Pthread Parallel K-means

Barbara Hohlt [*]

CS267 Applications of Parallel Computing

UC Berkeley

December 14, 2001

1 Introduction

K-means is a popular non-hierarchical method for clustering large datasets. The time requirements increase linearly with the size of the data set which make it particulary suited for extremely large datasets such as those found in digital libraries. The method was developed by MacQueen [4] in 1967. In our project we take a uniprocessor k-means algorithm and implement a parallel k-means algorithm using pthreads. The algorithm we use is from the Normalized Cuts (Ncuts) [6] codebase of the Vision Group at UC Berkeley.

The rest of this paper is organized as follows: We start by providing information about Ncuts and k-means. In Section 3.2, we describe our software testbed, and present our design and implementation of the parallel k-means algorithm. Then in Section 3.3 we show our performance measurements and results. We conclude and discuss our plans for future work in Section 4.

[*] `hohltb@cs.berkeley.edu`

2 Background

In this section we present a brief background of Normalized Cuts [6] and the k-means algorithm in the Ncuts codebase.

2.1 Normalized Cuts and the Image Segmentation

Normalized Cuts [6], or Ncuts, is an approach developed by the UC Berkeley Vision Group for segmenting images into disjoint regions of coherent brightness and texture. The image segmentation is treated as a graph partitioning problem and a novel global criterion, the normalized cut, is used for segmenting the graph. The details of Ncuts and the publications can be found off the Vision Group website [1].

Ncuts image segmentation has several processing stages. In the earliest stage, a k-means algorithm is used to convert an image of pixels into an image of textons [5]. A texton is defined as the elementary unit of vision perception.

2.2 Uniprocessor K-means

What is K-means? K-means is an iterative, non-hierarchical method for clustering very large data sets. It was developed in 1967 by J.B. MacQueen [4]. K-means computes a partition of a data set into K clusters. K is given as input. The Ncuts uniprocessor k-means algorithm goes as follows:

1. Randomly assign memberships, 1 thru K, to all the data items.
2. Compute the means of each random cluster group.
3. The algorithm iterates thru the following steps until a minimum RMS error or a maximum iteration is reached:
    (a) Proceed through the data items, assigning an item to a cluster whose mean is nearest.

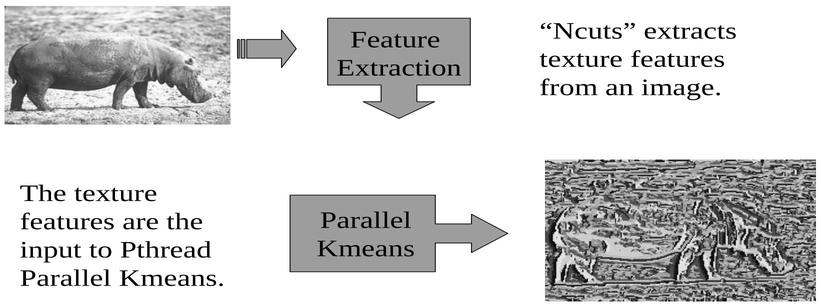
Feature Extraction
"Ncuts" extracts texture features from an image.
The texture features are the input to Pthread Parallel Kmeans.
Parallel Kmeans

Figure 1: The Parallel K-means Testbed

(b) Recalculate the new mean of each cluster group.

In the case of Ncuts, the values of 30 iterations, and 20 K clusters are used. The data items are texture feature vectors of 39 dimensions.

3 Design and Implementation

In this section, we discuss our testbed, design, and implementation of the parallel k-means algorithm.

3.1 The Test Bed

Our first task at hand was to familiarize ourselves with the Ncuts codebase and develop a software testbed for the parallel k-means algorithm. Two codes were written; a uniprocessor base case in C++, and a parallel version in C++/pthreads.

Figure 1 illustrates our testbed. The parallel k-means algorithm takes as input texture features generated by the Ncuts filterbank program. The features are partitioned into K clusters in parallel and then the memberships are converted into color channels for display. The image on the lower right has been partitioned into

20 clusters with each color representing a cluster group. The membership colors are assigned at random. The image represents the texton image.

3.2 The Parallel Design and Implementation

Our design is driven by three factors: a) parallel machine architecture b) programming model and c) performance bottlenecks. We decided to use pthreads, a shared memory programming model, so that our code would easily integrate with the existing Ncuts code. Linux threads is installed on the Millenium [2] cluster, and there are 4 way SMPs. After some intitial measurements of our base case uniprocessor code it became obvious the bottleneck was in computing the cluster memberships. Figure 2 shows the breakdown of the code by component.

We created parallelism by programming with pthreads and using the the single program multiple data (SPMD) strategy. Since the SMPs are 4 way, we create 4 threads and partition the data by 4. Communication is handled via shared data structures. In our algorithm three data structures are shared a) feature data b) membership array, and c) global centers of mass (centroids).

The feature data and membership array did not need synchronization as they were partitioned accross threads. However, access to the global centroids needed to be synchronized. For this we used mutexes and barriers. The barrier code was taken from Butenhoff [3].

The parallel k-means algorithm goes as follows:

1. Divide the feature data by N threads / D features.
2. Randomly assign memberships, 1 thru K, to all the features.
3. Compute K global means of each random cluster group.
4. The algorithm iterates thru the following steps until a minimum RMS error or a maximum iteration is reached:

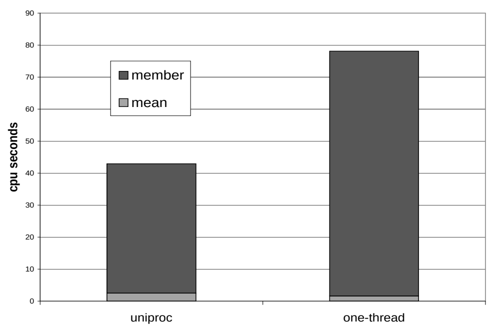

Figure 2: Base Performance of Two Implementations.

(a) In parallel, proceed through the feature vectors, assigning each to a cluster whose mean is nearest.

(b) In parallel, sum the feature vectors for each cluster K.

(c) Synchronized, gather sums and compute K new means of each cluster group.

In the next section 3.3 we show that the gather step (the synchronization step) has near zero performance impact.

3.3 Analysis

All of our measurements were run on a Pentium III machine with 4 katmai processors running Linux kernel 2.4.1. Three data set sizes were used: 120x80, 240x160, and 480x320 pixel images. One feature vector represents each pixel and each feature vector is 39 dimensions. Unless otherwise noted, the 480x320 images were used in the graphs.

For our baseline we developed uniprocessor code useing the Ncuts libraries. We verified our baseline code against the original Ncuts code for accuracy. In both codes the measurements were the same. It was in these initial measurements that it became clear the bottleneck was in the membership component of the

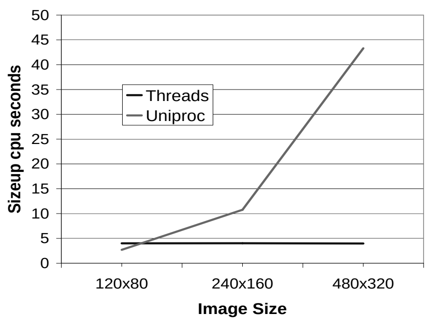

Figure 3: Performance Gains.

code.

Figure 2 shows the times of the basic code components in the baseline uniprocessor code and a single thread of the parallel pthread code. In both cases the membership assignments dominate the processing time, over 95 percent. Time spent in thread overhead and gathers was negligible, so in the graphs we only show the measures of calls to k-means.

Unfortunately, the pthread implementation running one thread is almost twice as slow. This anomaly has yet to be explained.

In Figure 3 we evaluate how the algorithm behaves when the workload is increased. For the parallel code, the dark lower line represents the ratio of the cpu time of a single thread when running one thread to a single thread when running 4 threads. The lighter upper line shows the cpu time of the k-means call in the baseline uniprocessor code.

The baseline uniprocessor code shows poor sizeup, the cpu seconds increase quadradically as the image size increases linearly. However, the parallel code running 4 pthreads shows a near constant time per thread, or a 4 times speedup, indicating a good sizeup property.

Figure 4 shows the speed up property of the parallel code. The dark upper line shows the speed up of

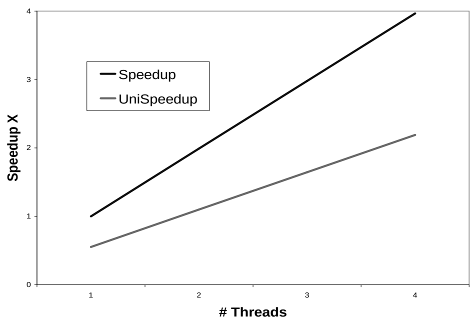

Figure 4: Speedup.

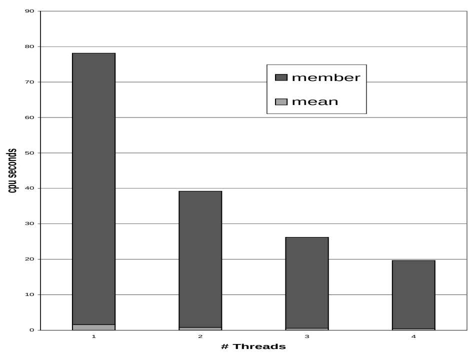

Figure 5: Pthreads and Components.

the parallel code as it relates to one thread. The lighter lower line shows the speed up as the parallel code relates to the baseline uniprocessor code. Both indicate good speedup properties.

Figure 5 shows the breakdown of time spent in each part of the parallel code in a single thread. Only the membership and mean calculation times can be seen as all other times in gathers and thread overhead are negligible. The workload remains constant and again we see the nice speed up property.

4 Conclusion

In this paper, we present a parallel k-means algorithm implemented with pthreads. Our design and analysis show that k-means lends itself well to parallelization. The parallel implementation demonstrated good speedup and performance. The one disappointment was the comparison of the baseline code to the parallel code running a single thread. We hope to eventually resolve this anomoly and to contribute our implementation to the Ncuts code base.

There are two more optimizations we would like to add to the parallel k-means implemenetation. The first is to simply parallelize the computation of the K centers of mass by having each thread work on K/N threads centroids in parallel. The second is a serial optimizaton. In our initial measurements we observed the first iteration is significantly longer that the rest. Beginning with a good estimate of the starting K centers of mass would improve the start up time. A good approximation would be to run k-means on a fraction of the data.

A future area of research would be to explore parallel k-means distributed accross nodes in a cluster.

5 Acknowledgments

We would like to thank Professor Kathy Yelick, David Bindel, and David Martin for their ideas, suggestions, and many discussions on this project.

References

[1] *http://HTTP.CS.Berkeley.EDU/projects/vision/Grouping/overview.html*. The Vision Group, UC Berkeley.

[2] *http://www.millennium.berkeley.edu*. The Millennium Research Group, UC Berkeley.

[3] BUTENHOFF, D. R. *Programming with POSIX Threads*. Addison-Wesley, 1997.

[4] MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5-th Berkeley Symposium on Mathematical Statistics and Probability, UC Berkeley* (1967).

[5] MALIK, J., BELONGIE, S., SHI, J., AND LEUNG, T. Textons, contours and regions: Cue combination in image segmetation. In *Proceedings of the International Conference on Computer Vision* (September 1999).

[6] SHI, J., AND MALIK, J. Normalized cuts and image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (June 1997).