A Self-Optimising Coprocessor Model for Portable Parallel Image Processing

D Crookes[1], *T J Brown*[1], *Y Dong*[1], *G McAleese*[2], *P J Morrow*[2], *D K Roantree*[2], *I T A Spence*[1]

[1]Department of Computer Science, Queen's University Belfast, Belfast BT7 1NN, UK
[2]Department of Computing Science, University of Ulster, Coleraine BT52 1SA, UK

Abstract This paper describes an approach taken to achieve both portability and efficiency in the context of image processing. It outlines the EPIC architecture (Extensible Parallel Image Coprocessor), an image processing-specific software architecture capable of efficient implementation on a range of parallel machines. EPIC is currently being implemented using a network of C40 processors.

1 Introduction

Much research in parallel programming aims to reconcile two conflicting objectives: software portability, and efficiency of parallel implementation. Often this is addressed by choosing a standard programming language (e.g. a variant of Fortran), and seeking to improve efficiency using auto-parallelisation. In fact, in addition to portability and efficiency, there is a third objective: expressive power of the programming notation. This suggests an alternative, application-specific approach. Instead of starting with full expressive power and portability (e.g. with Fortran), the EPIC project starts with portability and efficiency, and seeks to increase the expressive power of the notation EPIC provides only programming abstractions which can be implemented efficiently and, within this constraint, aims to make the abstractions more generally applicable.

This paper firstly outlines briefly the programming model which EPIC provides to an application developer. Then, the key strategy for retaining efficiency through the extensibility and self-optimising facility of the EPIC environment is presented. Finally, some outline indicators of the performance of the EPIC optimiser are given.

2 The EPIC Programming Model

EPIC aims to provide the image processing developer with the power of parallelism without the responsibility for expressing it. The basic EPIC model comprises an image coprocessor which implements the high level abstractions (in parallel), controlled by a (sequential) general-purpose host. In our implementation, the parallel coprocessor runs on a network of C40 processors. The user program runs on the host, and is written in C++. Access to the instructions of the coprocessor is provided via C++ classes which constitute the programmer's interface to the coprocessor.

The high level programming abstractions which EPIC provides for image processing are based on those of Image Algebra [1]. The power of the notation comes

primarily from the facilities for template-based neighbourhood operations. The EPIC abstractions are a significant advance on previous notations such as IAL [2] and I-BOL [3], in that EPIC supports variant (rather than static) templates, and other template constructors. The shape and size of templates can vary with image position and other parameters (though they are not image data dependent). In the context of implementation on a (distributed memory) parallel architecture, the definition of such templates provides *predictability* of data access. All data locations to be accessed by a neighbourhood operator can be pre-calculated (and the data pre-fetched, if appropriate) as though at the start of a BSP-style superstep [4].

Using these abstractions, it is possible to build a single template operator to carry out an image transform (such as Hadamard or Fourier) in one 'instruction' [5].

3 Efficiency through Extensibility

Although the abstract machine model makes for simpler, more portable programming, it is traditionally thought to result in inevitable inefficiencies. For instance, an image scaling operation which might be coded as:

*Result = NewMin + (Image1 - Min1) * ScaleFactor;*

would typically require three image instructions, each having the inherent overhead of the loop control needed for their implementation.

Another situation where the usual coprocessor model results in inevitable inefficiencies arises with the use of template operators, where a hand coding could take advantage of the specific weight values to give a more efficient implementation.

Generating New Instructions Dynamically

If optimal efficiency is required then a traditional coprocessor model is not a solution. However, to retain the portability advantages of the coprocessor model while obtaining the efficiency of specific hand coded operations, EPIC is designed to be an *extensible* coprocessor, which can automatically generate new, optimised instructions to implement the compound operations actually occurring in the user program.

The heart of the EPIC approach involves several steps, which are made invisible to the application developer through the extensive use of operator overloading in C++:

- As library operations are called, a syntax tree is built. Upon assignment, if this is the first time occurrence of the tree, it will be executed (albeit inefficiently).
- Later, from this tree, optimised code for a new routine is generated (and compiled, etc.), to give a new instruction – thus extending the coprocessor's instruction set.
- The next time this compound operation comes to be executed, the new, optimised instruction is retrieved and executed.

When observing the behaviour of an application program running in the EPIC environment, it will usually be noticed that the second time a program is run, it will do so faster than the first execution. The potential problem of encountering a non-optimised operation when running 'live' can be avoided by explicitly extracting such expressions, and defining them as functions separately.

The EPIC Architecture

To implement the self-optimising, dynamically extensible coprocessor model above, which is the core of the EPIC environment, the software architecture in Fig. 1 is used.

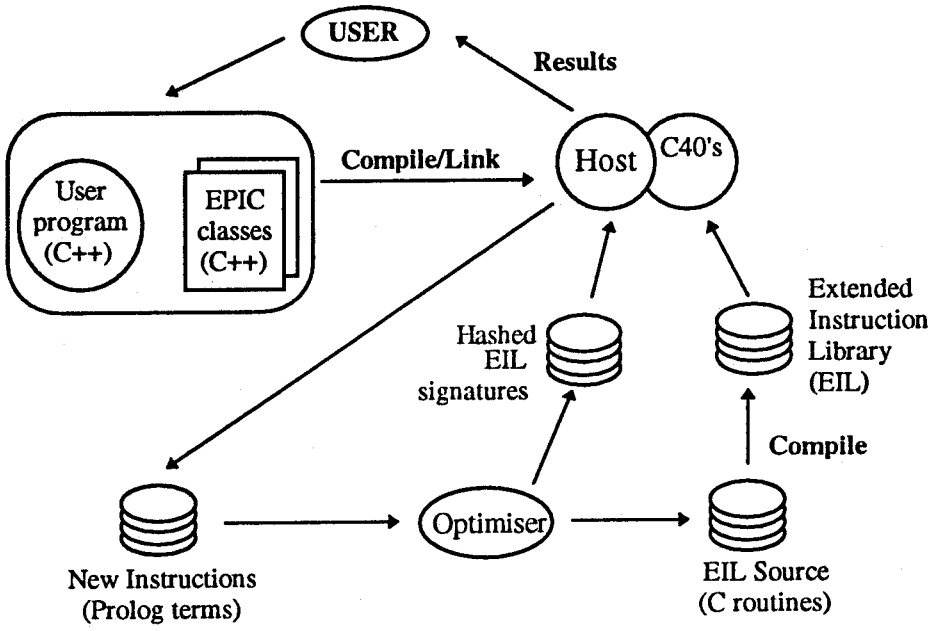

Figure 1 *Architecture of the EPIC Environment*

The instruction set of the image coprocessor is represented by an extensible library (the EIL). Initially this will comprise the primitive operators of EPIC, but it will be dynamically extended as a program is run. For the system to know at execution time whether or not the EIL contains a previously-generated routine for a compound operation (for instance), a quick search has first to be made. This requires the use of a signature for each EIL routine, which is essentially a representtaion of the syntax tree it evaluates. There is inevitably some overhead compared with a system where the linking is done entirely at comple time.

4 Performance of the EPIC Optimiser

Three examples of typical image processing operations which would be inefficiently implemented by a traditional coprocessor model will serve to illustrate the advantages of the dynamically extensible nature of the EPIC architecture.

(i) Image Scaling. The EPIC coding given above is optimised to eliminate the loop overheads. Initial timings for this implementation and for the unoptimised version were taken (on a Pentium PC) and gave a speed up factor of 4.9. On a parallel C40 system, since there is no communication, we would expect a similar speed up factor.

(ii) Sobel Edge Detection. In EPIC, the Sobel operation is coded as:

Edge = (absconv(Image1, Sh)) + (absconv(Image1, Sv))) > Threshold;

EPIC generates a single routine which eliminates loop overheads and takes advantage of the specific weights in the Sobel templates. Comparison of timings for the optimised and non-optimised versions (on a Pentium PC) yield a speed up of 5.3.

(iii) Hadamard Transform. The optimised version of the high level coding of the Hadamard Transform [5] is 95% as efficient as a hand coded version.

5 Conclusions

This paper has presented a novel model for improving the state of the art in achieving both portability and efficiency in parallel software, in the context of image processing. The concept of an abstract coprocessor model is very useful for achieving portability, but it traditionally introduces inefficiencies when compound operations are required. To retain the benefits of the coprocessor model without this efficiency loss, we have defined an *extensible* parallel image coprocessor (EPIC), together with an optimiser which can generate optimised versions of new instructions which are dynamically detected in a user's program. Initial results suggest a speed up factor in the region of 5 can readily be achieved purely through the operation of the optimiser.

Although this paper has not concentrated on the programming abstractions in EPIC, separate investigations are being carried out into how these abstractions can be made more powerful, with a view to being suitable for a wider range of applications.

Acknowledgements

The authors wish to express thanks for financial support for this work to the UK's Engineering and Physical Sciences Research Council, under its Portable Software Tools for Parallel Architectures Programme, and also to Transtech Parallel Systems.

References

1. Ritter G X, Wilson J N and Davidson J L, 'Image Algebra: an overview', *Computer Vision, Graphics and Image Processing*, No. 49 (1990) pp 297–331.
2. Crookes D, Morrow P J and McParland P J, 'IAL: a parallel image processing programming language', *IEE Proceedings*, Part I, Vol 137 No 3 (June 1990) pp 176–182.
3. Brown T J and Crookes D, 'A high level language for image processing', *Image and Vision Computing*, Vol 12 No 2 (March 1994) pp 67–79.
4. McColl W F, 'Scalable Parallel Computing – a grand unified theory and its practical development', IFIP Transactions A: Computer Science and Technology, 1994, Vol.51, pp.539–546.
5. D Crookes, I T A Spence and T J Brown, 'Efficient parallel image transforms: a very high level approach', in Transputer Applications and Systems, Proc 1995 World Transputer Congress, IOS Press, September 1995. pp.135–143.