CMSSL conserves storage and enhances load balance by storing each conjugate-symmetric sequence in $N/2$ complex numbers. In the case of multidimensional transforms, the fact that two real sequences are stored in one complex number after the transform along the first axis results in some interesting alternatives for the storage of data for subsequent transforms.

In short, each new dimension transformed generates two new mingled conjugate-symmetric sequences, mingled since they have been transformed with ordinary complex sequences. A method has to be found that, in a manageable way and in-place, unmingles and stores these two new sequences of wave-numbers in the same amount of space as any of the wave-numbers of their companion complex sequences require. The following method has been implemented: for each transform after the first, store elements of the two conjugate-symmetric sequences in a fashion such that the position in the array gives the wave-number for one or the other sequence. [Thi 1993]

9.2 Parallelizing the Fast Wavelet Transform

Mats Holmström
TDB, Uppsala University

The interest for wavelets and wavelet techniques has grown enormously over the last few years, both in theoretical and applied areas. In image compression wavelets are used as an alternative to Fourier techniques. In numerical analysis wavelets are used for solving integral equations and partial differential equations. To understand the basic properties of wavelets it is of value to make a comparison of similarities and differences between wavelets and the more familiar Fourier basis.

If we have a time dependent signal and want to gain information about its frequency content, the standard solution is to use the Fourier transform. One drawback of the Fourier transform is that we do not get any information about *where* in time these frequencies are located. A short pulse cannot be located in time by examining the Fourier spectrum of the signal. In signal analysis one usually solves this problem by using a windowed Fourier transform.

Wavelet analysis provides another approach to this localization problem by using basic building blocks that are smaller for higher

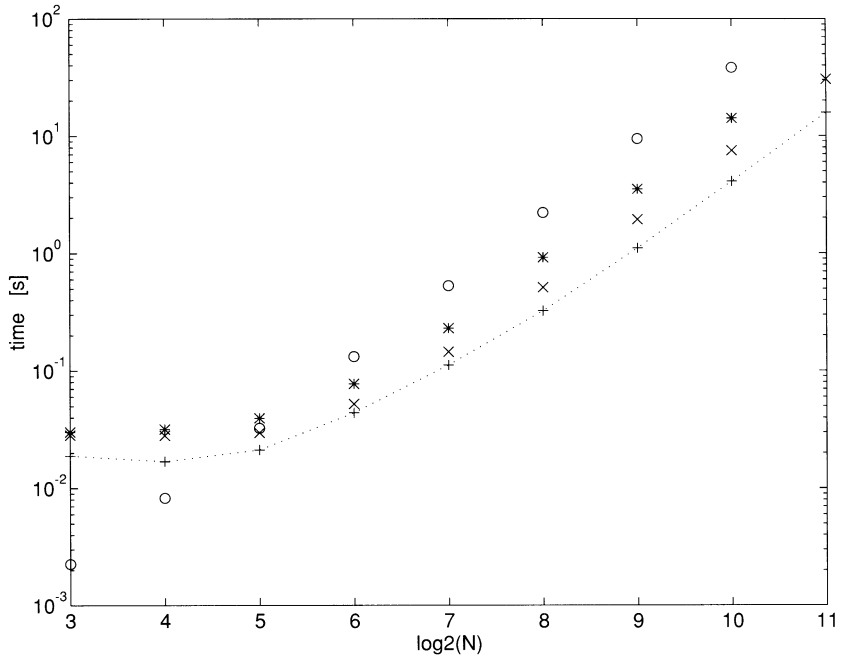

Figure 9.1. Comparison of execution times for some two-dimensional FWT algorithms. The FWT is done in three stages on a square with N^2 points. Legend: '$\circ$' a sequential algorithm on a Sun SPARC-10; '$*$' Algorithm 1; '$\times$' Algorithm 2; '$+$' Algorithm 3. All three algorithms were executed on a CM200, configured with 128 FPUs.

frequencies. These building blocks, or basis functions, obey a relation of the following type

$$\psi_{j,k}(x) = 2^{-j/2}\psi(2^{-j}x - k).$$

All basis functions are scaled and translated versions of a single *mother wavelet*, $\psi(x)$. The scaling corresponds to index j and the translations to index k. We can then represent a function as a linear combination of these basis functions, $f(x) \approx \sum_{j,k} b_{j,k}\psi_{j,k}(x)$.

Since we usually deal with sampled values of functions, i.e. we know a functions values $f(x)$ at certain points x_i, $i = 1, 2, \ldots, N$, we need an efficient way of calculating these wavelet coefficients $b_{j,k}$, given the function values $f(x_i)$. The algorithm for doing this is called the Fast Wavelet Transform (FWT). The time for executing the FWT on N points is proportional to N. This can be compared to the Fast Fourier Transform which has an execution time that is proportional to $N \log(N)$.

When using wavelet methods on large scale problems the time to execute the FWTs can be prohibitively long, although the FWT has a linear time complexity on sequential computers, as noted above. One solution is to use massively parallel computers, but

we are then faced with the problem of constructing an efficient FWT algorithm for such computers.

The reason for the need of different algorithms for parallel computers is that new considerations, such as communication time has to be taken into account when constructing parallel algorithms. In CM Fortran the elements of an array are distributed on the processors according to a virtual grid. It is desirable to reduce the amount of communication between the processors by trying to make the computations local. When we need communication we prefer NEWS-communication (shifting the whole array on the virtual grid) since it is fast on the CM.

Taking the above communication considerations into account, two new algorithms for doing the FWT on a CM were constructed, and compared in execution time with a previously published algorithm. The first new algorithm provided a speedup by a factor of two and is suited for parallel computers in general (actually it is also suited for sequential computation). The second new algorithm uses a feature of the CMs hypercube topology: the ability to quickly shift arrays by a distance that is an even power of two and achieved a speedup by a factor of four compared to the previously published algorithm. In Figure 9.1 the execution times for the three algorithms are presented. Note that the test problem is two-dimensional. This does not present a problem, since when we have an algorithm for the one-dimensional FWT it is easy to extend it to two or more dimensions.

A great help when programming on the CM200 is the debugger Prism. By using Prism one can, in addition to debugging facilities, get timing information on a per-line basis, thus allowing the programmer to evaluate the communication and computation cost of each individual CM Fortran statement as well as statistics for the whole program.

The two new algorithms for the FWT on the CM shows that it is possible to implement an efficient FWT on the CM200. There still remains a lot of work to be done in terms of testing and fine-tuning, and the ultimate goal should be to provide an efficient and robust "black-box" FWT algorithm on the CM200, much like the FFT algorithm that is provided in the CMSSL library.