Application level microcode to speed data base management

P. A. Dearnley

School of Computing Studies, University of East Anglia, Norwich NR4 7TJ

The paper discusses the impact of user microroutines on machines hosting data base systems. The particular areas of such systems which hold up query processing are identified. The use of static and dynamic microroutines to speed up processing is illustrated by a series of examples. Results are presented showing the advantages gained over low and high level language versions of the same routines. The results show substantial advantages in the use of static microroutines and that dynamic microroutines can be advantageously constructed for even modest size bases.

(Received November 1977)

1. Introduction
Users of current data base systems often find that these systems are slow and large compared with the conventional sequential filing systems they replace. Some of the research work on the data base area is directed towards improving data base system performance. This work can be divided into three general categories: use of data models, incorporation of intelligence and the specialisation of host computer systems. Work in the use of various data models is directed towards improving the logical and physical data structures selected by the user (Codd, 1970; CODASYL, 1971). The incorporation of intelligence into data base systems tries to ensure that future systems will monitor their own performance and direct it towards some agreed goal such as minimum overall operational cost (Stocker and Dearnley, 1974).

Work on specialisation in host computer systems divides into dedicated systems used as 'back end' computers (Canaday et al., 1974) and the design of special hardware or firmware. Current work at the University of East Anglia combines both features in an investigation of firmware for a self-organising data base system resident in a back end computer.

Little analysis has been published on where the bottlenecks in existing systems are. Some measurements have been made on the pilot system built at the University of East Anglia (Dearnley, 1978a). The principle features of this system, apart from self-organisation, are its tabular view of data akin to the relational model and its implementation as a self-contained (rather than host language) system. Detailed study of the measurements reveals that, not surprisingly, most of the time is taken in waiting for disc accesses. The processing which can be overlapped with disc accesses is not a significant problem. However there remain a number of processes which hold up the request for disc access; for example, scanning indexes. Special hardware for scanning discs is becoming available (Coulouris et al., 1972) and development work is continuing (Lin et al., 1976). Research on the back end self-organising system at the University of East Anglia assumes that such hardware will eventually become common and can be incorporated in the system by altering the I/O subsystem and the appropriate costing formulae. The implementation is being performed on a Data General Eclipse S200 computer; this machine is micro-programmed and has a user microprogramming feature, the 'Writable Control Store' (WCS). This feature is being exploited to minimise the hold ups between requests for disc processes by implementing microcode for this specific application. The implementation techniques are discussed elsewhere (Dearnley, 1978b).

2. Application level microcode
The usual use of microcode is to provide a particular machine language environment, as in the IBM System 360 series computers, or to allow additional microroutines to extend the usual repertoire; both techniques take no particular heed of the type of application being run at any one point in time. Application level microcode is provided specifically to enhance a given application and may be re-tailored to this application as it changes. Static microcode is realised by microroutines loaded into WCS at computer start-up time and remaining unchanged whilst the application is running; any parameters required by the microroutine are accessed through registers or main memory. Dynamic microcode is realised by microroutines which are loaded, and possibly changed, at the time that the application is running; the precise nature of the microroutine is tailored to the runtime requirements of the application.

Type 1 dynamic microcode has its microroutine constructed at runtime in main memory; the variables used and tests made vary with the requirements of a particular part of a run of the application program. The microcode is loaded into WCS and the tailored routine is executed—usually many times.

Type 2 dynamic microcode has a skeleton microroutine loaded at computer start-up time; at runtime changes are loaded from main memory prior to each sequence of executions.

All three kinds of microcode have been used in the investigations at the University of East Anglia.

3. Examples
Some of the results from testing and timing a wide variety of microroutines are given in the next section. The microroutines vary from a simple byte string move to the central part of a merge sort; they are discussed briefly below and summarised in Table 1.

1. Byte string move
The Eclipse S200 has hardware for loading or storing individual bytes whilst the data base system holds text in variable length byte strings. The byte string move microroutine is provided as a static routine with all its parameters in registers.

2. Zeroise page buffer
The data base system uses pages of 256 and 248 words, frequently such pages have to be initialised before, say, index entries are placed on them. Again this is a static microroutine with parameters in central processor registers.

3. Page search
This microroutine searches a page to locate all occurrences of a given key fragment. The routine is also given the length and number of records on the page, the position of the key field in
Table 1  Representative examples of microcode

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Method</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte string move</td>
<td>3 in CP registers; subject address; object address; length.</td>
<td>Static</td>
<td>Simple routine required for all data base systems</td>
</tr>
<tr>
<td>Zerosize page</td>
<td>2 in CP register; page address; page type</td>
<td>Static</td>
<td>Simple routine peculiar to specific implementation</td>
</tr>
<tr>
<td>Page search</td>
<td>6; page address; key offset, no. of recs, length of recs, key fragment, pointer to address stack.</td>
<td>Static Dynamic Type 1 and Dynamic Type 2</td>
<td>More complex routine, too many parameters for CP registers, very frequently used algorithm but widely different parameter values</td>
</tr>
<tr>
<td>List merge sort</td>
<td>3; no. of items, list address; address of table of records.</td>
<td>Static</td>
<td>Complex algorithm, too many intermediate results for CP registers.</td>
</tr>
</tbody>
</table>

Table 2  Summary of measurements of static microcode

<table>
<thead>
<tr>
<th>Test</th>
<th>Language</th>
<th>Time per test (msecs)</th>
<th>Relative speed</th>
<th>Theoretical estimate (msecs)</th>
<th>Number of statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte string move (for 256 bytes)</td>
<td>Microcode</td>
<td>0.64</td>
<td>1</td>
<td>0.68</td>
<td>17</td>
</tr>
<tr>
<td>Zerosize page</td>
<td>Asserber</td>
<td>1.65</td>
<td>2.6</td>
<td>1.48</td>
<td>26</td>
</tr>
<tr>
<td>Page search</td>
<td>Microcode</td>
<td>0.041</td>
<td>1</td>
<td>0.041</td>
<td>28</td>
</tr>
<tr>
<td>Merge sort</td>
<td>Microcode</td>
<td>0.33</td>
<td>1</td>
<td>0.23</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Asserber</td>
<td>1.9</td>
<td>5.8</td>
<td>1.4</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>FORTRAN</td>
<td>29</td>
<td>87.8</td>
<td></td>
<td>59</td>
</tr>
</tbody>
</table>

the record and a pointer to a stack for the addresses of records containing the given key fragment. Thus the routine examines a page of index or data for likely records. If no such records exist the next page can be examined with the minimum of delay. If any likely records are pointed to the full key specification can be checked and the index addresses or data extracted. This microroutine has been tested using static and both types of dynamic microcode. The static version uses six parameters on a data stack in main memory. The Type 1 dynamic version is built with the parameters placed into the immediate part of the microinstruction. The Type 2 dynamic version changes parts of the instructions in a skeleton search which is preloaded.

4. Merge sort
This microroutine takes a set of records and produces a linked list giving the records in ascending order before output during a disc merge sort. The microroutine is implemented by static microcode; the three parameters required are held in central processor registers.

4. Results of tests on static microcode
To measure the advantage gained from using microcode in an application the microroutines were also coded in assembly language and, except for the byte string move, in FORTRAN IV. The algorithms used for the different routines were, as far as possible, the same for each programming language. The micoroutines were prepared using a specially written micro-assembler and loader package. The assembly language was prepared using Data General Macro-assembler and the FORTRAN using the normal Data General extended FORTRAN IV compiler. The same test harness and test data were used for each version of the routines. The test harness times itself and any initilisation required without executing the routine under test to allow the overhead of timing to be deducted; each test is then repeated a large number of times and the average execution time calculated. The results are summarised in Table 2.

All the measurements were taken on a Data General Eclipse S200 with the memory mapping and protection feature (MAP) and, both two- and four-way interleaved core memory. The interleaving and MAP make accurate theoretical estimates difficult; those given in Table 2 are based on instruction timing for a four-way interleaved memory ignoring the effect of the MAP feature. The number of statements coded is included to give some feel for the size of the algorithms tested.

The micoroutines are faster than their assembler equivalent due to the parallelism provided in the Eclipse microprogrammable processor and the faster fetch time of the WCS compared with the main memory. A microroutine involving a loop can expect to make considerable savings on the repeated instruction fetch provided that any accesses to main memory can be overlapped with micro-order execution.

5. Results of tests on dynamic microcode
The tests for the Type 1 dynamic microcode consisted of timing the construction and loading of a search page microroutine and executing the routine. The Type 2 dynamic microcode tests excluded preloading the microroutine skeleton and
Table 3  Summary of measurements of dynamic microcode

<table>
<thead>
<tr>
<th>Name</th>
<th>Load time (msecs)</th>
<th>Change time (msecs)</th>
<th>Execution time (msecs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>0.65</td>
<td>—</td>
<td>0.028</td>
</tr>
<tr>
<td>Search page</td>
<td>—</td>
<td>0.27</td>
<td>0.028</td>
</tr>
</tbody>
</table>

only timed the changes to the skeleton and subsequent execution. The results are summarised in Table 3. It will be seen that both the dynamic microcode executions save 0.013 milliseconds per search over the static equivalent. Thus the time required to load the Type 1 microroutine can be recovered if the search is of 50 or more pages. Similarly the time required to change a Type 2 microroutine can be recovered by a search of 21 or more pages.

The dynamic microroutines are faster than static routines due to the 'tailoring' done when they are created. Variables accessed by static routines can be replaced by constants built into the micro-orders of dynamic routines. Similar indicators which may be interpreted at runtime to show, for example, the type of condition to be evaluated can be replaced by the correct test field in part of a micro-order.

Conclusions

The measurements of static microcode show that the execution time of critical parts of the database system which hold up request for disc access can be reduced substantially; the worst figure being a time of around 40% of that taken by assembler for the byte string move and the best, 10% for the page search. Similarly the results of measurements of dynamic microcode show that quite small volumes of data, less than 6K words, are required for the dynamic Type 2 method to be better than the static method.

Since the work on microprogramming and the development of the back end database system are being done in parallel the impact of microcoding on the complete new system will still not be known for some time. However, it is interesting to note that other systems under development such as PRTV and System R (Todd, 1975; Astrahan, 1976) are processor limited and hence the use of special application level microcode on such systems could be applied to a much wider range of operations.

References


CODASYL DATA BASE TASK GROUP, April 1971 Report.


Book review


Despite its description as 'An Advanced Course' this book is much more akin to a set of conference proceedings; the 'course', given in July 1977 and again in April 1978, takes the form of a series of presentations bordering on, or even consisting of, research topics in a number of areas connected with the entire spectrum of operating systems. The presentations, aimed squarely and unashamedly at those who have had both instruction on operating systems, and experience of them, are reproduced here as lecture notes, with a minimum of linking editorial matter.

The course organisers, the editors of this book and its several contributors are to be congratulated. There is a minimum of the repetition which so often mars such a product, and yet the internal linkages between the various facets are present. I found much that was new, and appealing, in terms of new techniques, new results and especially valuable, new ways of looking at familiar situations. The main emphasis is on the abstract modelling of three aspects of current advances in operating systems: the invocation, allocation and control of resources; improvements in reliability by co-operation between software and specific functions of the hardware; and the provision of efficient and secure interprocess communications, especially where the communicating processes are under distinct local managements.

Within its limited objectives, of bringing the specialist or would-be specialist up to date in a rapidly moving subject area, this book succeeds well. Like many similar books, it is produced by photographic reproduction of contributors' typescripts, but the reproduction is of an acceptable standard, although some of the diagrams might benefit from retouching. As an unintended by-product one is given a handy comparison of the quality of the word processing facilities available to each contributor—on balance, that available to Brian Randell is arguably the best. All in all a good buy for the specialist, or the library; not recommended reading for the beginner looking for an introduction to the subject.

M. WELLS (Leeds)