Transactions on Information and Communications Technologies vol 3, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Toward an automatic mapping of DSP algorithms onto parallel processors

M. Razaz, K.A. Marlow
University of East Anglia, School of Information Systems, Norwich, UK

ABSTRACT

With ever increasing computational requirements of complex DSP algorithms and applications, implementation on multiprocessor platforms becomes a necessity. The main problem is lack of necessary software tools for multiprocessor mapping. We present the main features of a prototype design environment which allows direct mapping of complex DSP applications, designed for implementation on a single processor, onto a multiprocessor platform. We currently use a configurable network of MIMD machines but essentially any platform and interconnection topology can be specified by the user. Experimental results are presented and discussed for automatic mapping of an adaptive differential pulse code modulation (ADPCM) system to a multiprocessor platform with different number of processors and interconnection topologies.

INTRODUCTION

A typical cycle of DSP design and implementation starts with the generation of system specification in an abstract fashion. At this stage one is interested in the design feasibility and not in details of hardware implementation. The next step is to develop a design that meets the required specification. The design is then verified by simulation before implementation on DSP hardware. If it does not meet the specification then the design-simulation step is repeated . If on the other hand the simulation is successful the design is implemented on a single DSP chip and it is then further tested. Again as before the implementation-testing step may have to be iterated several times until testing is successful, otherwise a new design is

needed to meet testing requirements. The shortcomings of this conventional design methodology include the following:

(i) A long cycle from specification to the final product development. This aspect is particularly an important consideration in an industrial environment where time-to-market is crucial for competitiveness and commercial exploitation.

(ii) Hardware dependence and hence lack of portability to different DSP platforms. Efficient software implementation also requires low-level DSP programming skills which is often a rare commodity.

(iii) Lack of exploitation of algorithmic and architectural parallelism for DSP applications.

Besides there are also many complex and computationally intensive applications where the speed of a single hardware platform is a major limiting factor such as speech synthesis and recognition, high definition TV, multimedia communication and image processing. This is also true for real-time DSP applications where there is a need for very high speed processing power. Although high speed DSP chips with limited multiprocessing capabilities are becoming commercially available such as DSP96002 and TMS320C40[2,3], the necessary software tools for supporting the target multiple processor platforms do not exist or are primitive where task allocations have to be done manually by the designer.

In order to address these issues we have used a structured methodology to develop a prototype integrated system for DSP design and development, called Taurus. This new system overcomes the shortcomings of traditional DSP methodology and has distinct features such as the capability of implementing DSP applications to a multiprocessor platform, independence from the hardware processors, exploitation of concurrency and post-implementation performance analysis. When our prototype system is fully developed, it will have the capabilities of:

i) automatically mapping DSP applications to multiple parallel processors with a variety of architectures;

ii) allowing the user to modify schedules and analyse the system performance, and;

iii) prototyping real DSP applications in a multiprocessor environment.

DESIGN ENVIRONMENT

Figure 1 shows the block diagram of our integrated design environment, Taurus, whose main constituent modules are the front-end CAE system, Converter, Platform Independent Support

Software, Multiprocessor Platform, Performance Analyser and Graphical Schedule Editor. We present here a brief description of the modules; more details can be found in [6,11,13].

The user interface to our system is via a commercially available CAE system, SPW[9, 10]. It has a comprehensive range of software facilities for design capture using block diagrams, simulation, and code generation for specific target DSP platforms.

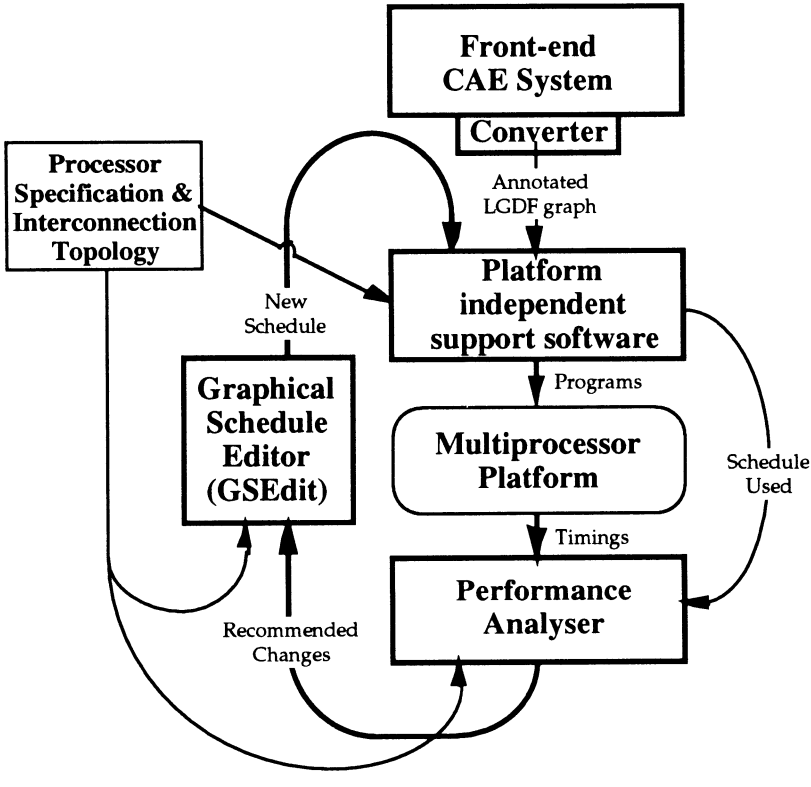

Figure 1. The block diagram of the multiprocessor design environment.

The Converter translates a DSP-based application generated by the user interface into an equivalent large grain data flow(LGDF) graph[1]. The latter is an effective graph representation which allows, using the scheduler, direct algorithm mapping to the Multiprocessor Platform. A node in the LGDF graph represents a task. This can be a basic operation like add and multiply or a more

complex functional block such as FFT, convolution and so on. The flow of information from one node to another and therefore their interdependencies is represented by a directed edge. A node is data driven in that it fires when sufficient tokens i.e. input samples are available to perform a task. For the LGDF graphs to be statically schedulable we assume they are acyclic (i.e. they do not contain any loops) and independent of data.

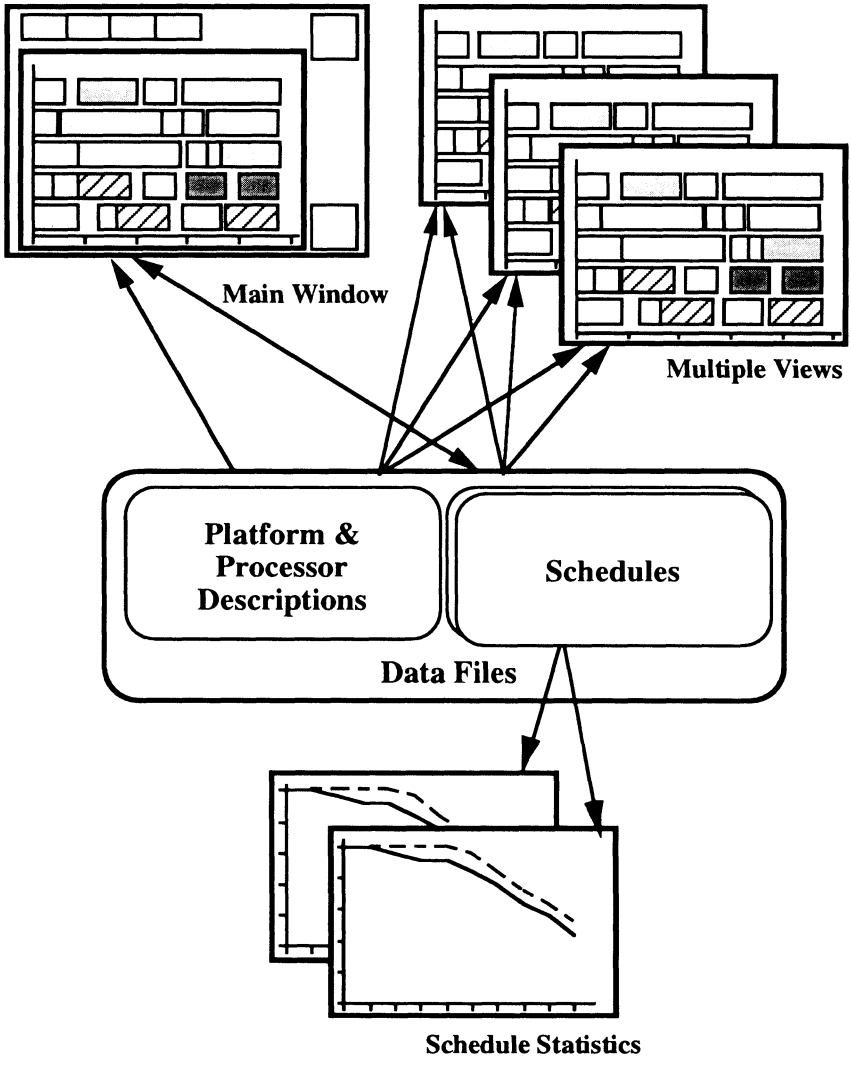

Figure 2 Operational Schematic of Graphical Schedule Editor

Transactions on Information and Communications Technologies vol 3, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

The Platform Independent Support Software consists of Scheduler and Precompiler. The Scheduling system [6]performs the major function of task co-ordination and scheduling, and consists of the Scheduler, Schedule Verifier and Graphical Schedule Editor. The main function of the Scheduler, is to assign systematically the functional blocks(nodes in the LGDF graph) to various processors in the hardware platform. Various forms of scheduling algorithms were considered [4-8] but the static scheduling was chosen as it is performed at compile time and resource requirement in terms of memory and dynamic time is not demanding and hence is ideally suited for DSP applications.

The Schedule Verifier checks if a schedule is permissible i.e. it can be executed to completion without a deadlock or livelock. Deadlock occurs when a processor sends more tokens in a loop than can be consumed by the following processors. Deadlock could also occur when no node in the precedence list has data on its input buffers

The Graphical Schedule Editor (GSEdit) provides the user with a central coherent interface for the checking and editing of multiprocessor schedules with a view to improve efficiency and throughput. Editing a schedule is allowed as long as the changes result in a permissible new schedule. When GSEdit is first executed a Main Window is displayed containing the complete Gantt chart for the first schedule to be operated on. By clicking upon a task in the Gantt chart information is displayed in a subwindow on either what the task actually is or what dependencies it has . By using the mouse this Gantt chart can be manipulated to zoom in on a region of the schedule, and hence displaying greater detail. This region can then be moved up and down the Gantt chart to display different parts of the schedule. The form of display can also be changed to group like tasks by colour and to display the intercommunications occurring. It is also possible to create new views onto the schedule independent of the view displayed in the Main Window.

The user can perform, through the graphic interface, direct operations upon a schedule using a select-drag-drop technique. In addition to manual editing of the schedule it is possible for GSEdit to perform several optimisations upon the schedule under the full control of the user; at every stage GSEdit will select those tasks affected and display the effect of the changes for approval by the user.

GSEdit is being further developed to allow for the user to load in more than one schedule at a time and to cross compare their efficiency or speed-up by displaying graphs of schedule statistics in independent windows.

The Precompiler [13] uses the information from the current schedule description file together with the LGDF and precedence graphs to create the necessary C source programs and control files for the compiler and linker in the target Multiprocessor Platform. The resulting executable programs implement the DSP application.

The Multiprocessor Platform is a Meiko Computing Surface [14] consisting of a configurable network of processors, each processor being a transputer [15] which has its own local control unit, program and memory. This is a message passing parallel computer with multiple instruction, multiple data streams (MIMD) architecture. The network interconnection topology is configured in software.

The Performance Analyser, which is currently being implemented, provides an indication of how good the implementation schedule is [12] and how to interact with the system in a closed-loop iterative fashion in order to modify the schedule and hence improve throughput and efficiency. During the execution of a current DSP implementation, a list of performance measurements is collated which include such parameters as the start and stop times of tasks on different processors, and times when messages were transmitted and received. These measurements, once the execution is completed, are passed to the Performance Analyser which carries out various transformations on them to remove any performance deficiencies. The suitable transformations are then selected and presented to the user for interaction with the system. At this stage the user is also permitted to enter any alternative transformations. The suitably transformed schedule is then used for the next implementation, and the whole process is repeated until the desired implementation efficiency is achieved. However if these modifications do not lead to an improved schedule, then the original DSP design must be changed. By employing a database to handle the underlying files, a designer would be able to easily backtrack or undo previous transformations, and thus explore various options from a list of possible transformations.

RESULTS AND DISCUSSION

To demonstrate the capabilities of Taurus, an ADPCM system [16] as shown in Figure 3, was scheduled to two different simulated platforms: bus connected processors and the Meiko Computing Surface. The principal difference between these two platforms being one of the costs of interprocessor communication; for the bus connected platform the costs are over an order of magnitude less than those for the Meiko platform.

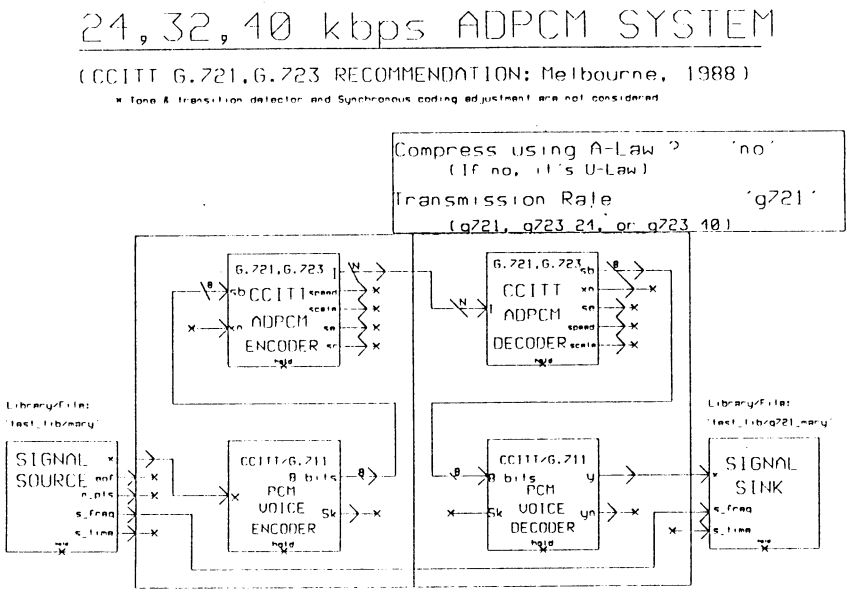

Figure 3. Schematic diagram of ADPCM system taken from SPW.

Figure 4 shows the resulting schedule for the bus connected platform with 6 processors. As can be seen a speed up of 5.78 with a processor utilisation of 96% was achieved. Figure 5 shows the resulting schedule for the same platform but with 8 processors; the resulting speed up and processor utilisation were 7.58 and 94%. This near linear increase in speed up can be attributed to two factors, namely the low communication costs and the high degree of parallelism inherent in the ADPCM system.

When the scheduler is used to map an application to a platform with high interprocessor communication costs, in our case the Meiko Computing Surface, the true effect of such communications on the parallelism achieved can be seen. In figure 6 a schedule to such a platform with 6 transputers in shown, the speed up achieved is 4.29 with a processor utilisation of 71%. Figure 7 shows the schedule to the same platform, but using 8 transputers; the speed up being 4.75 and the achieved utilisation 59%. Both these diagrams show well the effects of high interprocessor communication on the schedules.

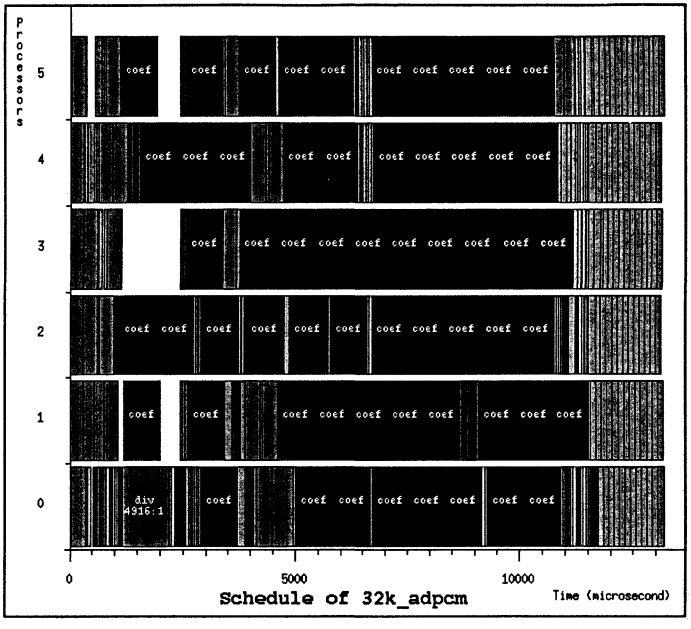

Figure 4. The ADPCM system scheduled to 6 processor bus-connected platform. Speed up 5.78 and utilisation 96%

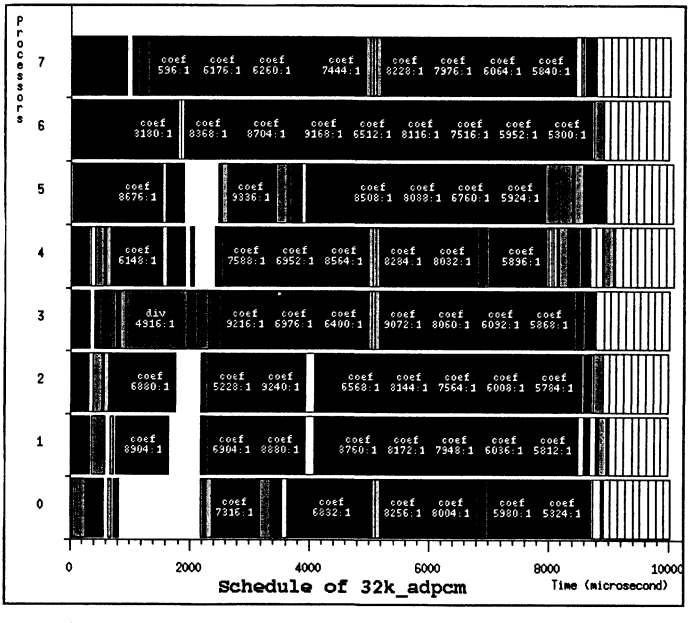

Figure 5. The ADPCM system scheduled to 8 processor bus-connected platform. Speed up 7.58 and utilisation 94%.

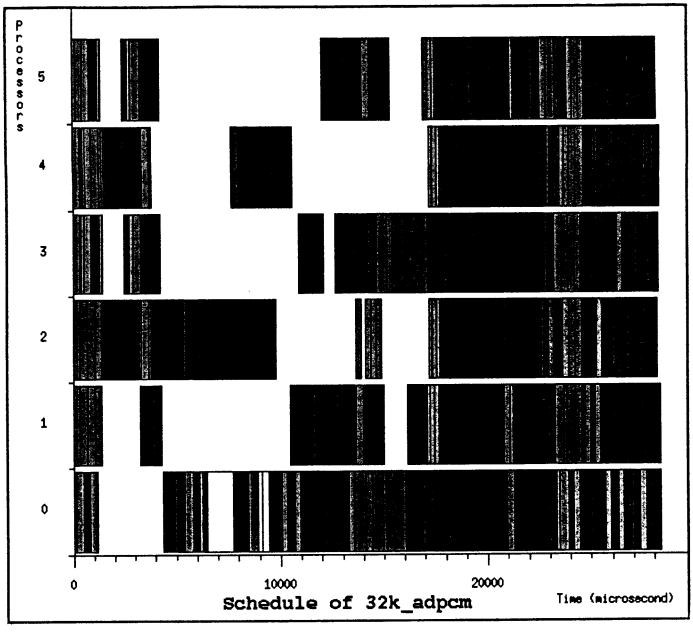

Figure 6. The ADPCM system scheduled to 6 transputer platform. Speed up 4.29 and utilisation 71%.

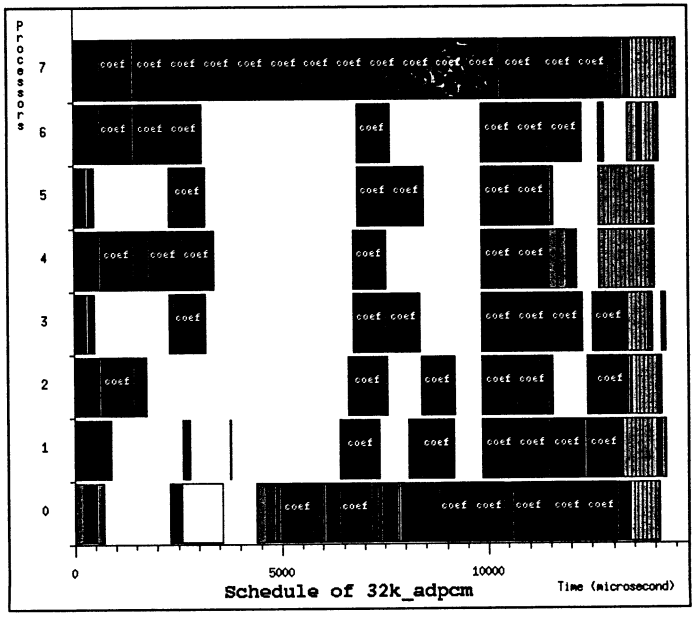

Figure 7. The ADPCM system scheduled to 8 transputer platform. Speed up 4.75 and utilisation 59%.

High interprocessor communications have such a dramatically detrimental effect on the final efficiency of the schedules created due to the increased effect of task-to-processor misplacement during scheduling. A task-to-processor misplacement occurs when a task is scheduled to a processor such that its dependants and predecessors have to incur a higher total cost for communication (or longer execution span) than if it was placed onto a more suitable processor. This is a factor which currently our scheduler cannot take into account during scheduling, although we are researching into using various heuristic methods to 'encourage' tasks with similar dependants to group together on the same processor or closely connected processors.

To improve the ability of our system as a whole to deal effectively with high interprocessor communication times we are currently researching into two forms of post implementation performance analysis:

i) analysis of the schedules created by the scheduler, and;
ii) post analysis of the schedules used on the Multiprocessor Platform with reference to the actual timings measured during the execution of the programs.

ACKNOWLEDGMENTS

The authors would like to thank the Science and Engineering Research Council and British Telecom for their support.

REFERENCES

1. Davis, A. L., and Keller, R. M. "Data flow program graphs", *IEEE Comput.*, vol. 15, Feb. 1982.
2. *DSP96002 User's Manual*, Motorola, Inc.
3. *TMS320C40 User 's Guide*, Texas Instrument, Inc.
4. French, S. *Sequencing and scheduling*, Ellis Horwood, 1982.
5. Hu, T. C., "Parallel sequencing and assembly line problems", *Oper. Res.*, pp. 841-848, 1961.
6. Razaz, M. and Marlow, K. A. "Scheduling DSP algorithms for parallel multiprocessor environment", *3rd IMA Conf. Maths. in Signal Processing*, Dec. 1992.
7. Chen, N. F. and Liu, C. L. " On a class of scheduling algorithms for multiprocessor computing systems", *Proc. Sagamore Comp. Con. on Parallel Processing*, pp.1-16, Springer Verlag, N. Y., 1974.
8. Adam, T. L. et al "A Comparison of List Schedules for Parallel Processing Systems" *Comm. ACM 17*, pp. 685-690, 1974.

Transactions on Information and Communications Technologies vol 3, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

9. Comdisco Systems Inc.
10. Mitchell, J. A. " A development environment for DSP", *Electronic Prod. Design*, pp. 43-48, June 1990.
11. Razaz, M. and Marlow, K. A. "Design tools for mapping DSP algorithms onto concurrent architectures", submitted to *Int. Conf. Application Specific Array Processors*, ASAP'93, Italy, Sept. 1993.
12. Vrsalovic, D. F., et al " Performance prediction and calibration for a class of multiprocessors", *IEEE Trans. on Comp.*, vol. 37, No. 11, pp. 1353-1365, 1988.
13. Marlow, K.A. and Razaz,M. "A new precompiler for mapping DSP applications to multiprocessing systems"; submitted to *World Transputer Conference, WTC'93*, Germany, Sept. 1993.
14. Meiko Scientific Ltd., *Meiko hardware reference guide*, Bristol 1992.
15. INMOS Ltd., *Transputer reference manual*, Prentice-Hall, 1988.
16. Proakis, J. G., *Digital Communication, 2nd Ed.*, McGraw-Hill, 1989.