Shared-Memory Parallelization for Content-based Image Retrieval

Christian Terboven[1], Thomas Deselaers[2], Christian Bischof[3], and Hermann Ney[2]

[1] Center for Computing and Communication, RWTH Aachen University, Aachen, Germany
terboven@rz.rwth-aachen.de

[2] Human Language Technology and Pattern Recognition – Computer Science Department, RWTH Aachen University – Aachen, Germany
{deselaers,ney}@cs.rwth-aachen.de

[3] Institute for Scientific Computing – Computer Science Department, RWTH Aachen University, Aachen, Germany
bischof@sc.rwth-aachen.de

Abstract. In this paper we show how modern shared-memory parallelization techniques can gain nearly linear speedup in content-based image retrieval. Using OpenMP, few changes are applied to the source code to enable the exploitation of the capabilities of current multi-core/multi-processor systems. These techniques allow the use of computationally expensive methods in interactive retrieval scenarios which has not been possible so far. In addition, these ideas were applied to a clustering algorithm where substantial performance improvements could be observed as well.

1 Introduction

With the enormously growing amount of digitally available image data, the need for adequate methods to access, sort, and store the data is heavily increasing. For example medical doctors have to access immense amounts of images daily [1] and home-users often have image databases of thousands of images [2]. Currently the data are usually accessed by meta-data, e.g. the date when the image was taken. But content-based methods, though computationally more expensive, promise interesting possibilities [3].

In the computing industry, on the other hand, performance increases in computers due to increased clock speed are tapering off. Instead, the compute power is increased by replicating processing units, thus making parallel computing a necessity even for 'pizza-box' sized computers [4].

Related work. Traditional parallelization approaches for image retrieval follow the idea of distributed computing where several computers, connected by a network, share the work. Various groups have proposed approaches to parallel image retrieval: The Abacus group from City University of Hong Kong[4] presents an approach where a Beowulf cluster is used for a distributed image retrieval system [5]. In [6] the authors present a method where features are dynamically extracted in a distributed manner on a Linux cluster. In [7] methods to merge results from distributed retrieval systems are presented.

In contrast to that, here we present an approach that uses several processors in one computer that share the same memory. This approach has the considerable advantage, as will be described in the sequel, that the changes that have to be applied to the program are far less drastic and that the runtime overhead incurred in parallelization is much smaller.

Furthermore, the same approach was applied to an expectation maximization clustering algorithm and the results obtained are also very good.

In the remainder of this paper, we shortly discuss the content-based image retrieval system FIRE [8], the clustering algorithm, possible approaches to parallelization, and experimental results. Lastly, we summarize our findings and their importance in the light of upcoming architectural developments.

2 FIRE – Flexible Image Retrieval System

The Flexible Image Retrieval Engine (FIRE)[5] has been developed at the Human Language Technology and Pattern Recognition Group of the RWTH Aachen University. It is designed as a research system with flexibility in mind. That is, it is easily extensible and highly modular. The FIRE system was successfully used in the ImageCLEF 2004 and 2005 content-based image retrieval evaluations [9, 10]. The retrieval process is organized as follows:

Given a query image Q and the goal to find images from a database that are similar to the given query image, we calculate a score $S(Q, X)$ for each image $X \in \mathcal{B}$ from the database $\mathcal{B}$:

$$S(Q, X) = \exp\left(\sum_{i=1}^{I} w_i \cdot d_i(Q_i, X_i)\right) \tag{1}$$

Here, Q_i and X_i are the ith features of the images Q and X, respectively, d_i is the corresponding distance measure, and w_i is a weighting coefficient. For each d_i, $\sum_{X \in \mathcal{B}} d_i(Q_i, X_i) = 1$ is enforced by re-normalization. The K database images with the highest score $S(Q, X)$ are returned. This approach can easily be extended toward user interaction with relevance feedback by considering several positive and negative query images and combining the individual scores.

3 Expectation Maximization Clustering

Another application that could be greatly sped up using this parallelization approach is an expectation maximization clustering algorithm for Gaussian mixture

[4] `http://abacus.ee.cityu.edu.hk/`

[5] available under the terms of the GPL at `http://www-i6.informatik.rwth-aachen.de/~deselaers/fire.html`

densities [11] as it is used for the training in automatic speech recognition [12]. The particular program that we consider here is used for object detection and recognition [13].

The algorithm proceeds as follows: Initially, the data is described by one Gaussian. Then this density is iteratively split until a certain criterion is reached. This criterion might be e.g. “the variance is below a certain threshold” or “there are less than a certain number of observations in a cluster”. After each split, the densities are reestimated for several iterations using the expectation maximization algorithm. This reestimation consists of two steps: (i) In the *expectation step*, for each observation the distances to all cluster centers are calculated and (ii) in the *maximization step*, the means and the variances are recalculated.

4 Parallelization

In this section we present the approaches to parallelization that were considered and the results we obtained using the parallelized implementations.

4.1 Image Retrieval

Given the image retrieval system, three different layers can be identified that offer potential for parallelization:

1. Queries tend to be mutually independent. Thus, several queries can be processed in parallel. This is of interest, if several users access the system at the same time or if several queries are run in *batch mode*.
2. The scores $S(Q, X_n)$ for the database images can be calculated in parallel as the database images are independent from each other. This parallelization has a strong impact if the number of images is large in comparison to the number of threads as is normally the case.
3. Parallelization is possible on the feature level, because the distances $d_i(Q_i, X_i)$ for the individual features can be calculated in parallel.

In this work, only the first two layers are considered as the third may require larger changes in the code for some distance functions and we do not expect it to be profitable as the parallelization in the first two layers already leads to sufficient performance improvements in normal situations.

The FIRE system is written in the C++ language which allows a wide choice of tools for parallelization. Our goal is to achieve a parallelization with as few modifications to the source code as possible, in order to avoid interference with ongoing development. Shared-memory parallelization is more suitable than distributed-memory parallelization for the image retrieval task, as the image database can then be accessed by all threads and does not need to be distributed.

Furthermore, shared-memory parallelization leads to better throughput as most of the computers now have 2 or more processors which share the memory. If one process consumes the whole memory of a machine, the remaining processors cannot be used by other jobs as there is no memory available. In addition,

Program 1 The part of fire where parallelization on the second layer is applied.

```
1  #pragma omp parallel
2  {
3  #pragma omp for schedule(static)  private(imgDists)
4   for(long i=0;i<long(N);++i) {
5     vector<double>&d=distMatrix[i];
6     imgDists=imgComp.cmp(q,db[i]);
7     for(long j=0;j<long(M);++j) {
8       d[j]=imgDists[j];
9      }
10   } // end of for-worksharing
11 } // end omp parallel
```

several program instances loading the image database concurrently might put heavy pressure on the file server, as the image database can be several GB in size.

Given these constraints and objectives, we considered three possibilities for parallelization [14]:

- *UPC*[6] is an extension to the C programming language and as such can be applied to C++ code as well. To obtain optimal performance using UPC, for the proposed parallelization approaches, the memory management would have to be rewritten and thus large changes to the existing code would have been necessary [15].
- *POSIX threads*[7] are a POSIX-conforming method for multi-threaded programs and are provided as a library. As such they can independently be used in every programming language. To use POSIX threads, the changes to the code would be moderate, but dynamic workload distribution on the higher parallelization level would have to be implemented manually [16].
- *OpenMP*[8] requires the fewest modifications to the source code, because it mainly consists of compiler directives [17].

Among these options, OpenMP seems to be the ideal choice as it only requires minor modifications to the source code and is supported by most current C++ compilers. Compilers that do not support these directives just print a warning and ignore them; as a result, the portability of the code is not negatively affected. In Program 1, an excerpt of the code and the necessary modifications are shown for the second layer, where one query image `q` is compared to the database images `db[i]`. It can be seen that only 4 lines (lines 1, 2, 3 and 11) are added in total, 2 of them containing only braces and the other ones containing compiler pragmas.

In line 1 the directive specifies that a parallel region starts. From this point the code is processed in parallel. The parallel regions ends in line 11. Line 3

[6] `http://upc.lbl.gov/`
[7] `http://standards.ieee.org/catalog/olis/posix.html`
[8] `http://www.openmp.org`

specifies that the `for` loop is executed in parallel and that the variable `imgDists` has to be instantiated as a local variable for each of the threads. The loop variable `i` is instantiated for each thread automatically. With `static` scheduling the number of loop iterations is distributed equally to the threads, i.e. for p threads each of them processes roughly $\frac{N}{p}$ loop iterations.

A very similar structure can be found in batch mode for the processing of queries. There, a loop over the queries to be processed is run in parallel. That is, instead of processing query by query, several queries can be processed in parallel. A combination of these two approaches can be achieved by allowing a certain number P_1 of queries to be processed in parallel such that these queries can be processed in the second layer with P_2 database images being processed in parallel per query. Therefore the number of threads has to be $P_1 \times P_2$ and thus, this approach is applicable only for high numbers of processors or relatively small P_1 and P_2.

Parallelization in the third layer could be implemented in a similar way as shown above, but this parallelization would only lead to a better load balancing if the number of queries and the number of database images is lower than the number of processors available, which currently is usually not the case.

In contrast to this, the parallelization for the batch mode is on a very high level and has hardly any effect on the structure of the program. A disadvantage of this approach is that it can only lead to optimal speedup if the number of queries to be processed is a multiple of the number of threads, otherwise the load balancing is suboptimal and some processors are idle. The parallelization on the second level is very advantageous in interactive mode as the number of database images usually is very high in comparison to the number of processors available and nearly linear speedup can be expected.

The object-oriented programming paradigm as employed in the FIRE C++ code simplified the parallelization of both levels. Though the code was not designed with parallelism in mind, the datatype encapsulation originating from the mathematical model of the image retrieval task prevents unintended data dependencies and supports the data dependency analysis as well. In addition, we observed a positive impact on the data locality for non-uniform memory architectures (NUMA), as for example the AMD Opteron based systems.

We evaluated the performance of our parallelization on two architectures with different characteristics, which are both available in the RWTH compute cluster at the Center for Computing and Communication[9]. The Sun Fire E6900 servers consist of 24 dual-core UltraSPARC-IV processors running at 1.2 GHz clock speed with a total of 96 GB of RAM. The dual-core processors are treated as two completely independant processors by the Solaris 9 operating system. Therefore, from the user perspective, the Sun Fire E6900 systems have 48 processors.

Furthermore, two different types of Sun Fire V40z machines were used. They have 4 AMD Opteron 875 dual-core processors and 16GB of RAM or 4 AMD 848 processors and 8GB of RAM, respectively. These processors have 2.2GHz clock speed. The dual-core machines are running the Solaris 10 operating system

[9] `http://www.rz.rwth-aachen.de/computing/info/sun/primer/`

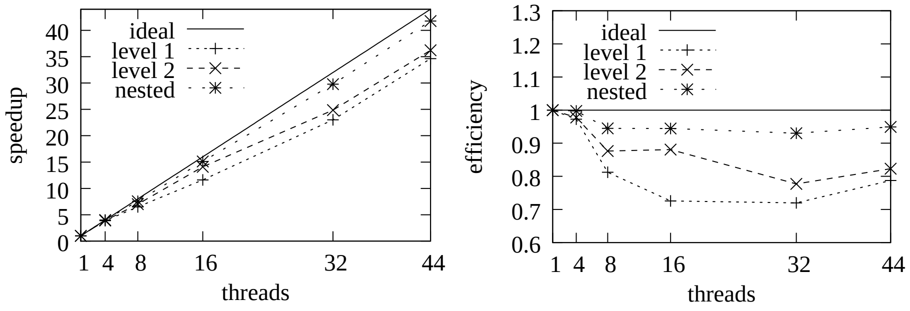

Fig. 1. Speedup and efficiency for image retrieval on a SF E6900 system depending on the number of threads.

that treats the cores as individual processors and the single-core machines have Linux 2.6 as operating system.

While the Sun Fire E6900 systems provide a flat memory model, the Sun Fire V40z systems have a NUMA architecture where data locality is important. On the Solaris systems we used the Sun C++ Studio 10 compiler, on the Linux systems we used the Intel C++ 9.0 compiler, as these both proved to deliver the best serial performance with the FIRE code.

The effect on the retrieval speed can be seen in Figure 1 for the UltraSPARC-IV machines and in Figure 2 for the Opteron machines with Solaris. The figures show the *speedup* and the *efficiency* depending on the number of threads used. The *speedup* is $\frac{T_p}{T_1}$, where T_p denotes the runtime of the parallel program using p threads. The *efficiency* is the speedup divided by the number of processors used. In both cases, the speedup is nearly optimal: using n processors, the runtime is nearly reduced by a factor of n. From the graphs it can be seen that combining (*nesting*) the two proposed parallelization techniques slightly outperforms the use of a single level of parallelization. The UltraSPARC-IV machines were intentionally not used with 48 threads in parallel to prevent a performance dropdown due to operating system effects.

The database used here consists of 9,000 medical radiographs and 100 queries are processed. The images are compared directly using the image distortion model [18]. One image comparison needs approximately 0.012 seconds on the Opteron machines and 0.034 seconds on the UltraSPARC-IV machines, respectively. For one query, 9,000 image comparison have to be executed and thus, the processing of one query takes 110 seconds for the Opteron machines and 303 seconds for the UltraSPARC-IV machines. Using the maximal number of threads measured, the processing time for the complete batch reduces from 3 hours to 23 minutes or 8.5 hours to 12 minutes for the two different systems, respectively.

For all these results it is important to note that the retrieval result is not altered. That is, the results are the same no matter if parallelization is used or not.

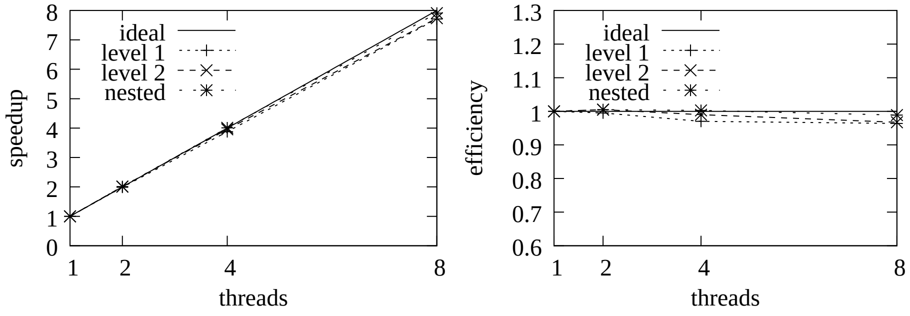

Fig. 2. Speedup and efficiency for image retrieval on a SF V40z system depending on the number of threads.

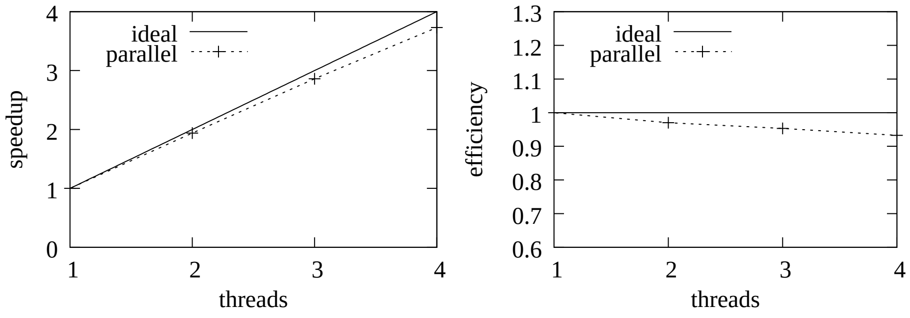

Fig. 3. Speedup and efficiency for clustering on a SF V40z system depending on the number of threads.

4.2 Clustering

The computationally most expensive part of the clustering algorithm are the distance calculations in the expectation phase. The structure of the code is very similar to the above described part of the image retrieval system and the distances are calculated independently for the observations. Thus, the parallelization is possible in exactly the same way as described above. That is, the distance calculations are distributed among the threads.

In the maximization phase, the means and the variances are reestimated by calculating the empirical means and variances for the clusters. This phase could also be parallelized but we have not done this yet, as more than 90% of the computing time is needed for the distance calculation and thus the effect of parallelizing the first phase was sufficient for our application.

The impact of using multiple threads on the performance of the program can be seen in Figure 3. The clustering was only tested on the Opteron machines under Linux but we expect the scalability on other machines to be similar to the scalability of the image retrieval task. It can clearly be seen that the speedup is nearly linear in the number of threads.

5 Conclusion and Outlook

In this work, we presented our approach to using shared-memory based parallelization techniques in content-based image retrieval. Using OpenMP the performance increase is nearly linear in the number of the processors used with minimal modifications to the source code, thus not impacting either the algorithmic structure or the portability of the code. It is clearly shown that shared-memory parallelization is a suitable way to substantially sped up applications in computer vision.

Experimental results with the FIRE code on a 8-processor Opteron Sun Fire V40z and a 48-processor UltraSPARC-IV Sun Fire E6900 show almost perfect speedup. The use of shared-memory parallelism is becoming also more and more important, as recent architectures such as the Sun Fire T2000[10], which has 8 processors, each capable of executing 4 threads, on a shared memory in a 1-unit rack ('pizza-box') form factor, are likely to be available before not too long with substantial floating point capabilities.

With these expected future developments in the computing industry and the results presented it is most probable that methods that currently cannot be used interactively due to their high computational demands might be applicable for interactive use in the not too far future.

Further speedup can likely be obtained using methods like pre-filtering results and work is currently in progress.

Acknowledgement

This work was partially funded by the DFG (Deutsche Forschungsgemeinschaft) under contract NE-572/6.

References

1. Müller, H., Michoux, N., Bandon, D., Geissbuhler, A.: A review of content-based image retrieval systems in medical applications – clinical benefits and future directions. International Journal of Medical Informatics (73) (2004) 1–23
2. Sun, Y., Zhang, H., Zhang, L., Li, M.: Myphotos a system for home photo management and processing. In: ACM Multimedia Confernce, Juan-les-Pins, France (2002) 81–82
3. Smeulders, A.W.M., Worring, M., Santini, S., Gupta, A., Jain, R.: Content-based image retrieval: The end of the early years. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(12) (2000) 1349–1380
4. Sutter, H.: The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobb's Journal **30**(3) (2005)
5. Cheung, C.H., Po, L.M., Wong, K.M.: Web-based beowulf-class parallel computing on image database indexing and retrieval system. In: International Symposium on Intelligent Multimedia, Video and Speech Processing, Hong Kong (2001) 457–460

[10] `http://www.rz.rwth-aachen.de/computing/hpc/hw/niagara.php`

6. Kao, O.: On parallel image retrieval with dynamically extracted features. Journal of Parallel Computing (2005)
7. Berretti, S., Bimbo, A.D., Pala, P.: Merging results for distributed content based image retrieval. Multimedia Tools and Applications **24** (2004) 215–232
8. Deselaers, T., Keysers, D., Ney, H.: Features for image retrieval – a quantitative comparison. In: DAGM 2004, Pattern Recognition, 26th DAGM Symposium. Number 3175 in Lecture Notes in Computer Science, Tübingen, Germany (2004) 228–236
9. Clough, P., Müller, H., Sanderson, M.: The CLEF Cross Language Image Retrieval Track (ImageCLEF) 2004. In: Fifth Workshop of the Cross–Language Evaluation Forum (CLEF 2004). Volume 3491 of LNCS. (2005) 597–613
10. Clough, P., Mueller, H., Deselaers, T., Grubinger, M., Lehmann, T., Jensen, J., Hersh, W.: The clef 2005 cross-language image retrieval track. In: Workshop of the Cross–Language Evaluation Forum (CLEF 2005). Lecture Notes in Computer Science, Vienna, Austria (2005) in press
11. Linde, Y., Buzo, A., Gray, R.: An algorithm for vector quantization design. In: IEEE Transactions on Communications. Volume 28. (1980) 84–95
12. Jelinek, F.: Statistical Methods for Speech Recognition. MIT Press (1998)
13. Deselaers, T., Keysers, D., Ney, H.: Discriminative training for object recognition using image patches. In: IEEE Conference on Computer Vision and Pattern Recognition. Volume 2., San Diego, CA (2005) 157–162
14. Terboven, C.: Shared-Memory Parallelisierung von C++ Programmen. Diploma thesis, RWTH Aachen University, Aachen, Germany (2006)
15. UPC Consortium: UPC Language Specifications, v1.2. (2005)
16. Portable Application Standards Committee: IEEE Standard 1003.1, 2004 Edition. (2004)
17. OpenMP Architecture Review Board: OpenMP Application Program Interface, v2.5. (2005)
18. Keysers, D., Gollan, C., Ney, H.: Local context in non-linear deformation models for handwritten character recognition. In: International Conference on Pattern Recognition. Volume 4., Cambridge, UK (2004) 511–514