A Hardware Implementation of Hierarchical Clustering *

Shobana Padmanabhan, Moshe Looks, Dan Legorreta, Young Cho, and John Lockwood
shobana@arl.wustl.edu,mlooks@cse.wustl.edu,dan.legorreta@charter.net,{young,lockwood}@arl.wustl.edu
Department of Computer Science and Engineering
Washington University in St. Louis

Non-hierarchical k-means algorithms have been implemented in hardware, most frequently for image clustering. Here, we focus on *hierarchical* clustering of *text* documents based on document similarity. To our knowledge, this is the first work to present a hierarchical clustering algorithm designed for hardware implementation and ours is the first hardware-accelerated implementation.

1. Hierarchical Clustering

Clustering partitions a set of items into subsets with similar characteristics by maximizing intracluster similarity while minimizing intercluster similarity, according to a given similarity metric. In hierarchical clustering, each sub-cluster is recursively clustered. A well-studied special case is where the items correspond to text documents and the similarity metric is based on viewing a document as a "bag-of-words", or "bag-of-word-disjunctions" [4, 5]. That is, documents are points in a high-dimensional space where each dimension corresponds to the frequency/occurrence of one or more words in the document.

Input to Hierarchical Partitioning (HP) clustering is a set of N k-dimensional points (vectors) in a binary space and output is a binary tree where each point corresponds to exactly one leaf. Points (from tree root downward) are partitioned into sets $left$ and $right$, maximizing $score(left) + score(right)$. The algorithm terminates when $left$ and $right$ are individual points. The function $score$ maps sets of points to reals, and measures cohesiveness. Specifically, for a set of points $C = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n\}$, let

$$\vec{sum}(C) = \sum_{i=1}^{n} \vec{a}_i \tag{1}$$

in

$$score(C) = \frac{1}{n} \sum_{i=1}^{n} \frac{\vec{sum}(C) \cdot \vec{a}_i}{|\vec{a}_i|}. \tag{2}$$

Figure 1 shows the clustering results for a subset of 160 documents from the popular 20-newsgroup dataset [1] (the first 40 documents each from the groups rec.sport.hockey, rec.sport.baseball, talk.politics.mideast, and rec.motorcycles). The tree has been automatically pruned based on cluster quality at each level. Data dimensionality is reduced to 4000-wide vectors (from a dimensionality in the hundreds of thousands) via a word mapping table [2].

1.1. Adaptation for Embedded Systems

To approximate the optimal partitioning, hill-climbing is used; the set of points is randomly partitioned by randomly assigning each point to either $left$ or $right$. Moves of a single point between $left$ and $right$ which increase the sum of the scores are taken until no more such moves are possible (i.e., a local minimum is reached). With embedded computing, often, integer operations are preferred over floating-point. Accordingly, $score(C)$ is approximated by

$$\left\lfloor \left(\frac{1}{n} \sum_{i=1}^{n} \left\lfloor \frac{\vec{sum}(C) \cdot \vec{a}_i \cdot k}{|\vec{a}_i|} \right\rfloor \right) \right\rfloor. \tag{3}$$

The magnitudes, $|\vec{a}_i|$, may be easily precomputed and cached in an array. For each computation of $score(C)$, we only need to compute $\vec{sum}(C)$ once. Each entry in the summation vector is stored in only eight bits [2]. Thus, we obtain the following C code:

```
score_t score(iterator from,iterator to)
  score_t sc=0;
  base_t sum[K];
  iterator it;
  set_all_to_zero(sum);
  for (it=from;it!=to;++it)
    add_to(*it,sum);
  for (it=from;it!=to;++it)
    sc+= (K*dot_prod(sum,*it))/magnitude(*it);
```

*This research was sponsored by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under Contract number MDA972-03-9-0001. Liquid architecture work is sponsored by National Science Foundation, under grant ITR–0313203. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

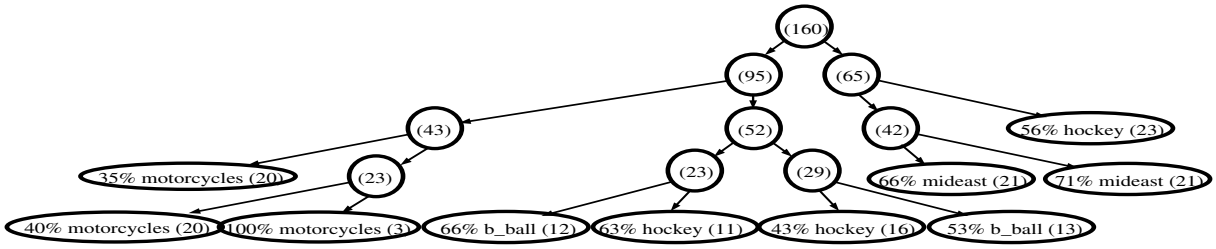

Figure 1. Clustering results

#docs	Runtime (sec) based on 5000 runs each on		
	unaccelerated LEON clocked at 25MHz	accel LEON at 25MHz	Opteron PC at 2.2GHz
16	54	5	0.53
64	196	24	2.08
256	767	113	8.26

Figure 2. Hierarchical clustering runtimes

```
sc/=distance(from,to);
return sc;
```

The asymptotically dominant operations here are the calls to the add_to function (which adds a k-wide vector of $base_t$s with a k-wide vector of bits) and the calls to the dot_prod function (which computes the dot product of a k-wide vector of $base_t$s with a k-wide vector of bits). Experimentally, almost all of the algorithm's execution time is evenly divided between add_to and dot_prod.

2. Performance

The scoring circuit was implemented as a coprocessor to LEON2 on Xilinx Virtex XCV2000E of Liquid architecture platform [3]. The runtimes are compared in Figure 2.

For the performance gain from the circuit, let us first consider the case of the *unaccelerated* algorithm. Given a set of n k-dimensional vectors, the summation requires k add operation for every vector, giving a total of nk operations (not counting operations that are common to both the unaccelerated and accelerated versions). Computing the dot products requires an add and a multiply for each element (thus $2nk$ operations) – giving an overall computational overhead of $3nk$ operations. For the *accelerated* circuit, we need $1.5nk/64$ operations to input data, simultaneously computing $s\vec{u}m$. 1.5 arises from each *pair* of vectors requiring $3k/64$ operations (two loads and a send times $k/64$). Dot-product computation is pipelined and so the addition overhead from this step is only $2k/64$ operations. Finally, $4n$ operations are needed to read back, normalize, and sum the results. The total is thus $0.023nk + 0.31k + 4n$ operations.

The actual *speedup* for clustering N vectors depends on k, as well as how deep a clustering hierarchy we want to construct. Consider $k = 4000$; in this case, the unaccelerated algorithm takes $12000n$ operations, while the accelerated version takes $0.023{\cdot}4000{\cdot}k + 0.31{\cdot}4000 + 4n = 96n + 1240$ operations. Thus, the speedup factor is between 9 $(12000/(96{+}1240))$ and 125 $(12000/96)$ i.e., the speedup improves as n grows.

3. Redesign for High Performance

Using embedded processor cores (i.e., Virtex-4 FX200), we hypothesize that the performance bottleneck will be the memory bandwidth and no longer the processor speed. With the larger FPGA, we can instantiate multiple acceleration engines – up to 45 and use them in parallel to fully utilize the available memory bandwidth. We exploit parallelism in more than one way – scale horizontally, parallelize hierarchical clustering, or parallelize multiple independent hierarchical clusterings. With the speedup gained from such scaling, we predict that we will significantly outperform the commodity general purpose processors.

References

[1] http://people.csail.mit.edu/jrennie/20newsgroups/.

[2] J. W. Lockwood, S. G. Eick, D. J. Weishar, R. Loui, J. Moscola, C. Kastner, A. Levine, and M. Attig. Transformation algorithms for data streams. In *IEEE Aerospace Conference*, 2005.

[3] S. Padmanabhan, P. Jones, D. V. Schuehler, S. J. Friedman, P. Krishnamurthy, H. Zhang, R. Chamberlain, R. K. Cytron, J. Fritts, and J. W. Lockwood. Extracting and improving microarchitecture performance on reconfigurable architectures. *International Journal of Parallel Programming*, 33(2–3):115–136, June 2005.

[4] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. 2000.

[5] A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Proceedings of the 17th National Conference on Artificial Intelligence: Workshop of Artificial Intelligence for Web Search (AAAI 2000)*. AAAI, 2000.