Parallel Computing 17 (1991) 1247–1255
North-Holland

Parallel Loops – A test suite for parallelizing compilers: Description and example results

Jack Dongarra [a], Mark Furtney [b], Steve Reinhardt [b], and Jerry Russell [b]

[a] *Computer Science Department, University of Tennessee, Knoxville, TN 37996-1301, USA and Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA*
[b] *Cray Research, Inc., 655-F Lone Oak Drive, Eagan, MN 55121, USA*

Abstract

Dongarra, J., M. Furtney, S. Reinhardt and J. Russell, Parallel Loops – A test suite for parallelizing compilers: Description and example results, Parallel Computing 17 (1991) 1247–1255.

Several multiprocessor systems are now commercially available, and advances in compiler technology provide automatic conversion of programs to run on such systems. However, no accepted measure of this parallel compiler ability exists. This paper presents a test suite of subroutines and loops, called *Parallel Loops*, designed to (1) measure the ability of parallelizing compilers to convert code to run in parallel and (2) determine how effectiveley parallel hardware and software work together to achieve high performance across a range of problem sizes. In addition, we present the results of compiling this suite using two commercially available parallelizing Fortran compilers, Cray and Convex.

Keywords. Multiprocessor systems; parallelizing compilers; parallel test suite; results; CRAY; CONVEX.

1. Introduction

Within the past several years, many vendors have produced computer systems with multiple central processing units (CPUs) with shared memory. Alliant, BBN, Convex, Cray Research, DEC, IBM, Intel, and Sequent are examples of machines in this category. While some of these machines can be used most simply as throughput machines, with each program using only one CPU, often a single program will need the extra speed made possible by using multiple CPUs. For multiprocessing to be effective, the compiler must be able to detect the parallelism that exists in the program. Moreover, the software and hardware must be able to exploit that parallelism for the problem size that the user defines.

Various systems exploit parallelism in different ways. For example, Cray Research systems have multiple, segmented functional units (which can be concurrently active), multiple memory ports and chained vector operations for exploiting parallelism on a single CP, and a simple set of hardware and software features for exploiting parallelism on multiple CPUs. However, no commonly accepted measure of these abilities exists. Several tests measure compiler vectorization abilities in different ways, but none of them concentrates on *parallelization* abilities. Other tests measure performance, but generally only for one problem size.

We present a test suite, called *Parallel Loops*, made up of two parts: (A) a set of subroutines and (B) a set of DO-loops. We have used this suite effectively to measure the ability of software to detect parallelism and the ability of hardware and software to work together to exploit this parallelism. The executable code in the subroutines is dominated by DO-loops (often nested three or more levels deep). Run-time performance of the subroutines provides a metric with which we can measure compiler capabilities to recognize and exploit parallelism. Each of the DO-loops is executed with a set of array sizes and trip counts, and each can be used to measure parallel system performance on a wide variety of problem sizes.

The majority of the loops in this test suite were extracted from existing code (from Part A), and about 20 loops were built synthetically. No effort was made to choose loops that cover the space of parallelizable constructs. Although these types of loops could be a useful addition to this set, we have preferred to stay with loops that we find in practice. These loops reflect constructs for which parallelization ranges from easy to challenging to extremely difficult. We

Table 1
36 subroutines (Cray Research Y-MP8 and Autotasking ^{TM} compiling system, CF77 4.0)

Subr. number	Floating Point Ops. (000's)	Vector 1 CPU		Vector 8 CPUs		Speedup Scalar (8)/ Scalar (1)	Speedup Vector (1)/ Scalar (1)	Speedup Vector (8)/ Vector (1)	Speedup Vector (8)/ Scalar (1)
		MFLOPS	% Peak	MFLOPS	% Peak				
1	7303	32.7	10	36.3	1	1.28	2.50	1.11	2.77
2	5660	56.8	18	52.3	2	0.95	3.66	0.92	3.37
3	126307	151.3	48	573.9	22	5.56	9.45	3.79	35.85
4	270513	168.4	53	1071.2	42	6.58	10.59	6.36	67.39
5	17668	105.0	33	147.2	5	1.74	5.38	1.40	7.54
6	173757	155.9	49	859.7	34	5.10	8.50	5.52	46.90
7	291647	180.9	57	1024.8	40	5.95	7.87	5.66	44.57
8	24518	130.1	41	124.6	4	0.95	9.06	0.96	8.68
9	11256	77.5	24	150.7	6	5.75	6.91	1.94	13.43
10	10452	70.1	22	249.4	9	3.67	6.48	3.55	23.02
11	28981	192.1	61	184.4	7	0.92	10.90	0.96	10.46
12	101543	103.9	33	563.7	22	5.65	11.02	5.42	59.75
13	10304	68.6	21	68.1	2	1.09	5.64	0.99	5.60
14	221141	265.0	84	1061.1	42	5.59	5.98	4.00	23.93
15	12815	45.2	14	60.9	2	1.55	3.78	1.35	5.09
16	33089	203.3	64	195.7	7	0.90	5.29	0.96	5.09
17	243753	204.4	65	1114.3	44	6.21	9.74	5.45	53.12
18	10121	66.6	21	407.8	16	6.67	5.45	6.12	33.38
19	190660	149.4	47	800.3	31	5.63	6.47	5.36	34.64
20	6913	55.4	17	99.8	3	2.73	3.61	1.80	6.51
21	23763	146.0	46	142.8	5	0.98	7.67	0.98	7.50
22	130846	96.2	30	650.1	25	6.44	7.20	6.76	48.69
23	4904	28.2	8	142.3	5	1.68	1.76	5.05	8.89
24	196496	120.6	38	782.1	31	7.08	4.09	6.49	26.50
25	22563	130.1	41	113.1	4	1.05	6.14	0.87	5.34
26	12913	92.7	29	90.3	3	0.96	5.11	0.97	4.97
27	1323	10.2	3	25.9	1	2.42	2.26	2.53	5.72
28	13717	97.1	31	248.7	9	1.50	2.93	2.56	7.51
29	180614	134.5	42	231.7	9	1.93	6.31	1.72	10.87
30	334903	232.7	74	1526.7	60	6.00	8.86	6.56	58.13
31	136813	230.2	73	635.4	25	3.12	9.08	2.76	25.05
32	122346	134.8	43	522.2	20	5.13	7.55	3.87	29.27
33	2528	13.7	4	13.8	1	0.96	1.00	1.01	1.01
34	27661	157.5	50	279.4	11	6.54	8.11	1.77	14.38
35	36070	162.6	51	205.7	8	1.27	7.84	1.27	9.92
36	16585	57.9	18	373.3	14	6.42	3.94	6.44	25.38
Aggregate	3062461635	149.3	47	479.7	19	3.83	7.43	3.21	23.87
Maximum		265.0	84	1526.7	60	7.08	11.02	6.76	67.39

Table 2
64 parallel loops (NSIZE1 = 1000, NSIZE2 = 100, NSIZE3 = 10)
(Cray Research Y-MP/8 and Autotasking ^{TM}Compiling System, CF77 4.0)

DO-loop number	DO-loop label	Floating Point operations	Vector 1 CPU		Vector 8 CPUs		Speedup Scalar (8)/ Scalar (1)	Speedup Vector (1)/ Scalar (1)	Speedup Vector (8)/ Vector (1)	Speedup Vector (8)/ Scalar (1)
			MFLOPS	% Peak	MFLOPS	% Peak				
1	1000	100000	46.9	14	46.7	1	1.00	9.02	0.99	8.97
2	1100	2000000000	312.3	99	2477.7	98	7.93	1.00	7.93	7.93
3	1200	300000	23.1	7	23.1	1	1.00	1.00	1.00	1.00
4	1300	100000	6.9	2	52.3	2	7.95	8.15	7.57	61.71
5	1400	9904950	199.9	63	200.0	7	1.00	10.71	1.00	10.71
6	1500	791208	210.5	67	1601.5	63	7.51	12.32	7.61	93.75
7	1600	395604	116.4	37	899.0	35	5.89	10.13	7.73	78.28
8	1700	100000	113.5	36	793.1	31	7.55	8.98	6.99	62.78
9	1800	200000	232.9	74	1589.6	63	7.35	11.25	6.83	76.80
10	1900	200000	231.7	74	1594.1	63	7.46	11.11	6.88	76.45
11	2000	200000	140.6	44	953.7	38	7.38	8.81	6.78	59.72
12	2100	200000	173.1	55	1076.1	42	6.97	10.30	6.22	64.02
13	2200	200000	119.4	38	896.9	35	6.71	7.28	7.51	54.70
14	2300	400000	126.9	40	961.0	38	8.05	13.48	7.57	102.07
15	2400	11300000	202.8	64	1545.9	61	7.55	12.12	7.62	92.41
16	2500	300000	46.6	14	355.5	14	4.90	5.61	7.63	42.80
17	2600	400000	143.1	45	689.2	27	6.13	7.67	4.82	36.95
18	2700	1500000	186.8	59	1383.8	55	6.43	8.82	7.41	65.34
19	2800	1400000	177.5	56	177.5	7	0.99	11.36	1.00	11.36
20	2900	11400000	172.6	55	522.5	20	2.63	8.40	3.03	25.43
21	3000	100000	23.5	7	208.1	8	7.36	9.61	8.84	84.94
22	3100	100000	20.0	6	148.8	5	7.68	6.96	7.44	51.81
23	3200	100000	20.0	6	148.0	5	7.68	6.96	7.41	51.53
24	3300	100000	18.8	5	132.9	5	7.39	5.86	7.08	41.48
25	3400	80000000	306.2	97	2355.1	94	7.69	1.00	7.69	7.69
26	3500	100000	15.0	4	111.2	4	7.68	6.40	7.41	47.38
27	3600	100000	21.5	6	160.0	6	7.37	9.29	7.44	69.13
28	3700	250000	10.0	3	80.8	3	3.52	3.24	8.09	26.19
29	3800	450000	32.1	10	252.8	10	7.30	8.00	7.88	63.04
30	3900	250000	4.2	1	4.0	1	1.00	1.00	0.96	0.96
31	4000	199800	91.5	29	91.5	3	1.00	15.94	1.00	15.94
32	4100	199800	122.6	39	122.6	4	0.99	5.51	1.00	5.50
33	4200	200800	8.8	2	8.4	1	0.94	1.67	0.95	1.58
34	4300	101000	110.3	35	110.2	4	1.00	7.72	1.00	7.71
35	4400	100000	93.0	29	547.6	21	6.00	17.31	5.89	101.90
36	4500	150000	142.6	45	942.5	37	7.25	16.27	6.61	107.52
37	4600	50000	91.3	29	603.9	24	7.10	14.84	6.61	98.09
38	4700	500400	24.9	7	25.6	1	0.99	5.03	1.03	5.16
39	4800	500000	43.2	13	24.5	1	1.00	1.71	0.57	0.97
40	4900	39000	31.8	10	31.9	1	0.79	1.35	1.00	1.35
41	5000	158400000	248.5	79	1851.2	73	6.49	8.60	7.45	64.10
42	5100	900000	215.9	68	1650.1	65	6.91	9.06	7.64	69.28
43	5200	601300	287.5	91	288.3	11	1.00	7.24	1.00	7.26
44	5300	80060000	213.8	68	1616.5	64	6.00	6.54	7.56	49.46
45	5400	160040000	255.9	81	2083.1	83	6.62	8.40	8.14	68.38
46	5500	200000	192.5	61	1983.3	79	19.25	15.62	10.30	160.91
47	5600	40000000	187.3	59	187.2	7	1.00	16.52	1.00	16.50
48	5700	3000	135.2	43	135.2	5	0.99	9.63	1.00	9.63
49	5800	4000000	92.1	29	669.0	26	6.96	8.02	7.27	58.25
50	5900	70000000	198.6	63	186.3	7	1.00	9.24	0.94	8.67
51	6000	200100	234.9	75	234.0	9	0.99	9.41	1.00	9.38
52	6100	4351644	213.6	68	1547.0	61	6.36	9.59	7.24	69.48
53	6200	891000	132.5	42	132.5	5	1.00	11.02	1.00	11.02
54	6300	1600000	20.1	6	20.2	1	0.99	1.10	1.01	1.11

Table 2 (continued)

DO-loop number	DO-loop label	Floating point operations	Vector 1 CPU		Vector 8 CPUs		Speedup Scalar (8)/ Scalar (1)	Speedup Vector (1)/ Scalar (1)	Speedup Vector (8)/ Vector (1)	Speedup Vector (8)/ Scalar (1)
			MFLOPS	% Peak	MFLOPS	% Peak				
55	6400	1000000	205.6	65	1663.2	66	6.64	10.56	8.09	85.38
56	6500	5200000	132.8	42	389.0	15	2.50	11.58	2.93	33.94
57	6600	1500000	225.6	72	1661.1	66	5.59	9.10	7.36	66.99
58	6700	160000	79.9	25	79.8	3	1.00	8.42	1.00	8.41
59	6800	20580000	113.8	36	113.8	4	1.00	14.38	1.00	14.38
60	6900	6000000	128.6	41	1243.0	49	11.82	12.23	9.67	118.22
61	7000	7001000	30.7	9	234.8	9	7.70	10.65	7.65	81.53
62	7100	30000000	316.7	99	319.7	12	1.00	9.29	1.01	9.38
63	7200	31000000	304.2	97	2095.5	83	7.48	6.35	6.89	43.75
64	7300	3200000	201.5	64	573.1	22	2.50	9.94	2.84	28.27
Aggregate		2749870606	270.0	86	1176.4	46	2.47	3.76	4.36	16.40
Maximum			316.7	99	2477.7	98	19.25	17.31	10.30	160.91

Table 3
36 subroutines (Convex C-2, 1 CPU Vs. 4 CPUs and compiling system, cf 4.1)

Subroutine Number	Floating Point Ops. (000's)	Convex 4		Convex 1		Convex 4/ Convex 1
		MFLOPS	% Peak	MFLOPS	% Peak	
1	7303	3.9	1	2.2	4	1.77
2	5660	6.6	3	6.0	12	1.10
3	126307	33.0	16	10.1	20	3.27
4	270513	36.0	18	8.6	17	4.19
5	17668	44.0	22	15.9	31	2.77
6	173757	44.6	22	11.7	23	3.81
7	291647	65.4	32	17.8	35	3.67
8	24518	40.1	20	21.4	42	1.87
9	11256	3.5	1	9.2	18	0.38
10	10452	9.0	4	1.6	3	5.62
11	28981	19.0	9	6.7	13	2.84
12	101543	35.1	17	10.0	20	3.51
13	10304	8.6	4	6.9	13	1.25
14	221141	40.7	20	8.3	16	4.90
15	12815	10.0	5	5.0	10	2.00
16	33089	5.3	2	4.5	9	1.18
17	243753	80.9	40	19.1	38	4.24
18	10121	17.3	8	4.9	9	3.53
19	190660	65.6	32	21.6	43	3.04
20	6913	12.6	6	4.1	8	3.07
21	23763	31.9	15	18.7	37	1.71
22	130846	27.6	13	6.6	13	4.18
23	4904	5.1	2	3.6	7	1.42
24	196496	60.8	30	15.4	30	3.95
25	22563	29.0	14	12.7	25	2.28
26	12913	7.6	3	9.8	19	0.78
27	1323	3.2	1	1.3	2	2.46
28	13717	25.6	12	10.3	20	2.49
29	180614	32.9	16	11.8	23	2.79
30	334903	68.5	34	16.4	32	4.18
31	136813	38.1	19	17.3	34	2.20
32	122346	60.6	30	15.4	30	3.94
33	2528	4.2	2	3.7	7	1.14
34	27661	55.7	27	19.0	38	2.93
35	36070	28.9	14	11.8	23	2.45
36	16585	[a]		[a]		
Aggregate	3045876	35.9	17	11.5	23	3.12
Geometric mean		20.7		8.6		

[a] The timing for this loop was too inaccurate to give reliable performance numbers.

Table 4
64 parallel loops (NSIZE1 = 1000, NSIZE2 = 100, NSIZE3 = 10) Convex C-21 CPU vs, 4 CPUs compiling system, fc 4.1

DO-loop number	DO-loop label	Floating Point Ops. (000's)	Convex 4		Convex 1		Convex 4/ Convex 1
			MFLOPS	% Peak	MFLOPS	% Peaks	
1	1000	100	4.4	2	1.2	2	3.67
2	1100	2000000	95.2	47	11.3	22	8.42
3	1200	300	2.4	1	2.4	4	1.00
4	1300	100	2.0	1	0.5	1	4.00
5	1400	9904	60.8	30	20.4	40	2.98
6	1500	791	59.7	29	20.4	40	2.93
7	1600	395	30.6	15	8.8	17	3.48
8	1700	100	26.9	13	7.8	15	3.45
9	1800	200	56.2	28	15.8	31	3.56
10	1900	200	51.9	25	15.7	31	3.31
11	2000	200	34.8	17	9.4	18	3.70
12	2100	200	42.9	21	11.8	23	3.64
13	2200	200	14.1	7	5.9	11	2.39
14	2300	400	3.2	1	3.2	6	1.00
15	2400	11300	90.9	45	23.3	46	3.90
16	2500	300	18.0	9	5.4	10	3.33
17	2600	400	52.4	26	15.2	30	3.45
18	2700	1500	81.2	40	21.2	42	3.83
19	2800	1400	77.3	38	22.9	45	3.38
20	2900	13300	81.8	40	22.8	45	3.59
21	3000	100	20.3	10	5.9	11	3.44
22	3100	100	18.0	9	4.8	9	3.75
23	3200	100	17.7	8	4.6	9	3.85
24	3300	100	17.9	8	4.7	9	3.81
25	3400	80000	107.1	53	35.9	71	2.98
26	3500	100	7.4	3	1.8	3	4.11
27	3600	100	8.9	4	2.5	5	3.56
28	3700	250	2.6	1	0.8	1	3.25
29	3800	450	15.6	7	3.9	7	4.00
30	3900	250	2.7	1	0.7	1	3.86
31	4000	199	19.6	9	7.9	15	2.48
32	4100	199	10.4	5	8.0	16	1.30
33	4200	200	1.9		1.7	3	1.12
34	4300	101	20.8	10	7.9	15	2.63
35	4400	100	25.5	12	7.9	15	3.23
36	4500	150	42.6	21	11.8	23	3.61
37	4600	50	28.6	14	8.0	16	3.58
38	4700	500	12.9	6	3.6	7	3.58
39	4800	500	3.4	1	3.3	6	1.03
40	4900	39	11.6	5	11.1	22	1.05
41	5000	158400	109.0	54	24.9	49	4.38
42	5100	900	90.4	45	23.2	46	3.90
43	5200	601	143.4	71	41.3	82	3.47
44	5300	80060	90.3	45	23.0	46	3.93
45	5400	160040	173.3	86	42.7	85	4.06
46	5500	200	44.8	22	15.7	31	2.85
47	5600	40000	64.2	32	18.3	36	3.51
48	5700	3	8.6	4	11.0	22	0.78
49	5800	4000	34.5	17	10.1	20	3.42
50	5900	70000	36.0	18	21.2	42	1.70
51	6000	200	38.8	19	17.5	35	2.22
52	6100	4351	77.4	38	21.2	42	3.65
53	6200	891	55.8	27	14.9	29	3.74

Table 4 (continued)

DO-loop number	DO-loop label	Floating Point Ops. (000's)	Convex 4		Convex 1		Convex 4/ Convex 1
			MFLOPS	% Peak	MFLOPS	% Peaks	
54	6300	1600	28.1	14	9.3	18	3.02
55	6400	1000	64.0	32	26.8	53	2.39
56	6500	5200	15.7	7	15.3	30	1.03
57	6600	1500	78.8	39	27.1	54	2.91
58	6700	160	15.0	7	7.1	14	2.11
59	6800	20580	57.1	28	18.6	37	3.07
60	6900	6000	50.7	25	17.0	34	2.98
61	7000	7001	72.8	36	73.7	147	0.99
62	7100	30000	25.5	12	11.7	23	2.18
63	7200	31000	135.6	67	38.7	77	3.50
64	7300	4500	124.3	62	31.2	62	3.98
Aggregate		2753070	86.7	43	13.2	26	6.58
Maximum			173.3	86	73.7	147	8.42
Geometric Mean			27.7		9.7		

have collected the results from compiling and executing these loops using commercially available, parallelizing Fortran compilers on a variety of supercomputers, mini-supercomputers, and mainframes.

This paper is organized as follows. Section 2 discusses our motivation for developing the test suite. Section 3 categorizes the collection of loops used in the test. Section 4 describes the methodology used to perform the test. Section 5 explains how the results were scored, and Section 6 presents the results of our testing. Section 7 discusses the value of this test suite and presents suggestions for future research.

2. Motivation

Over the past five years, several vendors have introduced compilers that have the ability to automatically parallelize Fortran programs. Naturally, the designers of these parallel compilers would like to know where to direct their next development efforts to have the most positive effect. (This is the constant challenge of trying to define 'typical' applications.) Moreover, they are interested in measuring the compiling system in a way that will translate into expected improvements for customer codes.

Unfortunately, the traditional collections used to test performance do not satisfactorily measure the concurrency abilities of hardware and software systems in handling such applications:

1. The Livermore Fortran kernels [1], for example, provide a proven method for deriving information about vectorization from run-time data. However, the loops are simple enough and iteration counts are small enough that converting them to run in parallel is sometimes less efficient than running them on a single CPU (at least for machines that have hardware with significant ability to exploit parallelism within a CPU, e.g. vector registers).
2. The Vector Loops by Levine et al. [2] measure the ability of a compiler to vectorize Fortran constructs.
3. The LINPACK Benchmark [3] measures both serial and parallel performance and acknowledges that different problem sizes are appropriate for machines of different power, hence the

100 × 100 and 1000 × 1000 problem sizes. However, we wish to measure a wider variety of loop constructs, with more emphasis on performance and parallel speedup.

4. The Perfect Benchmarks [4] measures the performance of entire programs. However, the problem sizes of the Perfect Benchmarks have been scaled down so that systems from many vendors (with a wide variation in peak and sustained performance) can run the tests in a reasonable time. This bias in problem size strongly affects the decision to use uniprocessing or multiprocessing to get the best performance for individual DO-loops and loop nests. Additionally, the Perfect Benchmarks do not illustrate the effects that different problem sizes will have, and this information is often important to different users.

3. The parallel test suite

3.1. Selection

We have collected subroutines and loops that contain significant parallelism and are representative of real applications. As stated before, a few of the constructs are synthetic, but for the most part they are drawn from existing applications. A variety of scientific disciplines are represented; no single discipline dominates the sample. No alterations to the source code have been performed, except to make 'cleanup' changes (that is, adding uniform indenting and statement labeling, and changing variable names). No manual optimization has been carried out, and no directives have been added. This test is meant to be run *as is*, although vendors will have to supply their own wall-clock timing function which is called throughout the source. We have chosen two parts to the test, 'Whole Subroutines' and 'Loop Nests', as the best way to represent real applications.

3.2. Whole subroutines

Part A of the test suite consists of whole subroutines, because these are what the end user most often encounters. The suite has a total of 36 subroutines, 36 driver routines, 3 utility routines, and a main routine. The 36 subroutines contain over 750 DO-loops in about 6,500 lines of Fortran. The routines have significant nesting levels: over half are doubly nested, and more than a third are nested three or more deep. Iteration counts have been adjusted so that most of the routines take about the same amount of time on a CRAY Y-MP/8 system. The routines conform to the ANSI Fortran 77 standard.

The focus of Part A is principally on the ability of the compiler to convert code to run concurrently. The larger pieces of code involved may give sophisticated compilers a better opportunity to schedule parallelism as appropriate for a given machine (since parallelism may be exploitable with a larger granularity). However, whole subroutines often make it harder for compilers to find exploitable parallelism.

3.3. Loop nests

Part B of the test suite consists of 64 individual loop nests, most of which were chosen from the subroutines in Part A. The amount of work done in each loop nest varies. Eleven cases with different iteration counts are executed per DO-loop to provide appropriate coverage of the ways these loops are used in many applications.

The focus of Part B is on the ability of the hardware and software to work in unison to provide good performance across a range of problem sizes. Because the loops are easier to analyze than the subroutines, we expect more parallelism to be found and exploited. Difficulty may arise, however, because the granularity of parallelism (for a particular problem size) may limit the efficiency, though this should be less of a difficulty as the problem size grows.

4. Testing methodology

The *Parallel Loops* test suite is modeled after the Livermore Fortran kernels in several respects. First, it is intended to be run unaltered except for the timing routines; that is, no code changes or directives are to be included. Second, the result arrays of each subroutine are summed, and the sum is compared to known correct answers. Third, the speed of each routine is calculated as the number of floating-point operations (as measured by the CRAY-Y/MP hardware performance monitor [5], and included in the source code) divided by the wall-clock time. The programs are self-contained and require no input data. Multiple executions of the test are required to collect the data necessary for the speedup calculations (once with one CPU, and once more for each multi-CPU configuration to be tested).

5. Loop scoring

Vendors were mailed a magnetic tape containing the *Parallel Loops* collection. They were asked to compile the loops without making any changes, using only compiler options for automatic parallelization. Thus, the use of compiler directives or interactive compilation features to gain additional parallelizations was not tested.

Vendors returned their output to us after compiling and running the suite. The output was further refined and is printed in the appendix of this report. The objective of this test suite has been to provide a measure of system performance on these loops; both raw computational speeds and speedups for several problem sizes are reported and made available for comparison. Users of this report are urged to consider these results carefully when making comparisons, and in particular to be careful when comparing speedups – which, after all, are only ratios and are difficult to compare effectively.

6. Interpretation of the results

Two types of information were gathered: the raw computation rates (for 1 and N_{max} CPUs) and the speedup ratios. Note that 'Aggregate' and 'Maximum' columns of data are also supplied, where 'Aggregate' is defined as the total operation count divided by the total time. The raw computation rates are an effective measure of how well a particular system (hardware and software) performs with a particular problem type. Included with these raw computation rates is the percentage (1-99) of peak performance achieved. The speedup ratios compare elapsed (wall-clock) times of serial, vector, parallel, and parallel-vector execution. These ratios represent the ability of software and hardware to work together to exploit the parallelism represented in the subroutines and DO-loops.

For each vendor, we include several tables in the appendix. *Tables 1* and *3* show the results of Part A, the subroutine portion, of the *Parallel Loops*. The raw computation rates may be useful for a variety of comparisons (for example, comparing Cray's 4.0 compiler release with the older 3.0 release). *Tables 2* and *4* show the full results for the largest of the 11 iteration count sets of Part B (1000 × 100 × 10).

7. Conclusions

We have developed a test suite, called *Parallel Loops*, to serve as a metric of parallel compiler and hardware performance. In doing so, we have tried to choose a set of routines that will test the strength of a computer system (compiler, run-time system, and hardware) in a variety of disciplines. Our initial goal has been twofold:
(1) to compare the ability of different Fortran compilers to automatically parallelize various loops, and
(2) to measure the parallel performance of full systems (hardware and software) on real problems of varying sizes.

A copy of the source code used in the test is available from *netlib* at Oak Ridge National Laboratory. To receive a copy of the code, send electronic mail to netlib@ ornl.gov. In the mail message, type

send parallel from benchmark

We intend to update and expand the results presented here. In particular, we plan to develop a check to verify the correctness of the compiler-generated code.

We may also add new loops. We believe the *Parallel Loops* are a good measure of how well compilers and hardware work together to run common loop constructs in parallel. However, the test is definitely focused on the types of parallelism that can be detected by today's compilers. For example, the whole test is defined to measure parallelism that occurs within the bounds of a single subroutine. Today interprocedural analysis is a matter of significant research interest, but few existing production compilers have the ability to detect and exploit cross-subprogram parallelism. In the future this ability may become common, in which case this test should change to reflect the new technology.

We solicit the aid of outside parties in reviewing our routines, deciding whether they constitute a representative set for numerically intensive disciplines, disseminating the routines to interested vendors, and moderating the results on a continuing basis.

References

[1] F.H. McMahon, The Livermore Fortran Kernels: A computer test of numerical performance range, Lawrence Livermore National Laboratory Report UCRL-53745, October 1986.
[2] D. Levine, D. Callahan and J. Dongarra, A comparative study of automatic vectorizing compilers, Mathematics and Computer Science Division Preprint MCS-P218-0391, Argonne National Laboratory, March 1991.
[3] J.J. Dongarra, Performance of various computers using standard linear equations software in a Fortran environment, University of Tennessee Report CS-89-80, 1990.
[4] M. Berry et al., The perfect club benchmarks: Effective performance evaluation of supercomputers, *Internat. Supercomputer Applications* 3(3) (Fall 1989) 5-40.
[5] Cray hardware reference manual.