Using eSkel to implement the multiple baseline stereo application

Anne Benoit, Murray Cole, Stephen Gilmore and Jane Hillston

School of Informatics, The University of Edinburgh, James Clerk Maxwell Building,
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK
abenoit1@inf.ed.ac.uk
http://homepages.inf.ed.ac.uk/abenoit1/eSkel

Abstract: We give an overview of the Edinburgh Skeleton Library *eSkel*, a structured parallel programming library which offers a range of skeletal parallel programming constructs to the C/MPI programmer. Then we illustrate the efficacy of such a high level approach through an application of multiple baseline stereo. We describe the application and show different ways to introduce parallelism using algorithmic skeletons. Some performance results will be reported.

Keywords: High-level parallel programming, algorithmic skeletons, Edinburgh Skeleton Library, pipeline, deal, multiple baseline stereo application, image matching.

Structured parallel programming with eSkel

The skeletal approach to parallel programming is well documented in the research literature (see [2, 3, 5, 6] for surveys). It observes that many parallel algorithms can be characterised and classified by their adherence to one or more of a number of generic patterns of computation and interaction. For instance, a variety of applications in image and signal processing are naturally expressed as process pipelines, with parallelism both between pipeline stages, and within each stage by replication and/or more traditional data parallelism [7].

Skeletal programming proposes that such patterns be abstracted and provided as a programmer's toolkit, with specifications which transcend architectural variations but implementations which recognise these to enhance performance. This level of abstraction makes it easier for the programmer to experiment with a variety of parallel structurings for a given application, by enabling a clean separation between these structural aspects and the application specific details.

In the *eSkel* (Edinburgh Skeleton Library) project, motivated by our observations [3] on previous attempts to implement these ideas, we have begun to define a generic set of skeletons as a library of C functions on top of MPI. A description of the library and relevant documentation can be found in [1].

The multiple baseline stereo application

We present the multiple baseline stereo application, and we discuss several possible implementations of the application that we are developing with the *eSkel* library. We focus on the programming features offered by a high level parallel programming library to parallelise any application easily and efficiently.

Presentation of the application

The application we are considering is the multiple baseline stereo described in [4, 8]. The algorithm aims to measure depth in a scene accurately, with the help of several cameras. The cameras have various baselines, thus allowing to obtain precise distance estimates with a stereo matching method.

The input consist of three images, acquired from three horizontally aligned, equally spaced cameras. One image is the *reference image*, while the other two are *match images*. For each of 16 disparities $d = 0..15$, the first match image is shifted by d pixels, and the second image by $2d$ pixels. Then, a *difference image* is created by computing the sum of squared differences between the reference image and the shifted match images, for each pixel. The *error image* is obtained by replacing each pixel in the difference image with the sum of the pixels in a surrounding 13x13 window. Finally, the *disparity image* is formed by finding, for each pixel, the disparity that minimizes error. We then know the depth of each pixel, which depends on this disparity.

Parallel implementation with eSkel

We provide in this abstract the outline of our implementations. Several parallel schemes are compared. A sequential version of the application is the reference implementation. The parallel versions are closely related to the sequential code, and simply add parallelism by using the skeletons of the *eSkel* library.

Since we are interested in computing the disparity image successively for different triplets of images, we can easily switch to a parallel version of the application by using the well-known *Pipeline* skeleton [1]. The operations that we need to perform on the input images can then be done in parallel at each iteration. The *eSkel* library offers to the programmer a data model which allows to transfer easily the various images from one pipeline stage to another.

However, some of the stages are more computationally intensive, and we can easily improve the performance by *replicating* the most demanding pipeline stages with a *Deal* skeleton. It is similar to a traditional farm, but with tasks distributed strictly cyclically to workers. It is thus useful while nested in a pipeline, since the deal semantics require the ordering of outputs from the skeleton to match that of the corresponding inputs, irrespective of the internal speed of the workers.

Different configurations will be compared. We can use *persistent* nesting for stage replication, which means that all data arriving into a stage is distributed directly to the workers, and similarly the output data is transferred from the worker to the next stage. We will also illustrate the use of *transient* nesting, for example to parallelise for a given set of input the computation of the difference images (one difference image per disparity). The deal is then called within a stage function, with the data explicitly passed as a parameter. This allows to pre-process the data before calling the nested skeleton. Performance results will be reported for both approaches.

Conclusions and overview of the paper

In this extended abstract, we have briefly introduced the structured parallel programming library *eSkel*, and how it can be used to easily parallelise an application, as illustrated with the multiple baseline stereo application.

The paper will give more details on the parallel algorithms used, through a detailed presentation of *eSkel* and the offered algorithmic skeletons. Moreover, some numerical results will show the efficacy of this approach. We have already obtained promising results showing a good speedup of the application with an implementation using *Deal*.

The use of the library not only allows an easy design of parallel applications, but performance models can also be associated with each skeleton, allowing to reach even better performance for highly demanding computing applications.

References

[1] A. Benoit and M. Cole. *eSkel's web page*. 2005. `http://homepages.inf.ed.ac.uk/mic/eSkel`.

[2] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press & Pitman, ISBN 0-262-53086-4, 1989. `http://homepages.inf.ed.ac.uk/mic/Pubs/skeletonbook.ps.gz`.

[3] M. Cole. Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. *Parallel Computing*, 30(3):389–406, 2004.

[4] M. Okutomi and T. Kanade. A Multiple-Baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–363, April 1993.

[5] S. Pelagatti. *Structured Development of Parallel Programs*. Taylor & Francis, ISBN 0-7484-0759-6, London, 1998.

[6] F.A. Rabhi and S. Gorlatch, editors. *Patterns and Skeletons for Parallel and Distributed Computing*. Springer Verlag, ISBN 1-85233-506-8, 2003.

[7] J. Subhlok, D. O'Hallaron, T. Gross, P. Dinda, and J. Webb. Communication and memory requirements as the basis for mapping task and data parallel programs. In *Proceedings of Supercomputing '94*, pages 330–339, Washington, DC, November 1994.

[8] J.A. Webb. Latency and Bandwidth Considerations in Parallel Robotics Image Processing. In *Supercomputing'93*, pages 230–239, Portland, OR, November 1993.