Load Adaptive Algorithms and Implementations for the 2D Discrete Wavelet Transform on Fine-Grain Multithreaded Architectures

Ashfaq A. Khokhar, Gerd Heber, Parimala Thulasiraman, Guang R. Gao
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE 19716, USA
{ashfaq,heber,thulasir,ggao}@eecis.udel.edu

Abstract

In this paper we present a load adaptive parallel algorithm and implementation to compute 2D Discrete Wavelet Transform (DWT) on multithreading machines. In a 2D DWT computation, the problem sizes reduces at every decomposition level and the lengths of the emerging computation paths also vary. The parallel algorithm proposed in this paper, dynamically scales itself to the varying problem size. Experimental results are reported based on the implementations of the proposed algorithm on a 20 node multithreading emulation platform, EARTH-MANNA. We show that multithreading implementations of the proposed algorithm are at least 2 times faster than the MPI based message passing implementations reported in the literature. We further show that the proposed algorithm and implementations scale linearly with respect to problem and machine sizes.

1. Introduction

Wavelet theory provides a unified framework for a number of techniques developed in multiresolution analysis [4] and subband coding. Wavelet transforms have recently generated a great deal of interest as a new form of multiresolution representation for 1-D signals and 2-D images [4].

The boom in practical applications of wavelet transforms is greatly attributed to the discovery of a fast algorithm for computing a discrete wavelet transform (DWT) [4]. The DWT operates on sampled (or discrete) data in one or more dimensions. Our interest lies in the use of wavelet transforms for analyzing the information content of 2-D images. We therefore focus on the 2-D DWT.

The 2-D DWT is computationally intensive and operates on large data sets. These factors, coupled with the demand for real time operation in many image processing tasks, have necessitated the use of parallel processing to provide high-performance at a reasonable cost. The parallel solutions proposed earlier in the literature for DWT [2, 3, 5] do not adapt to the irregular computation structure of the wavelet transform and are for traditional message passing computing paradigms. In this paper we present a load adaptive algorithm for 2-D DWT on multithreaded architectures.

Recently, multithreaded architectures have been promoted due to their tolerance of inherent parallel execution latencies. In a 2-D DWT, the computation structure consists of different path lengths and the amount of computation within and across the paths varies. This semi-irregular computation structure of the wavelet transform and the embedded fine-grained parallelism is well-suited for multithreading. This paper presents a load adaptive parallel algorithm and multithreading implementation of the 2-D DWT on the EARTH-MANNA multithreaded platform using the Threaded-C language. For this paper we only consider separable wavelet filters. In terms of analytical results we describe a load adaptive algorithm which dynamically scales to the varying problem size. We show that multithreading implementations of the algorithm are at least 2 times faster than the message passing implementations reported in [6] and and that it scales linearly with respect to problem and machine sizes.

Section 2 describes the computational structure and sequential algorithms for the 2-D DWT. Section 3 describes the proposed parallel algorithm and examines the scalability issues. The implementation results are presented in Section 4. Section 5 concludes the paper.

2. Parallel Algorithms for the 2-D DWT

This section describes the wavelet decomposition of a signal using a QMF (Quadrature Mirror Filter) bank. We begin by describing the 1-D subband decomposition scheme and extend these concepts to the case of 2-D wavelet decomposition of image data.

1-D Wavelet Decomposition. Mallat [4] shows that the computation of the wavelet representation can be accomplished with a pyramidal algorithm based on convolutions with quadrature mirror filters. The orthogonal wavelet representation of a discrete signal $x(n)$ can be computed by convolving with the lowpass filter $H(z)$ and highpass filter $G(z)$ and retaining every other sample of the output. The process of decomposing the sequence into two sequences at half resolution can be iterated on either or both sequences. To achieve better resolution at lower frequencies, the scheme is commonly iterated on the lower band The output from the lower band of the k-th stage $y_h^k(n)$ is the input for stage $k+1$ of the wavelet decomposition. In general, K stages of wavelet decomposition result in a $(K+1)$-band wavelet decomposition of the original signal $x(n)$ [6]. The signal can also be reconstructed from a wavelet representation with a similar pyramidal algorithm.

2-D Wavelet Decomposition. In order to apply wavelet decompositions to images, 2-D extensions of wavelets are required. This can be achieved by the use of separable or non-separable wavelets. We consider separable wavelets only in this paper. A separable filter implies that filtering can be performed in one dimension (rows), followed by filtering in another dimension (columns). A 2-D wavelet transform can be computed with a separable extension of the 1-D decomposition algorithm [4]. We first convolve the rows of $x(n,m)$ with a 1-D filter, retain every other column, convolve the columns of the resulting signals with another 1-D filter, and retain every other row. The filters used in this decomposition are the 1-D QMF filters described below. Further stages of the 2-D wavelet decomposition can be computed by recursively applying the procedure to the lowpass filter of the previous stage. In general, K stages of wavelet decomposition result in a $(3K+1)$-band wavelet decomposition of the original image $x(n,m)$.

The computational structure of a three level wavelet decomposition using separable wavelet kernels is shown in Figure 1. The LPF and HPF are lowpass and highpass filters respectively. An interesting computational characteristic of the wavelet structure is that after the application of a row or column filter, the resultant output is down-sampled by a factor of 2 for subsequent processing at the next stage. Thus, at every stage the problem size reduces by a factor of 2 and only output of the low-pass filter is used for the next level decomposition.

3 Dynamic Load Adaptive Parallel Algorithms

There are three characteristics to developing load-adaptive scalable algorithms: 1) the problem size reduces as the computation proceeds, 2) the computation structure is irregular, i.e. not all the computation threads are of equal length, 3) one path (critical path or longest path) in the computation structure spawns other paths. The input image is distributed in a block-row fashion. For an image consisting of n rows and n columns of pixels, each processor initially contains n/p adjacent rows, where p is the number of processors. Further assume that the wavelet kernel is of size L for both row and column kernels. The aim is to compute

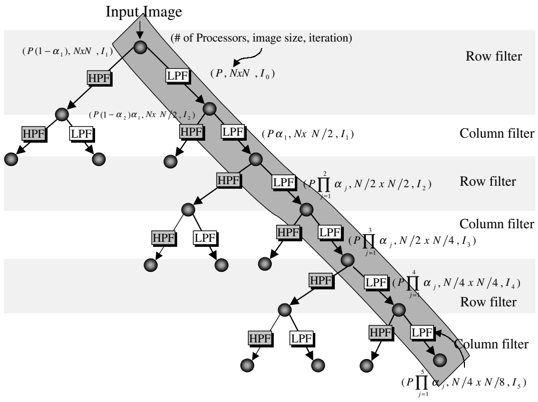

Figure 1. The proposed scheduling of the computation structure on a multithreaded system.

the critical path (highlighted in Figure 1), as fast as possible. During the computation of this path, since the data size is reducing at every node, keeping the same number of processors assigned to this path may create communication overheads for the stages where data size is small. We can shed-off processors along this path and assign them to compute non-critical paths.

Given the input image, all the available p processors compute LPF on the n/p rows and down sample each row by a factor of 2. Since the initial distribution is block-row, this phase does not require any remote data. We can accomplish this task by initiating n/p local threads, one thread for each row. Since the next filter, LPF on the columns, is working on half the data size, we assign only $p\alpha_1$ processors to this task, where $0 < \alpha_1 \leq 1$. The remaining $p(1-\alpha_1)$ processors are assigned to work on the computations which are not on the critical path and are also not dependent on future computations on the critical path. The data currently available at these processors, and that which are needed in future computations on the critical path, are read via remote threads in a totally asynchronous fashion. An efficient schedule is generated to minimize the delay due to remote threads. We interleave these remote threads with the local threads such that by the time local threads (threads that do not request any remote operations) are exhausted, data needed from the remote threads are already available to perform the next step of the computation. The rate at which processors are shed-off (specified by α_i) can be varied from one computation level to the next. Figure 1 shows the status of the algorithms at different stages, i.e. # of processors assigned at each level, the input size, iteration number, etc. If $\alpha_1 = \alpha_2 = \cdots \alpha_{2d} = 1/2$ then the # of processors at iteration I_i is half of the processors assigned at iteration I_{i-1}. Here $1 \leq d \leq \log_4 n$ refers to the number of possible decomposition levels in the wavelet transform and $2d$ is the depth of the computation structure.

Theoretical Analysis. Using the QRQW-PRAM model for analysis, the queue length in terms of number of local and remote threads at any processor along the critical path goes up to at most $O(n)$ threads since there are no synchronization points between the iterations. The, number of remote threads, however, is bounded by $O(n/p)$, the largest subimage assigned to a processor. Since the computation structure is highly asynchronous and the number of processors also vary at different stages of the computation, we will only comment on the total work (in terms of remote and local threads) along the critical path. We assume that the filtering on each row or column of the input images is expressed as one local thread and is independent of the size of the row or column. Similarly remote write or read of a row or column is also considered as one remote thread. The number of local threads per processor during filtering are as follows (**r** - rows, **c** - columns):

$$\begin{aligned} \mathbf{r} &: \frac{n}{p} + \frac{n}{2p\alpha_1\alpha_2} + \cdots + \frac{n}{np\alpha_1\alpha_2 ... \alpha_{2d}} \\ \mathbf{c} &: \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{np\alpha_1\alpha_2 ... \alpha_{2d}} \end{aligned}$$

Similarly, the number of remote threads per processor during this phase:

$$\begin{aligned} \mathbf{r} &: (\frac{n}{p} + L) + (\frac{n}{2p\alpha_1\alpha_2} + L) + \cdots + \frac{n}{np\alpha_1\alpha_2 ... \alpha_{2d}} \\ \mathbf{c} &: (\frac{n}{2p\alpha_1}) + (\frac{n}{4p\alpha_1\alpha_2\alpha_3}) + \cdots + \frac{n}{np\alpha_1\alpha_2 ... \alpha_{2d}} \end{aligned}$$

Case 1: when $0 < \alpha_i \leq 0.5$. This case implies that the rate at which processors are shed off and assigned to non-critical paths is equal to or greater than the rate of reduction in image size during subsequent iterations. Therefore the ratio of the local work to the processors available remains the same. The total number of local threads per processor during row and column filters is $O(n)$. The total number of remote threads per processor during the row filter is $O(\log nL + n)$ and column filter is $O(n)$. The ratio of local threads = $O(1)$ for $L << n$. The total number of local computations per processor (the number of threads $\times$ the size of each thread) during row filter is $O(L\frac{n^2}{p}\log n)$ and column filter is $O(L\frac{n^2}{p})$.

Case 2: when $0.5 < \alpha_i \leq 1$. In this case the number of processors assigned to the critical paths decreases slower than the rate of reduction in the image size. Therefore, the load in the first iteration dominates the overall workload. The total number of local threads per processor during row filter is $O(n/p)$ and column filter is $O(n)$. The total number of remote threads per processor during row filter is $O(\log nL + n/p)$ and column filter is $O(n/p)$. With larger values of α_is, the ratio of local and remote threads is $O(p)$ for $L << n$. The total number of local computations per processor during row filter is $O(L\frac{n^2}{p})$ and column filter is $O(L\frac{n^2}{p})$. The performance of the algorithm lies in the balance between the number of remote and local threads and on their overlapped asynchronous scheduling. If the ratio of the number of local and remote threads is ≥ 1, the interleaving of these threads provides an opportunity to hide remote data fetch latencies. For the first case, when $\alpha_i \leq 0.5$, the rate of reduction in number of processors is equal to or faster than the rate of reduction in the problem size, therefore the number of local threads is $\geq$ the number of remote threads. For the second case, when $\alpha_i > 0.5$, the number of remote threads will be initially lower compared with the number of local threads. However, as the computation progresses this balance is reversed. A careful observation would thus be required to specify different values for the α_is for each level i. These values would depend on architectural and problem parameters. Considering the total number of computations, the algorithm scales linearly with p, i.e. the number of processors. In the following we, report implementation results for the above algorithm on a fine-grained multithreaded platform, the EARTH-MANNA system.

4 A Multithreaded Implementation

The focus of the implementation work is to demonstrate the efficiency of multithreaded architectures in solving image and signal processing problems that employ irregular computation structures.

The EARTH Platform and Language. The EARTH [1] (*Efficient Architecture for Running Threads*) platform supports a multithreaded program execution model in which code is divided into threads that are scheduled atomically using dataflow-like synchronization operations. Once a thread is started, it runs to completion, and instructions within it are executed in sequential order. Therefore, a conventional processor can execute a thread efficiently, even when the thread is purely sequential. For this reason, it is possible to obtain single-node performance close to that of a purely sequential implementation [1]. Conceptually, each EARTH node consists of an *Execution Unit* (EU), which executes the threads and a *Synchronization Unit* (SU), which performs the EARTH operations requested by the threads. The implementations reported in this paper have been developed using Threaded-C and the performance results have been obtained on the EARTH-MANNA (*Massively parallel Architecture for Numerical and Non-numerical Applications*) platform.

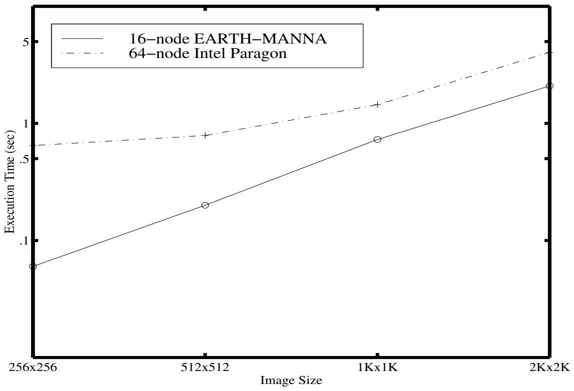

Figure 2. Performance Comparison between Intel-Paragon and EARTH-MANNA for filter/kernel size of 24.

Performance Results. In this section, we present performance results for the load adaptive algorithm proposed in Section 3. The results are reported for only one-level decomposition of the 2D Wavelet transform. Figure 2 compares the performance of the proposed algorithm with the MPI-based message passing implementations reported in [6]. Compared to the 64 node Paragon performance, the execution on a 16 node EARTH-MANNA system is on the average 5 times faster for smaller image sizes and it is about 2 times faster for large image sizes. This large difference could be attributed to two factors: 1) fairly small overhead for initiating remote threads on the EARTH MANNA system (900 ns) compared with approximately 40 microseconds software overhead for sending messages on the Intel Paragon, and 2) the ability of the EARTH system to overlap communication with computation at the fine grain levels. Also note that the available performance results for the Intel Paragon used in the comparison are for a bigger machine size (64 nodes). In such a scenario, for small image sizes, let's say 256 x 256, the value of n/p is 3. Therefore in a block row distribution, each processor will have 3 rows of the image and for column convolution on a filter of size 24, it would require data from 8 consecutive processors. With the decrease in the image size at the next level of decomposition, this situation will be further gross. Therefore for an algorithm, which does not adapt itself to the varying problem size and for a Paragon like architecture which has large message passing overheads, the performance will degrade sharply for small image sizes. Our

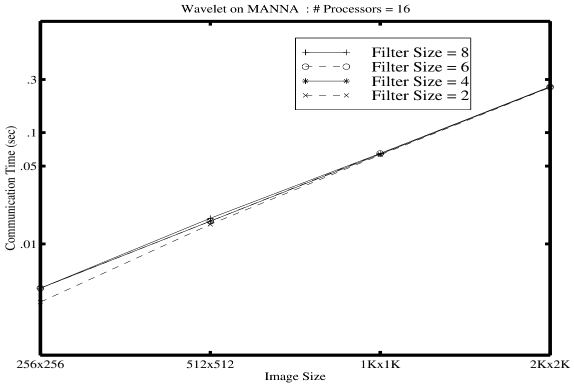

Figure 3. Image size vs. Communication overhead time.

scheduling policy in the proposed algorithm dynamically adapts to the changes in the problem size in subsequent iteration, i.e. if the problem size reduces, the number of processors assigned to the problem are also reduced. In this case, the amount of communication remains uniform until the problem size reduces to fit in a single processor. Figure 3 plots the total communication overhead for different problem sizes. Note that regardless of the change in filter size, the communication time is dominated by the shuffling of the image data. Figure 4 shows the performance results for a fixed problem size while varying the machine size. It shows a linear speedup as we increase the number of processors. Compared with the time on a single EARTH-MANNA node, the speed up is within $0.75 \times p$ on a p processor machine. The performance of the algorithm scales linearly, as shown in Figure 5, for different problem sizes on a fixed machine size. Figure 6 shows the scalability results with respect to filter size. The impact of the filter size on the execution time is sublinear for all image sizes. This is also supported by analytical results because the computation complexity $O(n^2/p)$ is independent of L, the filter size, for small values of L compared to the image size.

5 Conclusions

We have presented a parallel load adaptive algorithm for computing irregular computation structure of the 2D

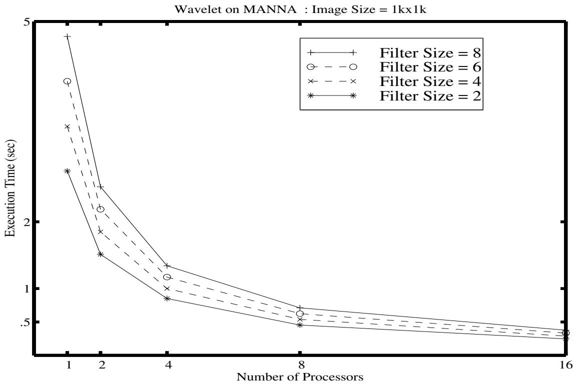

Figure 4. Scalability with respect to machine size; problem size fixed.

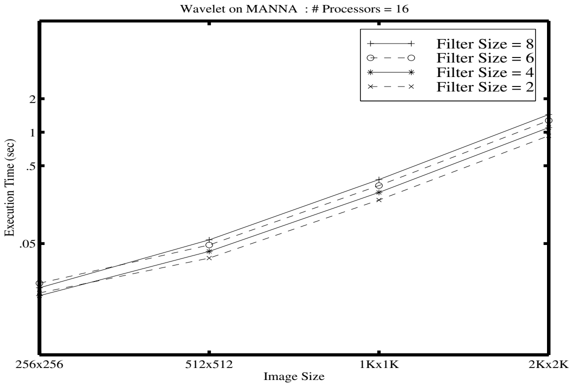

Figure 5. Scalability with respect to problem size; machine size fixed.

wavelet transform for multithreading paradigms. The algorithm is capable of providing efficient schedule for overlapping computation and communication. The proposed algorithm has been implemented on an emulated multithreaded system, EARTH-MANNA, with a run time system, EARTH (Efficient Architecture for Running THreads), specifically designed for fine-grained multithreaded paradigms. Our work is one of the early efforts in designing and developing signal processing applications on multithreaded platforms. The results are encouraging and provide ample evidence to pursue multithreaded architectures as a serious candidate for real-time processing of non-linear image and signal processing applications in multimedia systems.

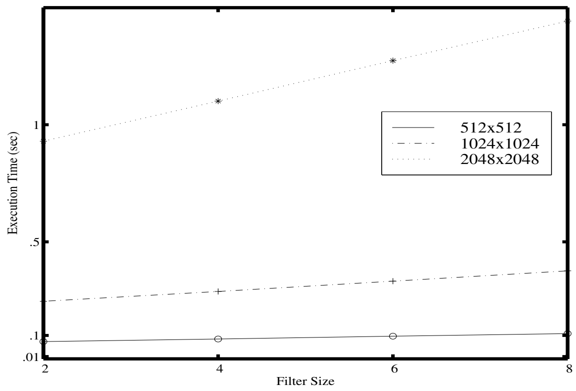

Figure 6. Scalability with respect to filter size; machine size fixed.

References

[1] H.J. Hum et al., "A Study of the EARTH-MANNA Multithreaded System", in *Intl. J. of Parallel Programming*, 24(4):319-347, Aug.1996.

[2] D. Krishnaswamy and M. Orchard. Parallel algorithms for the two-dimensional discrete wavelet transform. In *IPPS94*, pages III–47–54, St. Charles, IL, August 1994.

[3] J. Lu. Parallelizing mallat algorithm for 2-d wavelet transforms. *Information Processing Letters*, 45(5):255–259, April 1993.

[4] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[5] M. Misra and T. Nichols. Computation of the 2-d wavelet transforms on the connection machine-2. In *Proceedings of the IFIP WG10.3 Working Conference on Applications in Parallel and Distributed Computing*, pages 3–12, Caracas, Venez, April 1994.

[6] J. N. Patel, A. A. Khokhar, and L. H. Jamieson. Scalability of 2-d wavelet transform algorithms: Analytical and experimental results on coarse-grained parallel computers. In *Proceedings of the 1996 VLSI Signal Processing Workshop*, volume VLSI Signal Processing IX, pages 376–385, San Francisco, California, October 1996.