A UNIFORM AND RECONFIGURABLE FRAMEWORK FOR THE MULTIDIMENSIONAL FOURIER TRANSFORM

Ron Coleman and Michael Post
Department of Electrical Engineering and Computer Science
Polytechnic University
Brooklyn, New York 11201
(718) 260 - 3440

Alan Waksman
Plex Systems Research, Incorporated
New York City, NY 10016
(212) 686 - 9600

ABSTRACT

The computational demands and real-time constraints of the multidimensional Fourier transform (FT) make it an ideal candidate for attack by massive parallelism. The appeal of parallelism has led to the consideration of a variety of architectures including systolic arrays, data flow architectures, arrays of digital signal microprocessors and so-called hybrid architectures. These designs have been advanced for d ≤ 2 dimensions or rely complex and inflexible hardware such as array transposers and peripheral rotation networks. Moreover, they include specific optimizations for the FT or assume the need for course-grain, high-speed computational resources. The Hypercomputer supercomputer is a reconfigurable, massively parallel architecture, a 9,072 processor prototype currently planned. The hypercomputer architecture family is based on arrays of a simple and autonomous unit logic entity, the universal cell. Physically wired in a uniform, eight-degree mesh, the universal cell is a pipelined, 8-bit microarchitecture. The mesh array is supported by a separate 3D, parallel I/O network. The surprising conclusion we show is that simulated multidimensional FT algorithms on the hypercomputer are fast, achieve optimal space and time complexity even though the approach is uniform and does not incorporate any specific architectural optimizations for the FT. Furthermore, each cell in our design is simple and fine grain (8-bits), and is to be implemented with high latency circuits (≥ 200 ns). We exploit these hardware limitations through massive parallelism at the instruction level and the algorithm level. Through the reconfiguration, we address specialized computation and communication, tailoring the degree of both to accommodate each other as well as the FT of different problem sizes and with different space-time constraints.

HYPERCOMPUTERS

The hardware foundation of the hypercomputer family centers on a uniform, eight-degree mesh of universal cells. This mesh array is supported by a three dimensional, global I/O network which provides real-time I/O in the planar dimension and distributed configuration and synchronization along the polar dimension. Physically, each cell is an autonomous, 8-bit microarchitecture with a four stage, pipelined data path and a 128x32-bit control store, general purpose registers, control registers and flags, and an interface to its neighbors through locally shared regions collectively called the synapse. These simple hardware features are the physical basis of low level reconfiguration which captures the functionality of processor elements, switch latches, adjunct memory, etc., as a single conceptual entity. These attributes also serve a much more powerful abstraction concept we call computational holism in which objects or actors (Ref. 10) pool their limited resources functionally and hierarchically, forming cooperative associations of cell ensembles or "chunks" that behave as a single, continuous whole. The software counterpart of this model is embodied in an object-oriented, visually interactive environment, called hyperware which composes arbitrarily abstract and complex actors.

ALGORITHM

The discrete Fourier transform (DFT) for the N-cube signal space of d dimensions is computed by Equation 1 where S(n1,n2,...,nd) is a datum in X k1k2...kd = N1 N2 ... Nd W N1{k1} W N2{k2} ... W Nd{kd} k1 k2 ... kd

The signal space, N = N1 = N2 = ... = Nd, N is a power of 2 and W N = exp(2πi k/N) for m,k = 0,1,...,N-1. Equation 1 can be re-written in a form which is more suitable for direct implementation (Ref. 9) as Equations 2-4 where R is a cyclic permutation function: that is, R(S(a1,a2,...,an)) = S(a2,a3,...,an,a1). Functionally, R is a perfect shuffle routing map. Physically, R is a rotation of the signal cube through one dimen-
cell. However, when local groups of cells chunks pool their limited resources through cooperative computation, complicated tasks like the d-dim simplifies the task enormously. At the highest level, we analyse the d-dim FT communication which maps to two tightly coupled, complex actors: the

\[ X_{x_n} = e^{-j \omega n} = R \left( \sum_{x_n} X_{x_n} \cdot e^{-j \omega n} \right) \tag{2} \]

\[ X_{x_n} = e^{-j \omega n} = R \left( \sum_{x_n} X_{x_n} \cdot e^{-j \omega n} \right) \tag{3} \]

\[ X_{x_n} = e^{-j \omega n} = R \left( \sum_{x_n} X_{x_n} \cdot e^{-j \omega n} \right) \tag{4} \]

Figure 1. Cooperating subproblems for the d-dim FT.

To achieve the theoretical lower bound complexity, \(O(d N \log N)\) (Ref. 12), Equations 2-4 are implemented to take full advantage of the inherent opportunities for parallelism: first, the parallel, fast Fourier transform (pFFT) is used instead of the serial DFT and second, in the (d-1)-dim plane, N^d-1 pFFTs are computed in parallel. However, in practice, this lower bound cannot be achieved without communication overhead costs since the speedup gained through parallelism is obtained at the expense of increased data movement in the pFFT and the cube rotation. The cost of this overhead is architecture dependent. For our implementation, we present overhead results in the "Performance" section.

IMPLEMENTATION

In view of the above algorithm (Equations 2-4), the d-dim FT on the hypercell presents a challenge, enough to easily overwhelm any single cell. However, when local groups of cells chunks pool their limited resources through cooperative computation, complicated tasks like the d-dim FT can be viewed as various levels of abstraction which as a whole, simplifies the task enormously. At the highest level, we analyse the d-dim FT problem as two smaller cooperative subproblems in computation and communication which map to two tightly coupled, complex actors: the FT actor and the shuttle actor. See Figure 1. These complex actors are hierarchically integrated as a composition of progressively simpler, more fundamental actors which "bottom-out" at the level of primitive actors; i.e., actors whose behavior is derived directly from a single universal cell.

FT Actor

The FT actor is composed of \(N^{d-1}\) 1-dim pFFT actors. A multistage version of the pFFT actor for \(N = 8\) is shown in Figure 2. The pFFT version we implement is recirculating; that is, there is one stage which behaves like multiple stages by way of an interconnection network to "circulate" the data flow. In our case, this recirculation is carried out by the shuttle actor. We prefer the recirculating network of actors because of its significant area advantage compared to the multistage version; that is, for the pFFT, \(O(N)\) vs. \(O(N \log N)\). The pFFT actor is composed of \(N/2\) butterfly actors (e.g., the black objects in Figure 2) each of which compute log \(N\) output pairs, \((A + B) \text{ and } (A - B)\) (see Figure 3). A, B are inputs and \(w^k\) is a 'twiddle factor' constant which corresponds to the \(k\)-th root of unity for the complex number \((x + iy)^k\).

A butterfly actor is composed of three primitive actors: \(i, m, \text{ and } t\). See Figure 4. The main task for the \(i\)-actor is to provide a control interface between the rest of the butterfly actor and its counterparts in the shuttle actor. As a secondary task, the \(i\)-actor computes \((A + B)\) of the butterfly output after forwarding \(A, B\) to the m-actor. Since the FT-shuttle network is recirculating, the \(i\)-actor also retains one of the outputs, either \((A + B)\) or \((A - B)w^k\) depending upon its logical address in the butterfly network. The primitive \(i\)-actor provides an active store for the twiddle factors and forwards to the m-actor one constant for each of the \(\log (N/2)\) outputs. The twiddle factors are stored in a register chain table using the general purpose registers and are retrieved using very long shifts. Although very long shifts are slow (2-bits per cycle), the computation of \(w^k\) in the \(i\)-actor and \((A + B)\) in the \(i\)-actor are completely overlapped by the computation of \((A - B)\) in the m-actor. The impact of this bottleneck in the butterfly can be reduced by parallel arithmetic (Ref. 11) is employed to accelerate the multiplication step. In this scheme, the butterfly actor is reconfigured to perform two 16x32-bit or four 8x32-bit partial multiplications in parallel, then integrate the partial results.

Shuttle Actor

The shuttle actor is a general purpose, message-processing assembly of simpler, \(S\) and \(N\) actors which allow parallel data flow from one input to another output. On behalf of the FT actor, the shuttle actor recirculates the data flow for both butterfly and perfect shuffle rotation networks: the actor reconfigures itself as necessary. The \(S\) actor pipelines data generally 'southbound' and the \(N\) actor pipelines data generally 'northbound.' Variations on these theme allow data to be transferred in lock-step fashions that are function routing dependent. For example, Figure 5 shows the step-wise data flow for the \(N = 16\) butterfly network. A similar data flow is used for the perfect shuffle rotation network.

130
PERFORMANCE

In this section, we derive the area and time performance of the multi-dimensional FT on the hypercomputer and compare these data with existing production systems: specifically, commercial DSP microprocessors and the AT&T Signal Processing Ensemble or ASPEN architecture.

Area, Time

The area performance for the N-cube signal space is determined by the butterfly actor type (i.e., the configurable degree of parallelism, \( q \), used by the built-in subactor multiplier), the shuttle actor area and the \( d \) dimensional cube volume. These relationships are expressed in Equation 5. This

\[
A_{\text{area}} = (q+2)N^{d}/2
\]  

(5)

area complexity, \( O(N^d) \) is optimal (Ref. 12). The run-time performance for the N-cube signal space is determined by the pipelined computation time for the 1-dim pFFT (\( T_{\text{pFFT}} \)) along each dimension and the communication delay in the 1-dim pFFTs (\( D_{\text{pFFT}} \)) and the rotation (\( D_{\text{rot}} \)). (All times are reported in system clock cycles except where noted otherwise.) The pipelined computation time is given in Equation 6 (where \( t_{\text{rot}} \), measured empirically, is the butterfly stage computation time for the \( q \) version of the butterfly actor). The values for \( t_{\text{rot}} \) are given in Table 1. The total

\[
T_{\text{p}} = t_{\text{rot}}d N^{d-1} \log(N/2)
\]  

(6)

delay, \( T_{\text{p}} = dD_{\text{pFFT}} + (d-1)D_{\text{rot}} \), is elaborated in Equation 7 (where \( d = 6 \) is the cell-to-cell routing delay).

\[
T_{\text{p}} = t_{\text{rot}}d N^{d-1} \log(N/2) + (d-1)(2Nt_{\text{cD}} + t_{\text{rot}})
\]  

(7)

The above performance model assumes a \( k \) cell hypercomputer such that \( N/2 \leq k \). However, this number of cells may be impractical for the problem size such that only \( N/2 \leq k < N^{d/2} \) cells are available. To illustrate our principle for dealing with this problem we suggest a model for the simpler case of \( k = (q+2)/N/2 \) where \( k \) is now just large enough to compute the 1-dim FT directly. We employ the global IO network which is able to input data through the planar dimension or the polar dimension in parallel. See Ref. 6 for details. We use this network to pump inputs into the mesh array and carry outputs (e.g., temporary results) away from the mesh array. The following example is provided in Ref. 6.

\[
T_{\text{Sim}} = \frac{d \log^2(N/2)}{2} + (d-1)(2Nt_{\text{cD}} + t_{\text{rot}})
\]  

(8)

delay, Equation 9 where \( t_{\text{cD}} \) is the delay steps to input or output a single datum item and \( t_{\text{rot}} \) is the delay steps to rotate the signal cube. The final performance of this system will depend upon the architecture of the global IO network.

Comparisons

We use the above models to compute the performance of our prototype machine with a 200ns clock for the 1-dim 1024 point FT and the 2-dim 32x32 point FT. Note, the 1-dim 1024 point FT is unaffected by \( N/2 \leq k < N/2 \). We compare these results with available data in the literature for commercial 32-bit digital signal processors (Ref. 7) and the prototype AT&T Signal Processing Ensemble or ASPEN computer (Ref. 5) with 25 32-bit AT&T DSP32 microprocessors. As can be seen in Tables 2 and 3, the Hypercomputer would excel in the 1-dim FT compared with uniprocessor signal processors and would deliver nearly equivalent performance for the 2-dim FT (for \( q = 3 \)) compared with the Aspen Processor. This example would easily fit into our machine. Table 3 further shows the effects of serially computing the 1-dim FTs over the two dimensional signal space for "reasonable" values of \( t_{\text{rot}} \); we assume that the IO and rotation are pipelined so that the rotation is overlapped. Thus, \( t_{\text{rot}} = 0 \).

CONCLUSIONS

We have considered a complete analysis of the \( d \)-dim FT on the hypercomputer architecture. Our main goal has been to highlight implementa-
tion and performance and demonstrate that despite the machine's proposed simplicity and generality, it delivers competitive response. These results are surprising and may be even counterintuitive yet it underscores what we believe are the delightful possibilities when reconfiguration and massive parallelism are married.

Table 2. Hypercomputer vs. uniprocessor DSPs.

<table>
<thead>
<tr>
<th>Microprocessor</th>
<th>Clock (ns)</th>
<th>T (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP5600</td>
<td>50</td>
<td>5.0</td>
</tr>
<tr>
<td>TMS320C25</td>
<td>40</td>
<td>7.1</td>
</tr>
<tr>
<td>ADSP2100</td>
<td>31</td>
<td>7.2</td>
</tr>
<tr>
<td>TMS3210</td>
<td>50</td>
<td>30.0</td>
</tr>
<tr>
<td>hypercomputer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q=1</td>
<td>200</td>
<td>0.96</td>
</tr>
<tr>
<td>q=2</td>
<td>200</td>
<td>0.60</td>
</tr>
<tr>
<td>q=4</td>
<td></td>
<td>0.37</td>
</tr>
</tbody>
</table>

Table 3. Hypercomputer vs. Aspen Parallel Processor.

<table>
<thead>
<tr>
<th>Architecture</th>
<th># PEs</th>
<th>T (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASPEN</td>
<td>25</td>
<td>0.80</td>
</tr>
<tr>
<td>hypercomputer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q=1</td>
<td>2,560</td>
<td>1.40</td>
</tr>
<tr>
<td>q=2</td>
<td>3,072</td>
<td>1.09</td>
</tr>
<tr>
<td>q=4</td>
<td>4,096</td>
<td>0.88</td>
</tr>
<tr>
<td>q=1, b=32</td>
<td>2,560</td>
<td>9.37</td>
</tr>
<tr>
<td>q=1, b=10</td>
<td>2,560</td>
<td>7.68</td>
</tr>
<tr>
<td>q=1, b=1</td>
<td>2,560</td>
<td>6.99</td>
</tr>
</tbody>
</table>

ACKNOWLEDGEMENTS

This work has been funded in part by Applied Concepts, Inc.: we appreciate its generosity in supporting our work. We also thank the staff members of Applied Concepts, Inc., Sally Huns and Maria Gonzalez-Coleman for their spirit of encouragement and assistance, and Haldun Hadimoglu and the Polytechnic Parallel Processing Group for providing insightful technical criticisms and invaluable feedback throughout the evolution of many of the ideas embodied in hypercomputers.

REFERENCES