DIRECTED GRAPH METHODS IN COMPUTER SYSTEMS EDUCATION

F. G. Heath

Introduction

I have been working actively in directed graphs applied to all aspects of computer science and engineering since 1970, and it has been on the whole a frustrating time. I became convinced fairly swiftly that in all situations except where the computation is constrained by design to be single-stream that directed graphs are the proper analytical tool for powerful CAD methods equally at home in hardware, software or any mixture.

I became convinced that directed graphs were a good thing, and told other designers that they were a good thing, they agreed that they were a good thing, promised to try to use them and went straight back to their bad old ways. Heriot-Watt still has a directed-graph research program funded partly from the SRC and partly from departmental money (there's conviction for you) which is adding register-transfer language methods to the earlier work (based on the LOGOS project at Case-Western Reserve University run by Ted Glaser who was the real innovator).

If working designers will not accept the methods then we must do the next best thing and slip directed graph teaching into our syllabuses. This is the main topic of my lecture, but in order to help you to appreciate the importance of directed graphs in our subject I intend to spend about half my time on a paper which I presented at the 1976 NATO conference on computer architecture at San Rafael (Appendix). This can of course be read at more length separately.

The Technical Features of Directed Graphs

It must be true to say that almost all computer design, hardware and software, is based on Boolean algebra, programming or flow charts. Boolean algebra is totally unsuitable as a method of designing large complex circuits. Higher level algorithms derived from the algebra (for example minimisation rules) are far-removed from common sense, frequently needing computer processing. Their results baffle the maintenance man. So designers work with familiar high level concepts, stitched together with very elementary boolean functions. Programming is not much better in assembler language programming, since the functions are logical in style and often can be incomprehensible even after a careful reading of the manual. A PDP 11 instruction to write 11001111 to the low byte of a register fills the high byte with 1's. It is hard to convince an engineer that this is a good idea. (I do know why.)

Flow-charts begin to make sense, except that for many people the flow-chart is just an excuse to meander from input to output through a series of boxes representing ill-defined functions. Even so the method gives a good start to design, and in fact the directed graph methods which I want to discuss are really about a

flow-chart which is split into control and data sections and which has sufficient formalism for us to derive valuable system properties from the structure of the computation.

I shall now go quickly through the Appendix, pointing out the essential formalism which has to be introduced, also the analytical methods which can be realised because of the formalism and the useful benefits to the designer in the form of information about his design.

Page 8 of the appendix indicates the general requirements of such a system, followed by (pp. 10-11) a flow-chart for Euclid's algorithm (see ref. 4) which is developed into a block structure diagram (Figure 3b) and then into the full twin directed-graph (bigraph, schema) in Figure 4.

The left-hand graph represents the control function, and may be regarded loosely as a microprogram in logic, with cells (bistables) alternating with simple logic operators. However, the logic operators are not simple, although at first sight the AND seems to be a la Boole. These functions are defined such that as the flow of control goes in one direction (say downward from operator OR1 to operator 2) there is a flow of information upwards from cell 3 to cell 1 which allows OR1 to be activated again.

This is quite essential for orderly design of control nets, and has been endlessly discovered and given different names by designers. Another typical requirement is that the blockhead must have a connection to all operators, and all operators must have a connection to the block-end.

The data graph can be loosely viewed as the data-flow diagram of, for instance, a CPU. However, there can be control nets buried in each data operators as long as there is only one signal-pin which can start the operation and one pin which signals completion. Thus control function 2 is linked to the divide operation by two signals, as shown in Figure 11. Because of the formalism this whole graph can be shrunk to two operators at the next higher level of design (Figure 5), giving an unlimited hierarchy.

Turning to analysis methods, pp. 15-17 of the Appendix show that one can either use forms of Warshall's algorithm (matrix closure) or grow the entire tree of vector state possibilities in order to find out useful properties of the design, and each block can be treated individually. The first is preferable whenever possible, since it needs about one hundredth of the time and a tenth of the memory of the second method.

A very simple analysis will check for hung control states, as well as races/hazards in the design.

Hardware designers may say that all the bistables slow things down - if we can make microcomputers work in parallel we do not need to squeeze the last ounce, or perhaps I should say 32 grams, out of our designs.

It is also possible to do timing calculations in the form

of least time, mean time and max. time for the block.

Resource allocation and fault-tolerant design (really the same calculation) are both possible but there is no avoiding the lengthy tree-growing procedure.

Lastly as a result of work by Douglas Bain, it is possible to design a computation in completely serial form and then process it by the fast matrix method into its maximally parallel form. This is important, not just for the obvious reasons but because Bain has therefore proved that schemas of this type have a canonical reduction.

Before leaving this portion let me answer one possible objection. We know (you may say) that a computation can never be proved correct by its structure : it must be run (or simulated). This is true, and the directed graph methods above will never tell you whether the computation is correct. However, the fact of each block being proved logically sound means that testing or simulation can be brought down to very manageable times. (Of course, being a universal design method you can, if you want, design a system using bigraphs which is quite unreasonable to test.)

Computer Education

There is no doubt that in the early days nobody bothered at all about consistency. As one Ferranti engineer said in the early fifties "We are all going to make a fortune writing articles about logic design", and some people did, usually writing basically the same article several times in different places.

What I have observed since then is that discipline has had to be introduced to the art of hardware design. Each stage of discipline was resisted initially by the practitioners, however once accepted a great improvement in design productivity ensued.

Key design stages where the workers revolted were:-

1) Going from wired chassis to packages (1956).
2) Not being allowed to design new logic for each project (1962).
3) Not being allowed to have an unlimited range of circuits from the same family in a design (1966).
4) Having to use a CAD system for logic layout and interconnection (1968).

What is noticeable about these? We've never taught any of them! So if directed graphs are the new discipline for system designers, why should we treat it any differently than the hardware predecessors?

I think that directed graphs are quite different in quality from these previous design landmarks, and quite definitely worth introducing into teaching methods, both undergraduate and postgraduate. In a way, historical hardware was a necessary step to computer systems since there was no other way. Today however each LSI chip is a separate processor of one sort or another, and most behave like a data-operator in the graphs. The natural design

environment is therefore multi-processor, and all the problems of synchronisation and sequencing need a design discipline.

On the other hand there is little need for a higher level of element - the specification can be processed into MSI/LSI by one designer process, and the directed graph discipline is very suitable.

What about software? Our work has shown me that talking about linking processors often turns imperceptibly into talking about co-operating processes, and sometimes a pair of graphs which started as hardware finish up as software. Sometimes it goes the other way, and a program finishes up as a PROM with simple sequences. So, in my view, the new environment needs directed graphs, and if you can't afford the CAD, using the formal definitions gives considerable insight into what is going on. It is worth noting that the combination of a control operator with data operator and input and output registers is the equivalent of one statement in a Register Transfer language, with affinities with APL.

At Heriot-Watt we have taught the methods to electrical engineers at final Honours and M.Sc. level for the last few years. The course follows one on sequential circuits and is concerned with the design of control networks with proper behaviour. At the M.Sc. research level we have designed and instructed a set of control modules with which we can plug-up a wide range of circuits. These are similar to the macro-modules of Wesley Clark and the PDP 16 "evoke" modules except that certain disadvantages have been avoided by the formal approach. Several Ph.D. students have written their theses on the development of the methods while others have seen the work and incorporated it in their own activities.

Our current view is that a happy mixture of register-transfer language statements and graphics is needed to give the modern designer what he needs, and this is the present aim of our research work. However, it no longer seems impossible, at present hobbies prices, to give each designer his own colour-graphs microcomputer graphics design system and link a number of designers into one more powerful processor, holding the design data-base and carrying out all the major algorithms design tests.

Discussion

Rapporteur Mr. E. Best

Professor Dijkstra : What you show is counter to all my experience, intuition and prejudices. The problem is: I have a feeling that if in a discipline pictures are used for representing anything non-spatial, that is a sign of immaturity of that profession. The second thing I mean is the degree with which you seem to promote the use of gadgetry. You wish us people to use computer aided designing tools. I'm a little bit amazed by that.

Professor Heath : Are you against CAD?

Professor Dijkstra : I'm not for or against; but most certainly I would never use it.

After extensive experiments on how people solve problems, one of the things that I learned was that there is a great tendency to make a mess of it as soon as they resort to mechanical aids such as pencil and paper; and as soon as they use more mechanical aids I expect the mess becomes greater.

Professor Heath : Well I remember a famous lecture that Sydney (Professor Michaelson) gave; it was to about 500 people in Edinburgh (IFIP 68 I think). I remember his saying there that really the way to write programs is to have himself and his two best chaps sitting in a pub with an endless supply of beer, and they could then solve almost any programming problem in the world far better than IBM could. And then I remember the chief programmer for IBM standing up and saying: it's not quite like that, because people have to work together.

My feeling is that what you say is fine if it's just you who is involved. Anyone of us can get to know a machine and its programming so well that pencil and paper become almost superfluous to us; but in industry, or in university where the next lot of students have to mend what the last lot left behind them, recording information is terribly important. Most computers could not be built from a non-CAD basis (the diagrams we all know are never fully accurate). The only reason I have for introducing this is to try and get the computers to a wider audience, not just to experts.

People who are not so immature do use a lot of diagrams and a great number of practising electronic engineers like nothing better than a schematic diagram.

Professor Dijkstra : "Like" - that's true. But I interpret that as a symptom of the way people have been educated.

Professor Heath : Well suppose you take a register transfer language stream of statements, say, defining some specific logic block. Can you quickly modify it from that register transfer listing? I know lots of people who say they cannot.

Dr. Glaser : I think we have to be a bit more careful about the choice of graphics. I'm perhaps a funny guy to say that I'm purely picture-driven. But, seriously, what most people appear not to be aware of is that the braille language which is used by the blind is in essence a string language. There is a rather mature notation that most of us do not think about as being graphical, it is called music. One of the reasons, it actually has been looked at by several people - including Shannon and others - is that it is one of the most dense, most mature coding forms ever developed by man. Braille is purely symbolic and string; and there has never been an adequate format for music for the blind because braille is only one-dimensional. I picked that only as an example. I think that when Fred (Professor Heath) talked about pictures he has integrated some of his work. I think that another way of looking at it might be to say that given the fact that he replaced (potentially) a technology of pencil and paper (or typewriter - I don't want to go back to an older technology such as stone chipping or cuneiform) with the ability to deal with a two-dimensional representation in a two-dimensional language, the formalism in terms of the kinds of operators that can be applied to the structure gains

considerably in significance, and looking at it from the standpoint of symbolic representation we can design much better, the difference being that we are not being destroyed by one dimension.

Professor Edwards : I have been involved in reasonably small machines with about 20,000 integrated circuits; I do not see how you can make that sort of thing without some form of CAD. It's just absolutely impossible.

Dr. Glaser : I can form those in my head, but it is not recommended; it hurts.

Professor Edwards : If you give me a tool, as a designer there is always scope for using it properly as an aid or using it badly; and I would not want to make excuses for designers who use aids in an attempt to do the design, instead of thinking themselves. However, some form of CAD is such an invaluable aid that, as I said, you can't really do without it. The problem that I really want to put to you is why should we use LOGOS as opposed to other techniques which are currently available. What you have not given me is any figure of merit or scale of device that you design, the size of machine that you need to run on, the time involved in doing the process, and the sort of information that you go through to come out with a successful design; or any comparison of that with other exercises like simulation. That is really why, as it were, I can't put LOGOS on some sort of comparitive scale.

Professor Heath : That's a valid criticism. When we discovered that we were in a no-sale situation about 18 months ago, we realised that the great flow of opinion - and justifiably so - was for a register transfer language statement at some stage of the proceedings. It really does seem quite essential. The diagrams are not enough for many aspects of design or manufacture and therefore we changed our tack and decided to add that onto it; whereas, I think, perhaps in your group you were doing register transfer work earlier. Possibly you did not see that problem. But certainly I had one student go through the design of the 8008, for instance, and I'm quite sure that he did it much quicker than Intel did.

The thing that used to get me at ICL (you know the factory floor I'm talking about) was that, say, they would put down the prototype, the designers would sit down with it, and I don't know how many man-months were consumed really getting it off the ground. Now that's the problem. Manufacturers are used to the overheads working in that way, therefore they think it is part of nature. It's not. If you can take a design that is block-structured all the way, and if you know from the analysis that each of those blocks may not do the right thing but is logically sound and works, then you have a great way of getting into simulating it on the one hand or working it up from scratch on the other hand; because (you know what it is like) a piece of equipment gives a different answer every time, you don't know if there is a timing problem or what else have you. With these systems you do get chunks that definitely work if the hardware is functional, and they always do the same thing. That seems to me a terribly valuable feature for shortening simulation.

Professor Edwards : You have not convinced me, though.

Professor Heath : I'm sure I have not. Would you like to know, for instance, that each block of a design is logically sound? If you say, I don't want to know that . . .

Professor Edwards : Let's make the scale ridiculous. If I can only design a decade counter with it, that is not very much use to me, and what I want to know is where does your system start running into difficulties; I want to know the snags.

Dr. Glaser : I think I can answer that. You know I started the work and I am out of it now and I'm probably coming back in as a customer and picking up some of Fred's things. There are two points I can make in terms of an active, ongoing designer right now.

First, there is no simulation for doing what you were saying. In the particular system we are building (probably several versions of it) the simplest one will have 100 independent processors; that's the baby. The big one will have exactly 3500. They range in size from simple chip processors to one with several boards and are dedicated to a very specific task. They are highly asymmetrical, some of them are close-coupled, some of them are not. We have got to get the first one out in approximately a year's time, and there are no other tools around that can give us a chance of doing it. We have done enough simulation to find that simulation in a useful form does not work.

And the other count: by being able to develop a hierarchy of interfaces so that we could say: if this block works correctly it cannot in any way be changed by anything external to it, or, equivalently; if this block works incorrectly then it will still be consistent and will not in any way damage other blocks; now this returns the problem from a totally global interaction of processes to a Markov-type process where we can look at direct linkages. It means that we have reduced the combinatorics to something which is feasible. In itself that's not a very helpful statement. But you know I am in a commercial environment; we have looked at virtually everything else that is available. There is nothing that will touch it. We went once and tried to do it on our simulation model. We currently have a 158 and the company said I could not have it, because that's about what it would take to do the job - full time.

Mr. Pinnell : I should like to commend the use of directed graph methods not so much as a visual tool, but as a means of achieving computability. There was a case in IBM where the specifications of a communications interface were audited by first expressing the interface control sequences as a directed graph, and then tracing the paths so as to find any conditions which had not been provided for. The designer then knew that all possible sequences had been checked.

Professor Heath : And that was without machine analysis?

Mr. Pinnell : No, with machine analysis.

Professor Dijkstra : It took 18 seconds or minutes or so,

and they had to investigate 25000 transition pairs, and the technique was such that they wanted to investigate a network of 3 nodes. The number of transition triples to be investigated would be well over 800,000 to investigate a four-node network, according to that technique - now they were absolutely helpless.

Professor Heath : That sounds like a tree of states.

Professor Dijkstra : And it shows the complete silliness of that whole approach.

Professor Heath : I'm told to close, so let me just do so by saying that after I gave this lecture with great hope in San Rafael last year I came home and heard nothing, until one day a letter came from Eduard Mumprecht in Switzerland, saying substantially: since listening to your lecture, my work on real time programming has been transformed for the better.