Multi-dimensional Array Operations for Signal Processing Algorithms

Paolo Bientinesi, Nikos P. Pitsianis, and Xiaobai Sun

Duke University, Department of Computer Science,
P.O.Box 90129, Durham, NC 27708-0129, USA
{pauldj,nikos,xiaobai}@cs.duke.edu

Abstract. Game and graphics processors are increasingly utilized for scientific computing applications and for signal processing in particular. This paper addresses the issue of efficiently mapping high-dimensional array operations for signal processing algorithms onto such computing platforms. Algorithms for fast Fourier transforms and convolutions are essential to many signal processing applications. Such algorithms entail fast high-dimensional data array operations. Game and graphics processors typically differ from general-purpose processors in memory hierarchy as well as in memory capacity and access patterns. We characterize the memory structures from the perspective of high-dimensional array operations, identify the mismatch between algorithmic dimension and architectural dimension and then describe the consequent penalty in memory latency. An architecture supports d-dimensional accesses if a d-dimensional data array can be accessed along one dimension as fast as along any other dimension. We introduce an approach to reducing latency and provide experimental results on the STI Cell Broadband Engine as supporting evidence.

Key words: Multi-dimensional Array Operations, Signal Processing, FFT, Convolution, Game and Graphics Processors, Cell Broadband Engine.

1 Introduction

In this paper we present an approach for efficiently mapping certain important signal and image processing algorithms to multi-core processors, including the game and graphics processing processors. We illustrate the approach with the Fast Fourier transform (FFT). The FFT has a ubiquitous presence in signal and image processing applications, it also represents a special class of fast algorithms that factor the operator, which is the Discrete Fourier Transform (DFT) matrix in this case, in order to reduce the amount of arithmetic operations. In computation, the mathematical factorization invokes partition, permutation and reintegration of data arrays and incurs a change in memory reference pattern and latency. Reducing or minimizing the latency in memory references without compromising the arithmetic complexity or accuracy has been a constant effort

in research and development as the computer architectures undergo evolutionary changes.

We address the memory latency problem by characterizing the relationship between algorithms and architectures in terms of a match or mismatch in data array dimensionality. First we review the FFT algorithms in terms of multi-dimensional array operations. We then introduce the concept of architectural dimensions with respect to memory accesses to multi-dimensional arrays. From this perspective, many traditional memory systems are limited to one-dimensional array access. Modern game and graphics processing processors typically support two or higher dimensional array accesses at one memory level. Finally, we identify the dimensional mismatch as a source for high latency in memory accesses and introduce an approach for resolving a mismatch via algorithm transformations. Experimental results on the Sony-Toshiba-IBM Cell processor are provided.

2 Multi-dimensional Array Operations

2.1 The mathematical array dimension in the FFT

We review the FFT algorithms in terms of multi-dimensional array operations. A multi-dimensional FFT involves multi-dimensional data arrays. We start, however, with the multi-dimensional array operations in the one-dimensional (1-D) FFT, $v := F_n\, u$, where F_n is the DFT matrix of order n with elements $F_n(k,k') = e^{-\sqrt{-1}2\pi kk'/n}$, $k, k' = 0, 1, \cdots, n-1$. The input and out data arrays u and v, respectively, may be real or complex numbers, depending on particular applications. In the FFT, the DFT matrix of a composite size, say, $n = pq$, is represented and applied to the input data in the factored form

$$v = F_n u = (F_q \otimes I_p)\; D_{q,p}\, (I_q \otimes F_p)\, P_{n,q}\, u. \tag{1}$$

Here the factors are applied to the input data u from right to left, with $\otimes$ denoting the Kronecker product. First, $P_{n,q}$ permutes u with stride q; next, $I_q \otimes F_p$ specifies the independent application of the DFT of size p to q vector segments, where I_k is the identity matrix of order k. The diagonal matrix $D_{q,p}$ is for the "twiddle" scaling, which we will specify in detail shortly. The last factor in the Kronecker product form specifies the independent application of the DFT of size q to p vector segments, each segment consisting of elements in stride p.

In order to facilitate the mapping from algorithms to architectures, the factorization (1) can be described in terms of two-dimensional data array operations

$$V_{p\times q} := [\, W_{p\times q} \odot (F_p U_{p\times q})\,]\, F_q. \tag{2}$$

The $p\times q$ data array $U_{p\times q}$ is obtained by folding u row-wise, which corresponds to the stride-q permutation. The simultaneous transforms of the columns in $U_{p\times q}$ correspond to the application of the Kronecker product factor $(I_q \otimes F_p)$. The operator $\odot$ denotes element-wise multiplication, the $p \times q$ array W is the leading $p \times q$ subarray of F_n, which unfolds column-wise into the diagonal elements of

$D_{q,p}$. The post-multiplication of F_q to the rows in the scaled 2D array corresponds to that by $F_q \otimes I_p$ in the 1D-array expression of (1). Finally, The array $V_{p\times q}$ unfolds column-wise to render the vector v.

1D and 2D FFTs are related through a simple relationship from a 2D-array operation perspective. Consider the $p \times q$ FFT,

$$V_{p\times q} = F_p \, U_{p\times q} \, F_q. \tag{3}$$

First, the 2D FFT can be seen as consisting of two sweeps of 1D FFTs, one on the columns of size p, the other on the rows of size q. Secondly, when we replace the scaling array W in (2) with all-1s, the 2D-array expression for 1D FFT of size $n = pq$ becomes that for the 2D FFT. In addition to the omission of the element-wise scaling, there is also an omission of the ordering between the transforms from both sides with F_p and F_q, respectively.

In fact, higher dimensional array operations are involved in the 1D FFT, or the 2D FFT, when the basic factorization in (1) is recursively applied to F_p and F_q. Every factored transform involves array operations along a dimension. Except the ordering determined by the twiddle factors among the dimensional transforms, a transform along one dimension operates in the same way as one along any other dimension.

2.2 Dimensional match and mismatch

When memory access latency is uniform across the multi-dimensional array in the FFT, the cost in memory latency is proportional to the cost in arithmetic operations. This is the ideal case when the mathematical and architectural dimensions match. On many modern hierarchical memory structures, however, memory access latency is not uniform. In the extreme case of 1D array memory accesses, one sequentializes a 2D data array, for example, row-wise or column-wise. Assume the data placement is row major. Then the row-wise accesses are much faster than those column-wise and the latency for the transform along the major axis is much lower than that along the minor axis. We refer to such a situation as a dimensional mismatch. A common technique to reduce the overall memory latency cost due to dimensional mismatch is to carry out the row-wise transform first, then transpose the array, followed by another row-wise transform. The transposition exposes additional latency in memory accesses, in addition to that associated directly with arithmetic operations. Recent game and graphics processors support equal memory accesses to two or higher dimensional arrays. We now extend the concept of dimensional match to high-dimension memory architecture. When the mathematical and architectural dimensions mismatch, one may map a high-dimensional array to a lower-dimensional memory by data placement in subarrays. In the previously discussed case of mapping a 2D array onto a 1D memory, the subarrays in data placement are rows. The dimension of the subarrays matches the architectural dimension. The dimensional axes of the subarrays are major ones. The memory accesses across the subarrays, or along a minor axis, may incur substantially higher latency. The corner turn of a 2D

array to dimensional mismatch in general to swap a minor axis to a major one, with respect to the architectural dimension.

It is important to identify the architectural dimension at each memory level in which a multi-dimensional array resides and the relative match or mismatch with respect to the mathematical dimension of the array. In the case of a match, unnecessary dimensional swaps should be eliminated; for a mismatch instead, one shall determine a data placement scheme and a FFT factorization to reduce unnecessary dimensional swaps and subsequent data movements. In a parallel architecture, one has an additional opportunity to reduce the overall latency because the data accesses across the subarrays in the major axes can be potentially carried out in parallel with arithmetic operations. In the next section we illustrate this approach with a distributed FFT on the STI-Cell.

3 Distributed FFT on the Cell

The Sony-Toshiba-IBM Cell (the Cell) supports parallel computation in multiple fashions at different memory levels. Here we focus on the parallel FFT at the level of distributed memory consisting of the local stores at the eight Synergistic Processing Elements (SPEs).

The m-dimensional FFT involves 1D FFTs along each dimensional axis of an m-dimensional data array. For data placement, our model for the architectural dimension at the distributed memory level suggests subarrays that are not necessarily rows or columns. As an example, a three dimensional data array, is subdivided into 8 subarrays, one per SPE. In terms of factorization, we have selected the first factor F_2 in every dimensional axis. By (2), the dimension of the initial data array is increased by 3, the array size along each of the newly introduced axis is 2, and the array size along the initial axis is halved. Memory accesses within the subarray at each local store are much faster than those across the subarrays at different local stores. We omit in this paper the detailed difference in memory accesses on the local store.

The architecural model and the distributed algorithm will be elaborated in the full paper. The following experimental results[1] are obtained using a scheme to overlap communication and arithmetic computations.

[1] The results shown here are preliminary; final results will appear in the full paper.

Array Size	Parallel on 8 SPEs			
	Total GFLOPs	GFLOPs/SPE	cycles$\times 10^3$	ms
128×256	26.2	3.27	299	93.5
256×256	30.3	3.79	552	172.5
$64\times64\times8$	25.4	3.18	299	93.5
$64\times64\times16$	29.5	3.69	554	173.2

Table 1. Performance in GFLOPs and execution time, in 10^3 cycles or milliseconds, for distributed 2D and 3D FFTs.