A Computational Framework and SIMD Algorithms for Low-Level Support of Intermediate Level Vision Processing

Martin C. Herbordt*, Charles C. Weems*, Michael J. Scudder*

Abstract: Computation on and among data sets mapped to irregular, non-uniform, aggregates of processing elements (PES) is a very important, but largely ignored, problem in parallel vision processing. Associative processing [2] is an effective means of applying parallel processing to these computations [5], but is often restricted to operating on one data set at a time. What we propose is an additional level of parallelism we call multi-associativity as a framework for performing associative computation on these data sets simultaneously. In this extended abstract (see [3]), we introduce algorithms developed for the CAAPP [6] to simulate efficiently within aggregates of PEs simultaneously the associative algorithms typically supported in hardware at the array level. Some of the results are: the efficient application of existing associative algorithms (e.g. [2]) to arbitrary aggregates of PEs in parallel, and the development of new multi-associative algorithms, among them parallel prefix and convex hull. The multi-associative framework also extends the associative paradigm by allowing operation on and among aggregates themselves, operations not defined when the entity in question is always an entire array. Two consequences are: support of divide-and-conquer algorithms within aggregates, and communication among aggregates. Numerous multi-associative algorithms are presented.

It is generally accepted that computer vision requires multiple data representations and algorithms to manipulate those representations. Some important tasks involve relating different representations: one is segmentation of features into regions and obtaining a map-like representation in the form of relational structures among features; another is applying grouping processes to these structures to generate more complex objects for use in model matching [4, 1]. As the process of constructing and manipulating features and regions is necessarily iterative and top-down as well as bottom-up, these tasks require that the pixel plane representations be constantly used to recompute, manipulate, and collect information about sets of data.

It is also accepted that real-time vision requires massive parallelism. In parallel low-level vision, the pixel planes are commonly mapped to SIMD arrays of processing elements (PES). This approach has been successful at preprocessing operations that operate on uniform local regions of data, but in the tasks detailed above, the PEs mapped to data sets (corresponding to features of interest) form irregular, non-uniform aggregates. PEs within aggregates must cooperate with each other; aggregates of PEs must also be able to exchange data and to merge with each other. These aspects of parallel vision computation are often avoided, with most emphasis on window-based operations; some notable exceptions are [5, 7].

Associative processing is an effective way for performing operations on arbitrary aggregates of PEs holding related data. The prototypical associative operation is for the controller to broadcast a query to the array, and to receive a response from the matching elements either in the form of Some/None (global OR) or a Count. But associative processing, as opposed to the familiar associative memory operations, also enables the conditional generation of symbolic tags based on the values of data, and the use of those tags to constrain further processing. Only subsets of the data are involved in any particular operation, but all pixels and features with a given set of properties are processed in parallel. The use of associative processing as a paradigm for parallel vision computation is described in [2, 5]; the fundamental operations are:

1. Global Broadcast—Local Compare—Activity Control
2. Some/None Response
3. Count Responders
4. Select a Single Responder

This method is much more efficient than performing these operations serially, but still yields an obvious bottleneck. Especially with the increased application of parallel processing to higher level vision functions, we want the performance in the data extraction tasks to match that of pre-processing; we want to operate on all PE aggregates simultaneously. This involves an addi-
tional level of parallelism; how is this possible with only one thread of control?

We propose that algorithms can be constructed on the CAAPP [6] for the fundamental associative operations (listed above) to run efficiently on multiple aggregates of PEs simultaneously. In particular, algorithms are necessary for SelectFirst and GetCount that have a small branching factor with relation to the shapes of the aggregates. The other associative primitives are applied to multiple aggregates directly through use of the coterie network. Once obtained, we use these multi-associative primitives as building blocks for more complex algorithms.

Multi-associativity can now be described: PEs (mapped to sets of data) form aggregates, and within those aggregates perform the operations outlined above. The role of the controller as the querier and collector of feedback is taken over either by a “leader” PE, or by a set of PEs within each aggregate. We also define operations on the aggregates themselves: Split/Merge Aggregate, and data transfer between neighboring aggregates. We now have six capabilities defining multi-associative processing:

1. Multicast by a sub-aggregate—Local Compare—Activity Control
2. Some/None Response to a sub-aggregate
3. Count Responders to a sub-aggregate
4. Select a Single Responder
5. Split/Merge aggregates
6. Data Transfer Between Aggregates

The first four operations map all associative algorithms defined over processor arrays to multi-associative PE aggregates in parallel. The split and merge operations give the model new power; there are three basic advantages: (1) enabling the use of divide and conquer, (2) saving different partial results in individual PEs, and (3) taking advantage of ordering implicit in the shape of the aggregate. The sixth capability has obvious utility in region-merging segmentation algorithms.

We emphasize the major restriction of the multi-associative model: there is still only one controller. Therefore, only algorithms with a branching factor << than the number of associative sets will run efficiently. However, due to the apparent large number of useful algorithms with this property, we view this restriction from the positive side: there is greater hardware efficiency because only a single controller is needed. The bulk of [3] involves a description and analysis of the algorithms listed below (and others).

- Create Connected Components
- Separate Borders
- Separate Lines
- Elect Leaders
- Collect (Sparse) Info In Leaders
- Get Sorted Lists
- Collect Ordered Curve Info
- Parallel Prefix On Curves
- Reduction On Curves

These operations are built up into the following vision algorithms:

- Find Smallest Circumscribing Rectangles
- Label Connected Components
- Trace Borders
- Create Border-Corner Lists
- Get Adjacent Region Labels
- Merge Regions
- Count Selected PEs
- Get Medians/Means
- Histograms Within Regions
- Parallel Prefix Within Regions
- Reduction Within Regions
- Convex Hulls

Performance is a difficult issue here: although complexities are often easily derived using big-O notation, they are not always meaningful. In the simplest analysis, we know that multi-associativity is likely to improve performance if the slowdown of the primitive operations, in relation the array equivalents, is a factor smaller than the number of aggregates typically being processed simultaneously. Since the former is on the order of from 1 to 100, and the latter often in the thousands, we believe that good results are likely.

References