A Parallel Goodness of Fit Test Algorithm for Realtime Applications
U. Nagaraj Shenoy  Alok N. Choudhary  Xiaohui Shen  Donald Weiner  Pramod Varshney

CPDC-TR-9811-022
©1998 Center for Parallel and Distributed Computing
June 1998.

Center for Parallel and Distributed Computing
Department of Electrical & Computer Engineering
Northwestern University
Technological Institute
2145 Sheridan Road, Evanston IL-60208
A Parallel Goodness of Fit Test Algorithm for Real-time Applications*

U.Nagaraj Shenoy    Alok N. Choudhary    Xiaohui Shen
ECE Department
Northwestern University
Evanston, IL 60208

Donald Weiner    Pramod Varshney
EECS Department
Syracuse University
Syracuse, NY 13244

Abstract

Ozturk algorithm is one of the goodness of fit tests used to check consistency of a null hypothesis against a given sample of random data. For certain real-time applications of the Ozturk algorithm, it is essential that the processing time is within some acceptable limits. Employing parallel machines seems to be one way of reducing the processing time.

In this report we discuss the parallelization of this algorithm on a variety of parallel machines. Our current results indicate that the algorithm scales well up to 6-8 processors depending on the architecture of the parallel machine. Further scaling is impeded by some inherently sequential portions of the algorithm.

1 Introduction

In the analysis of random data, situations are encountered where there may be various statistical models or hypotheses that need to be checked against the data. In a typical scenario, one would like to check whether a particular distribution (the null hypothesis) consistently represents the data from a certain experiment. Several tests have been proposed for this purpose and the test developed by Professor Aydin Ozturk (commonly known as Ozturk Algorithm [1]) is the focus of this report.

Ozturk algorithm is one of the goodness of fit tests used to check consistency of a null hypothesis against a given sample of random data. Since the algorithm is often used to process real time data, it is necessary to reduce the time needed to process each sample of data. For large sample sizes, it is observed that the computation time of the Ozturk algorithm is not acceptable.

One way to achieve improved processing speed is by employing multiple processors.

---

*This work was supported by Air Force Materials Command under contract F30602-97-C-0026
The algorithm in the original sequential form exhibits very little parallelism which can be exploited by conventional parallelizing compilers on distributed and distributed shared memory parallel machines. However, there exists significant amount of parallelism which can be exploited by hand parallelizing the code. In the current work we attempt to improve the execution time of this algorithm by employing multiple processors.

As the original sequential code changed later, so we presented two sets of results.

In the next few sections we discuss our initial experience in parallelizing this algorithm on a variety of parallel machines for the original code: IBM SP2 - a 16 processor distributed memory multiprocessor, IBM J30 - an 8 processor symmetric multiprocessor, SGI Origin 2000 - an 8 processor distributed shared memory multiprocessor and a cluster of HP 9000 work-stations. For the new code, the results from SP2 and HP work-stations are presented.

To ensure portability across different parallel architectures we have adopted the message passing style of parallel programming. Further, we have used standard message passing interface namely the MPI [2, 3, 4, 5].

The rest of the paper is organized as follows. In Section 2 we look more closely at the original sequential implementation of the Ozturk algorithm and study how the computation time is distributed among various phases of the algorithm. With this in mind, we evolve a parallelization strategy which is discussed in Section 3. Section 5 describes the code changes from the original sequential code. The results of running our parallel Ozturk algorithm on a variety of parallel machines are presented in Section 4 and 6. Finally we conclude with Section 7.

2 Breakup of computation time

Before deciding the parallelization strategy, the sequential algorithm was profiled to study the breakup of computation time in different phases of the algorithm. The original algorithm which was implemented in FORTRAN has around 50 functions. None of these exhibit significant loop level parallelism which can be exploited by conventional parallelizing compilers. Manual parallelization seems to be the only way to parallelize this algorithm.

Among these 50 functions, two top level functions namely mst at and eexpuv together contribute to more that 80% of the execution time (the function mst at along with the functions called by it contributes to more than 50% of the time and eexpuv along with the functions called by it contributes to 30% of the time). There also exist other parts of the algorithm, which contribute to roughly 15-20% of the time, that are difficult to parallelize either due to I/O or due to fine granularity.

Our current parallelization strategy mainly concentrates on portions of the algorithm which contribute to roughly 80% of the time.

3 Parallelization Strategy

Our emphasis is on arriving at a portable parallel implementation of the Ozturk algorithm which can run on a variety of parallel machines with little or no modification. We found that a message passing style of parallel programming met this goal. We used the standard message passing interface namely MPI as a portable interface due to its wide availability. The overview of our parallel implementation is shown in Figure 1.

The circles marked by $P_0 \ldots P_n$ indicate pro-
cessors $P_0...P_n$. The sequential parts of the algorithm are run on $P_0$. The computation in mstar and eexpuv is shared by all the processors in parallel. Whenever a parallelizable phase of the algorithm is encountered, the computation as well as the data is farmed out to all the processors. These processors work on their portions of the data in parallel and join together at the end of the parallel phase. Different MPI primitives like MPI_BCAST, MPI_GATHER and MPI_REDUCE are used to achieve computation partitioning as well as interprocessor communication. The algorithm proceeds as a sequence of alternating sequential and parallel phases.

4 Experimental Results of Old Code

We have ported our parallel Ozturk algorithm on a range of parallel architectures - loosely coupled network of work stations to tightly coupled shared memory multiprocessors. Table 1 lists these architectures and some of the related numbers.

Three different sizes (100, 250 and 500 points) of input sample data were tried to see how the algorithm scales with sample size. Figure 2 shows the processing time for these samples on different number of processors for these parallel machines.

execution time on SGI Origin 2000 is the least among all the four parallel architectures. This is both because of the higher performance of each processor (195 MHz) as well as better communication bandwidth supported (60 MB/sec) by this machine. The MPI running on Origin is a special implementation of the standard MPI which probably makes best use of the distributed shared memory architecture.

Though the IBM J30 is also a shared memory machine, the performance of the algorithm is not as good as that on Origin primarily because of the lower processor speed.
<table>
<thead>
<tr>
<th>machine</th>
<th>architecture</th>
<th>processor</th>
<th>clock</th>
<th>comm b/w(^1)</th>
<th>nodes</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP 9000</td>
<td>NOW(^2)</td>
<td>PA7200</td>
<td>120 MHz</td>
<td>800 KB/sec</td>
<td>8</td>
<td>MPICH</td>
</tr>
<tr>
<td>IBM SP2</td>
<td>multicomputer</td>
<td>P2SC</td>
<td>120 MHz</td>
<td>24 MB/sec</td>
<td>16</td>
<td>IBM MPI</td>
</tr>
<tr>
<td>SGI Origin</td>
<td>DSM(^3)</td>
<td>R10000</td>
<td>195 MHz</td>
<td>60 MB/sec</td>
<td>8</td>
<td>SGI MPI</td>
</tr>
<tr>
<td>IBM J30</td>
<td>SMP(^4)</td>
<td>PowerPC 604</td>
<td>112 MHz</td>
<td>3 MB/sec</td>
<td>8</td>
<td>MPICH</td>
</tr>
</tbody>
</table>

\(^1\)Effective, point to point  
\(^2\)Network of workstations  
\(^3\)Distributed shared memory multiprocessor  
\(^4\)Symmetric multiprocessor

Table 1: Platforms on which the Old parallel Ozturk algorithm has been ported

![Graphs showing execution times for different data sizes on various machines](image)

Table 2: Old code: Execution times for different data sizes on various machines
112 MHz) and also the poor communication bandwidth (3 MB/sec). On the other hand, though the SP2 processor is slightly faster (120 MHz) and supports a much higher communication bandwidth (24 MB/sec), J30 always performs slightly better. We suspect that this could be due to I/O overheads since the programs running on SP2 were accessing a remote file system as compared to the programs on J30 which accessed a local file system. Also, the higher communication bandwidth of SP2 may not have given an edge since the algorithm appears to be compute bound rather than communication bound.

Our observation that communication bandwidth does not play a major role in this application stands justified when we compare the performance of the algorithm on SP2 and the network of HP workstations. Architecturally both are distributed memory multiprocessors except that SP2 has a high bandwidth communication fabric (24 MB/sec) as compared to the low speed LAN (800 KB/sec) connecting the HP NOW. For small number of processors, both perform almost identically. The difference shows up for larger number of processors probably because of the reduced concurrency in the communication on the HP NOW.

For small input sizes, only Origin seems to keep up with the near linear speed up. However, SP2 catches up and to some extent scales better than Origin for larger input sizes. Overall, both Origin as well as SP2 seem to scale up very well. Among the machines, HP NOW seems to saturate early.

The speed-up curve seems to taper off beyond 6 to 7 processors in most of the cases. This is understandable given the fact that the parallelized code still has some sequential part which would definitely start dominating as we increase the number of processors irrespective of the architecture.

5 Changes of the New Code

The original code has been changed later. The changes include:

ORIGINAL MODS: (1) added subroutine to read/process an ASCII parameter file (2) removed references to encode/decode; replaced with formatted read/writes

FOLLOW-ON MODS: (1) explicitly declared variable types in the common blocks (2) modified position of -o argument in the makefile; make would ignore -o option if last argument on the line. (3) removed references to 'structure' keyword in subroutine 'initialize-pgm.f'.

The changes of ozturk code do not affect the parallelization method, so the new parallel code did the same way of parallelization as the old one, i.e. concentrates on mstar and expuv functions which contribute to most of the execution time.

6 Experimental Results of New Parallel Code

We have ported the new parallel Ozturk algorithm on Network of HP workstations and two IBM SP2 machines: one is a 16-node machine at Center for Parallel and Distributed Computing (CPDC) at Northwestern University and one is a 80-node machine at Argonne National Lab. Table 3 lists these architectures.

We also tried three different sizes (100, 250 and 500 points) of input sample data to study the scalability. Figure 4 shows the processing time for these samples on different number of processors for these parallel architectures.

Our results show that the execution time on Argonne's SP2 is the best of all parallel architectures and the network of HP workstations is the worst. This is because net-
work of workstations is loosely coupled with poor communication bandwidth. The Argonne’s SP2 has better communication bandwidth (104Mbytes/s) than CPDC’s although they have the same kind of CPU, so it always outperforms CPDC’s SP2.

In terms of scalability, all parallel architectures demonstrate good scalability. The only exception is Network of workstations on the small input size (100), but when the input size is large enough, its execution time also decreases appropriately as the number of nodes increases. One observation is that the larger input size, the better speedup can be achieved. Another observation is that the rate of performance gain decreases as the number of node becomes large, this is due to the sequential part of the code may dominate the performance.

Note, the old results cannot be compared to the new results since the SP2 and SGI Origin systems have been upgraded.

7 Conclusion

In this report we discussed the parallelization of the Ozturk algorithm on a range of multiprocessors. The algorithm exhibits significant amount of parallelism needing very little interprocessor communication. We presented the results from our implementation on IBM J30, IBM SP2, SGI Origin 2000 and a network of HP 9000 work-stations for the old code and results on IBM SP2 and network of HP 9000 work-stations for the new code. Our experiments show that the algorithm scales almost linearly for reasonable input sizes and on a small number of processors.

The current algorithm concentrates only on the most compute intensive parts of the original sequential algorithm and tries to parallelize them. The remaining parts are left sequential and are executed by one of the processors (processor 0). This results in the early saturation of the algorithm for larger number of processors since the sequential portions dominate. We observe that some of the sequentiality is introduced by I/O which probably could be eliminated.

References


<table>
<thead>
<tr>
<th>Machine</th>
<th>Architecture</th>
<th>Processor</th>
<th>Clock</th>
<th>Comm b/w</th>
<th>Nodes</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP 9000</td>
<td>NOW(^2)</td>
<td>PA7200</td>
<td>120 MHz</td>
<td>800 KB/sec</td>
<td>8</td>
<td>MPICH</td>
</tr>
<tr>
<td>IBM SP2</td>
<td>Multicomputer</td>
<td>P2SC</td>
<td>120 MHz</td>
<td>24 MB/sec</td>
<td>16</td>
<td>IBM MPI</td>
</tr>
<tr>
<td>IBM SP2(Argonne)</td>
<td>Multicomputer</td>
<td>P2SC</td>
<td>120 MHz</td>
<td>104 MB/sec</td>
<td>80</td>
<td>IBM MPI</td>
</tr>
</tbody>
</table>

\(^1\)Effective, point to point  
\(^2\)Network of workstations  
\(^3\)Distributed shared memory multiprocessor  
\(^4\)Symmetric multiprocessor

Table 3: Platforms on which the New parallel Ozturk algorithm has been ported

Table 4: New Code: Execution times for different data sizes on various machines