**ON THE COMPLEXITY OF
PATTERN RECOGNITION ALGORITHMS
ON A TREE-STRUCTURED PARALLEL COMPUTER**

By

A.L. Gorin

D.B. Roe

and

A.G. Greenberg

IMA Preprint Series # 594

October 1989

This paper will appear in the IMA
Volumes in Mathematics and its
Applications Volume 22 on Signal
Processing: Part 1 – Signal
Processing Theory

**ON THE COMPLEXITY OF
PATTERN RECOGNITION ALGORITHMS
ON A TREE-STRUCTURED PARALLEL COMPUTER***

A.L. GORIN, D.B. ROE and A.G. GREENBERG†

1. Introduction. Pattern Recognition is the problem of automatically classifying an unknown pattern as one of a known set of reference patterns. Applications include speech recognition, speaker recognition, and shape recognition of imaged objects. As the number and variability of the reference patterns increases, so does the computational burden, leading one to consider parallel processing in order to speed up the computation. Several researchers [B 79], [S 84], [S 85], [G 87] have proposed trees of processors for accelerating pattern recognition computations in a variety of applications, such as databases, expert systems, speech and vision. One reason is that the important interprocessor communication operations that arise in pattern recognition can be executed quickly and simply on a tree interconnection. We call these operations *communication kernels*.

This paper describes the theory and practice of efficiently implementing the communication kernels for pattern recognition on a tree machine. We first define algorithms for these kernels, then derive complexity models that express execution time as a function of problem and device parameters. In particular, rather than a traditional asymptotic complexity analysis, we explicitly model and evaluate the interaction between communications and processing parameters in several application scenarios. In the worst cases, one observes logarithmic communications overhead in the parallel algorithms. We demonstrate, however, that in some important situations this overhead can be made essentially zero by pipelining the communications kernels. Furthermore, we show that these tree kernels are minimal in comparison to all other architectures whose interconnection topology is of bounded-degree.

Examples of the communications kernels that we describe arose in application studies of real-time large-vocabulary speech recognition on the AT&T BT-100 parallel processor, which is a tree machine developed by AT&T's ASPEN Project with the Defense Advanced

*This work was partially supported by DARPA Contract No. N00039-86-C-0173.

†AT&T Bell Laboratories, Murray Hill, New Jersey 07974

Research Projects Agency (DARPA). Speech recognition is an important example of pattern recognition, which we will draw upon for motivation and illustration throughout this paper. Our experience has shown that while the details of the pattern recognition computation vary greatly amongst applications, the pattern recognition communication kernels remain remarkably similar. Benchmark analyses of several communications kernels on the BT-100 will be presented to support and quantify the theoretical complexity models.

Complexity models of the communications kernels are of practical use for at least two reasons. The first is to predict improvements in performance as device parameters of an architecture evolve. The second is to provide guidelines within which to optimize the hardware for the kernel operations that occur often in practice. Interestingly, this is an application of the *reduced instruction set* concept to the optimization of interprocessor communications networks.

Independent of the application, trees have two attractive and distinguishing characteristics within the set of parallel computer architectures. First, the sparseness of a tree interconnection allows for compact high-density implementations. For example, as described by Gorin and Shively [G 87], the ASPEN Project's BT-100 has a current computing density of 2 GigaFLOPS per cubic foot, with an eventual goal of 40 GigaFLOPS per cubic foot [S 88]. Second, the tree is universal in the sense that it can be embedded within any lattice of processors while maintaining communications diameter. This is important from the perspective of providing reconfigurability and fault-tolerance in parallel computers, as described by Snyder [S 82] and Pradhan [P 85].

The paper has four sections. Section 2 provides a brief introduction to tree machines, emphasizing those well-known characteristics that make them scalable and well-suited to VLSI and advanced packaging. In particular, the processing element and board-level modules do not change as the machine size is scaled, maintaining a small and constant pinout via Leiserson's expansion scheme [L 81]. Section 3 describes how to partition and map pattern recognition algorithms onto a tree, building upon the pattern matching paradigm of Bentley and Kung [B 79] to encompass structural pattern recognition via dynamic programming search. We describe a host-orchestrated software environment based on Single Program Multiple Data (SPMD) processing which suffices to support these parallel algorithms, and allows us to treat separately their computational and communications complexity. In Section 4, we present algorithms for the communication kernels, derive complexity models, and show minimality. Finally, Section 5 reports on experimental benchmarks on the BT-100 tree machine that quantify and support these models, in particular demonstrating the advantages of pipelined communications.

2. Tree Machines. This section briefly reviews the definition and important properties of a tree machine. The goal is to motivate our focusing on a tree interconnection by appeal to its scalability and suitability for VLSI. These properties lead to compact, high-density implementations.

(cf. Figure 1). This 4-port property extends recursively to larger submodules such as cabinets. The small and constant pinouts of the PEs and submodules make the tree well-suited for miniaturization via VLSI and advanced packaging. The BT-100 illustrates the high-density computing that can be achieved via this approach. Each PE comprises an 8 MFLOP programmable 32-bit processor and 64 kbytes of memory. The board-level module comprises 8 PEs on an $8'' \times 13''$ board, with a 127 PE GigaFLOP processor housed in 0.5 cubic feet. This computing density, 2 GigaFLOPs per cubic foot, is the highest known to the authors for a programmable parallel computer.

3. Pattern Recognition on a Tree Machine. *Pattern Recognition* is the problem of automatically classifying an unknown input pattern as one of a known set of reference patterns. In some cases, the references can be effectively enumerated and stored in a library. Then, one can proceed, as for example described by Duda and Hart [D 73], by comparing the unknown pattern to each reference and then classifying it as that reference to which it best matches. The comparison can be via a distance function, wherein the best match is a minimum distance, or via a probability scoring, wherein the best match is a maximum likelihood. This is the basic pattern matching problem, with extensions to be described below. As the number of patterns in the reference library increases, so does the computational burden of the system. The goal is therefore to exploit parallelism to maintain execution speed as the problem size increases.

In this paper, we restrict ourselves to viewing an individual pattern match function as an atomic computation, and focus on the nature and complexity of the operations necessary to distribute the matches over many processors. Thus, our results are independent of the details of the match function, depending only on surface characteristics such as the memory required to describe the patterns and the execution time of the match function.

The computational core of a pattern recognizer is the matching of the unknown pattern against the library of reference pattern. Expanding on the paradigm introduced by Bentley and Kung [B 79], one partitions and maps the computation onto a tree as follows. The machine is initialized by first distributing the library of reference patterns amongst the PEs, storing a subset in each PE's memory. The unknown pattern is then broadcast from the tree's root to all PEs. Each PE then concurrently matches the input patterns with its subset of the reference library. The best score is then resolved by bubbling it up the tree. Associated information on the model with the best score is then reported up the tree. In this scheme, the matching is performed concurrently, at a communications cost proportional to the tree depth. This cost is small, since the tree depth grows only as the logarithm of the tree size. I.e., a complete binary tree with 1023 PEs has a depth of merely 10.

We now describe a software architecture that both supports this pattern-matching control strategy and also provides program scalability. Both processing and global communications are orchestrated via function calls from a serial host program, so that data

flow within the tree is controlled by the host, independently of the tree size. Therefore, the application source code is unchanged with the addition of more reference patterns and PEs. Furthermore, as long as the number of PEs is scaled proportional to the problem size, the computation time remains the same, and one can separately analyze the logarithmic growth of the communications functions, as we do in section 4.

The major parallel construct is a *sliced procedure*, in which the identical program is executed simultaneously in each PE, but on different data sets. The multiple executions of this single procedure can follow different instruction streams (still within the procedure, of course) depending upon the data. These potentially different instruction streams are forced to converge and synchronize at the completion of the sliced procedure. This synchronization scheme is known as barrier synchronization, and the control strategy known as Single Program Multiple DATA (SPMD) processing. For adherents of Flynn's [F 66] SIMD/MIMD categories, SPMD may be alternatively viewed as coarse grain SIMD or as data-driven MIMD.

In support of the SPMD processing mode, we use several global burst communications functions. These communications are invoked by function calls from the host, and are examples of traffic-specific interprocessor communications. *Broadcast* transmits data from the host to the PEs. *Resolve* identifies the PE in the tree with the maximum value of some variable, such as a probability score. The *Report* command, typically following a *Resolve*, transmits data from the identified PE to the host, such as the name of the reference pattern that has the maximum probability. Such a software architecture has been implemented on the BT-100, where the host program and sliced procedures are programmed in standard C, and the invocation of sliced procedures and communication functions are via C function calls from the host program.

Structural Pattern Recognition. When a reference set becomes too large to effectively list, one can instead describe it as a set of structural patterns formed by gluing together simpler patterns according to some set of rules. For example, in speech recognition [L 85], one usefully views a sentence as a structural pattern comprising concatenated words, rather than as a gestalt. Similarly, in large-vocabulary recognition, one views words as concatenations of subword units such as phonemes. One does not try to match the unknown pattern with all such structural patterns, since that is a combinatorially explosive search problem. The basic approach used in speech recognition to prune the search while still guaranteeing an optimal match is dynamic programming, as described by Rabiner and Levinson [R 81].

Bentley and Kung's pattern matching paradigm has been extended to accelerating dynamic programming search for structural pattern recognition, and then applied to real-time speech recognition by Gorin, Shively and Roe [G 87], [G 88]. The concept underlying this extension is as follows. Dynamic programming is a graph search, whose outer loop in this context comprises an iteration of the following.

1. starting at some state(s) in the search space, evaluate subsequent states via matching the next input epoch against the library of reference pattern primitives;
2. select the best or several best matching references;
3. make the transition(s) to subsequent states in the search space.

As shown in Figure 2, step 1 is performed in parallel via a sliced procedure, step 2 via a resolve/report, and step 3 via a broadcast. In many searches, one wishes to maintain multiple hypotheses. This motivates a K-best resolve/report function, which will be described in Section 4.

Set-Up: Load Templates

Broadcast Data Epoch

Match in Parallel

Resolve Best Match(es)

Broadcast New State(s)

Broadcast Next Epoch
and Iterate

Figure 2

Parallel Dynamic Programming
on a Tree Machine

For illustration, we briefly review the connected-word recognition benchmark described by Gorin and Roe [G 88]. That algorithm is a dynamic programming search that computes a Viterbi scoring of the probability that an unknown spoken utterance was generated by a reference Hidden Markov Model [L 85]. For real-time execution, the search must complete the above three-step cycle for each 15 milisecond input frame of speech. I.e., all processing and communications for an input frame needs be completed before the next frame arrives. A set of 25 short-term acoustic spectral features are measured from the frame, then broadcast to all PEs. The local pattern recognition computation involves evaluation of a continuous probability density function defined via a 9-component Gaussian mixture in the 25-dimensional feature space. For each of several syntactic hypotheses, the K -best scores are resolved and reported, then broadcast to all PE's along with the next frame of spectral features. Both the number of hypotheses and K depend greatly upon the vocabulary and grammar. For a digit-recognition example, there were 7 hypotheses maintained with the 3 best scores each. These parameters can increase a hundred-fold as vocabulary and language complexity increase.

4. Complexity Models. In this section, we derive formulae for execution time of the pattern recognition communication and processing kernels on a tree, expressed in terms of problem and device parameters. We also discuss application scenarios and show minimality. In particular, we will show that in several important application scenarios, the parallel pattern recognition algorithm achieves perfect linear acceleration of the match computation, while incurring a communications cost proportional to the number, K , of scores resolved at each cycle of the search plus a logarithmic pipeline fillup term.

We will show that this is the best that can be achieved on any architecture whose interconnection topology is of bounded degree. We say that an interconnection is of bounded degree b if each PE is connected to at most b other PEs. This will be true, for some value of b , for all physically realizable machines. For example, a binary tree of any size has degree 3, a hypercube with 1024 PEs has degree 10, and a rectangular array has degree 4.

4.1 Broadcast. The first stage in the basic parallel pattern recognition algorithm is to *broadcast* the set of features characterizing the unknown pattern. This is achieved by each PE receiving a value from its parent, storing it locally, and then retransmitting it to its children. For a single data element, the delay between entering the root and arriving at the leaves is proportional to the logarithm of the number of PEs, i.e the tree depth. We will show, however, that for many situations of interest, pipelining can effectively eliminate the apparent logarithmic communications cost.

Let B denote the number of bits in the feature vector. (We choose the units of bits for exposition's sake. It is straightforward to modify these arguments to, for example, 16-bit wide pathways.) Let c denote the communications time to propagate a bit one level down the tree. Let N denote the number of PEs, and $\log_2 N$ the depth of the tree. Finally, let S denote the host startup/shutdown costs.

One method to measure elapsed time for a broadcast is from initiation until the last bit arrives in the leaves, denoted by T_{B_L} and given by

$$(1) \quad T_{B_L} = S + c(B + \log_2 N),$$

costing cB cycles for the transfer of the data to the root, and then $c \log_2 N$ cycles for a pipeline shutdown time as the data propagates down the tree.

However, in practice, a broadcast function call is embedded within a host C program. The actual time of interest is *not* T_{B_L} , but *rather* when the host program proceeds on to its next instruction. Since the broadcast function returns no value to the host, the host instruction counter is incremented as soon as the last bit enters the root of the tree. Thus, in *all* situations of interest, the pipeline shutdown occurs concurrently with the next program step, and can be ignored. Denote this effective execution time for a broadcast by T_B , given by

$$(2) \quad T_B = S + cB.$$

Minimality. Consider the problem of broadcasting B bits of data to all PEs in some parallel architecture. I.e., at completion, the data should be resident in local memory at each PE. Assuming that writing to a PE's local memory is a serial process, then it requires time at least cB to accomplish this write to any individual PE. Thus, a lower bound for completing the global broadcast is cB , which is achieved modulo a constant factor by the tree broadcast kernel.

Finally, observe that when the block size is large, the host startup cost $S \ll cB$, and T_B is approximated by

$$(3) \quad T_B \approx cB.$$

In the AT&T BT-100, experimental data bear out formulae (2) and (3), as will be seen in Section 5.

Example. In a speech recognition application, one broadcasts measures of the short-term acoustic spectra, which in the example of [G 88] comprise a batch of 25 32-bit numbers at 15 millisecond intervals. On the BT-100, communications is achieved via software protocols between PEs, and as will be shown in Section 5, the communications parameter c is 1.4 microseconds/bit/level, and the startup cost, S , on the host PC is 120 microseconds. Thus, summing, the broadcast time T_B is 920 microseconds, well within the 15 millisecond real-time budget. A significantly faster value for c is projected for the machine described by Shively and Gorin in [S 88], based on 16-bit wide data path clocked at 20 Mhz, yielding 320 Mbits per level per second. In that case, the broadcast time for 25 32-bit numbers is 2.5 microseconds. While this is overkill for a single real-time speech channel, it extends the applicability to pattern recognition of higher-bandwidth signals.

4.2 Match. The second phase in parallel pattern recognition is for each PE to concurrently *match* the unknown against the subset of reference patterns in that PE. The communication kernel associated with invocation of a sliced procedure is twofold: first, initiation of the process, then global synchronization at completion. We will show that for many situations of interest, the logarithmic initiation/synchronization overhead is negligible, and that one can achieve essentially perfect linear acceleration of the match computation.

Let L denote the number of reference patterns and N the number of PEs. Since in this paper we treat a match function as an atomic computation, we assume that there is at least one reference per PE. I.e. the number of PEs is assumed less than or equal to the number of references. For simplicity of exposition, we assume that L is an integral multiple of N , so that each PE has the same number of reference patterns. Thus, there are L/N reference patterns per PE.

Let m denote the number of machine cycles necessary to perform a single match, and let p denote the time necessary for a PE to execute one of those cycles. For simplicity of exposition we assume that m is the same for all reference patterns. Thus, each PE will spend time pmL/N computing its matches. For initiation and resynchronization, the communications overhead is $2c \log_2 N$, where c is the communications time per level as before. Denote by $T_M(L, N)$ the total execution time for computing L matches distributed over N PEs, which is given by

$$(4) \quad T_M(L, N) = pmL/N + 2c \log_2 N.$$

Observe that the time to execute these matches on a single PE is given by

$$(5) \quad T_M(L, 1) = pmL.$$

The best that one can expect in accelerating the match computation is to achieve *perfect linear acceleration*, i.e., $T_M(L, N) = T_M(L, 1)/N$. Let us examine the deviation from perfect linear acceleration in equation (4), rewriting it as

$$(6) \quad T_M(L, N) = [T_M(L, 1)/N][1 + (2cN \log_2 N)/(pLm)].$$

Thus, in order to approach perfect linear acceleration in the match phase, the deviation term $(cN \log_2 N)/(pLm)$ must be much smaller than unity. We consider three different sufficient cases, by no means exhaustive, for which this is true. Each is based on a situation in which the amount of computation that the PE performs is much greater than the communications costs. To clarify the exposition, we first rewrite equation 6 as

$$(7) \quad T_M(L, N) = [T_M(L, 1)/N][1 + (\log_2 N/m)(2c/p)(N/L)].$$

The first case is if the match function is complex, i.e. the number of machine cycles per march, m , is large. For sake of example, assume that communications and processing are

equal, i.e. $c = p$. Upon recalling that $(N/L) \leq 1$, then if the ratio $\log_2 N/m$ is very small then so is the deviation. I.e., the deviation from perfect linear acceleration is small when the number of machine cycles per match is much greater than the tree depth. This occurs in many situations of interest. For example, the computation of a Gaussian likelihood is quadratic in the dimension of the feature vector. In speech recognition, a typical acoustic feature vector comprises 25 dimensions, implying on the order of 625 floating point computations to perform the match. This is much larger than any conceivable value of $\log_2 N$, and thus leads to essentially perfect linear acceleration of the match phase. The efficiency of the speech benchmarks on the BT-100 in [G 88] are due to the complex match functions found in signal pattern recognition.

The second case is if there are many references per PE. Again, for sake of example, assume that $c = p$. Then many references per PE implies that N/L is very small, again leading to a small deviation and thus essentially perfect linear acceleration. This situation occurs, for example, when there are large numbers of simple matches to perform. E.g., if there are millions of references, and we distribute them over thousands of PEs, then each PE will have plenty of work to do regardless of how simple the match might be.

The third case, although not as interesting, occurs if one links together relatively slow processors with a relatively fast communications network, i.e. $p \gg c$. This will lead to perfect linear acceleration, but would be the result of an unbalanced design yielding an efficient albeit slow implementation. For example, consider a situation in which $p > 10c$, i.e. a processor instruction cycle requires more than ten times the communications time per level. Then, in a 1023 PE tree of depth 10, an initiation or synchronization communication would appear to be instantaneous, since $c \log_2 N$ is less than one instruction cycle.

4.3 Resolve. The resolve kernel has two components: first, initiation of the process, then bubbling up of the best score via local three-way comparisons. Both require time proportional to the logarithm of the number of PEs, which we will show is minimal over all parallel architectures of bounded degree.

As before, let L be the number of reference patterns and N be the number of PEs. Let c be the communication time per word between levels, and p the processing time for a two-way comparison at each PE. Denote by $T_R(L, N)$ the time for resolving a single global best score from L values evenly distributed over an N -PE tree, which is given by

$$(8) \quad T_R(L, N) = pL/N + 2(c + p) \log_2 N,$$

since pL/N is the time for each PE concurrently to select the best of its local L/N references, $c \log_2 N$ the time to initiate the process, and $(c + 2p) \log_2 N$ the time to bubble up the global best score to the root.

Although Resolve is a communications function, it involves computation as well. Thus, it is reasonable to compare the time required to select the best of L scores on 1 PE to the time on N PEs. Perfect linear acceleration of the resolve phase would occur when

$T_R(L, N) = T_R(L, 1)/N$, where

$$(9) \quad T_R(L, 1) = pL.$$

Minimality. The deviation from perfect linear acceleration of the resolve operator is the term $2(c + p) \log_2 N$ in equation (8). We will show that, when $L = N$, i.e. for one reference pattern per PE, this deviation is minimal over all architectures whose interconnections are of bounded degree. Consider an interconnection whose branching factor is at most $b + 1$, and pose the problem of resolving the minimum of a set that is distributed over the ensemble with one value per PE. The worst-case time for gathering that value to a specific PE is the diameter of the interconnection, which is at least $\log_b N$. Thus, for any interconnection, the resolve operation requires at least logarithmic time.

4.4 K-Best Resolve. Thus far, we have shown that the basic pattern matching problem of finding the single best match can be computed minimally on a tree of processors. In some applications, however, one requires not merely the single best score, but the K-best scores. Furthermore, in many applications it is desirable to maintain several alternative contextual hypotheses that will weight the decision via modification of prior probabilities, and one is interested in the K-best scores for each of these several alternate hypotheses. In these situations, it is possible to pipeline the multiple resolves up the tree, absorbing the logarithmic overhead in a pipeline. We now consider the problem of finding the K-Best scores, given a set of L scores distributed evenly among the N nodes of the tree.

We simplify the problem for sake of exposition: Consider a collection of L values, $z_1, z_2, \dots, z_L$, distinct real numbers, distributed among the N nodes. Assume that L is an integral multiple of N , and each node initially contains L/N values. The problem is to produce at the root a decreasing list of the K largest values, $z_{i(1)}, \dots, z_{i(K)}$ such that $z_{i(1)} > z_{i(2)} > \dots > z_{i(K)} > z_j$, for all $j \notin \{i(1), i(2), \dots, i(K)\}$. All of what follows is easily adapted to the case where N need not divide L and the z_i need not be distinct, and a generalization to an arbitrary tree of processors is also discussed. In a typical instance of the problem for machines that should be soon be available, one might have $K \approx 100$, $N \approx 1000$, and $L \approx 100,000$.

We will consider and compare two solutions to finding the K-best scores. The first is to merely iterate the Resolve function. The second is to pipeline the Resolve operations. We denote these times by T_{nopipe} and T_{pipe} respectively.

Non-Pipelined K-Best Resolve. The simplest solution is to repeat the following steps K times, where $M = \min(K, L/N)$.

1. *Local select and sort:* In parallel each node computes the list of the M largest local values in decreasing order.
2. *Merge sort via iterated resolves:* Do the following K times. Select the maximum of the values at the heads of the N local lists via a resolve, broadcast this global

maximum, remove it from the head of the local list where it originated, and promote the next largest local value to the head of that list.

Let $T_s(L/N, M)$ denote the time for the first phase: the serial computation, concurrently in each PE, of the list of M largest values from its set of L/N values. The second phase involves a sequence of K resolves, whose individual times were given in equation (8). Thus, the worst case running time $T_{npipe}(L, N, K)$ of the algorithm is

$$(10) \quad T_{npipe}(L, N, K) = T_s(L/N, M) + K[S + 2(p + c)] \log_2 N,$$

where p and c are processing and communication times respectively, and S is a startup cost. We will discuss the complexity of the local serial sort, T_s , in the next subsection.

For example, consider the case of one reference pattern per PE, where $L = N$. Then, $T_s = 0$ and

$$(11) \quad T_{npipe}(N, N, K) = K(a_1 + a_2 \log_2 N),$$

where a_1 and a_2 are constants derived from S, p and c in formula 10. Observe that the time is linear in K , but the slope depends on $\log_2 N$. This behavior is illustrated quantitatively in the benchmarks reported on in the Section 5.

Pipelined K-Best Resolve. We now define and analyze a pipelined version of the above algorithm. We show that, in the analog of equation (11), execution time will still be linear in K , but the slope will be independent of $\log_2 N$. Rather, the $\log_2 N$ term appears as a very small additive bias representing a pipeline fillup and shutdown time. This makes the communications time essentially independent of tree size, and thus scalable. This behavior is illustrated quantitatively in the kernel benchmarks of Section 5. The pipelined algorithm is as follows, where as before $M = \min(K, L/N)$.

1. *Local select and sort:* In parallel each node computes the list of its M largest local values in decreasing order. (This is the same as in the non-pipelined algorithm.)
2. *Merge sort via pipelined resolves:* The nodes merge (as described below) the lists computed in phase one to produce a list of the K largest values at the root.

First consider the local select and sort function of phase 1, which is performed concurrently in all PEs and is the same as in the non-pipelined algorithm. The pipelining impacts only the merge sort phase of the algorithm, but it will be useful to understand the relative magnitudes of the two phases. It is difficult to give a single sorting method that would be best in practice for all problem sizes, however, the following is asymptotically minimal. Blum's algorithm [A 74] can be utilized to select the M -th largest element in time $O(L/N)$ and heapsort [A 74] used to sort the M largest values in time $O(M \log_2 M)$, to achieve, where A is some constant,

$$(12) \quad T_s(L/N, K) \leq A(L/N + M \log M) \leq A(L/N + K \log_2 K).$$

The Merge Phase. After the first phase of the algorithm, each node holds a list of its M largest local values in decreasing order. We now describe how to carry out the second phase where the N lists are merged, using three auxiliary buffers per node, each with the capacity to store up to K values. At the end of this subsection we adapt the algorithm to use less memory.

In the course of the algorithm, a node manipulates three FIFO queues:

- q_0 : used to hold values received from the left child;
- q_1 : used to hold values received from the right child;
- q_2 : used to hold local values.

Initially, q_2 holds the list of the K largest local values obtained via the local sort and select phase, padded with $-\infty$ if $K > L/N$. Queues q_0 and q_1 are initially empty. The algorithm is such that all three queues will always be in decreasing order; the largest values will be first and the smallest values last.

The task of a leaf is to write its local list q_2 to its parent one value at a time. An internal node first waits until each child has a value to send upwards. Then, for K steps, the PE accepts two values from its two children and enqueues the two in q_0 and q_1 . The PE then selects a maximum value from among the three at the heads of q_0, q_1 , and q_2 , removes that value from its queue, and outputs the value to its parent. What follows is the program (a sliced procedure involving communication) that each PE concurrently executes within the K-Best kernel.

The program is data driven, in that when one PE wants to communicate with another it waits until the other PE is ready and then carries out the communication. In this way, the outputs of the leaves trigger the data bubbling up to the root.

```

do  $K$  times {
  if not a leaf then {
    wait until left child supplies a value; insert that value in  $q_0$ 
    wait until right child supplies a value; insert that value in  $q_1$ 
     $v_0$  = value at the head of  $q_0$ 
     $v_1$  = value at the head of  $q_1$ 
     $v_2$  = value at the head of  $q_2$ 
     $j$  = index in  $\{0, 1, 2\}$  such that  $v_j = \max\{v_0, v_1, v_2\}$ 
  }
  else  $j = 2$ 
  remove  $v_j$  from  $q_j$ 
  if a root then append  $v_j$  to the solution list
  else wait until parent is ready and then supply  $v_j$  to parent
}

```

Denote the running time of the merge phase of the pipelined algorithm by $T_{merge}(L, N, K)$, which is given by

$$(13) \quad T_{merge}(L, N, K) = S + 2(c + p)(K + \log_2 N),$$

where c and p are communications and processing parameters respectively, and S is the host startup cost. The root PE is inactive for $\log_2 N$ cycles while waiting for the initial values to bubble up from the leaves.

In the case of one reference pattern per PE, where $L = N$, then $T_s = 0$ and we can model the total execution time for the pipelined algorithm, T_{pipe} , as

$$(14) \quad T_{pipe}(N, N, K) = b_0 + b_1 K + b_2 \log_2 N,$$

where b_0 , b_1 and b_2 are constants derived from S , c and p in formula 13. Comparing this time to that for the non-pipelined algorithm in equation (11), we observe that execution time is still linear in K , but the $\log_2 N$ term appears additively rather than multiplicatively. Thus, the execution time is essentially independent of the tree size if K is large. This behavior will be supported quantitatively by a benchmark in Section 5.

We combine equations (12) and (13) to bound the total run time of the pipelined K-Best Resolve,

$$(15) \quad T_{pipe}(L, N, K) \leq A (L/N + M \log M + \log_2 N),$$

where A represents some positive constant and $M = \min(K, L/N)$.

Minimality. We do not know, for all parameter values, if T_{pipe} is within a constant factor of minimal. However, for particular scenarios, it is indeed minimal to within a constant factor. This occurs when the parameter values are such that the cost of the local sort phase is proportional to L/N , (the number of data elements per PE), rather than $L/N + M \log_2 M$. One example of such a scenario is for a 2-Best resolve, i.e. $K = 2$. A second example is when there is only one reference pattern per PE, i.e. $L = N$. In those cases, the net running time is bounded by $T_{pipe}(L, K, N) \leq A(L/N + K + \log_2 N)$. To show minimality, a simple fan-in argument yields that for A_1 some positive constant, at least time $A_1(L/N + K + \log_b N)$ time is needed on any network of processors where the K largest must be collected at one processor and each processor communicates directly with at most $b + 1$ other processors. Thus, in the stated scenarios, the K-Best Resolve kernel is minimal, within a constant factor, over all parallel architectures whose interconnections are of bounded degree.

It is straightforward to show the merge algorithm is correct:

THEOREM. *For each node x and each $i = 1, 2, \dots, K$, the i -th output of node x to its parent is the i -th largest value stored among all nodes in the subtree rooted at x .*

Proof. We say that a leaf has level 0, and any other PE has level one greater than the maximum level of its children. We proceed by induction on the level index. By construction, the Theorem holds for the i -th output of each leaf, for each $i = 1, 2, \dots, K$. Suppose that, for some $j \geq 1$, the Theorem holds for the i -th output of each node at level j , for each $i = 1, 2, \dots, K$. It then follows that during the K steps at which each node x at level $j + 1$ is active, the node computes and then outputs a three way merge of

- the sorted list consisting of the K largest values in the subtree rooted at the left child of x ,
- the sorted list consisting of the K largest values in the subtree rooted at the right child of x ,
- the sorted list of values local to node x ,

which establishes the result at level $j + 1$, and by induction at all levels. $\square$

In practice it would be best to use two buffers to implement q_0 and q_1 , where each consists of K consecutive slots, the unit of memory needed to represent a value. In that case, the algorithm requires at least $2K$ slots of auxiliary, or scratch, memory to support the two queues.

The algorithm is easily modified to use just $2s$ auxiliary slots, for any $1 \leq s \leq K$, at the cost of some extra communication. More precisely, q_0 and q_1 would each be implemented as a FIFO queue with capacity s using a circular buffer [A 74] of s slots. If, for example, at

a given node x , queue q_0 is at capacity, then node x blocks the output from its left child. (That blocked data will not figure into the calculation of the current output of node x anyway.) A blocked output is simply left at the head of the queue from which it arose and retried at each step until it is unblocked and accepted. This modification hardly changes the processing time. The worst case running time approximately agrees with (16), except that the constant c is amplified to account for the blocking and unblocking signals.

To adapt the algorithm for an arbitrary tree of processors, a node having b children maintains $b + 1$ queues: one for local values and one for values received from each child. A $b + 1$ way maximization replaces the three way one. This changes the running time of the merge phase to $(c_b + p_b(K + H))$, where H is the depth of the tree, and c_b and p_b represent communication and processing parameters dependent on the maximum branching factor.

5. Experimental Kernel Benchmarks. These benchmark experiments were performed on an AT&T BT-100 processor running at a 20 MHz clock rate attached to an AT&T PC6300+ host. The processor was reconfigured to provide (sometimes skewed) trees of depths ranging from 3 to 9, corresponding to balanced trees of sizes 7 through 511. The times were measured by embedding the kernel function call within a loop in the host C program, and then averaging. The first set of benchmarks show that the broadcast kernel is indeed invariant to tree depth, regardless of block size. The second set of benchmarks demonstrate the speedup and insensitivity to tree depth obtained via the pipelined K-Best Resolve as compared to the nonpipelined algorithm.

5.1 Broadcast. We recall that equation (2) predicted that, for a broadcast instruction, elapsed time until the next function is executed would be independent of tree depth and linearly dependent on the broadcast block size. Figure 3 shows execution time for integer broadcasts on blocks of size 1, 10 and 100, for trees of depth 3, 5 and 9. Observe that, for a fixed block size, the broadcast times are essentially constant with respect to tree depth. Figure 4 shows the same execution times plotted as a function of block size. Observe the linear relationship. There is only one plot in Figure 4 because the values for the three tree-depths overlap.

Figure 3

Execution Times for an Integer Block Broadcast

Figure 4

Execution Times for an Integer Block Broadcast

Figure 5

Benchmark Comparison of K-Best Resolve/Report

A data analysis yields a broadcast communications parameter of $c = 22$ microseconds/word/level, or equivalently 1.4 microseconds/bit/level. As observed in Section 4.1, this unremarkable value arises since communications is accomplished via software through a bit-serial I/O port. A more reasonable value is projected for the next generation machine [S 88], which executes the kernel in a semi-custom router chip, yielding a 16-bit wide data path at 20 Mhz. Finally, one can determine the startup overhead incurred by the current PC host, which is $S = 120$ microseconds.

5.2 K-Best Resolve Benchmarks. In this section, we report on several experiments that support the performance models for the K-best resolve algorithms of section 4.4. We consider just the second phase (merging) of the algorithms, since the first phases (local sorting) were identical and only involve processing. Recall that the execution times for the non-pipelined and pipelined K-best resolve algorithms were modeled by equations (11) and (14) as

$$(16) \quad T_{no\text{pipe}} = K(a_1 + a_2 \log_2 N),$$

$$(17) \quad T_{pipe} = b_0 + b_1 K + b_2 \log_2 N.$$

Each score was a 32-bit floating point quantity, with three associated 16-bit integers to be reported. Thus, the measured times are for a K-Best Resolve/Report. The times were measured to an accuracy of 0.5 % by looping several times through C code on the host that invoked the communication functions. Results for the pipelined and non-pipelined algorithms are shown in Figure 3 for values of K between 1 and 20 and for tree depth ranging from 3 to 9. Observe that the pipelined approach is much faster even when K is small. Observe also that the slopes of the linear regressions for the pipelined approach are constant for all N , confirming that the time is independent of tree depth, and thus of the number of PEs. We can fit these timing measurements to the predicted behavior via a regression analysis. The coefficients in formulae 16 and 17 are given in milliseconds with a correlation coefficient of 0.99 as:

$$(18) \quad T_{no\text{pipe}} = K(0.70 + 0.061 \log_2 N)$$

$$(19) \quad T_{pipe} = 0.204 + 0.132K + 0.050 \log_2 N.$$

REFERENCES

- [A 74] AHO, A., HOPCROFT J., AND ULLMAN, J., *The Design and Analysis of Computer Algorithms*, Addison Wesley, New York (1974).
- [B 79] J.L. BENTLEY AND H.T. KUNG, *A Tree Machine for Searching Problems*, Proc. International Conference on Parallel Processing, IEEE New York (1979), pp. 257–266.

- [B 80] S.A. BROWNING, *The Tree Machine: A Highly Concurrent Computing Environment*, Ph.D. Thesis, CalTech (Jan 1980).
- [D 73] R.O. DUDA AND P.E. HART, *Pattern Classification and Scene Analysis*, Wiley (1973).
- [F 66] M.J. FLYNN, *Very High-Speed Computing Systems*, Proc. of the IEEE 54, 12 (Dec. 1966), pp. 1901–1909.
- [G 87] A.L. GORIN AND R.R. SHIVELY, *The ASPEN Parallel Computer, Speech Recognition, and Parallel Dynamic Programming*, Proc. 1987 International Conference on Acoustics, Speech and Signal Processing (ICASSP) (April 1987), pp. 976–979.
- [G 88] A.L. GORIN AND D.B. ROE, *Parallel Level-Building on a Tree Machine*, Proceedings of ICASSP 88 (April 1988), pp. 295–298.
- [L 81] C.E. LEISERSON, *Area-Efficient VLSI Computation*, Ph.D. Thesis, Carnegie Mellon University (1981).
- [L 85] S.E. LEVINSON, *Structural Methods in Automatic Speech Recognition*, Proceedings of the IEEE (Nov. 1985).
- [L 87] M. B. LOWRIE AND K. FUCHS, *Reconfiguration Tree Architectures using Subtree Oriented Fault Tolerance*, IEEE Trans. on Comp., c-36, 10, pp. 1172–1182 (Oct. 1987).
- [P 85] PRADHAN, D.K., *Dynamically Restructurable Fault-Tolerant Processor Network Architectures*, IEEE Trans. on Comp., c-34, 5 (May 1985).
- [R 81] L.R. RABINER AND S.E. LEVINSON, *Isolated and Connected-Word Recognition-Theory and Selected Applications*, IEEE Trans. on Communications, 29, 5 (May 1981).
- [S 82] L. SNYDER, *Introduction to the Configurable, Highly Parallel Computer*, Computer (January 1982).
- [S 84] D. SHAW, *SIMD and MSIMD variants of the Non-Von Supercomputer*, Proc. of ComCon, San Francisco, CA (Feb. 1984).
- [S 85] S.J. STOLFO AND D. MIRANKER, *DADO: A Parallel Processor for Expert Systems*, Advanced Computer Architecture (Agrawal, ed.) IEEE Comp. Soc. Press (1985).
- [S 88] R.R. SHIVELY AND A.L. GORIN, *A Fault-Tolerant Reconfigurable Systolic Signal Processor*, Proc. ICASSP 89 (May 1989), pp. 2397–2340.
- [T 85] Y. TAKAHASHI, ET AL, *Efficiency of Parallel Computation on the Binary Tree Machine CORAL '83*, Journal of Information Processing, 8, 4 (1985), pp. 288–299.

Recent IMA Preprints

#	Author/s	Title
516	P. Singh, Ph. Caussignac, A. Fortes, D.D. Joseph and T. Lundgren,	Stability of Periodic Arrays of Cylinders Across the Stream by Direct Simulation
517	Daniel D. Joseph,	Generalization of the Foscolo-Gibilario Analysis of Dynamic Waves
518	A. Narain and D.D. Joseph,	Note on the Balance of Energy at a Phase Change Interface
519	Daniel D. Joseph,	Remarks on inertial radii, persistent normal stresses, secondary motions , and non-elastic extensional viscosities
520	D. D. Joseph,	Mathematical Problems Associated with the Elasticity of Liquids
521	Henry C. Simpson and Scott J. Spector,	Some Necessary Conditions at an Internal Boundary for Minimizers in Finite Elasticity
522	Peter Gritzmann and Victor Klee,	On the 0-1 Maximization of Positive Definite Quadratic Forms
523	Fu-Cho Pu and D.H. Sattinger,	The Yang-Baxter Equations and Differential Identities
524	Avner Friedman and Fernando Reitich,	A Hyperbolic Inverse Problem Arising in the Evolution of Combustion Aerosol
525	E.G. Kalnins, Raphael D. Levine and Willard Miller, Jr.,	Conformal Symmetries and Generalized Recurrences for Heat and Schrödinger Equations in One Spatial Dimension
526	Wang Jinghua and Gerald Warnecke,	On Entropy Consistency of Large Time Step Godunov and Glimm Schemes
527	C. Guillopé and J.C. Saut,	Existence Results for the Flow of Viscoelastic Fluids with a Differential Constitutive Law
528	H.L. Bodlaender, P. Gritzmann, V. Klee and J. Van Leeuwen	Computational Complexity of Norm-Maximization
529	Li Ta-tsien (Li Da-qian) and Yu Xin,	Life-Span of Classical Solutions to Fully Nonlinear Wave Equations
530	Jong-Shenq Guo,	A Variational Inequality Associated with a Lubrication Problem
531	Jong-Shenq Guo,	On the Semilinear Elliptic Equation $\Delta w - \frac{1}{2}y \cdot \nabla w + \lambda w - w^{-\beta} = 0$ in R^n
532	Andrew E. Yagle,	Inversion of the Bloch transform in magnetic resonance imaging using asymmetric two-component inverse scattering
533	Bei Hu,	A Fiber Tapering Problem
534	Peter J. Olver,	Canonical Variables for BiHamiltonian Systems
535	Michael Renardy,	A Well-Posed Boundary Value Problem for Supercritical Flow of Viscoelastic Fluids of Maxwell Type
536	Michael Renardy,	Ill-Posedness Resulting from Slip As a Possible Explanation of Melt Fracture
537	Michael Renardy,	Compatibility Conditions at Corners Between Walls and Inflow Boundaries for Fluids of Maxwell Type
538	Rolf Rees,	The Spectrum of Restricted Resolvable Designs with $r = 2$
539	D. Lewis and J.C. Simo,	Nonlinear stability of rotating pseudo-rigid bodies
540	Robert Hardt and David Kinderlehrer,	Variational Principles with Linear Growth
541	San Yih Lin and Yisong Yang,	Computation of Superconductivity in Thin Films
542	A. Narain,	Pressure Driven Flow of Pure Vapor Undergoing Laminar Film Condensation Between Parallel Plates
543	P.J. Vassiliou,	On Local Equivalence for Vector Field Systems
544	Brian A. Coomes,	On Conditions Sufficient for Injectivity of Maps
545	Yanchun Zhao,	A Class of Global Smooth Solutions of the One Dimensional Gas Dynamics System
546	H. Holden, L. Holden and N.H. Risebro,	Some Qualitative Properties of 2×2 Systems of Conservation Laws of Mixed Type
547	M. Slemrod,	Dynamics of Measured Valued Solutions to a Backward-Forward Heat Equation
548	Avner Friedman and Jürgen Sprekels,	Steady States of Austenitic–Martensitic–Domains in the Ginzburg–Landau Theory of Shape Memory Alloys
549	Avner Friedman and Bei Hu,	Degenerate Hamilton–Jacobi–Bellman Equations in a Bounded Domain
550	E.G. Kalnins, Willard Miller, Jr., and M.V. Tratnik,	Families of Orthogonal and Biorthogonal Polynomials on the N-Sphere
551	Heinrich Freistühler,	On Compact Linear Degeneracy
552	Matthew Witten,	Quantifying the Concepts of Rate and Acceleration/Deceleration of Aging
553	J.P. Albert and J.L. Bona,	Total Positivity and the Stability of Internal Waves in Stratified Fluids of Finite Depth
554	Brian Coomes and Victor Zurkowski,	Linearization of Polynomial Flows and Spectra of Derivations

Recent IMA Preprints (Continued)

#	Author/s	Title
555	Yuriko Renardy , A Couette-Poiseuille Flow of Two Fluids in a Channel	
556	Michael Renardy , Short wave instabilities resulting from memory slip	
557	Daniel D. Joseph and Michael Renardy , Stokes' first problem for linear viscoelastic fluids with finite memory	
558	Xiaxi Ding , Superlinear Conservation Law with Viscosity	
559	J.L. Ericksen , Liquid Crystals with Variable Degree of Orientation	
560	F. Robert Ore, Jr. and Xinfu Chen , Electro-Optic Modulation in an Arbitrary Cross-Section Waveguide	
561	M.V. Tratnik , Multivariable biorthogonal continuous-discrete Wilson and Racah polynomials	
562	Yisong Yang , Existence of Solutions for a Generalized Yang-Mills Theory	
563	Peter Gritzmann, Laurent Habsieger and Victor Klee , Good and Bad Radii of Convex Polygons	
564	Martin Golubitsky, Martin Krupa and Chjan. C. Lim , Time-Reversibility and Particle Sedimentation	
565	G. Yin , Recent Progress in Parallel Stochastic Approximations	
566	G. Yin , On H-Valued SA: Finite Dimensional Approximations	
567	Chien-Cheng Chang , Accurate Evaluation of the Effect of Diffusion and Conductivity in Certain Equations	
568	Chien-Cheng Chang and Ruey-Ling Chern , The Effect of Viscous Diffusion in Discrete Vortex Dynamics for Slightly Viscous Flows	
569	Li Ta-Tsien (Li Da-qian) and Zhao Yan-Chun , Global Existence of Classical Solutions to the Typical Free Boundary Problem for General Quasilinear Hyperbolic Systems and its Applications	
570	Thierry Cazenave and Fred B. Weissler , The Structure of Solutions to the Pseudo-Conformally Invariant Nonlinear Schrödinger Equation	
571	Marshall Slemrod and Athanasios E. Tzavaras , A Limiting Viscosity Approach for the Riemann Problem in Isentropic Gas Dynamics	
572	Richard D. James and Scott J. Spector , The Formation of Filamentary Voids in Solids	
573	P.J. Vassiliou , On the Geometry of Semi-Linear Hyperbolic Partial Differential Equations in the Plane Integrable by the Method of Darboux	
574	Jerome V. Moloney and Alan C. Newell , Nonlinear Optics	
575	Keti Tenenblat , A Note on Solutions for the Intrinsic Generalized Wave and Sine-Gordon Equations	
576	P. Szmolyan , Heteroclinic Orbits in Singularly Perturbed Differential Equations	
577	Wenxiong Liu , A Parabolic System Arising In Film Development	
578	Daniel B. Dix , Temporal Asymptotic Behavior of Solutions of the Benjamin-Ono-Burgers Equation	
579	Michael Renardy and Yuriko Renardy , On the nature of boundary conditions for flows with moving free surfaces	
580	Werner A. Stahel , Robust Statistics: From an Intellectual Game to a Consumer Product	
581	Avner Friedman and Fernando Reitich , The Stefan Problem with Small Surface Tension	
582	E.G. Kalnins and W. Miller, Jr. , Separation of Variables Methods for Systems of Differential Equations in Mathematical Physics	
583	Mitchell Luskin and George R. Sell , The Construction of Inertial Manifolds for Reaction-Diffusion Equations by Elliptic Regularization	
584	Konstantin Mischaikow , Dynamic Phase Transitions: A Connection Matrix Approach	
585	Philippe Le Floch and Li Tatsien , A Global Asymptotic Expansion for the Solution to the Generalized Riemann Problem	
586	Matthew Witten, Ph.D. , Computational Biology: An Overview	
587	Matthew Witten, Ph.D. , Peering Inside Living Systems: Physiology in a Supercomputer	
588	Michael Renardy , An existence theorem for model equations resulting from kinetic theories of polymer solutions	
589	Daniel D. Joseph and Luigi Preziosi , Reviews of Modern Physics: Addendum to the Paper "Heat Waves"	
590	Luigi Preziosi , An Invariance Property for the Propagation of Heat and Shear Waves	
591	Gregory M. Constantine and John Bryant , Sequencing of Experiments for Linear and Quadratic Time Effects	
592	Prabir Daripa , On the Computation of the Beltrami Equation in the Complex Plane	
593	Philippe Le Floch , Shock Waves for Nonlinear Hyperbolic Systems in Nonconservative Form	
594	A.L. Gorin, D.B. Roe and A.G. Greenberg , On the Complexity of Pattern Recognition Algorithms On a Tree-Structured Parallel Computer	
595	Mark J. Friedman and Eusebius J. Doedel , Numerical computation and continuation of invariant manifolds connecting fixed points	
596	Scott J. Spector , Linear Deformations as Global Minimizers in Nonlinear Elasticity	