Journal of Statistical Physics, Vol. 70, Nos. 3/4, 1993

Vectorized Search for Single Clusters

Hans Gerd Evertz[1]

Received August 5, 1992

Breadth-first search for a single cluster on a regular lattice is shown to be vectorizable. It is applied to construct clusters in the single-cluster variant of the Swendsen–Wang algorithm. On a Cray-YMP, total CPU time has been reduced by factors of 3.5–7 in large-scale applications. A multiple-cluster version is also described.

KEY WORDS: Cluster search; breadth-first search; vectorization; Swendsen–Wang algorithm.

1. INTRODUCTION

The vectorization described in this paper applies to "breadth-first search"[1] on a regular lattice in general. We shall describe it in the framework of cluster construction in a spin system. Monte Carlo simulations of many discrete- and continuous-spin systems have been revolutionized in recent years by the advent of cluster algorithms that eliminate or strongly reduce critical slowing down[2,3] (for reviews see, e.g., refs. 4). As an improvement over the original multicluster Swendsen–Wang algorithm,[2] the *single-cluster variant*[3] further reduces critical slowing down for many systems. Where possible, it is now often the method of choice in computer-time-intensive simulations.

The Swendsen–Wang algorithm is computationally dominated by the task of identifying all clusters. This is equivalent to the well-known problem of connected component labeling, important for, e.g., image processing. For the *multicluster* version there exist several SIMD parallel (i.e., vectorizable) algorithms to do this efficiently[5–7] (see also the overviews in refs. 8 and 9).

[1] Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida 32306.

The *single-cluster* variant, however, seemed to have the severe drawback of not being efficiently vectorizable. The corresponding loss in speed often greatly reduced the original gain over critical slowing down. The problem here is that only a (small) fraction of all lattice sites is inside the single cluster on average, so that algorithms which require processing of *all* lattice sites become inefficient.

In the present paper we avoid this problem. We treat the dynamically favorable *single-cluster* algorithm and show how to efficiently vectorize over "generations" in the corresponding breadth-first search. The procedure can also be iterated to construct multiple clusters.

2. CLUSTER CONSTRUCTION AND BREADTH-FIRST SEARCH

For simplicity, let us work on a Euclidean "square" lattice of arbitrary size and dimension. More general graphs are possible. Let lattice sites be labeled by a set of integers. The above-mentioned cluster algorithms specify a procedure for defining "*bonds*" to be *on* or *off* between two lattice sites. The sites are connected if the bond is *on*. The connected components of lattice sites are called clusters.

In the single-cluster algorithm, an initial site is chosen at random, and the cluster to which it belongs is then determined (constructed) by a search. Bonds are often evaluated only during this search. Two commonly used search algorithms are "depth-first" and "breadth-first" search. We shall use the latter.

Breadth-first search:

```
(1) Start a list $C$ with one entry $i$, where $i = i_0$ = initial site.
(2) For each neighbor $j$ of site $i$ that does not yet belong to the cluster:
    — determine if bond $\langle ij \rangle$ is *on* or *off*;
    — if bond is *on*, then:
        add $j$ to list $C$ of cluster members;
        mark site $j$ as belonging to cluster.
(3) Repeat (2) for $i$ = next entry in list $C$, until list is exhausted.
```

3. VECTORIZED SEARCH

The above search contains "generations" of sites, where the first generation is $\{i_0\}$, and each following generation consists of those direct neighbors of the previous generation that are newly entered into the list C.

This list can now be employed to *vectorize over each generation* in order to create the next generation. There is one possible vector conflict, namely that a new site could be neighbor of more than one site of the current generation, and could thus be added to the list more than once. This conflict is easily avoided by considering only neighbors in *one direction* during each vectorized loop, and then treating directions in an outer loop.[2]

What follows is the Fortran code for vectorized cluster search, as it runs on a Cray-YMP. The initializations are only sketched. The cluster construction is written out in full.

```
** Initializations (sketch):
*  Once:
C  define array:          neighbor_site(site,direction)  !lattice geometry
C  define function:       bond_is_on(site_1,site_2)
C  declare array          list_entry(site)               !the list 'C'
*  For each cluster:
C  (re)initialize array:  site_is_in_cluster(site)=.false.
C  define                 initial_site

** Cluster construction by Breadth First Search (full code):
        list_end=1
        list_entry(list_end)=initial_site
        end_of_generation=0

    10 start_of_generation=end_of_generation+1      ! Loop
        end_of_generation=list_end
        do 20 direction=1, number_of_directions
CDIR$ IVDEP ! tell compiler to ignore vector dependencies in "do 30"
            do 30 index=start_of_generation, end_of_generation
              site_i=list_entry(index)
              site_j=neighbor_site(site_i,direction)
              if(site_is_in_cluster(site_j)) goto 30
              if(bond_is_on(site_i,site_j)) then
                site_is_in_cluster(site_j)=.true.
                list_end=list_end+1
                list_entry(list_end)=site_j
              endif
    30      continue
    20    continue
        if(list_end .gt. end_of_generation) goto 10
```

After the search terminates, array "list_entry" contains the list C of all sites in the cluster. Each cluster site occurs once in C. The existence of this list can be quite helpful, as it can be used to perform subsequent vectorized operations on all cluster sites.

[2] Thus the vectorization will work on any graph in which the number of neighbors per site is bounded (preferrably constant) and in which there is a one-to-one map "neighbor_site(site,direction)."

Note that the program contains multiple indirect addressing. It vectorizes on a Cray-YMP. Its complexity can be reduced to simple indirection by using additional temporary arrays.

The code can easily be extended to do *multicluster* searches: Just replace the definition of "initial_site" with an outer loop starting like

```
do 5 initial_site=1,lattice_size
   if(site_is_in_cluster(initial_site)) goto 5
   .....
```

This will work well as long as there are not too many very small clusters. An integer array "site_is_in_cluster" might be used in order to label clusters. In Swendsen–Wang multicluster dynamics a large fraction of all clusters normally has size 1. Small clusters like these are more efficiently treated in initial vectorized scans over the whole lattice.

4. DISCUSSION

The vector length during a cluster search is initially 1. It will rise one or more times and in the end be small again. Maximum and average vector length are influenced by cluster size, cluster shape, and dimensionality of the lattice. A small additional gain in speed (about 10–30% on a Cray) can be obtained be letting small loops run in scalar mode (below a length of about 4 on the Cray-YMP).

Our vectorization works very well for the *single-cluster* Swendsen–Wang algorithm(3) because that algorithm has a large average cluster size.(4) We did not evaluate the performance of a multicluster version.

The actual gain in CPU time will be machine dependent. On a given machine it is determined by the distribution of vector lengths, which in turn depends on the simulated physical system through its cluster properties. Let it therefore suffice to quote a few real-life examples in order to show that the *vectorization* works well.

We have been using the vectorized algorithm in several large-scale Monte Carlo simulations on a Cray-YMP. For a two-dimensional spin model(10) at an average cluster size of 1000, the observed average vector length was 34, and the complete update routine ran about 3.5 times faster than the optimized nonvectorized version. A three-dimensional simulation(11) of an Ising-like system with complicated inner loop (complicated function "bond_is_on") ran about 7 times faster at average cluster size 1000. A similar gain was observed for a four-dimensional $O(4)$ model.(12)

We have thus shown how a rather small modification will *efficiently vectorize the breadth-first-search algorithm*, resulting in a large gain in CPU time.

ACKNOWLEDGMENTS

I am grateful to Clive Baillie and Paul Scholten for helpful comments.

REFERENCES

1. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, Massachusetts, 1990).
2. R. H. Swendsen and J. S. Wang, *Phys. Rev. Lett.* **58**:86 (1987).
3. U. Wolff, Collective Monte Carlo updating for spin systems, *Phys. Rev. Lett.* **62**:361 (1989); see also P. L. Leigh, Cluster size and boundary distribution near percolation threshold, *Phys. Rev. B* **14**:5046 (1976).
4. U. Wolff, Critical slowing down, in Lattice '89, Capri 1989, N. Cabbibo *et al.*, eds., *Nucl. Phys. B* (*Proc. Suppl.*) **17**:93 (1990); A. D. Sokal, How to beat critical slowing down: 1990 update, in Lattice '90, Tallahassee 1990, U. M. Heller *et al.*, eds., *Nucl. Phys. B* (*Proc. Suppl.*) **20**:55 (1991).
5. Y. Shiloach and U. Vishkin, An $O(\log n)$ parallel connectivity algorithm, *J. Algorithms* **3**:57 (1982).
6. J. L. C. Sanz and R. Cypher, Data reduction and fast routing: A strategy for efficient algorithms for message-passing parallel computers, *Algorithmica* **7**:77 (1992).
7. R. C. Brower, P. Tamayo, and B. York, A parallel multigrid algorithm for percolation clusters, *J. Stat. Phys.* **63**:73 (1991); J. Apostolakis, P. Coddington, and E. Marinari, A multi-grid cluster labeling scheme, *Europhys. Lett.* **17**:189 (1992); P. Rossi, and G. P. Tecchiolli, Finding clusters in a parallel environment, preprint, (October 1991); H. Mino, A vectorized algorithm for cluster formation in the Swendsen–Wang dynamics, *Computer Phys. Commun.* **66**:25 (1991).
8. R. G. Edwards and A. D. Sokal, Sequential and vectorized algorithms for computing the connected components of an undirected graph, in preparation.
9. C. F. Baillie and P. D. Coddington, Cluster identification algorithms for spin models—Sequential and parallel, *Concurrency: Practice Experience* **3**:129 (1991).
10. H. G. Evertz, M. Hasenbusch, M. Marcu, K. Pinn, and S. Solomon, Stochastic cluster algorithms for discrete gaussian (SOS) models, *Phys. Lett. B* **254**:185 (1991); High precision measurement of the SOS surface thickness in the rough phase, *J. Phys. I* **1**:1669 (1991).
11. H. G. Evertz, R. Ben-Av, M. Marcu, and S. Solomon, Critical acceleration of finite temperature $SU(2)$ gauge simulations, *Phys. Rev. D* **44**:2953 (1991).
12. M. Klomfass, private communication.

Communicated by D. Stauffer