Partitioning and Scheduling for Parallel Image Processing Operations

Cheolwhan Lee Tao Yang Yuan-Fang Wang
Department of Computer Science
University of California
Santa Barbara, CA 93106
{ chlee, tyang, yfwang}@cs.ucsb.edu

Abstract

Many computer vision and image processing (CVIP) operations can be represented as a sequence of tasks with nested loops, specified by the visual programming language `Khoros`. *This paper addresses the automatic partitioning and scheduling of such operations on distributed memory multiprocessors. The major difficulties in determining the optimal image data distribution for each task are that the number of processors available and the size of the input image may vary at the run time, and the cost of some image processing operations may be data-dependent. This paper proposes a compile-time processor assignment and data partitioning scheme that optimizes the average run-time performance of task chains with nested loops by considering the data redistribution overheads and possible run-time parameter variations. This paper presents the theoretical analysis and experimental results on a Meiko CS-2 distributed memory machine.*

1 Introduction

Many Computer Vision and Image Processing (CVIP) algorithms processes significant data parallelism [21, 22], and much research in parallel CVIP has focused on parallelizing *individual* algorithms [1, 2, 9, 16]. However, optimizing individual tasks in an operation does not guarantee the optimal global performance. In distributed memory architectures, each task may employ different data partitioning and processor allocation schemes, and data migration overhead between tasks may be significant. A programming tool which determines a schedule for each task to optimize the global performance is therefore highly desirable. However, there are many difficulties in achieving automated optimization. The task and resource parameters (e.g., the input image size and the number of processors available) may not be known at the compile time, the cost of some operations may be nondeterministic depending on the input images, just to name a few.

We have developed a scheduling scheme as a first step to realize a parallel visual programming system based on `Khoros` [12, 14]. Our goal is to facilitate the migration from a single-processor paradigm to a multiprocessor one in a *transparent* manner for parallel CVIP operations. The user employs `Khoros` visual programming language to specify CVIP operations as task graphs. The system will then determine suitable processor assignment and image data partitioning schemes and generates parallel codes by employing existing parallel routines such as ScaLAPACK [4]. In this paper, we assume that a task graph is a chain of data-parallel tasks with nested loops. The goal of optimization is to achieve the minimum overall processing time by assigning multiple processors to each task. We present an algorithm that determines task data distribution schemes to optimize the average global run-time performance.

Several research groups have recently studied the scheduling of CVIP operations. For example, [11] discussed static and dynamic design/scheduling strategies for image processing. DISC [23] uses a dynamic scheduling scheme for data-dependent tasks and conditional branches on a dynamic reconfigurable and repartionable machine (PASM). Library-based parallel systems were discussed in [8, 15]. Our work differs from the above in exploiting both the task and data parallelism on distributed memory architectures.

Scheduling task and data parallelism has recently been studied for program compilation [3, 13, 18, 19, 24]. The optimization function in [18, 19] is for maximizing the throughput with fixed task parameters. The work in [3, 13] deals with DAGs of fixed data distribution and processor parameters, but not with graphs with loops. In [5], techniques for optimizing the data distribution are presented for nested loops, which can be viewed as the optimization for one macro task. In contrast, our current scheduling scheme deals with the data distribution for task graphs with loops and the goal is to minimize the overall parallel time.

2 Problem Definitions

We assume that the target architecture is a distributed memory machine. Each processor has its own local memory, and processors communicate by send-

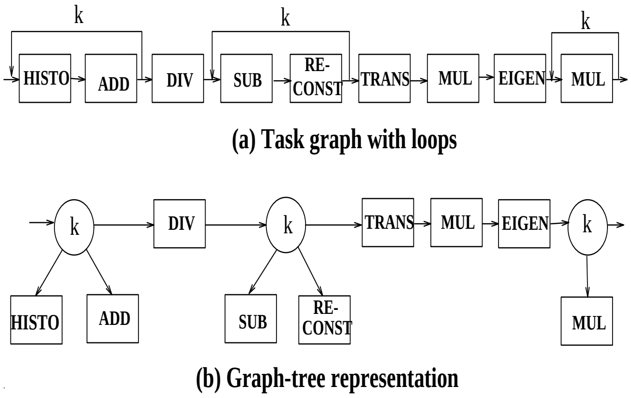

Figure 1: A task graph representing eigen image computation with simple loops.

ing messages among them. The message passing cost between processors can be modeled by the startup latency and transmitting speedup.

The input CVIP task graph is a linear chain of operations with any number of nested loops. A recursive definition of a task graph is: $G = (G_1, G_2, \cdots, G_h)$, which represents a sequence of computation subgraphs $G_1, \cdots, G_h$. Each subgraph G_i could be a single task T_i, or a loop structure $M_i(G)$ where M_i is the iteration number or conditions for executing subgraph G. We use a tree to depict the nested loop control structure in an input task graph. Fig. 1 gives an example of the task graph which represents the eigen image computation [20]. The first loop is for computing an average image of the input image ensemble. The second loop is for rasterizing individual images to form an image ensemble matrix $\mathbf{A}$ of size $n^2 \times k$, where n^2 is the size of an input image and k the number of images in the collection. $\mathbf{A^T A}$ is then computed to give a matrix of size $k \times k$ and the eigen analysis is performed on the resulting matrix. Finally, a third loop is for generating the k eigen images.

Let T be a task in a task graph G and let p be the number of processors available. We assume that task T will use all p processors and T employs one of the following six data distribution schemes based on the High-Performance Fortran standard [10]: row, column, row-cyclic, column-cyclic, block, and block-cyclic partitions. The assignment of computation in each task to the p processors is based on the data distribution pattern assumed for that particular task. Noted that an algorithm may have different performance using different distribution patterns. For example, the 2D "row-column" parallel FFT algorithm using the row or column partition outperforms the 2D decimation FFT algorithm adopting a block partition for small problem-size/machine-size ratios in coarse-grained machines [7]. Since we will employ existing parallel algorithm libraries to execute individual tasks in a graph, the possibility that some tasks may be implemented with a few fixed choices of data distribution must be considered. For example, if 2D FFT is implemented as a two-pass operation based on 1D FFT, which assumes either a row or a column partition, then only two data distribution schemes are possible for the 2D FFT. For the block-cyclic partitioning, we assume that the blocking factor b in the cyclic pattern is fixed such that the computation time is greater than the communication time for an efficient parallel implementation [6].

We model the total parallel time of task T_i as $PT(T_i) = f_i(D_i, p, n)$, where p is the number of processors used, D_i is a feasible data distribution strategy for T_i, and n is the size of the problem (i.e., an input image size of $n \times n$.). Notice the cost also depends on the architecture parameters such as the communication latency and bandwidth, and the CPU speed. Given a graph $G = (G_1, G_2, \cdots, G_h)$, the parallel time is modeled as $PT(G) = PT(G_1) + \sum_{i=2}^{h}(DR(G_{i-1}, G_i) + PT(G_i))$ where $DR(G_{i-1}, G_i)$ is the data redistribution cost between two consecutive stages G_{i-1} and G_i. More specifically, if the data redistribution patterns assumed in two adjacent stages are denoted as D_s to D_t, we can model the redistribution cost as $DR(D_s, D_t, p, n)$. If a subgraph has a loop, the loop iteration number need be factored in the computation of the parallel time. The goal of optimization is therefore to find the data distribution scheme for each task in a task graph such that the overall parallel time is minimized.

Some operations in CVIP applications have a time complexity depending on the value of image data, e.g., in the eigenvalue analysis. Another factor that complicates the optimization process is that the number of processors available and the size of the image may vary for each run. Our goal is to produce a schedule that has a good average performance, allowing the task parameters to vary at the run time. The advantage of finding a good average schedule is that there is no need to produce multiple codes tuned for different parameter settings.

3 Optimization Strategies and the Scheduling Algorithm

In order to develop an accurate cost model for each task, we have developed a taxonomy of CVIP operations [12] to classify data dependent and data independent operations and to tabulate the costs of many pixel and masking operations widely used in CVIP. The data redistribution cost can also be modeled based on all-to-all communication primitives, e.g. [17]. We have also derived data redistribution costs between any two data distribution schemes in the HPF standard [12]. We will first discuss our strategies in determining the data distribution for a task and then give

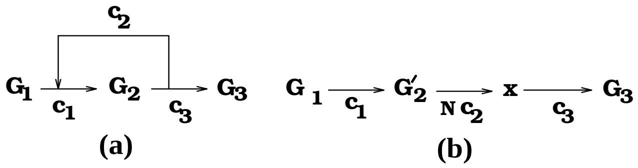

Figure 2: Loop linearization. (a) Original loop. (b) Flattened loop after linearization.

a summary of the scheduling algorithm by integrating those strategies.

3.1 Optimization based on the shortest path searching

We first investigate the optimization of a simple data-independent task chain $G = (T_1, T_2, \cdots, T_t)$ with fixed run-time parameters (p, n). We assume that each task has a deterministic cost function based on a given data distribution. Then $PT(G) = f_1(D_1, p, n) + \sum_{i=2}^{h}(DR(D_{i-1}, D_i, p, n) + f_i(D_i, p, n))$. We need to determine D_i for each task.

We construct a graph $S(G) = (V, E)$, called the *scheduling graph*, as follow. For each possible value of D_i for task T_i, we add a node in V with a weight $f_i(D_i, p, n)$ to represent this particular distribution for T_i. Then we add an edge between a pair of distribution nodes of two consecutive tasks and assign the edge weight as the data redistribution cost between these two data partitions. Then the length of the shortest path in this graph $S(G)$ represents the optimal parallel time. Thus the optimal solution for the data distribution of each task in the task graph can be derived.

3.2 Loop linearization and constrained shortest path searching

The above strategy only works for a task chain without loops. For a task chain with mixed loop control structures, the task graph contains cycles representing loops, and the shortest path algorithm will not go through these cycles. If loop bounds are known, our approach is to linearize the graph so that we can apply a constrained shortest path algorithm for the linearized chain.

We demonstrate our method for a simple case of a single loop. (For multiple nested loops, the same procedure can be applied in a bottom-up manner to linearize loops.) Let the task graph be $G = (G_1, N(G_2), G_3)$ as depicted in Fig. 2 (a) where N is a known loop iteration number. Let c_1, c_2, and c_3 be the data redistribution costs from G_1 to G_2, from the last operation of G_2 to the first operation of G_2, and from G_2 to G_3, respectively. Thus $PT(G) = PT(G_1) + c_1 + (N-1)(PT(G_2) + c_2) + PT(G_2) + c_3 + PT_3 = PT(G_1) + c_1 + N(PT(G_2) + c_2) + c_3 + PT(G_3) - c_2$. When N is large, $PT(G) \approx PT(G_1) + c_1 + N(PT(G_2) + c_2) + c_3 + PT(G_3)$.

Therefore we construct a new graph $G' = (G_1, G'_2, x, G_3)$ by linearizing this loop. G'_2 is identical to G_2 except that all weights in G_2 are enlarged by a factor of N, as depicted in Figure 2 (b). A dummy node x is added after G_2 and the node cost of x is zero. The redistribution cost from G_2 to x is adjusted as Nc_2 and the redistribution cost from x to G_3 is c_3. For this new graph, we construct its corresponding scheduling graph as before and search for the shortest path in this scheduling graph. But we impose an additional constraint in this shortest path such that the data distribution of the first node in G'_2 is the same as that for the dummy node x.

3.3 Graph node reduction

The constrained shortest path algorithm has a complexity proportional to the size of the scheduling graph. To reduce the complexity in finding the optimal schedule, we can eliminate some nodes by coalescing adjacent tasks into a single component task in a task graph. Note that by merging adjacent tasks into one node these operations will now assume the same data distribution scheme, thus merging should not affect the global performance optimality. The method we use is to coalesce a node representing a pixel operation with the operations immediately preceding or following it. Because the execution time of a pixel operation is independent of the data partitioning scheme used, a sequence of pixel operations can be merged into the adjacent masking or global operations without increasing the total parallel time.

Lemma 1 *Given a schedule of a task graph, the schedule can be transformed into a new schedule by merging a node representing a pixel operation in the task graph with its adjacent node without affecting the optimality of the resulting schedule.*

3.4 Scheduling for optimal average performance

So far, the optimal scheduling results are obtained for fixed parameters (p and n). If parameters are allowed to change, the scheduler may find different optimal paths. For example, the constrained shortest path for input image size of 64×64 could be different from that for image size of 1024×1024. Furthermore, the optimality may change if different numbers of processors are used. However, it is not advisable to produce different parallel codes for different parameters as there many be many parameters and many different settings of them. Generating a different parallel code for each possible setting of a parameter may lead to an exponential explosion of the code size. Hence, a more practical approach is to sample the possible parameter values and generate one parallel code which provides moderate performance variation over the sampled range of the parameter values.

Let R be all possible pairs of (p, n). Let $m = |R|$. Let $OPT(G, p, n)$ be the parallel time of the optimal assignment with respect to the parameter values p and n. Let $PT(G,p,n)$ be the parallel time of the schedule derived by our algorithm with respect to the parameter value p and n. Then a minimum variation schedule minimizes the metric mean performance ratios (MPR):

$$MPR = \frac{1}{m} \sum_{(p,n)\in R} \left|\frac{PT(G,p,n)}{OPT(G,p,n)} - 1\right|. \qquad (1)$$

A brute force method to compute such a schedule is to try all possible paths in the m scheduling graph variances. However, the cost can be as high as $O(6^l * m)$, where l is the number of nodes in the scheduling graph. Instead, we construct a new scheduling graph such that the topology of the graph is the same as $S(G)$ but the weight of a node corresponding to distribution D_i of task T_i is $\sum_{(p,n)\in R} \frac{f_i(D_i,p,n)}{OPT(G,p,n)}$. The weight of edge from node D_i to node D_{i+1} is $\sum_{(p,n)\in R} \frac{DR(D_i,D_{i+1},p,n)}{OPT(G,p,n)}$. This graph is called the *augmented scheduling graph*. Then the constrained shortest path searching in this graph can find the optimal schedule that minimizes the MPR. The total complexity is only $O(lm)$.

Lemma 2 *Given a task chain, the constrained shortest path searching in the augmented graph gives a schedule that minimizes the MPR with respect to a range of pre-sampled parameter settings.*

3.5 Isolating data-dependent operations

The above analysis has assumed that the cost of a task is data independent. If the costs are data dependent or the loop bounds are unknown, it is difficult to predict the performance and obtain a good schedule at the compile time. Our strategy is to apply our algorithm to optimize subgraphs in which weights and loop bounds are deterministic. Hence, the run-time data dependent parts are isolated and the maximal data-independent subgraphs are identified to which our optimization algorithm can be applied.

Given a graph $G = (G_1, \cdots, G_h)$, we search the corresponding tree structure in a bottom-up manner to identify data-dependent regions. If G_i contains a data-dependent task or an unknown loop iteration condition, we mark G_i accordingly. We apply the optimization method to each maximal consecutive subgraph sequence $G_i, \ldots, G_j$ such that all these subgraphs are data-independent.

For each subgraph G marked as data dependent, we recursively apply our strategy since the subgraph chains of this graph G may be data-dependent.

3.6 The scheduling algorithm

Now we are ready to put all the above optimization strategies together and present our scheduling algorithm as follows.

- **Step 1:** Traverse the tree control structure to identify a set of maximal subgraph chains which contain only data-independent tasks. For each of those chains, we apply the optimization technique described from Step 2 to Step 5.
- **Step 2:** Coalesce pixel operations to reduce the number of task nodes.
- **Step 3:** Traverse the tree control structure of each chain in a bottom-up manner to linearize loops, which results in a flattened task chain without loops.
- **Step 4:** After loop linearization, construct a scheduling graph for each parameter setting and derive the optimal assignment.
- **Step 5:** Construct an augmented scheduling graph for each chain and derive the optimal assignment with performance adaptive to the run-time variation of parameters.

Theorem 1 *Assume loop iteration numbers are large. For each maximal data-independent subgraph chain with loops, the above algorithm finds an asymptotically optimal schedule that minimizes the MPR with respect to a range of pre-sampled parameter settings.*

4 Experimental Results

We have implemented this partitioning and scheduling scheme and parallel versions of several CVIP algorithms on MEIKO CS-2 which is a distributed memory parallel machine with fat-tree based communication network. Each node uses a SUN SPARC Viking processor. We tested our algorithm on an image processing operation for noise reduction and image enhancement. The image processing operation used histogram equalization to stretch the image intensities back to the original range, followed by median filtering to remove “salt-and-pepper” additive noise. An ln operator was then applied to separate the multiplicative noise from the image itself. The multiplicative noise was removed by a filtering operation in the frequency domain (through Fourier and inverse Fourier transforms). Finally, the image was scaled back using an exponent operator. Figure 3 shows the experiment results on the Meiko CS-2 machine using 4, 16, and 25 processors. The graph on the left shows the parallel performance of the scheduled code in terms of speedup. Note that our scheduler produces only one processor assignment and one data mapping scheme. The run-time performance was evaluated using different task parameters (by varying the image size and the number of processors used). Our schedule leads to a good average performance for different parameter settings with an almost linear speedup. The right figure shows the performance improvement of the scheduled code over an unscheduled code, and the performance gain can be as high as 50%. This experiment shows the effectiveness of scheduling with different parame-

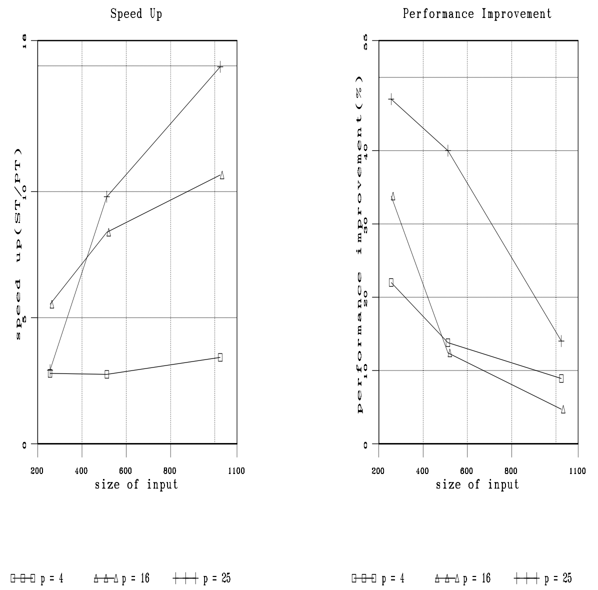

Figure 3: The left figure depicts the speedup of the scheduled parallel program over the sequential program in MEIKO CS-2. The right figure plots the performance improvement ratio over unscheduled code.

ter settings. More details can be found in [12].

5 Concluding Remarks

Our partitioning and scheduling schemes take advantage of domain knowledge and optimizes image data partitioning and processor assignment based on their global performance. We address the issue of runtime parameter variations to make static scheduling results more practical and flexible. Our future work is to extend this framework and incorporate the runtime scheduling approach [23] to perform both static and dynamic optimizations.

Acknowledgments

Cheolwhan Lee was partially supported by a Samsung Humantech fellowship, Y. F. Wang partially by the Alexandria Digital Library Project, and Tao Yang partially by NSF CCR-9409695 and the Alexandria Digital Library Project.

References

[1] M. Annaratone *et al.*, *The Warp Computer: Architecture, Implementation and Performance,* IEEE Trans. Computers, Vol. C-36, No. 12, Dec. 1987, pp. 1523-1538.

[2] P. K. Biswas, J. Mukherjee, and B. N. Chatterji, *Component Labeling in Pyramid Architecture,* Pattern Recognition, Vol. 26, No. 7, 1993, pp. 1099-1115.

[3] S. Chakrabarti, J. Demmel, and K. Yelick, *Modeling the Benefits of Mixed Data and Task Parallelism*, To appear in Proc. of ACM SPAA, Santa Barabra, 1995.

[4] J. Choi, J.J. Dongarra, R. Pozo, and D. W. Walker, *ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers*, Proc. of the 4th Symp. on Massively Parallel Computing, 1992, pp. 120-127.

[5] M. Gupta and P. Banerjee, *Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers,* IEEE Trans. Parallel and Distributed Systems, Vol. 3, No. 2, Mar. 1992, pp. 179-193.

[6] A. Gerasoulis and T. Yang, *On the Granularity and Clustering of Directed Acyclic Task Graphs,* IEEE Trans. Parallel and Distributed Systems, Vol. 4, No. 6, June 1993, pp. 686-701.

[7] L. H. Jamieson, *et al., Parallel Scalable Libraries and Algorithms for Computer Vision,* Proceedings of the 12th IAPR Int. Conf. on Pattern Recognition, Jerusalem, Israel, Oct. 1994, pp. 223-228.

[8] L. H. Jamieson, E. J. Delp, J. N. Patel, C.-C. Wang, and A. A. Khokhar, *A Library-Based Program Development Environment for Parallel Image Processing,* Proceedings of Scalable Parallel Libraries Conference, Mississippi State Univ., Mississippi, Oct. 1993, pp. 187-194.

[9] J. F. Jeng and S. Sahni, *Reconfigurable Mesh Algorithms for the Hough Transform,* Journal of Parallel and Distributed Computing, Vol. 20, 1994, pp. 69-77.

[10] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele Jr., and M. E. Zosel, *The High Performance Fortran Handbook*, MIT Press, Cambridge, Mass., 1994.

[11] S.-Y. Lee and J. K. Aggarwal, *A System Design/Scheduling Strategy for Parallel Image Processing,* IEEE Trans. PAMI, Vol. 12, No. 2, Feb. 1990, pp. 194-204.

[12] C. Lee, Y. F. Wang and T. Yang, *Static Global Scheduling for Optimal Computer Vision and Image Processing Operations on Distributed-Memory Multiprocessors,* Tech. Rep., TRCS94-23, Dept. of Computer Science, UCSB, 1994. A part of this paper will appear in Proc. of Inter. Conf. on Computer Analysis of Images and Patterns. Sep. 6-8, 1995.

[13] S. Ramaswamy, S. Sapatnekar, and P. Banerjee, *A convex programming approach for exploiting data and functional parallelism.* In Proc. of 1994 Inter. Conf. on Parallel Processing, Vol. 2, pp. 116-125.

[14] J. Rasure and S. Kubica, *The Khoros Application Development Environment,* Khoral Research Inc., Albuquerque, New Mexico, 1992.

[15] A. P. Reeves, *Parallel Programming for Computer Vision,* IEEE Software, Vol. 8, No. 6, Nov. 1991, pp. 51-59.

[16] H. J. Siegel, J. B. Armstrong, and D. W. Watson, *Mapping Computer-Vision-Related Tasks onto Reconfigurable Parallel-Processing Systems,* IEEE Computer, Vol. 25, No. 2, Feb. 1992, pp.54-63.

[17] R. Thakur and A. Choudhary, *All-to-all communication on meshes with wormhole routing,* Proc. of 1994 Inter. Parallel Processing Symposium, pp 561-565.

[18] J. Subhlok, D. O'Hallaron, T. Gross, P. Dinda, and J. Webb, *Communication and memory requirements as the basis for mapping task and data parallel programs.* In Proc. of Supercomputing '94, pp. 330-339.

[19] J. Subhlok and G. Vondran, *Optimal mapping of sequences of data parallel tasks*, To appear in Proc. of ACM PPoPP, Santa Barbara, 1995.

[20] M. Turk and A. Pentland, *Eigenfaces for Recognition*, In Journal of Cognitive Neuroscience, Vol. 3, No. 1, Mar. 1991, pp. 71-86.

[21] J. A. Webb, *High Performance Computing in Image Processing and Computer Vision,* Proceedings of 12th IAPR Int. Conf. on Pattern Recognition, Jerusalem, Israel, Oct. 1994, pp. 218-222.

[22] C. C. Weems, *et al., Parallel Processing in the DARPA Strategic Computing Vision Program,* IEEE Expert, Vol. 6, No. 5, Oct. 1991, pp. 23-38.

[23] F. Weil, L. H. Jamieson, and E. J. Delp, *Dynamic Intelligent Scheduling and Control of Reconfigurable Parallel Architectures for Computer Vision/Image Processing,* J. of Parallel and Distributed Computing, Vol. 13, 1991, pp. 273-285.

[24] T. Yang and A. Gerasoulis, *PYRROS: Static Task Scheduling and Code Generation for Message Passing Multiprocessors,* Proc. of 6th ACM Int. Conf. on Supercomputing, Washington D.C., Jul. 1992, pp. 428-437.