STOTT 3 G G I I g IG [1] e D STSTg D TDY [1] e D ST G SOTTSFG AT e D

James M. Conrad[1]
Dennis Bahler[2] and James Bowen[3]
Box 8206, North Carolina State University
Raleigh, NC 27695-8206

Abstract

Constraint satisfaction problems (CSPs) are ubiquitous in artificial intelligence; versions arise in areas such as vision, design, Boolean satisfiability, cryptarithmetic, and database retrieval. Most researchers have solved CSPs on sequential computers; relatively few have addressed the use of parallel computers for these problems. Among those who have investigated parallel approaches, several authors have solved CSPs using parallel tree search algorithms, while others have pre-processed constraint networks using parallel consistency algorithms. No one, however, has measured the specific work performed by the individual processors using arc consistency techniques to pre-process a constraint network. In this paper we introduce two Static Parallel Arc Consistency algorithms (SPAC-1 and SPAC-2), which ensure arc consistency of a finite domain binary constraint network, and which are designed for any general-purpose parallel processing computer. Through simulation, we measure work performed by each processor and compare it with work performed by existing sequential and parallel algorithms. Results show that our parallel arc consistency algorithm can be used to pre-process a constraint network with good speedup and utilization.

1 Introduction

Mackworth [11] describes *Constraint Satisfaction Problems* (CSPs) as "those in which one has a set of variables each to be instantiated in an associated domain and a set of Boolean constraints limiting the set of allowed values for speci ed subsets of the variables . enerally, straightforward backtracking algorithms are inade uate for solving a large constraint network because they exhibit an excessive amount of thrashing. ne approach to avoiding thrashing (but not the only one [9, 6, 8]) is to pre-process the problem by eliminating variable assignments which can never result in a solution. his can be done by using consistency algorithms to pre-process a network of constraints before the tree search. lgorithms have been described to achieve node, arc, and path consistency [11, 1].

lthough much has been said about solving CSPs on se uential computers, relatively little has been done on solving these problems on parallel processing computers. n fact, some researchers [10, 19] contend that constraint satisfaction algorithms are inherently se uential, and that using parallel processing computers cannot signi cantly improve the worst case performance. thers, however, believe that *a era e case* execution time can be reduced through parallelism. Several authors have solved CSPs using parallel tree search algorithms similar to forward-checking [16, 15] or depth- rst search with back umping [5, 2]. Still others have pre-processed constraint networks using parallel consistency algorithms [17, 7].

No one, however, has measured the speci c work performed by the individual processors using arc consistency techni ues to pre-process a constraint network. n this paper we introduce two new Static Parallel rc Consistency algorithms designed for any general-purpose parallel

t o o g g (o @ s 36h s s)
[2] t o o S (b@ s s)
[3] t o o S (j bow @ s s)

processing computer and, through simulation, measure work performed by each processor. Results show that our parallel arc consistency algorithm can be used to pre-process a constraint network with good results.

2 A onsist nc G orit s

e examine one class of algorithms for pre-processing constraint satisfaction problems, called consistency algorithms [11]. hese algorithms remove from variable domains values which could never participate in a solution of the constraint satisfaction problem.

o describe consistency algorithms, we refer to the network model of a CSP. e work with binary constraints, where the constraint between variables and represents an edge, and each edge consists of two directed arcs (,) and (,).

n arc (,) is *arc consistent* if, for each value in the domain D_i or , there is at least one value in D_j that satis es the constraint of the arc. rcs can be made consistent by performing the operation $D_i \leftarrow D_i \cup \{l_{i_x} | (\exists l_{j_y})(l_{j_y} \in D_j) \wedge R^2_{ij}(l_{i_x}, l_{j_y})\}$ for every speci c assignment l_{i_x} given to variable v_i from a domain D_i of possible items, where R^2_{ij} is a binary constraint relation containing tuples of values satisfying the constraint.

n this paper, we measure the size of a problem as follows the total number of variables in a problem is denoted by n; the number of edges (half the number of arcs) of a problem is e; the size of each domain is . e will analyze several algorithms below using these measures to count consistency checks. *consistency check* is a comparison of a possible value of one variable with the possible value of another variable. his measurement is used in Mackworth and Freuder's paper on consistency algorithm performance[12]. e address arc consistency only of constraint networks with nite domain variables and binary constraints. lthough each algorithm will ensure node consistency, we do not analyze node consistency algorithms. Similarly, path consistency is beyond our scope here.

Each variable is attached by constraints to the same number of variables. n particular we discuss the parallel arc consistency algorithm P C-1 developed by Samal [18]. n turn, both Samal's algorithm and our own make reference to Mackworth's se uential arc consistency algorithms [11]; se uential algorithms such as Mackworth's must in general be modi ed to operate efficiently on a parallel computer. e will not address other se uential or parallel arc consistency algorithms in this paper. n addition, u [7] has developed several arc consistency algorithms (which he calls Discrete Relaxation lgorithms) but these, unlike ours, re uire special-purpose hardware.

2.1 Existing Parallel Arc Consistency Algorithms

Nadel [14] has suggested that C-1 appropriately modi ed would be a good parallel algorithm. Samal [18] presented several parallel versions of se uential arc consistency algorithms. Samal assumes his model computer has an in nite number of processors available, and that the computer implements a shared memory scheme. Samal concluded that C-1, which he implemented on a small shared memory parallel processor, would also perform very well on larger parallel processors. Samal's algorithms use a parallel version of Mackworth's Revise, called Previse, shown in Figure 1. n Previse, all consistency checks of a single arc are executed concurrently. f the domain size of each variable of an arc is up to , then up to ^{2} checks are made in one time unit. herefore, up to ^{2} processors are needed and take $\mathbf{O}(1)$ time. herefore, computational complexity is $\mathbf{O}(\ ^2)$, the same as C-1 or C- .

```
boo          o        s (    ( , ))
begin
    h  nge  = F  LSE;
    ||for each l_ix  = 1 to   do
    begin
        suppo t[l_ix]  = F  LSE;
        ||for each l_jy  = 1 to   do
            if R^2_ij(l_ix, l_jy) then suppo t[l_ix]  =   RUE;
        if (¬suppo t[l_ix] ) then
        begin
            D_i  = D_i − {l_ix};
             h  nge  =   RUE;
        end
    end
    return( h  nge);
end
```

Figure 1 Samal's Parallel rc Consistency lgorithm Previse

he parallel version of Mackworth's C-1 is Parallel rc Consistency algorithm #1 (P C-1). P C-1 (Figure 2) rst ensures node consistency in parallel (PNC()). hen, each of the $2e$ arcs of the problem can be checked concurrently. Previse executes up to ^{2} consistency checks, so each iteration of the P C-1 loop re uires **O**(^2e) processors. f only one value is removed from each variable each loop, up to n loops are executed. he total time complexity is therefore **O**(n), but the computational complexity is **O**(^3ne).

```
p o  d      AC-1()
begin
    PNC();
     h  nge  =   RUE;
    while ( h  nge) do
    begin
         h  nge  = F  LSE;
        ||for each     ( ,  ) do
             h  nge  = Previse(    ( ,  )) ∨  h  nge;
    end
end
```

Figure 2 Samal's Parallel rc Consistency lgorithm P C-1

3 Static ara rc onsist nc

algorithm which dynamically assigns tasks would best balance the workload of a parallel processing computer, but the task assignment and recombination overheads may overwhelm any bene ts of load balancing [1]. static approach is desirable only if work can be assigned e uitably. e have therefore developed a static parallel version of C-1.

ur parallel arc consistency algorithms differ from Samal's P C-1 algorithm in that each processor works with one variable and the constraints which contain the variable. For example, processor will work with values l_{i_x} of variable v_i and all constraints connected to .

.1 tatic Parallel Arc Consistency Algorithm 1

he Static Parallel rc Consistency #1 algorithm (SP C-1) and Static Parallel Revise Boolean function #1 (Sprevise-1) are listed in Figure and Figure 4 respectively.

---

p o d AC-1- O ()
begin
    PNC();
    h nge = RUE; *no_solut on* = F LSE
    while(*h nge* ∧ ¬*no_solut on*) do
    begin
        h nge = F LSE;
        ||for variable v_i, *h nge* = *h nge*∧ SP C-1-N DE( );
    end
end

p o d AC-1- O ( )
begin
    for each directed v_i v_j do
        h nge = *h nge*∧ Sprevise-1((,));
    return(*h nge*);
end

Figure ost and Node Components of Static Parallel rc Consistency lgorithm #1

SP C-1 has two components, the host algorithm and the node algorithm. he host algorithm executes in only one processor and assigns work to other processors. he host maintains the *chan e* bit and determines if the arc needs to be rechecked. he host component of SP C-1 starts the node component. Each node of the computer executes the node component. Each node works with one constraint network variable and the arcs which emanate from that variable.

SP C-1 is similar to Mackworth's C-1 algorithm, but there are two differences. First, SP C-1 stops if the domain of a variable is empty, as no solution could ever be found. this case the global ag *no_sol tion* is set by Sprevise-1 to "on, and the arc consistency algorithm will halt because no solution could ever be found. Second, SP C-1 ends only when processors have veri ed local arc consistency. nce each processor has veri ed local arc consistency of the variable it was assigned, it will wait for the host to check consistency of the entire network. f any processor has set the *chan e* bit to "on, all processors will again ensure local arc consistency.

he Sprevise-1 function is similar to Mackworth's Revise function with one difference. he function will set a global ag *no_sol tion* if it nds a variable's domain is empty. his variable *no_sol tion* is used by the SP C-1- S algorithm.

```
boo         o   p    s -1(   ( , ))
begin
    for each l_ix ∈ D_i and (¬suppo t)
    begin
        suppo t  = F  LSE;
        for each l_jy ∈ D_j while (¬suppo t)
            if R^2_ij(l_ix, l_jy) then suppo t  =   RUE;
        if (¬suppo t) then
        begin
            D_i  = D_i − {l_ix};
            if D_i = φ then no_solut on  =   RUE;
             h nge  =   RUE;
        end
    end
    return( h nge);
end
```

Figure 4 Static Parallel Revise Function Sprevise-1

.2 tatic Parallel Arc Consistency Algorithm 2

SP C-1 operates in the same manner as C-1 and P C-1, in that all arcs are checked if a value has been removed from a variable's domain. herefore, SP C-1 still suffers from the inefficiencies of the other algorithms. t o better exploit the potentially distributed nature of C-1 and reduce the amount of unnecessary processing, we have developed another parallel arc consistency algorithm, SP C-2.

he SP C-2 algorithm is different in two ways from SP C-1. First, each variable (node) has its own *chan e* bit and SP C-2 running on processor will only set to "on a *chan e* bit of all processors where (,-) ∈ , the set of edges. herefore, processor does not cause processor to perform work when the results of processor v do not depend on the results of processor . his operation is performed by the Sprevise-2 function. he second difference is the introduction of an *acti e_ rocs* variable. his variable counter is used by each SP C-2 node to signify that the node is starting and stopping local arc consistency. hen a processor starts its local arc consistency, it will increment the counter. hen it completes local arc consistency, it will decrement the counter. hen each node processor running SP C-2-N DE observes that the *acti e_ rocs* variable is zero, it will end execution of SP C-2-N DE. he Sprevise-2 function is similar to the Sprevise-1 function, but there are two differences between the algorithms. First, in C-1 and SP C-1, the *chan e* bit is modi ed by the main arc consistency algorithm. n the SP C-2 algorithm, the *chan e* bits are actually modi ed by Sprevise-2. his means that in SP C-2 processors are noti ed earlier of variable value changes than if only the main arc consistency algorithm modi ed the *chan e* bit. Second, if a value is deleted from a variable's domain, the *chan e* ags of **o** the processors that use that variable are set to "on by Sprevise-2.

ith the SP C-2 algorithm changes are "localized, so only processors which have, an arc connected to a changing variable are informed of the change. his differs from the classic C-1, Samal's P C-1, and the SP C-1 algorithms. n the SP C-2 algorithm, all processors with a variable connected by a constraint to the changed variable will recheck of its arcs. his selective rechecking still prevents processors from checking their arcs, even if no changes to any connected variables are made. Processors which have veri ed local arc consistency, therefore, can immediately start useful processing when they are provided new variable domain data.

. Ad antages and isad antages o PAC 1 and PAC 2

mong the advantages of SP C-1 and SP C-2 over Samal's P C-1 are that SP C-1 and SP C-2 (i) are static and not as ne-grained as P C-1 and Previse; (ii) are easily distributed; (iii) are based on monotonic changes to data (hence a "one-write-many-read memory system can be used); and (iv) can be applied to a distributed memory parallel computer architecture, while Samal's P C-1 algorithm s run on a shared memory computer. t the same time, SP C-1 and SP C-2 have de ciencies, speci cally they (i) have worst case time complexity of $\mathbf{O}({}^3e)$; (ii) are not fault tolerant; and (iii) become less efficient toward the end of the consistency process.

. race es lts

s a rst test of the performance of SP C-1 and SP C-2, we traced their execution alongside that of P C-1 and the se uential C-1 on a simple network of four variables, four binary constraints, and domain size everywhere three. n comparing the various algorithms we make several assumptions. First, we count only consistency checks, not speci c steps in the algorithm. Second, the algorithm runs on any M MD parallel processing computer but does not take into account different memory access times of different computer architectures. hird, the deletion of a value from a variable's domain is available to all processors at the beginning of the next consistency check. Fourth, a "clock cycle is considered the time to make a consistency check, delete a value, and update all *chan e* bits, in that order.

1 AC s s s AC-1

By the end of the 15th clock cycle, SP C-1 removed all inconsistent values. By the end of the 1 th cycle, SP C-2 removed all inconsistent values. n the se uential version of C-1, only two of twelve inconsistent values were removed. De ning speedup as the uotient of se uential time and parallel time, and de ning "time as the number of consistency checks for the se uential algorithm and the number of time slice cycles for the parallel algorithm, SP C-1 speedup is 58 cycles/18 cycles = .2, and SP C-2 speedup is 58 cycles/15 cycles = .9. De ning utilization as the number of useful time slices divided by the product of total time slices and the number of processors, SP C-1 utilization is 6 /72 = 87.5% while SP C-2 utilization is 55/60 ≐ 91.7%. Finally, we de ne the work ratio as the uotient of parallel checks and se uential checks. he work ratio is a measure of how many total consistency checks are performed executing the parallel algorithm compared to the serial algorithm. his parameter shows the extra work, or super uous computation overhead [1], performed by the parallel algorithm. For example, the work ratio of the SP C-1 algorithm is 6 /58 = 1.09, and the work ratio of the SP C-2 algorithm is 55/58 = 0.95. he gure is rarely below 1.0; our results in the next section show the ratio is typically above 1.25.

2 AC s s s AC-1

Figure 5 compares the P C-1 algorithm running on 72 processors with the SP C-1 and SP C-2 algorithms running on four processors. lthough the P C-1 algorithm completes much uicker than SP C-1, SP C-2, or C-1, computer utilization is poor.

	P C-1 (72 procs.)	SP C-1 (4 procs.)	SP C-2 (4 procs.)
Cycles	4	18	15
Cons. Checks	110	6	55
ork Ratio	1.90	1.09	0.95
Utilization	8.2%	87.5%	91.7%
ctual Speedup	14.5	.2	.9
Possible Speedup	72	4	4
% of Possible Speedup	20.1%	80.0%	97.5%

Figure 5 Comparison of P C-1 with SP C-1 and SP C-2.

n fact, to fully implement P C-1, one would need 9000 processors to ensure arc consistency of a fully connected 10 variable constraint network with a domain size of 10 values. bviously, to work on large constraint networks, a way must be found to distribute work to a much smaller number of processors, similar to the way SP C-1 and SP C-2 distribute work.

Samal implemented what he called his P C-1 algorithms on a shared memory computer with only 16 processors. e attempted to balance the workloads by statically assigning arcs of the constraint problem to processors. e assigned the same number of arcs to each processor at the beginning of his computation. Each processor worked with several variables and their variable domains, and several arcs. is implementation, therefore, was actually different than his proposed algorithm.

Samal only measured the total start-to-stop execution time of his arc consistency algorithm on a shared memory parallel processing computer. hat Samal did not do is measure the number of consistency checks of his algorithms.

4 Si u ation su ts

s a second and more extensive comparative test of SP C-1 and SP C-2, a simulator has been developed [, 4] to examine the work performed by these algorithms on a parallel processing computer. Results reported in this section were obtained by executing the C-1 algorithm on a se uential computer, executing the SP C-1 algorithm on our parallel processing simulator, executing the SP C-2 algorithm on our parallel processing simulator, and counting consistency checks on the same set of constraint networks. hile gathering the experimental data, ten random constraint networks were run for each of eighteen (six variable three) solution con- gurations. Each constraint satisfaction network (180 networks in all) was executed using the C-1 algorithm, the SP C-1 algorithm, and the SP C-2 algorithm.

h results of our measurements suggest that our parallel algorithm is uit promising. Figures 6 and 7 show the speedup of our SP C-1 and SP C-2 algorithms over C-1. Simulation of our constraint networks indicate speedup between 51 and 66% of theoretically possible speedup for SP C-1 and between 5 and 72% of possible speedup for SP C-2. he utilization of SP C-1 and SP C-2 is also uite high; we never observed utilization below 70% for SP C-1 and 85% for SP C-2. e observed that the total amount of work performed by our SP C-1 is 28 to 79% higher than C-1 and the total amount of work performed by our SP C-2 is 26 to 88% higher

Figure 6 Speedup of SP C-1 over C-1

than C-1.

5 onc usions

e have reviewed several arc consistency algorithms for se uential and parallel processing computers. e introduced two parallel arc consistency algorithms, SP C-1 and SP C-2. e have described the SP O algorithms and provided examples of the operations of the algorithms. e have simulated the algorithms in a parallel processing environment and compared their performances with that of Mackworth's C-1 algorithm. SP C-1 achieved speedup of 51 to 66% of possible speedup over the C-1 algorithm and SP C-2 achieved speedup of 5 to 72% of possible speedup over the C-1 algorithm.

Some of our future work includes the development of a SP C algorithm based on Mackworth's C- algorithm, simulation of our algorithms on distributed memory machines, and making actual computation trials on a variety of parallel processors, including both local-memory and shared-memory architectures.

f r nc s

[1] Burkhart, elmar and Roland Millen, "Performance-Measurement ools in a Multiprocessor Environment, *ransactions on Com ters* (5), 725-7 7, May 1989.

[2] Collin, eev and Rina Dechter, " Distributed Solution to the Network Consistency Problem, *Proc S S* , 1990.

Figure 7 Speedup of SP C-2 over C-1

[ ] Conrad, James M. and Dharma P. grawal, "Simulation of eneric Multiprocessor Con gu-rations for synchronous lgorithms, submitted to the *nternational Jo rnal of Com ter Sim lation*, special issue on Multiprocessor Networks, 1992.

[4] Conrad, James M. and Dharma P. grawal, "Performance of a synchronous Parallel lgorithm on a eneric Multiprocessor Simulator, *Proc. Pittsb r h Conference on Sim lation an o elin* , Pittsburgh, P , May 1991.

[5] Freuder, Eugene C. and M.J. uinn, "Parallelism in an lgorithm that takes dvantage of Stable Sets of ariables to Solve Constraint Satisfaction Problems, ech. Rept. 85-21, Dept. of Computer Science, University of New ampshire, Durham, N , 1985.

[6] aschnig, J., "Experimental Case Studies of Backtrack vs. altz-type vs. New lgorithms for Satis cing ssignment Problems, *Proc n at Conf of the Cana ian Society for Com tational St ies of ntelli ence*, 1978.

[7] u, Jun, "Parallel lgorithms and rchitectures for ery Fast Search (Ph.D. Dis-sertation), ech. Rept. UUCS- R-88-005, PhD Dissertation, Dept. of Computer Science, University of Utah, Salt Lake City, U , 02 pp., ugust 1989.

[8] aralick, Robert M., and ordon L. Elliott, " ncreasing t ree Search Efficiency for Con-straint Satisfaction Problems, *rti cial ntelli ence* , 26 - 1 , 1980.

[9] Janakiram, irendra ., "Randomized lgorithms for Parallel Backtracking and Branch-and-Bound (Ph.D. Dissertation), Dept. of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 114 pp., 1987.

[10] asif, Simon, " O the Parallel Complexity of Some Constraint Satisfaction Problems, *Proc 5th ational Conf on rti cial ntelli ence* , 49- 5 , 1986.

[11] Mackworth, lan ., "Consistency in Networks of Relations, *rti cial ntelli ence* , 99-118, 1977.

[12] Mackworth, lan . and Eugene C. Freuder, " he Complexity of Some Polynomial Network Consistency lgorithms for Constraint Satisfaction Problems, *rti cial ntelli ence 5*, 65-74, 1985.

[1] Mohr, Roger and homas C. enderson, " rc and Path Consistency Revisited, *rti cial ntelli ence* , 225-2 , 1986.

[14] Nadel, Bernard ., "Constraint Satisfaction lgorithms, *Com tational ntelli ence 5*(4), 188-224, November 1989.

[15] Natara an, .S. and ivek Sarkar, "Processor Scheduling lgorithms for Constraint Satisfaction Search Problems, *Proc nternational Conf on Parallel Processin* , 140-147, ugust 1988.

[16] Natara an, .S., " n Empirical Study of Parallel Search for Constraint Satisfaction Problems, *esearch e ort PC 5* , BM J atson Research Center, orktown eights, N , 9 pp., 1987.

[17] Samal, shok and om enderson, "Parallel Consistent Labeling lgorithms, *nterna tional Jo rnal of Parallel Pro rammin* (5), 41- 64, 1987.

[18] Samal, shok umar, "Parallel Split-Level Relaxation (Ph.D. Dissertation), Dept. of Computer Science, University of Utah, 12 pp., ugust 1988.

[19] Swain, Michael J., "Comments on Samal and enderson Parallel Consistent Labeling lgorithms', *nternational Jo rnal of Parallel Pro rammin* (6), 52 -528, 1988.

Ad Cop o d gs o h I o pos o M hodo og s o I g s s, Ch o , C, O ob 1991 (p g -V g)