

PARALLEL IMPLEMENTATIONS OF OPTIMIZING NEURAL NETWORKS

ANDREA DI BLAS ARUN JAGOTA
RICHARD HUGHEY

Baskin School of Engineering, University of California at Santa Cruz
Santa Cruz, California

ABSTRACT

Hopfield neural networks are often used to solve difficult combinatorial optimization problems. Multiple restarts versions find better solutions but are slow on serial computers. Here, we study two parallel implementations on SIMD computers of multiple restarts Hopfield networks for solving the maximum clique problem. The first one is a fine-grained implementation on the Kestrel Parallel Processor, a linear SIMD array at the University of California, Santa Cruz. The second one is an implementation on the MasPar MP-2 according to the "SIMD Phase Programming Model", a new method to solve asynchronous, irregular problems on SIMD machines. We find that the neural networks map well to the parallel architectures and afford substantial speedups with respect to the serial program, without sacrificing solution quality.

INTRODUCTION

In this paper, we consider a particular discrete Hopfield neural network approach to the maximum clique problem, an NP-hard problem on graphs (Garey & Johnson, 1979) with applications in various fields (e.g., computer-aided design, computer vision, protein structure matching, etc). A *clique* in a graph is a completely connected subgraph and the maximum clique is the largest clique in the graph. There are sequential algorithms, based on different heuristics, that can find the exact maximum clique in a reasonable time (Coudert, 1997). However, the neural approach has two advantages. First, one can easily trade execution time for solution quality, so that large graphs can be handled in a reasonable amount of time. Second, the neural approach does not require backtracking, and can be efficiently implemented on a SIMD parallel computer. Since neural networks (NNs) intrinsically exhibit concurrency, much work has been done in the field of parallel implementations of NNs, both on Multiple Instruction-Multiple Data (MIMD) and Single Instruction-Multiple Data (SIMD) systems (Lin *et al.*, 1991; Margaritis, 1995; Misra, 1997; Prechelt, 1999). However, to the best of our knowledge, a SIMD implementation of the discrete Hopfield optimizing NN that approximates the maximum clique has not been reported.

THE OPTIMIZING NEURAL NETWORK

Given an undirected graph $G = \{V, E\}$, composed of a set of $N = |V(G)|$ vertices and a set of $|E(G)|$ edges, we build a corresponding discrete Hopfield network (Jagota, 1995) as in Fig. 1. The Hopfield network is completely connected, and weights w_{ij} are assigned to the edges according to the following rule:

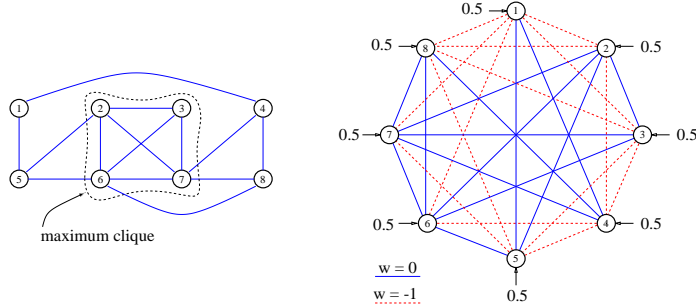


Figure 1: A sample graph (left) and the corresponding Hopfield neural network used to approximate its maximum clique (right).

$w_{ij} = 0$ if vertices i and j are connected by an edge, $w_{ij} = -1$ otherwise. A bias input to the nodes is initialized to $I_i = \frac{1}{2}$, $\forall i$.

A state $S_i \in \{0, 1\}$ associated to each node indicates whether the node is included in the clique under construction ($S_i = 1$). The input to node i is $n_i = \sum_{j \neq i} w_{ij} S_j + I_i$. Any serial update of the form:

$$S_i(t+1) = \begin{cases} 1 & \text{if } n_i(t) > 0 \\ 0 & \text{if } n_i(t) < 0 \\ S_i(t) & \text{otherwise} \end{cases}$$

minimizes the network energy: $E(\vec{S}) = -\frac{1}{2} \sum_{i,j} w_{ij} S_i S_j - \sum_i I_i S_i$ with lower energy minima corresponding to larger cliques. At each step, a node i : $\Delta E_i < 0$ is picked by a random roulette-wheel selection, with probability

$$p_i = \frac{\Delta E_i}{\sum_{j: \Delta E_j < 0} \Delta E_j}$$

where $\Delta E_i(t) = -(S_i(t) - S_i(t-1))n_i$ is the network energy decrease caused by the (possible) switch of node i .

The complete algorithm (Fig. 2) uses multiple restarts to avoid local minima. Moreover, the restarts are *adaptive*, to focus on the most promising areas in the network. If a larger clique is found in a restart, then the bias input of all the nodes in the clique is increased (the clique is *rewarded*), so that in the next restarts the search will focus on its neighborhood. Otherwise, the bias input of the corresponding nodes is decreased (the clique is *penalized*), so that the search gradually gets unbiased from the previous focus (Jagota, 1995; Jagota, 1996).

TWO PARALLEL IMPLEMENTATIONS

The basic algorithm would apparently require a computation time of the order of $O(N^2)$ for each restart loop. However, it can be easily worked out so that the total network energy does not need to be evaluated at each step. Moreover,

```

maxclique()
{
   $\forall i, I_i \leftarrow 0.5$  // initialize nodes' bias
   $\alpha_r \leftarrow 1, \alpha_p \leftarrow 1$  // initialize reward and penalize coefficients
   $C_{best} \leftarrow \Phi$  // initialize best clique to empty
  for  $r = 1$  to RESTARTS do
  {
    find_a_clique()
    if ( $|C| > |C_{best}|$ ) //  $C$  is the clq just found
    {
       $\forall i \in C, I_i \leftarrow I_i + \alpha_r * (|C| - |C_{best}|)$  // larger clq found: reward
       $C_{best} \leftarrow C$ 
    }
    else
       $\forall i \in C, I_i \leftarrow I_i - 1 - \alpha_p * (|C_{best}| - |C|)$  // penal. nodes in  $C$ 
    normalize_bias() // normal. all nodes' bias to ]0, 1[
     $\alpha_r \leftarrow \alpha_r * 1.001, \alpha_p \leftarrow \alpha_p * 0.999$  // adjust adaptation coefficients
  }
  return( $C_{best}$ )
}

```

Figure 2: The maximum clique algorithm with adaptive restarts.

```

find_a_clique()
1: {  $\vec{S} \leftarrow \vec{\Phi}$  // initialize clique to empty
2:    $\vec{n} \leftarrow \vec{I}$  // initialize inputs to bias
   do
3:   {  $\forall i, V_i \leftarrow \begin{cases} 1 & \text{if } n_i > 0 \text{ and } S_i = 0 \\ 0 & \text{otherwise} \end{cases}$  // node  $i$  can be selected
      // node  $i$  can not be sel.
4:      $\forall i, \Delta E_i \leftarrow -V_i * n_i$  // compute energy variations
5:      $k \leftarrow F(\Delta E)$  //  $F()$  picks next node
6:     if ( $k = 0$ ) return // no more selectable nodes, clique found
7:      $S_k \leftarrow 1$  // add node  $k$  to clique under construction
8:      $\vec{n} \leftarrow \vec{n} + \vec{w}_k$  // and update all nodes' input
   } while(1)
}

```

Figure 3: The network evolution function that finds a clique.

the inputs to the nodes are updated upon a node status change, rather than recomputed every time. These two modifications bring the complexity of a cycle down to $O(N)$ (Fig. 3, lines 4 and 8) at the expense of adding a “switch vector” \vec{V} , whose values are also computed in $O(N)$ time (line 3). The function $F()$ (line 5) performs the roulette-wheel selection and returns the selected node or 0 if no nodes can be added to the clique. Finally, the algorithm in Fig. 3 has been stripped of the instructions that take into account the possible *removal* of a node from the clique under construction. This condition can only happen when the network (the \vec{S} vector) is initialized to some non-empty state where some nodes are not part of the clique. The selection mechanism always selects nodes that are in $N(C)$ (where C is the clique under construction), therefore, starting from an empty clique, no node removal is necessary.

Fine-grain Kestrel Implementation

Kestrel is a 512-PE linear SIMD array on a single PCI board for NT/Linux/OSF platforms, designed and realized at the University of California, Santa Cruz, and currently used also by other academic and industrial institutions (Dahle *et al.*, 1997; Hirschberg *et al.*, 1998).

The mapping of the NN algorithm on the Kestrel parallel computer led to a “classic” fine-grain implementation, with one network node per PE. Therefore, the largest graph that can be solved with the current Kestrel system has 512 nodes. The key point in this implementation is that the steps at lines 1, 2, 3, 4 and 8 (Fig. 3) are performed in parallel on all N nodes (PEs), bringing down their time complexity to $O(1)$. The delicate point in the loop remains the selection function $F()$, whose complexity is still $\Theta(N)$.

“SIMD Phase Programming Model” MasPar Implementation

The MasPar MP-2 is a well-known parallel processor. It is a SIMD bi-dimensional array with toroidal wraparound composed of 1K, 2K, 4K, 8K or 16K PEs (Nickolls, 1990). A 4K-PE model was used for this program.

The amount of local memory in this system (64KB per PE) allows the implementation of the neural algorithm according to the “SIMD Phase Programming Model” (SPPM) for graphs of up to 512 nodes. The SPPM is an explicit parallel programming methodology that allows a simple coding of asynchronous, irregular problems on SIMD architectures (Di Blas & Hughey, 2000). In this implementation, each PE receives a copy of the whole graph, and acts independently like a serial machine. However, each PE follows a different stream of random numbers (used by the function $F()$ to select the nodes), and therefore explores a different region of the solution space. Adaptation, however, takes place globally. A synchronization barrier is placed after the call to the `find_a_clique()` function, and the largest clique found among the PEs is broadcast to the whole array. Adaptation can also be performed locally within each PE, but global adaptation has proved better for a small number of restarts.

EXPERIMENTAL RESULTS

Table 1 shows results on some of the DIMACS¹ benchmark graphs. For comparison, timings from a serial machine (143 MHz SUN Ultra-SPARC 1 with 256 MB RAM) running the same algorithm are also reported. Even though the serial machine is not a “state-of-the-art” computer, these results still prove that high speedups can be achieved using relatively slow parallel processors (Kestrel runs at 20 MHz and the MasPar at 12.5 MHz). All experiments were run with 8K total restarts, which, on the MasPar, means two restarts using 4K PEs.

The Kestrel implementation achieves an average speedup of about 15 with respect to the serial program and the MasPar implementation achieves a speedup of about 30, both on the graphs in Tab. 1 and on all 46 DIMACS maximum clique benchmarks. Both parallel implementations maintain the same average quality with respect to the serial program, with small differences accounting for the random mechanism of the algorithm.

The MasPar implementation permits a more elaborate experimentation. In our Kestrel fine-grain implementation there are always $N = |V|$ active PEs, but with the SPPM implementation a program can be run with any number of PEs. The plots in Fig. 4, left, show a typical behavior of the solution quality

¹Home page of *Center for Discrete Mathematics and Theoretical Computer Science*. <http://dimacs.rutgers.edu>.

Graph	# of PEs			Kestrel 512		MasPar 4096		Serial 1	
	$ V $	$ E $	γ^*	γ_k	t_k [s]	γ_m	t_m [s]	γ_s	t_s [s]
MANN_a27	378	70551	126	125.2	29.5	125.2	15.9	124.8	582.5
brock200_1	200	14834	21	18.2	4.9	19.8	2.8	19.2	42.2
brock400_1	400	59723	27	21.6	7.4	23.0	8.8	21.8	94.4
c-fat500-10	500	46627	126	126.0	41.7	126.0	15.3	126.0	838.2
hamming8-4	256	20864	16	16.0	4.2	16.0	3.4	16.0	33.5
keller4	171	9435	11	11.0	2.6	11.0	2.0	11.0	22.2
p_hat300-2	300	21928	25	21.3	3.4	24.6	4.4	22.7	58.2
p_hat500-3	500	93800	-	40.4	7.7	45.6	14.3	40.9	182.4
san200_0.7_1	200	13930	30	29.0	4.4	30.0	3.1	30.0	43.5
san400_0.9_1	400	71820	100	80.2	8.1	99.4	13.4	69.7	230.2
sanr400_0.7	400	55869	-	18.9	6.8	19.7	8.3	19.3	80.8

Table 1: Average performance of parallel and serial programs, for 8K total restarts on different graphs. When known, the optimal clique γ^* is indicated.

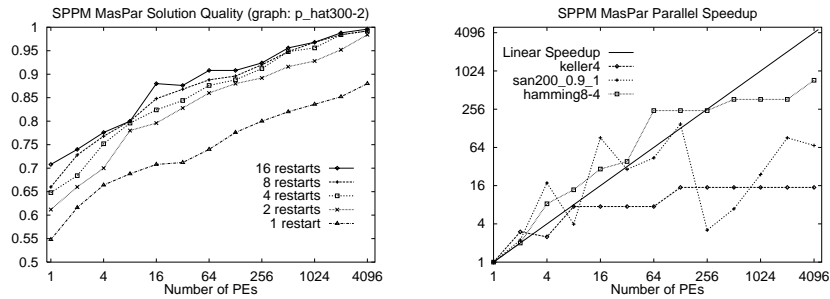


Figure 4: MasPar solution quality as a function of the number of PEs, for different number of restarts on a sample graph (left), and parallel speedup on three sample graphs (right).

as a function of the number of PEs used, for different numbers of restarts on a sample graph. With only one restart, there is no adaptation, and solution quality is generally poor. With two restarts, i.e. with a little adaptation, the solution quality substantially increases. Execution time, however, is only weakly dependent on the number of PEs, but increases linearly with the number of restarts. Therefore, when the number of PEs is much larger than the number of nodes in the graph, doing more than a few restarts is time-consuming without leading to a proportionally better solution.

As with other asynchronous applications, the “SIMD Phase Programming Model” implementation exhibits a good parallel speedup and efficiency; moreover, in this case, super-linear speedup is observed (Fig. 4, right). This phenomenon is common to parallel searches, when the algorithm finds just one solution and not all the possible solutions. This behavior is strongly dependent on the distribution of the solutions and on the random nature of the search (Rao & Kumar, 1993).

CONCLUSION

We have presented two different parallel implementations on two SIMD computers of a discrete Hopfield NN approach to approximating the maximum clique in a graph. The Kestrel fine-grain implementation exploits the intrinsic parallelism of the algorithm, and therefore performs better, with respect to the serial program, when graphs are large and with large cliques. The MasPar SPPM implementation offers a more flexible approach because all available PEs can be active and contribute to the solution at the same time, regardless of the number of nodes in the graph. The maximum clique problem has applications in many fields, and its parallel neural implementation can extend the scale of problems that can be efficiently solved. Moreover, the NN algorithm itself is somehow more generic (the problem-specific knowledge is encoded in weights and bias only), which can make it applicable to other combinatorial problems as well. This work was supported in part by the NSF grants CDA-9115268 and EIA-9722730.

REFERENCES

- Coudert, O. (1997). Exact coloring of real-life graphs is easy. In: *Proc. of the Design Automation Conf.* pp. 121–126, ACM Press.
- Dahle, D., Hirschberg, J. D., Karplus, K., Keller, H., Rice, E., Speck, D., Williams, D. H., & Hughey, R. (1997). Kestrel: Design of an 8-bit SIMD parallel processor. In: *Seventeenth Conf. on Advanced Research in VLSI* pp. 145–162.
- Di Blas, A. & Hughey, R. (2000). Explicit SIMD programming for asynchronous applications. In: *IEEE Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP2000)* pp. 258–267, IEEE Comp. Soc. Press.
- Garey, M. R. & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman.
- Hirschberg, J. D., Dahle, D. M., Karplus, K., Speck, D., & Hughey, R. (1998). Kestrel: A programmable array for sequence analysis. *J. of VLSI Sig. Proc.*, **19**, 115–126.
- Jagota, A. (1995). Approximating maximum clique with a Hopfield network. *IEEE Trans. on Neural Networks*, **6** (3), 724–735.
- Jagota, A. (1996). An adaptive, multiple restarts neural network algorithm for graph coloring. *European J. of Operational Research*, **93**, 257–270.
- Lin, W., Prasanna, V. K., & Przytula, K. W. (1991). Algorithmic mapping of neural network models onto parallel SIMD machines. *IEEE Trans. on Computers*, **40** (12), 1390–1401.
- Margaritis, K. G. (1995). On the systolic implementation of associative memory artificial neural networks. *Parallel Computing*, **21**, 825–840.
- Misra, M. (1997). Parallel environments for implementing neural networks. *Neural Computing Surveys*, **1**, 46–80.
- Nickolls, J. R. (1990). The design of the Maspar MP-1: A cost effective massively parallel computer. In: *COMPCON Spring 1990* pp. 25–28, IEEE Computer Society Press.
- Prechelt, L. (1999). Exploiting domain-specific properties: Compiling parallel dynamic neural network algorithms into efficient code. *IEEE Trans. on Parallel and Distributed Systems*, **10** (11), 1105–1117.
- Rao, V. N. & Kumar, V. (1993). On the efficiency of parallel backtracking. *IEEE Trans. on Parallel and Distributed Systems*, **4** (4), 427–437.