatically extracts 2-way SIMD parallelism from a sequential FFT program, and it emits assembly code.

3. The improvements to FFTW's "codelet" generator. Because of a new common-subexpression elimination algorithm, FFTW can now automatically derive straight-line code for cosine and sine transforms, starting from the code for a complex FFT. We show how this property can be used to produce an optimized implementation of the modified discrete cosine transform of length 18 used in the MP3 audio format.