

CONF-910326-6

ARGONNE NATIONAL LABORATORY

9700 South Cass Avenue

Argonne, Illinois 60439

ANL/CP--73599

DE91 016515

PERFORMANCE OF A BENCHMARK IMPLEMENTATION OF THE
VAN SLYKE AND WETS ALGORITHM FOR STOCHASTIC PROGRAMS
ON THE ALLIANT FX/8*†

K.A. Ariyawansa‡

Mathematics and Computer Science Division

Preprint MCS-P242-0591

April 1991

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

*This research was supported in part by DOE Grant DE-FG-06-87ER25045, NSF DMS-8918785, the Center for Research in Parallel Computation at Rice University, Houston, Texas, and by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

†A summary of this paper was presented at the Fifth SIAM Conference on Parallel Processing for Scientific Computing held in Houston, Texas, during March 25-27, 1991.

‡Permanent Address: Department of Pure and Applied Mathematics, Washington State University, Pullman, WA 99164

MASTER
DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. W-31-109-ENG-38. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

ABSTRACT

In [3], a benchmark parallel implementation of the Van Slyke and Wets algorithm for stochastic linear programs, and the results of a carefully designed numerical experiment on the Sequent/Balance using the implementation were presented. An important use of this implementation is as a benchmark to assess the performance of approximation algorithms for stochastic linear programs described in [4]. These approximation algorithms are best suited for implementation on parallel vector processors like the Alliant FX/8. Therefore, the performance of the benchmark implementation on the Alliant FX/8 is of interest. In this paper, we present results observed when a portion of the numerical experiment described in [3] is performed on the Alliant FX/8. These results indicate that the implementation makes satisfactory use of the concurrency capabilities of the Alliant FX/8. They also indicate that the vectorization capabilities of the Alliant FX/8 are not satisfactorily utilized by the implementation.

1. Introduction. The two-stage stochastic program with recourse, and with a discretely distributed random variable with a finite number of realizations, is the following:

Find $x^* \in \mathbb{R}^{n_1}$ such that when $x := x^*$

$z(x) := c^T x + \mathcal{Q}(x)$ is minimized, and

$Ax = b$, $x \geq 0$, where

$$\mathcal{Q}(x) := E[\mathcal{Q}(x, \mathbf{h}, \mathbf{T})] = \sum_{k=1}^K p^k \mathcal{Q}(x, h^k, T^k),$$

$$\mathcal{Q}(x, \mathbf{h}, \mathbf{T}) := \inf_{y \in \mathbb{R}^{n_2}} \{q^T y : My = h - Tx, y \geq 0\},$$

$A \in \mathbb{R}^{m_1 \times n_1}$, $b \in \mathbb{R}^{m_1}$, $c \in \mathbb{R}^{n_1}$, $q \in \mathbb{R}^{n_2}$, $M \in \mathbb{R}^{m_2 \times n_2}$ are deterministic and given and $\mathbf{h} \in \mathbb{R}^{m_2}$, $\mathbf{T} \in \mathbb{R}^{m_2 \times n_1}$ are random with (\mathbf{h}, \mathbf{T}) having the given probability distribution $F := \{((h^k, T^k), p^k), k = 1, 2, \dots, K\}$. (1)

Problem (1) arises in operations research problem areas including industrial management, scheduling and transportation; in control theory; and in economics. The monograph [5] for example, contains details of specific applications.

A popular algorithm for (1) is due to Van Slyke and Wets [8]. In [3], a parallel benchmark implementation of the Van Slyke and Wets Algorithm for (1), and the results of a carefully designed numerical experiment on the Sequent/Balance were presented. An important use of this implementation is as a benchmark to assess the performance of the approximate algorithms for (1) described in [4, 2]. These approximate algorithms are best suited for implementation on parallel vector processors like the Alliant FX/8. Therefore, the performance of the benchmark implementation on the Alliant FX/8 is of interest. In this paper, we present a slightly modified version of the implementation given in [3] that is better suited for the Alliant FX/8. We have performed a portion of the numerical experiment described in [3] using this implementation on the Alliant FX/8. We indicate the results observed.

2. A Parallel Benchmark Implementation of the Van Slyke and Wets Algorithm on the Alliant FX/8. As in [3], throughout the paper we shall make the following assumptions regarding (1).

- (A1) The set $\{x : Ax = b, x \geq 0\}$ is nonempty and bounded.
- (A2) The set $\{w : My = w, y \geq 0\} = \mathbb{R}^{m_2}$.
- (A3) The set $\{v : M^T v \leq q\}$ is nonempty.

It can be verified that when (A1), (A2) and (A3) are satisfied, (1) has a finite minimum. Therefore, issues of unboundedness and infeasibility of (1) do not arise.

We now present the pseudo code on which our Fortran implementation on the Alliant FX/8 is based.

Algorithm 1:*input:*

$m_1; n_1; m_2; n_2; K; A; c; b; M; q; F := \{((h^k, T^k), p^k), k = 1, 2, \dots, K\};$
 $nprocs; maxcut; tol.$
/ we assume that $K > nprocs$ */*

output: $x; z.$

```

begin
  /* begin initializations */
   $t := 0;$ 
  for  $l := 1$  to  $maxcut$  do
     $E^l := 0; e^l := 0;$ 
  end do;
  {call DPLO to solve the lp
    find  $x \geq 0, \theta \in \mathfrak{R}$  such that
     $c^T x + \theta$  is minimized, and
     $Ax = b,$ 
     $(E^l)^T x + \theta \geq e^l, l = 1, 2, \dots, maxcut$ 
    from scratch};
  call optimality-cut ( $m_2, n_2, K, M, q, F, x, nprocs, E, e, \mathcal{Q}$ );
   $t := t + 1;$ 
   $E^t := E; e^t := e;$ 
  call lowerbound ( $m_1, n_1, t, A, b, (E^l, e^l, l = 1, 2, \dots, t), maxcut, x, \theta, z$ );
  call optimality-cut ( $m_2, n_2, K, M, q, F, x, nprocs, E, e, \mathcal{Q}$ );
/* end initializations */
  while  $|\mathcal{Q} - \theta| / \max\{1, |\mathcal{Q}|\} < tol$  do
     $t := t + 1;$ 
    if  $t > maxcut$  then
      {report that more cuts than  $maxcut$  need to be added and stop};
    else
       $E^t := E; e^t := e;$ 
      call lowerbound ( $m_1, n_1, t, A, b, (E^l, e^l, l = 1, 2, \dots, t), maxcut, x, \theta, z$ );
      call optimality-cut ( $m_2, n_2, K, M, q, F, x, nprocs, E, e, \mathcal{Q}$ );
    end if;
  end while;
end.

```

optimality_cut:

input:

$m_2; n_2; K; M; q; F; x; nprocs.$

output:

$E; e; Q.$

```

begin
  Etemp := 0; etemp := 0; E := 0; e := 0;
  /* dimensions of Etemp and etemp are  $n_1 \times nprocs$  and  $nprocs$  respectively */

  bnchsz :=  $\lfloor K/nprocs \rfloor$ ;
  for bunch := 1 to nprocs in parallel do
    firstk := (bunch - 1) * bnchsz + 1;
    if bunch  $\neq nprocs$  then
      lastk := bunch * bnchsz
    else
      lastk := K
    end if;
    call dobnch(firstk, lastk,  $m_2, n_2, M, q, F, x, Etemp(1 : n_1, bunch), etemp(bunch)$ );
  end do;
  for bunch := 1 to nprocs do
    E := E + Etemp( $1 : n_1, bunch$ );
    e := e + etemp(bunch);
  end do;
  Q := e -  $E^T x$ ;
  return;
end.

```

dobnch:

input:

$firstk; lastk; m_2; n_2; M; q; F; x.$

output:

$E; e.$

```

begin
  for k := firstk to lastk do

```

```

 $w := h^k - T^k x;$ 
if  $k = firstk$  then
  {call DPLO to solve the lp
   find  $y \geq 0$  such that
    $q^T y$  is minimized, and
    $My = w$ 
   from scratch to obtain dual maximizer  $v$ };
else
  {call DPLO with restart procedure and with new options to prevent
   disk reads, and to prevent forming LU factorization of initial basis
   to solve the lp
   find  $y \geq 0$  such that
    $q^T y$  is minimized, and
    $My = w$ 
   to obtain dual maximizer  $v$ };
end if;
 $E := E + p^k (T^k)^T v;$ 
 $e := e + p^k (h^k)^T v;$ 
end do;
return;
end.

```

lowerbound:

input:

$m_1; n_1; t; A; b; E^l, e^l, l = 1, 2, \dots, t; maxcut.$

output:

$x; \theta; z.$

begin

{call DPLO with restart procedure, and with new option to prevent
disk reads, and with a specification that $(m_1 + t)$ -th row of constraint
matrix changes to $[(E^t)^T, 1]$, to solve the lp

find $x \geq 0, \theta \in \mathfrak{R}$ such that

$c^T x + \theta$ is minimized, and

$Ax = b,$

$(E^l)^T x + \theta \geq e^l, l = 1, 2, \dots, maxcut\};$

$z := c^T x + \theta;$

return;

end.

In Algorithm 1, $nprocs$ is the number of processors. The input parameter $maxcut$ essentially places an upper bound on the number of calls to routine *optimality-cut*, while the input parameter tol is a tolerance used in the stopping criterion. Justification for using these parameters is given in [3].

Algorithm 1 and Algorithm 2 of [3] are equivalent (in exact arithmetic). Algorithm 2 of [3] does not contain a routine *dobnch*, and the steps of routine *dobnch* of Algorithm 1 are explicitly included in routine *optimality-cut* of Algorithm 2 of [3]. This however, prevented the Alliant Fortran compiler from choosing the ‘parallel do loop’ in routine *optimality-cut* of Algorithm 2 of [3] for concurrent execution. The reorganization of Algorithm 2 of [3] into Algorithm 1 above by introducing the routine *dobnch* makes the Alliant Fortran compiler choose the ‘parallel do loop’ in Algorithm 1 for concurrent execution as desired.

The routine DPLO referred to in Algorithm 1 is an lp solver due to Hanson and Hiebert [6]. The modifications to DPLO mentioned in Algorithm 1 are all described in [3]. In addition to the modifications to DPLO described in [3], concurrent execution of the ‘parallel do loop’ in routine *optimality-cut* on the Alliant FX/8 requires modifications to eliminate certain COMMON blocks. (See [1, Section 5.7.11].) When we refer to DPLO in the statement of Algorithm 1 we mean a version in which we have made these additional modifications.

3. Performance Results. We have developed a Fortran implementation of Algorithm 1 on the Alliant FX/8. Using this implementation we have performed a portion of the numerical experiment described in [3]. The numerical experiment described in [3] is quite comprehensive consisting of four parts (a), (b), (c) and (d). Since the numerical results in [3] indicate that each part could be used to assess the performance of the implementation adequately, we selected part (b) as our experiment here. Part (b) of the experiment in [3] involves four problem sizes denoted by (i), (ii), (iii) and (iv). Problem sizes (i), (ii), (iii) and (iv) respectively correspond to dimension values $m_1 := 40$, $n_1 := 60$, $m_2 := 10$, $n_2 := 15$; $m_1 := 60$, $n_1 := 88$, $m_2 := 15$, $n_2 := 22$; $m_1 := 80$, $n_1 := 120$, $m_2 := 20$, $n_2 := 30$; and $m_1 := 100$, $n_1 := 148$, $m_2 := 25$, $n_2 := 37$. For each problem size we use a binomial-related multivariate distribution (see [3] for details) F with $K := 100$, $K := 1000$ and $K := 10000$. For each case characterized by problem size and value of K we generate three random instances of problem (1) and run our implementation on these three instances. Any cpu time value we quote henceforth is the average of the three cpu time values observed for the corresponding three random instances of (1). We observe two cpu time values for each case characterized by problem size and value of K : the cpu time for the overall algorithm, and the total cpu time used by routine *optimality-cut*. The latter cpu time is important because when K is ‘large’, the computations in Algorithm 1 is dominated by the computations in routine *optimality-cut* (see [3] for a discussion of this fact). In fact, as described in [3], the parallel implementation of the Van Slyke and Wets Algorithm given in [3] and in Algorithm 1 is motivated by this observation.

Before presenting timing results we mention four options that may be used to compile and link a code on the Alliant FX/8. More details about these options can be found in [1, Chapters 4,5]. These options are global optimizations (also referred to as general or scalar optimizations which generally optimize the code for better performance), associativity (which recognizes certain code forms that perform computations equivalent to intrinsic functions that have been implemented for good performance), vectorization and concurrency.

In Table 1 below we report the cpu time values observed with global optimizations, associativity, vectorization and concurrency used to compile and link the code and all eight processors used to run the resulting executable. These and all other timing results reported in this paper were obtained on the Alliant FX/8 at the Advanced Computing Research Facility of Argonne National Laboratory, Argonne, Illinois. In order to measure the effect of concurrency, we ran the same executable on a single processor. (This and all other single processor runs mentioned in this paper were made using the Alliant command *execute -c1*.) The observed cpu time values are reported in Table 2, while the resulting speed-up (referred to as speed-up 1) values are indicated in Table 3.

problem size	overall cpu time when K is			<i>optimality-cut</i> cpu time when K is		
	100	1000	10000	100	1000	10000
(i)	10.8	42.2	388.4	7.5	39.7	384.8
(ii)	21.1	152.7	1543.5	15.1	143.1	1529.7
(iii)	62.3	345.7	1039.4	43.1	321.2	1029.4
(iv)	80.8	254.6	1981.3	50.9	230.9	1955.4

Table 1. Cpu time (sec.) observed on eight processors with global optimizations, associativity, vectorization and concurrency

problem size	overall cpu time when K is			<i>optimality-cut</i> cpu time when K is		
	100	1000	10000	100	1000	10000
(i)	51.1	264.7	2451.3	47.4	261.8	2447.1
(ii)	101.4	931.2	9809.2	94.3	919.7	9792.8
(iii)	277.8	2162.4	7070.7	254.6	2132.3	7058.5
(iv)	325.9	1407.9	12486.9	289.1	1378.9	12454.9

Table 2. Cpu time (sec.) observed on a single processor with global optimizations, associativity, vectorization and concurrency

problem size	overall speed-up when K is			<i>optimality-cut</i> speed-up when K is		
	100	1000	10000	100	1000	10000
(i)	4.7	6.3	6.3	6.3	6.6	6.4
(ii)	4.8	6.1	6.4	6.2	6.4	6.4
(iii)	4.5	6.3	6.8	5.9	6.6	6.9
(iv)	4.0	5.5	6.3	5.7	6.0	6.4

Table 3. Values of speed-up 1

The speed-up values in Table 3 are satisfactory for an eight processor configuration and indicate that the concurrency capabilities of the Alliant FX/8 are utilized satisfactorily. They also compare well with similarly computed speed-up values quoted in [7] for an algorithm for bound constrained optimization problems.

As mentioned earlier, apart from concurrency, the Alliant FX/8 can enhance performance through global optimizations, associativity and vectorization. In order to see how our implementation utilizes these features, we obtained timing values by repeating part (b) of the experiment in [3], on a single processor with the code compiled and linked as follows: with no global optimizations, no associativity, no vectorization and no concurrency; with global optimizations, but with no associativity, no vectorization and no concurrency; and with global optimizations, associativity and vectorization but with no concurrency. The cpu time values observed are indicated in Tables 4, 5 and 6 respectively.

problem size	overall cpu time when K is			<i>optimality-cut</i> cpu time when K is		
	100	1000	10000	100	1000	10000
(i)	79.1	454.3	4755.0	72.9	449.6	4748.3
(ii)	164.5	1663.5	18733.5	153.1	1643.1	18703.7
(iii)	467.5	3917.6	13743.2	423.8	3860.2	13723.7
(iv)	542.0	2552.6	24023.4	481.0	2507.1	23973.3

Table 4. Cpu time (sec.) observed on a single processor with no global optimizations, no associativity, no vectorization and no concurrency

problem size	overall cpu time when K is			<i>optimality-cut</i> cpu time when K is		
	100	1000	10000	100	1000	10000
(i)	37.8	213.2	2198.1	34.8	210.9	2194.9
(ii)	76.6	760.8	8573.1	71.1	751.3	8559.5
(iii)	215.2	1776.8	6215.0	194.9	1750.9	6205.5
(iv)	245.0	1144.8	10729.1	217.1	1123.7	10705.9

Table 5. Cpu time (sec.) observed on a single processor
with global optimizations, but with no associativity,
no vectorization and no concurrency

problem size	overall cpu time when K is			<i>optimality-cut</i> cpu time when K is		
	100	1000	10000	100	1000	10000
(i)	35.4	182.1	1722.0	32.5	179.8	1718.9
(ii)	68.5	622.6	6558.9	63.3	613.8	6546.5
(iii)	185.4	1397.2	4598.6	166.4	1373.7	4589.2
(iv)	212.8	893.5	7965.2	185.0	871.9	7941.8

Table 6. Cpu time (sec.) observed on a single processor
with global optimizations, associativity and
vectorization, but with no concurrency

Before examining the cpu time values in Tables 4, 5 and 6, let us observe that a speed-up (referred to as speed-up 2) alternative to speed-up 1 above may be defined as follows to measure the utilization of concurrency by the implementation: speed-up 2 is cpu time observed on a single processor with the code compiled and linked with global optimizations, associativity and vectorization but with no concurrency divided by the cpu time observed on eight processors with the code compiled and linked with global optimizations, associativity, vectorization and concurrency. Therefore, speed-up 2 values are computed using cpu time values in Tables 6 and 1 instead of the cpu time values in Tables 2 and 1 used in computing speed-up 1 values. The values of speed-up 2 that result are reported in Table 7 below.

problem size	overall speed-up when K is			<i>optimality_cut</i> speed-up when K is		
	100	1000	10000	100	1000	10000
(i)	3.3	4.3	4.4	4.3	4.5	4.5
(ii)	3.2	4.1	4.2	4.2	4.3	4.3
(iii)	3.0	4.0	4.4	3.9	4.3	4.6
(iv)	2.6	3.5	4.0	3.6	3.8	4.1

Table 7. Values of speed-up 2

The speed-up 2 values in Table 7 compare well with similarly computed values reported in [9] for a quadratic programming algorithm.

Let us now consider the utilization of the vectorization capabilities of the Alliant FX/8 by our implementation. To measure this we computed a speed-up (referred to as speed-up 3) by dividing the cpu time observed with the code compiled and linked with global optimizations only and run on a single processor, by the cpu time for the code compiled and linked with global optimizations, associativity and vectorization and run on a single processor. Speed-up 3 values are thus computed using corresponding values in Tables 5 and 6. We report the values that result in Table 8 below.

problem size	overall speed-up when K is			<i>optimality_cut</i> speed-up when K is		
	100	1000	10000	100	1000	10000
(i)	1.1	1.2	1.3	1.1	1.2	1.3
(ii)	1.1	1.2	1.3	1.1	1.2	1.3
(iii)	1.2	1.3	1.4	1.2	1.3	1.4
(iv)	1.2	1.3	1.4	1.2	1.3	1.4

Table 8. Values of speed-up 3

The speed-up values in Table 8 are not satisfactory. (The possibility of vectorization speeding up computations by a factor of two to four is mentioned in [1, Section 1.2.2].) Since the computation in Algorithm 1 is dominated by work in routine *optimality_cut* (especially for large K), and routine *optimality_cut* essentially involves calls to DPLO, the poor vectorization indicated by speed-up 3 values in Table 8 is due to poor vectorization of DPLO. DPLO is a large package of subroutines and in the runs mentioned in this paper we have not attempted to manually introduce Alliant compiler directives into DPLO for better performance, i.e. only the automatic optimizations (including vectorization) introduced by the Alliant compiler were in effect. The speed-up 3 values in Table 8 suggest that we may be able to improve the performance by manually tuning routines in DPLO for better vectorization.

Let us now examine the effect of global optimizations on the performance of our implementation. A convenient way to do that would be by examining the values of a speed-up (referred to as speed-up 4) obtained by dividing the cpu time on a single processor for the code compiled and linked with no global optimizations, no associativity, no vectorization and no concurrency by the corresponding cpu time on a single processor for the code compiled and linked with global optimizations, but with no associativity, no vectorization and no concurrency. The values of speed-up 4 are indicated in Table 9 below.

problem size	overall speed-up when K is			<i>optimality-cut</i> speed-up when K is		
	100	1000	10000	100	1000	10000
(i)	2.1	2.1	2.2	2.1	2.1	2.2
(ii)	2.1	2.2	2.2	2.2	2.2	2.2
(iii)	2.2	2.2	2.2	2.2	2.2	2.2
(iv)	2.2	2.2	2.2	2.2	2.2	2.2

Table 9. Values of speed-up 4

Speed-up 4 values in Table 9 indicate that the global optimizations can speed-up the performance of the unoptimized code by a factor of about two. Again, since most of the computation takes place within subroutines of DPLO (especially when K is large) this performance improvement occurs in those subroutines.

We end this paper by observing that the above performance results indicate that the implementation satisfactorily utilizes all the major performance improvement features of the Alliant FX/8 except its vectorization capabilities, and that it may be possible to manually tune DPLO to make the implementation utilize vectorization better.

REFERENCES

1. Alliant Computer Systems Corporation, *FX/Fortran Programmer's Handbook*, 1988.
2. K. A. Ariyawansa, *Parallel processing in the solution of two-stage stochastic linear programs with complete recourse*, in Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing, J. J. Dongarra, P. Messina, D. C. Sorensen and R. G. Voigt, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1990, pp. 143–148.
3. K. A. Ariyawansa and D. D. Hudson, *Performance of a benchmark parallel implementation of the Van Slyke and Wets algorithm for two-stage stochastic programs on the Sequent/Balance*, Concurrency—Practice and Experience, 1991 (in print).
4. K. A. Ariyawansa, D. C. Sorensen and R. J-B. Wets, *Parallel schemes to approximate the values and subgradients of the recourse function in certain stochastic programs*, Manuscript, 1987 (in revision).

5. M. A. H. Dempster, *Stochastic Programming*, Academic Press, New York, 1980.
6. R. J. Hanson and K. L. Hiebert, *A sparse linear programming subprogram*, Report SAND81-0297, Sandia National Laboratories, Alburquerque, New Mexico and Livermore, California, 1981.
7. J. J. Moré, *On the performance of algorithms for large-scale bound constrained problems*, Preprint MCS-P140-0290, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1990.
8. R. Van Slyke and R. J.-B. Wets, *L-shaped linear programs with applications to optimal control and stochastic programming*, SIAM J. Appl. Math., 17 (1969), pp. 638–663.
9. S. Wright, *A fast algorithm for equality-constrained quadratic programming on the Alliant FX/8*, Annals of Operations Research, 14 (1988), pp. 225-243.