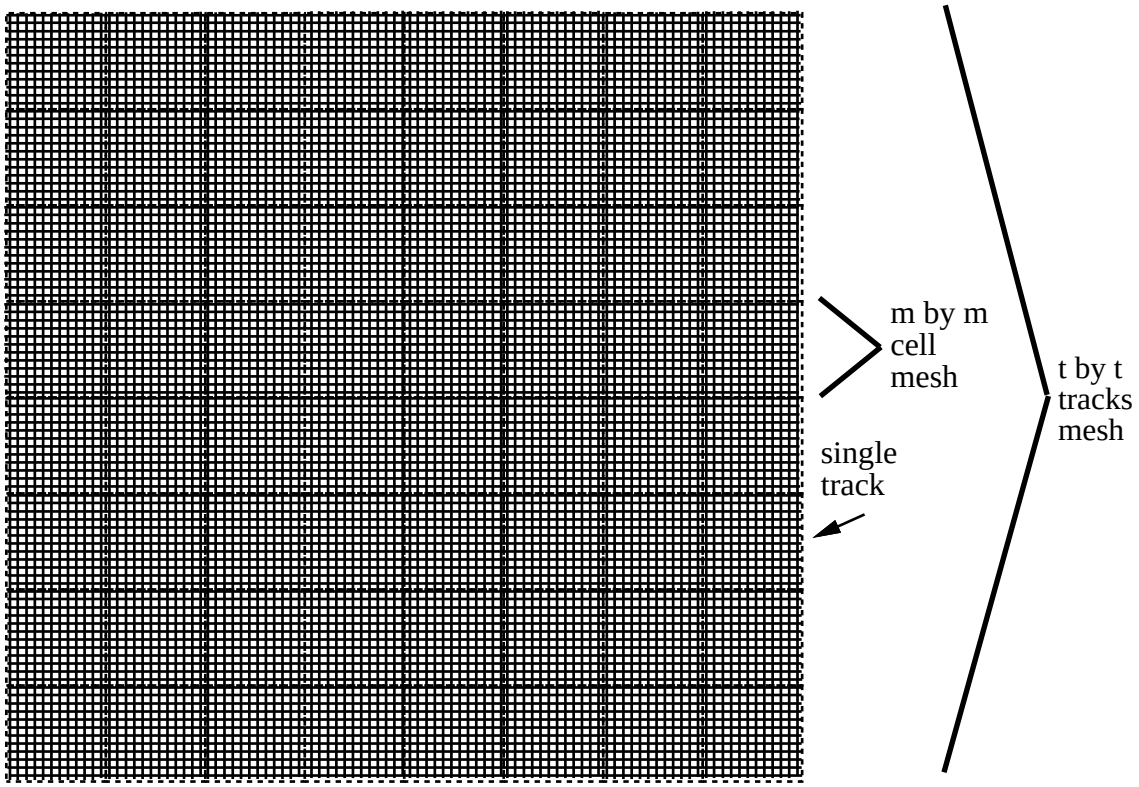

Figure 8.3: Example of Interconnection Layer of our Multi-chip Module Package

layer has tracks in the X- and Y- direction, and the number of tracks, t, is greater than c. Figure 8.3 shows the tracks of an interconnection layer.

Definition 8.5 *A* track mesh *is the collection of all X- and Y-direction tracks of a single interconnection layer. We assume that t tracks can be placed in one dimension, therefore communications tracks are drawn in an $t \times t$ mesh.*

Definition 8.6 *A* channel *is the collection of tracks in one dimension found beneath a row or column of chips. There are $t \div c = m$ tracks in a channel.*

Definition 8.7 *A* cell mesh *is the collection of all X- and Y-direction tracks beneath a single chip. If the communications tracks are drawn in an $t \times t$ mesh, and $t \div c = m$, then each of the c^2 chips has a $m \times m$ mesh below it.*

Definition 8.8 *A* point *is the identifiable position where two tracks cross. If the communications tracks are drawn in an $t \times t$ mesh, there are t^2 points per layer.*

Definition 8.9 *A specific MCM layout has s interconnections which are straight (travel only in the X- or Y-direction) and b interconnections which are bent (travel in both the X- and Y- direction).*

To review the measures of our MCM:

c is the number of chips in the X- or Y- direction of the chip layer.

c^2 is the number of chips of the chip layer.

t is the number of tracks in the X- or Y- direction of the interconnection layer.

t^2 is the number of points of the interconnection layer.

$m = t \div c$ is the number of tracks in the X- or Y- directional channel of the interconnection layer found below each chip.

l is the number of interconnection layers.

s is the number of straight interconnections.

b is the number of bent interconnections.

8.2 Overview of the Problem Solution

The input of our MCM interconnection assignment problem is the number of chips (c^2), the number of tracks per dimension of our interconnect layers (t), the number of interconnect layers (l), and a list of the straight and bent interconnections (source chip to destination chip). The output is the set of tuples identifying the starting point and ending point of an interconnection. Bent interconnections have two sets of tuples to represent the links needed in the X- and Y-direction. The specific format of these tuples is described below.

The execution steps of the problem are:

1. Assign straight interconnections based on the requirements described below.
2. Perform parallel node consistency on the network data using the requirements described below.
3. Perform parallel arc consistency on the network data using the currently assigned links as described below.
4. Perform parallel forward-checking search for the remaining interconnections, assigning a different part of the search tree to each processor, as described below. Impossible assignment data is shared among processors. Run until an end condition is met (see below).

We must first make some assignments in order to successfully use parallel arc consistency. Assigning straight interconnections is fairly straightforward, and can be handled in parallel on a multiprocessor. Once some assignments are made, parallel arc consistency can be used to reduce the domain size used for finding bent interconnections.

The basic assignment rules are:

- When connecting chips in the same row or column, one does not change direction or change levels.
- Links are assigned in an eastward direction if chips are on the same row.
- Links are assigned in a southern direction if chips are in the same column.
- For interconnections which are not in the same row and column, direction may only be changed once. One may assign different directions to different levels, but a single direction may not use two or more levels.
- If chip a is at location (i, j) and chip b is at location (k, l) of the chip level, the following rules for direction apply:

  The chip with the smaller column number is always the origin ($j < l$).

  If $i > k$, the direction assigned is either north-then-east (NE) or east-then-north (EN).

  If $i < k$, the direction assigned is either south-then-east (SE) or east-then-south (ES).

8.3 Data Specifications

Data representation facts:

- We assume square dimensions for all objects. The communications tracks are drawn in an $t \times t$ mesh, where $t \div c = m$, and each of the c^2 chips has a $m \times m$ mesh below it. Therefore, each chip has the same number of connection points below it in each of the l layers.

- All numbering of items begin with the number 0. Therefore, the indices of a specific chip is in the range $[0..c-1, 0..c-1]$. The indices of a specific point in the tracks mesh is in the range $[0..t-1, 0..t-1]$.

- Each start or end point of a link is represented by a tuple, (x, y, z), where (x, y) is a specific point in the $t \times t$ track mesh, and z represents the layer number.

- Interconnections are stored as two or four coordinate tuples. If the interconnection is for chip pairs with the same row or column, then only two tuples are needed. If two links in different directions are needed, the link in the first direction from the origin is stored as the first tuple pair, and the other link is stored as the second tuple pair.

- **Point Domains:** Each point in the tracks mesh is represented by a binary digit. The points in the tracks mesh is declared as an array of characters, allocated during execution of the program (malloc or calloc). The total number of bytes needed is $t \times t \times l$. This variable indicates if the point is occupied ('0') or available ('1').

- **Cell Domains:** A second hierarchical database quickly indicates if one can connect two cells via any available track. The domain size is 32-bits (the maximum number of tracks per cell). Each bit in the variable represents the possibility to travel from one cell to another on a specific track. Although we only allow travel in the west, north, and south directions, we create variables as if all four directions are allowed. This simplifies array references.

  We create a $c \times c \times l$ array of pointers, each representing the originating cell of a link. Each cell requires $2 \times c$ of the 32-bit numbers, so the total number of bytes needed to represent cell to cell links is $(8 \times c^3 \times l) + (sizeof(*\text{ unsigned long int }) \times c^2 \times l)$.

These variables are used to establish node consistency of our search problem. The results using these variables are used for the domains of our interconnections.

- **Interconnection Domains:** The node and arc consistency, as well as search, utilize the chip to chip interconnections as the search variables. These variables take a specific value from the domain of possible tracks as its assignment. The size of the domain is as follows:

  To connect two chips with a bent interconnection, one track in the X-direction and one in the Y-direction is required. Since a bent interconnection can start in either the X- or Y-direction, the second component is of the other direction. Therefore, in one level, $(m \times m) + (m \times m)$ different combinations of tracks are possible. In most cases, however, if a track in the first direction is unavailable, then there is no need to check the m tracks in the other direction. Therefore, in one level, if one just maintains each direction separately, only $4 \times m$ values are needed. A variable assignment is actually be an assignment pair, one component in the X-direction and one in the Y-direction. If we include multiple layers, we need $4 \times m \times l$ values per interconnection.

The total amount of memory needed is $t^2l + 8c^3l + 8c^2l + 4mlb$ bytes.

8.3.1 Constraint Network

The basic structure of our constraint network is:

Variables: Each chip to chip interconnection is a variable in our constraint graph.

Edges: The edges connecting the variables are defined as the constraining relations involving possible conflicting assignments of tracks, that is, where two link assign-

ments may intersect. Consider the following example: If chip(4,4) is connected to chip(5,5), the interconnection from chip(3,3) to chip(6,6) can never cross the interconnection from chip(4,4) to chip(5,5). Therefore, these two interconnection domains are not constrained by each other. The interconnection between chip(3,3) and chip(5,5) may cross the interconnection between chip(4,4) and chip(6,6). Therefore, the two interconnection domains would be connected by an edge.

Domains: The domains represent the possible link assignments of the specific interconnections. As mentioned above, each interconnection assignment requires an X- and Y-direction component, and may occupy different layers.

8.4 Execution Requirements

8.4.1 Initial Link Assignment Requirements

In this step, only straight (non-bending) interconnection links are assigned to specific MCM tracks. The input of this algorithm is the address of two ending cells of each straight interconnection. The outputs of this algorithm are the address of the starting and ending points of each link. Though the links are non-directional, the ending cell in the north or west is called the starting cell and is treated first. For each straight interconnection, the direction of the link is decided by its source and destination cells. Assigning horizontal and vertical links are handled separately. In fact, each row or column of cells and links can be assigned to separate processors of a multicomputer.

We must ensure that each channel has enough tracks so that all straight interconnection links in one direction can be assigned to one layer.

Theorem 8.1 *The number of tracks needed in a channel to assign straight intercon-*

nect links of all chips in a row or column of one layer when the row or column is fully connected is $c^2/4$.

Proof: In a fully connected row or column, the first chip needs an interconnection assignment of $(c-1)$ links. The second chip needs to connect to only $(c-2)$ other chips since is is already connected to the first chip. This connection pattern continues until the $(c-1)$th chip need only connect to the cth chip. In general, chip k need only connect to $(c-k)$ remaining chips. The total number of required links is represented as the series $(c-1)+(c-2)+\cdots+1$, or:

$$\sum_{k=1}^{c-1} k = \frac{c(c-1)}{2} \tag{8.1}$$

The first chip uses $(c-1)$ tracks to connect to the $(c-1)$ other chips. Some interconnection links are very short and do not use the entire length of the track. For example, the link between neighboring chips uses only two points and the section of the track between the two points. Clearly, the rest of the track is available for other chip-to-chip interconnection links. The second chip needs to connect to the other $(c-2)$ chips, but since the first chip did not use the entire length of many tracks, the links can occupy some of the unused points of the track. It actually uses $((c-2)-1)$ new tracks. In general, the number of new tracks needed for chip k is $(c-2k+1)$. No more tracks are needed for chips after chip $c/2$ of the row or column. The number of tracks required is:

$$(c-1)+\sum_{k=2}^{c/2}(c-2k+1) = \frac{c^2}{4} \tag{8.2}$$

□

If not enough tracks are available, another layer is needed to assign straight interconnections.

The procedure for assigning links for straight interconnections is:

1. Interconnections are sorted in decreasing order of their length.
2. The longest interconnection is removed from the list, and temporarily assigned a track. The list is then searched (in decreasing interconnection length) to find if another link can be placed on the same track. If so, this link is placed on the same track and the interconnection is removed from the list. The list is again checked to see if the same track can be assigned to other interconnections.
3. The previous step is repeated until all interconnections are assigned a track.
4. Boundary Link Rule: For those links starting and ending in the boundary cells, they are assigned to those point near the boundary.
5. Compute the number of open points per cell for each track. Permanently assign the outside-most tracks to those link-tracks which have the largest average number of available points.

8.4.2 Parallel Node Consistency Requirements

Each row of cells is assigned to one processor.

First, the point domains are built using the link and track assignment information from the Initial Link Assignment.

Next, the cell domains are built using the point domain information and employing a node consistency algorithm. The cell domains can then be copied to the individual link domains. Note that one cell domain may be copied to several link domains.

8.4.3 Parallel Arc Consistency Requirements

The constraint network has to be divided among the available chips using the Mean Field Annealing graph partitioning algorithm described in section 7.3. Perform the SPAC-3 algorithm based on the following:

- Sprevise-3 updates variable domains when a value is changed.
- Arcs are checked in the order of greatest to least connectivity.
- Since a shared memory scheme is to be used, all domain changes are immediately available.

8.4.4 Parallel Forward Checking Search

It is our plan to employ a Parallel Forward Checking Search algorithm to find a solution to the link assignment problem. We use the sequential algorithm developed by Haralick and Elliot [24], but we assign different parts of the search tree to the processors, and each processor can share the search information with each other.

While assigning links, the following requirements must be met:

- The first track assigned is the innermost track.
- The first direction assigned is north (if possible), then west, then south (if possible).
- Each assignment starts at the boundary of the cell.
- Each possible track on layer 1 is checked first (in the initial X- or Y-direction).

8.5 Miscellaneous Information

Cost information:

- The total of all wire lengths is a constant for a given chip connection scheme (if we do not consider changing levels).
- The cost of a solution can be:

  The number of layers used for the connection scheme.

  The total of all wire lengths, including the lengths of going from one layer to another.

The end of the problem is reached when:

1. At least one solution is found. Since the cost of all possible solutions are nearly equal, there is no need to continue the search past one solution, or
2. The entire search space is traversed and no solution is found.

8.6 Complexity Analysis

Backtracking search is a known NP-complete problem [19]. The fact that it is an exponential problem makes it intractable. We can use arc consistency to help reduce the search space of the MCM problem.

Theorem 8.2 *The complexity of backtrack search to find a solution of assigning links for MCM interconnections (without arc consistency pre-processing) is* $\mathbf{O}(ml)^{2sb}$.

Proof: The constraint satisfaction problem for our MCM problem involves assigning links for straight interconnections and assigning links for bent interconnections. If one examines a straight interconnection, the link is assigned any one of the m tracks available in the row or column channel of any of the l interconnection layers. Since there are ml possible assignments for each of the s interconnections, $(ml)^s$ assignment operations are possible.

Examining bent interconnections, two links must be assigned, one in the X-direction and one in the Y-direction. One can first assign one of the m tracks of the l layers in either the X- or Y-direction. Then, one can use one of the ml tracks in the other direction to complete the bent interconnection. Therefore, each of the b bent interconnections can use one of the $2m^2l^2$ possible tracks assignments. The total complexity of all interconnection assignments is $(ml)^s(2m^2l^2)^b$, or $\mathbf{O}((ml)^{2sb})$.□

Theorem 8.3 *The complexity of AC-3 for our MCM problem is bounded by* $\Omega((ml)^4b^2)$ *and* $\mathbf{O}((ml)^6b^2)$.

Proof: We reported in Section 2.2 that the complexity of AC-3 is bounded by $\Omega(a^2e)$ and $\mathbf{O}(a^3e)$. We have shown in Theorem 8.2 that the domain size $a = 2m^2l^2$. If all of our interconnections constrain each other, we have a fully connected constraint network. We showed in Equation 8.1 that a fully connected set of x points requires $x(x-1)/2$ links. If we have b bent interconnections, the number of edges of our constraint problem is $e = b(b-1)/2$. Therefore, the complexity is $\Omega((ml)^4b^2)$ and $\mathbf{O}((ml)^6b^2)$. □

Theorem 8.4 *The complexity of our MCM problem using parallel arc consistency and parallel search is* $\mathbf{O}(\frac{(ml)^6b^2}{p} + \frac{(\frac{ml}{2})^{2b}}{p})$

First we examine solving the MCM problem using sequential algorithms. If we assign the straight interconnection links first, and since we do not execute the straight interconnection as a search problem, the assignment takes a negligible amount of time. The search time for the bent links is now $(ml)^{2b}$. The complexity of the entire MCM assignment problem, if arc consistency does not remove any values, is:

$$\Omega(AC-3) + \mathbf{O}(search) = \mathbf{O}((ml)^4 b^2 + (ml)^{2b}) \tag{8.3}$$

If arc consistency does remove values, but makes no great impact on the domain size of the problem, the complexity is:

$$\mathbf{O}(AC-3) + \mathbf{O}(search) = \mathbf{O}((ml)^6 b^2 + (ml)^{2b}) \tag{8.4}$$

If we remove $1/2$ of the values of each domain, the complexity is now:

$$\mathbf{O}(AC-3) + \mathbf{O}(search/2) = \mathbf{O}((ml)^6 b^2 + (\frac{ml}{2})^{2b}) \tag{8.5}$$

It follows from Theorems 3.4 and 8.3 that the time complexity of SPAC-3 is $\Omega((ml)^4 b^2/p)$ and $\mathbf{O}((ml)^6 b^2/p)$.

If we effectively assign $1/p$ of the search problem to each of the p processors of a multicomputer, and if we removed $1/2$ of the domain values during parallel arc consistency, our time complexity of the entire problem is:

$$\mathbf{O}(SPAC-3) + \mathbf{O}(par-search/2) = \mathbf{O}(\frac{(ml)^6 b^2}{p} + \frac{(\frac{ml}{2})^{2b}}{p}) \tag{8.6}$$

□

Although we have still not reduced the problem to a polynomial complexity, we have reduced it enough that it may be solvable on a multiprocessor.

8.7 Future Directions of the MCM Problem

We have shown an example of using parallel arc consistency in a constraint satisfaction problem. The interconnection assignment of a MCM package is an important problem, and a successful implementation of the procedures described above could prove useful to the VLSI industry.

We plan to continue this research and implement this system to solve small problems. If we are successful, we will explore solving actual, complex MCM interconnection assignment problems.

Chapter 9

Conclusions

The major emphasis of this work is a thorough examination of parallel arc consistency algorithms. We have defined the realm of the multiprocessor environment and the characteristics of asynchronous algorithms. Constraint satisfaction problems, and the motivation of such problems, have been presented.

We have reviewed several arc consistency algorithms for sequential and parallel computers. Of the sequential algorithms, we examined Mackworth's AC-1 and AC-3 algorithms [44]. Samal's parallel arc consistency algorithms [55] were also examined in detail.

This work introduced three Static Parallel Arc Consistency Algorithms, SPAC-1, SPAC-2, and SPAC-3. The time and computational complexities of these algorithms are proportionally equivalent to existing sequential and parallel algorithms. The strength of this work is that our algorithms are demonstrated with step-by-step traces of parallel execution for sample constraint networks, while the performance is comparable to the other algorithms.

We have simulated the SPAC algorithms in a parallel processing environment and compared their performances with that of Mackworth's AC-1 and AC-3 algorithms. We then compared the simulated speedup with actual measurements taken on three

commercial parallel processors. As a further effectiveness demonstration of SPAC algorithms, we measured their performance with large constraint networks on a large distributed memory computer.

We also proposed the use of these Static Parallel Arc Consistency algorithms to aid in the design of inter-processor link assignment for multi-chip module packaging.

9.1 Review of Results

The results of our measurements on shared memory and hybrid memory multiprocessors suggest that our parallel algorithms provide good speedup. We compute SPAC-1, SPAC-2, and SPAC-3 speedup based on the AC-3 sequential algorithm.

Execution of SPAC-1 on a four-processor computer configuration yields speedup of between 2.74 and 3.07, SPAC-2 yields a range of 2.80 to 3.14, and SPAC-3 yields a range of 2.97 to 3.50. Execution of SPAC-1 on a ten-processor computer configuration yields speedup of between 5.43 and 6.41, SPAC-2 yields a range of 5.44 to 6.64, and SPAC-3 yields a range of 5.35 to 7.40. Execution of SPAC-1 on a twenty-processor computer configuration yields speedup of between 9.28 and 11.68, SPAC-2 yields a range of 10.46 to 11.84, and SPAC-3 yields a range of 10.44 to 12.68.

The results of our measurements on distributed memory multiprocessors suggest that our distributed parallel algorithms provide good speedup. We compute DSPAC-1, DSPAC-2, and DSPAC-3 speedup based on the AC-3 sequential algorithm.

Execution of DSPAC-1 on a four-processor computer configuration yields speedup of between 2.56 and 2.96, DSPAC-2 yields a range of 2.64 to 3.04, and DSPAC-3 yields a range of 2.84 to 3.36. Execution of DSPAC-1 on a ten-processor computer configuration yields speedup of between 4.40 and 5.92, DSPAC-2 yields a range of

4.53 to 6.27, and DSPAC-3 yields a range of 4.60 to 6.82. Execution of DSPAC-1 on a twenty-processor computer configuration yields speedup of between 6.24 and 9.62, DSPAC-2 yields a range of 6.63 to 10.41, and DSPAC-3 yields a range of 7.00 to 11.22.

The above results are based on a comparison of consistency checks, and not actual start-to-stop time. Measuring the time required to ensure the consistency of a constraint network, we found some surprising results. The DSPAC-1 algorithm finished in less time than the DSPAC-2 algorithm, and the DSPAC-2 algorithm finished in less time than the DSPAC-3 algorithm. These results also show a phenomenon called *super-linear speedup*, where the reported speedup is actually greater than the number of processors in the multiprocessor configuration [1]. This observation is also reported by Janakiram, et. al. [31] and Samal [55].

Although Mackworth and Freuder state that AC-3 is a more efficient algorithm, we have found the opposite. AC-3 requires 31% more time to finish as opposed to AC-1. We suggest that AC-3, as well as DSPAC-3, perform poorly in time trials because our earlier results did not take into account queue building and maintenance operations. The complexity analysis also does not take this into account. Samal also suspects this, and shows results which verify our premise [55].

We examined the effect of arc consistency on the execution time of search. Our results show that arc consistency can significantly reduce the search time needed to find at least one solution. Sequential arc consistency reduces the sequential search time to 3.2% of the time needed when arc consistency is not employed. Also based on our parallel search speedup assumption, parallel arc consistency reduces the parallel search time to 3.7% of the time needed when parallel arc consistency is not employed.

9.2 What We Have Learned

The process of conducting research is futile if one does not learn from the process and from the results. Because of this research:

- We have learned that our algorithms can be effectively used to pre-process a constraint network, but we must ensure that each processor has an equal amount of data to process. Although mean field annealing and the Kernighan/Lin algorithms distribute the work more effectively than simple load balancing, the run time required for the partitioning algorithms is not trivial.

- The algorithm with the best time complexity analysis may not be the best-running algorithm. Of course, as we mentioned above, we suspect this is due to the omission of some data operations in the time complexity analysis, especially the queue operations.

- SPAC-1 and SPAC-2 require no memory locking on shared memory machines because of the asynchronous nature of the algorithms. We can use multi-port memories with no danger of error.

- Caching is useful for domain data if cache writes are written-through to main memory, and then copied into the caches of other processors.

- SPAC-3 and DSPAC-3 require memory locking only when adding/deleting an arc to/from the queue.

- Distributed SPAC algorithms could benefit from a "local" bus structure. Each node could be represented by a working processor and a communications processor, working independently. Both could update domain data in memory.

- Performance can be improved with application-specific architectures. This is also proposed by Gu [22]. For example, performance could be improved with more interconnection between processors and shared memory, a sufficient instruction cache, and a large shared memory.

9.3 Future Work

We are pleased with our SPAC-1, SPAC-2, and SPAC-3 parallel algorithms and their performances in a simulation environment and on actual processors.

The constraint networks examined represent small problems. We believe that larger problems can be pre-processed using parallel arc consistency.

Due to the distributed nature of the algorithms, larger problems can be executed on a greater number of processors. Real world problems may require thousands or millions of variables and constraints, and a large number of processors are needed to efficiently process this work.

One specific goal has been to measure the algorithms' work on parallel machines. This included counting consistency checks and measuring time while comparing many single binary values. A more efficient way is to store data in groups of bits and form a large value with the domain information. Consistency checks can then be made with "masks" of variable and relation information (which is stored in the same manner). Relation masks and domain values can be compared using a simple bit-by-bit binary AND-operation, and then OR-operations on the results of the AND. Such a sequence is best programmed in assembly language. If the application warrants, one can even build special-purpose processors to handle these comparisons (similar to [22]).

The majority of time spent solving a constraint satisfaction time is in the actual

assignment of values, that being during the search phase. Although arc consistency is important to reduce the search space, we must either reduce the space to a level such that search becomes tractable, or we must find parallel solutions to the search problem. To achieve this goal, future work can include:

- Investigating the effect of constraint network characteristics on execution time of SPAC algorithms.
- Examining other problems which have n-ary constraints and/or infinite domains.
- Examining a dynamic load-balancing parallel arc consistency scheme.
- Examining the entire constraint satisfaction problem, especially search. We must analyze the inter-dependencies of arc and path consistencies and search.
- Investigating path consistency algorithms.
- Investigating existing parallel search algorithms, and developing new algorithms based on the type of problem.
- Developing specialized architectures with features that can improve execution time of arc consistency, path consistency, and search.

Research is important if it can be used to solve real-world problems. We believe our research, and related constraint satisfaction research, can be used to solve many of the complex problems shown to be intractable using sequential computers.

References

[1] Agrawal, Dharma P., *Advanced Computer Architecture (Tutorial Text)*, IEEE Computer Society Press: Washington, DC, 1986.

[2] Bitner, James R. and Edward M. Reingold, "Backtrack Programming Techniques," *Communications of the ACM*, Vol. 18, No. 11, pp. 651-656, November 1975.

[3] BBN Advanced Computers Inc., *Inside the TC2000 Computer*, Cambridge, MA, 1990.

[4] BBN Advanced Computers Inc., *Uniform System Programming in C*, Cambridge, MA, 1990.

[5] Borrmann, Lothar and Martin Herdieckerhoff, "A Coherency Model for Virtually Shared Memory," *Proceedings of the International Conference on Parallel Processing*, St. Charles, IL, pp. II-252-257, August 1990.

[6] Burkhart, Helmar and Roland Millen, "Performance-Measurement Tools in a Multiprocessor Environment," *IEEE Transactions on Computers*, Vol. 38, No. 5, pp. 725-737, May 1989.

[7] Collin, Zeev and Rina Dechter, "A Distributed Solution to the Network Consistency Problem," *Proceedings of the 1990 International Symposium on Methodologies for Intelligent Systems*, Tennessee, pp. 242-251, October 1990.

[8] Conrad, James M. and Dharma P. Agrawal, "Performance of an Asynchronous Parallel Algorithm on a Generic Multiprocessor Simulator," *Proceedings of the Pittsburgh Conference on Simulation and Modeling*, Pittsburgh, PA, pp. 1290-1297, May 1991.

[9] Conrad, James M., Dennis R. Bahler and James A. Bowen, "Static Parallel Arc Consistency in Constraint Satisfaction," *Proceedings of the 1991 International Symposium on Methodologies for Intelligent Systems*, Charlotte, NC, pp. 500-509, October 1991.

[10] Conrad, James M., Dharma P. Agrawal, and Dennis R. Bahler, "Scalable Static Parallel Arc Consistency Algorithms for Shared Memory Computers," *Proceedings of the Sixth International Parallel Processing Symposium*, Beverly Hills, CA, pp. 242-249, March 1992.

[11] Conrad, James M. and Dharma P. Agrawal, "Distributed, Scalable, and Static Parallel Arc Consistency Algorithms on Private Memory Machines," *Proceedings of the 12th International Conference on Distributed Computing Systems*, Yokohama, Japan, June 1992 (to appear).

[12] Conrad, James M. and Dharma P. Agrawal, "A Graph Partitioning-based Load Balancing Strategy for a Distributed Memory Machine," *Proceedings of the 21th International Conference on Parallel Processing*, St. Charles, IL, August 1992 (to appear).

[13] Conrad, James M. and Dharma P. Agrawal, "Simulation of Generic Multiprocessor Configurations for Asynchronous Algorithms," *International Journal in Computer Simulation, special Issue on Multiprocessor Networks*, 1993 (to appear).

[14] Cook, Todd A., "A Programming System for Parallel Digital Signal Processor Networks," M.S. Thesis, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 1989.

[15] Dechter, Rina, "Decomposing a Releation into a Tree of Binary Relations," *Journal of Computer and System Sciences*, Vol. 41, No. 1, pp. 2-24, August 1990.

[16] Finkel, Raphael and Udi Manber, "DIB - A Distributed Implementation of Backtracking," *ACM Transactions on Programming Languages and Systems*, Vol. 9, No. 2, pp. 235-256, April 1987.

[17] Freuder, Eugene C. and Michael J. Quinn, "Parallelism in an Algorithm that takes Advantage of Stable Sets of Variables to Solve Constraint Satisfaction Problems," Technical Report 85-21, Dept. of Comp. Science, University of New Hampshire, Durham, NH, 1985.

[18] Fujimoto, Richard M. and Campbell, William B., "Efficient Instruction Level Simulation of Computers," *Transactions of the Society for Computer Simulation*, Vol. 5, No. 2, pp. 109-123, April 1988.

[19] Garey, Michael R. and David S. Johnson, *Computers and Intractability: A Guide to NP-Completeness*, Freeman: San Francisco, CA, 1979.

[20] Gaschnig, John, "Experimental Case Studies of Backtrack vs. Waltz-type vs. New Algorithms for Satisficing Assignment Problems," *Proceedings of the 2nd Biennial Conference of the Canadian Society for Computational Study of Intelligence*, Toronto, Ont., pp. 268-277, 1978.

[21] Gilbert, John R. and Earl Zmijewski, "A Parallel Graph Partitioning Algorithm for a Message-Passing Multiprocessor," *International Journal of Parallel Programming*, Vol. 16, No. 6, pp. 427-449, December 1987.

[22] Gu, Jun, "Parallel Algorithms and Architectures for Very Fast AI Search," Technical Report UUCS-TR-88-005, PhD Dissertation, Department of Computer Science, University of Utah, Salt Lake City, UT, August 1989.

[23] Gu, Jun and Rok Sosic, "A Parallel, Optimal Arc Consistency Algorithm," *Proceedings of the International Conference on Parallel Processing*, St. Charles, IL, pp. III-599-600, August 1990.

[24] Haralick, Robert M., and Gordon L. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," *Artificial Intelligence*, Vol. 14, No. 1, pp. 263-313, August 1980.

[25] Ho, Jan Ming, Majid Sarrafzadeh, Gopalakrishnan Vijayan, and C.K. Wong, "Layer Assignment for Multichip Modules," *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 12, pp. 1272-1277, December 1990.

[26] Hwang, Kai and Fayé A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill: New York, NY, 1984

[27] Ibbett, Roland N. and Nigel P. Topham, *Architecture of High Performance Computers Volume II*, Springer-Verlag, New York, NY, 1989.

[28] Intel Scientific Computers Corporation, *iPSC/2 - iPSC/860 User's Guide*, Beaverton, OR, 1991.

[29] Intel Scientific Computers Corporation, *iPSC/2-iPSC/860 Programmer's Reference Manual*, Beaverton, OR, 1991.

[30] Janakiram, Virendra K., "Randomized Algorithms for Parallel Backtracking and Branch-and-Bound," PhD Dissertation, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 1987.

[31] Janakiram, Virendra K., Dharma P. Agrawal and Ravi Mehrotra, "A Randomized Parallel Backtracking Algorithm," *IEEE Transactions on Computers, Special Issue on Parallel Algorithms*, Vol. C-37, No. 12, pp. 1665-1676, December 1988.

[32] Johnson, David S., Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Scheron, "Optimization by Simulated Annealing: An Experimental Evaluation, Part I (Graph Partitioning)," *Operations Research*, Vol. 37, No. 6, pp.865-892, Nov-Dec. 1990.

[33] Kasif, Simon, "On the Parallel Complexity of Some Constraint Satisfaction Problems," *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Philadelphia, PA, pp. 349-353, 1986.

[34] Kasif, Simon, "On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks," *Artificial Intelligence*, Vol. 45, No. 3, pp. 275-286, October 1990.

[35] Kernighan, B.W. and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *The Bell System Technical Journal*, Vol. 49, No. 2, pp. 291-307, February 1970.

[36] Kheir, Naim A., ed., *Systems Modeling and Computer Simulation*, Marcel Dekker: New York, NY, 1988.

[37] Kirkpatrick, S., C.D. Gelatt, Jr. and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, pp. 671-680, May 13, 1983.

[38] Knuth, Donald E., "Estimating the Efficiency of Backtrack Programs," *Mathematics of Computation*, Vol. 29, No. 129, pp. 121-136, January 1975.

[39] Kumar, Vipin and V. Nageshwara Rao, "Parallel Depth First Search. Part II. Analysis," *International Journal of Parallel Programming*, Vol. 16, No. 6, pp. 501-519, December 1987.

[40] Kumar, Vipin, "Algorithms for Constraint Satisfaction Problems: A Survey," *AI Magazine*, Vol. 13, No. 1, pp. 32-44, Spring 1992.

[41] Kung, H.T., "Synchronized and Asynchronous Parallel Algorithms for Multiprocessors," in *Algorithms and Complexity: New Directions and Recent Results*, Joseph F. Traub, editor, Academic Press, New York, NY, pp. 153-200, 1976.

[42] Lee, Ruby Bei-Loh, "Empirical Results on the Speed, Efficiency, Redundancy, and Quality of Parallel Computations," *Proceedings of the International Conference on Parallel Processing*, St. Charles, IL, pp. 91-100, August 1980.

[43] Love, Andrew E., Jr. and Aburdene, Maurice F., "Simulation of a Distributed Algorithm using OCCAM," *Simulation*, Vol. 55, No. 2, pp. 86-95, August 1990.

[44] Mackworth, Alan K., "Consistency in Networks of Relations," *Artificial Intelligence*, Vol. 8, No. 1, pp. 99-118, February 1977.

[45] Mackworth, Alan K., "Constraint Satisfaction," In *Encyclopedia of Artificial Intelligence*, S. Shapiro - editor, Wiley: New York, NY, pp. 205-211, 1987.

[46] Mackworth, Alan K. and Eugene C. Freuder, "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems," *Artificial Intelligence*, Vol. 25, No. 1, pp. 65-74, January 1985.

[47] Mohr, Roger and Thomas C. Henderson, "Arc and Path Consistency Revisited," *Artificial Intelligence*, Vol. 28, No. 2, pp. 225-233, March 1986.

[48] Moresco, Larry L., "Electronic System Packaging: The Search for Manufacturing the Optimum in a Sea of Constraints," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, Vol. 13, No. 3, pp. 494-508, September 1990.

[49] Nadel, Bernard A., "Constraint Satisfaction Algorithms," *Computational Intelligence*, Vol. 5, No. 4, pp. 188-224, November 1989.

[50] Nadel, Bernard A., "Some Applications of the Constraint Satisfaction Problem," *AAAI-90: Workshop on Constraint Directed Reasoning*, Boston, MA, July 30, 1990.

[51] Natarajan, K.S. and Vivek Sarkar, "Processor Scheduling Algorithms for Constraint Satisfaction Search Problems," *Proceedings of the 1988 International Conference on Parallel Processing*, St. Charles, IL pp. II-140-147, August 1988.

[52] Natarajan, K.S., "An Empirical Study of Parallel Search for Constraint Satisfaction Problems," Research Report PC 13320 (#59642) 12/4/87, IBM TJ Watson Research Center, Yorktown Heights, NY, 1987.

[53] Natarajan, K.S. "Graph-Partitioning on Shared-Memory Multiprocessor Systems," *Proceedings of the 1991 International Conference on Parallel Processing*, St. Charles, IL pp. III-120-124, August 1991.

[54] Samal, Ashok and Tom Henderson, "Parallel Consistent Labeling Algorithms," *International Journal of Parallel Programming*, Vol. 16, No. 5, pp. 341-364, October 1987.

[55] Samal, Ashok Kumar, "Parallel Split-Level Relaxation," PhD Dissertation, Department of Computer Science, University of Utah, August 1988.

[56] Science Applications International Corporation, *ANSPEC: An Actor Oriented Environment for PDP Specification*, San Diego, CA, 1989.

[57] Sequent Computer Systems, Inc., *Guide to Parallel Programming on Sequent Computer Systems*, Prentice Hall: Englewood Cliffs, NJ, 1989.

[58] Swain, Michael J., "Comments on Samal and Henderson: 'Parallel Consistent Labeling Algorithms'," *International Journal of Parallel Programming*, Vol. 17, No. 6, pp. 523-528, December 1988.

[59] Teague, W. Michael, James M. Conrad, and Dharma P. Agrawal, "An Innovative Model of Generic Multiprocessors using Simulation," *Proceedings of the Pittsburgh Conference on Simulation and Modeling*, Pittsburgh, PA, May 1992 (to appear).

[60] Van den Bout, David E. and Thomas K. Miller, "Graph Partitioning Using Annealed Neural Networks," *IEEE Transactions on Neural Networks*, Vol. 1, No. 2, pp. 192-203, June 1990.

[61] Waltz, David L., "Understanding Line Drawings of Scenes with Shadows," In *The Psychology of Computer Vision*, Patrick H. Winston - Ed., McGraw-Hill: New York, NY, pp. 19-91, 1975.

[62] Winston, Patrick H., *Artificial Intelligence*, Addison-Wesley: Reading, MA, 1984.

[63] Zhang, Ying and Alan K. Mackworth, "Parallel and Distributed Algorithms for Constraint Satisfaction," Technical Report 91-6, Department of Computer Science, The University of British Columbia, Vancouver, BC, May 1991.

[64] Zhang, Ying and Alan K. Mackworth, "Parallel and Distributed Algorithms for Finite Constraint Satisfaction Problems," *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, Dallas, TX, pp. 394-397, December 1991.