Advanced Summer Institute 1994 in Computer Integrated Manufacturing and Industrial Automation ASI 1994.

AN EFFICIENT IMPLEMENTATION OF[1] MATERIAL REQUIREMENTS PLANNING BY USING CACHE MEMORY

D. Blonis
S. Nikoletseas
D. Sofotassios
A. Spiliou
P. Spirakis
D. Thilikos
V. Triantafillou

Coputer Technology Institute
Patras University, Greece
PO BOX 1122, 26110, Patras, Greece
email : spiliou@cti.gr

Abstract

This paper describes an efficient way to implement a Material Requirements Plan (MRP) which results in reducing the time needed to create this plan. Our approach is based on a) a Directed Acyclic Graph (DAG) structure for the representation of BOM (Bill Of Materials) and b) a complex structure of linked lists to keep demand for parts, while exploding BOM and calculating MRP. During the computation, both structures are stored in a cache memory which allows a very efficient data retrieval.

Keywords: Production Planning and Control, Bill-Of-Materials, Material Requirements Planning, Directed Acyclic Graph, cache memory, current requirements structure.

[1]This work was partially supported by the CEC ESPRIT Special Actions project DELTA-CIME and by the CEC STRIDE project TEXTILE.

1 Introduction

The Production Planning and Control (PPC) function has been a major research topic for many years. Any PPC concepts, in particular those based on the principle of co-ordinated material control such as Material Requirements Planning (MRP) and Manufacturing Resources Planning II (MRP II) [6] rely heavily on the availability of supporting information systems. The introduction of computerized information systems gave a new impulse to the development of new concepts because of the ability to store, manipulate and retrieve large amounts of data.

In most MRP II systems, all computations concerning production plans are based on standard relational tables of a database stored in a secondary memory. This is adequate for Master Production schedules or Capacity plans [6], where the calculations are relatively few. But when we refer to MRP plans, we deal with a great bulk of information due to the infinite time horizon of the production plans and the use of bucketless transactions during the MRP calculation [7]. So, an efficient implementation of the MRP module is the most significant part in the development of a Production Planning and Control System (PPCS).

Such an effort has been made in [8], where an efficient way to improve BOM [1][7] implementation is proposed. More specifically, linked lists are used to represent BOM, which are stored in RAM, in order to achieve a more efficient data retrieval.

This paper describes a complete MRP system using the Directed Acyclic Graph (DAG) structure to represent BOM and a complex structure of dynamic lists to store demand during MRP processing. The data structures are stored in the main memory, which serves as a cache memory, for faster data access. Manipulation of these data structures can be done efficiently through a high level programming language (eg. C).

The rest of the paper is organized in five sections. The first justifies the need to store data in main memory instead of secondary memory, in order to improve the time complexity of the retrievals. The second describes the structure that represents BOM, together with its operational interface. The third section demonstrates the whole MRP processing, including data structures and algorithms used. The fourth section gives an estimation of memory needed to keep the two data structures, and the last summarizes conclusions and presents some open problems.

2 The need of cache memory

In general terms, in a PPCS framework MRP explodes BOM to create the appropriate orders for semi-finished and/or purchased parts, in order to satisfy demand for end products. Assuming that a typical enterprise produces 400 end products which require for their production 5000 semi-finished and/or purchased parts, it is clear that accessing -with SQL queries- the relational tables in the secondary memory [5] to get the necessary information, has a heavy performance cost because:

- A complex BOM query (e.g. explosion of a part) may result in many sub-queries during the traversal of the BOM structure. Each sub-query has to pass the process of syntactic and semantic checking again.
- Each query has to access the database. This means that information should be sent from the application to the database and the results have to be sent back. This communication cost increases in a distributed environment.

To deal with this problem we need a) a storage environment which supports faster access than the secondary memory and b) efficient data structures to store the BOM, together with the necessary programming tools to handle them.
Based on this consideration we adopted the following approach :

- BOM data are stored permanently in relational tables in the secondary memory.
- These tables are transformed in an efficient data structure and are stored in main memory, which serves as a cache memory, while executing the application.
- There is also a dynamic structure (in the cache memory), which keeps the demand for parts of the current BOM level, while exploding BOM and calculating MRP.
- All operations necessary for BOM maintenance and MRP processing are implemented as a programming interface, using the C programming language.

Figure 1 : Product structure of black desk-lamp

3 Bill of Materials

In the manufacturing business, many different products have to be controlled. Not only final and purchased products are important, but also many semi-finished products such as sub-assemblies must be managed. Many products are related to each other, in a way that one product is required to manufacture another. The way in which a product is built up from purchased parts and/or semi-finished products (which in turn consist of other purchased parts and/or semi-finished products) is called the product structure of that product. Figure 1, depitcts the product structure of a final product, the black desk-lamp.

The relationships between the products represent the fact that a product is consumed in the process of manufacturing or assembling another product. A part C which is consumed in the production of product A, is called a component part of A and the product A which consumes C for its production, is called the parent part of C. Moreover, the set of all "A - component part" relationships, are called the Bill-Of-Material (BOM) of A [7].

Data retrieval can be obtained from a product structure by two basic functions named BOM-explosion and BOM-implosion (figure 2). The explosion function lists the component parts of a specific part and the implosion function lists the parents of a specific part. The above functions could be single-level or multiple-level [12].

In the following, an efficient way of implementing BOM, using complex data structures, is described. At this point, it should be emphasized that this paper does not deals with BOM generation which concerns manufacturing process improvements. On the contrary, it considers the product structures given and proposes solutions to faster data access, which concern reduction

Figure 2 : BOM retrievals

in MRP calculation time.

3.1 The BOM structure

The BOM should keep data fully describing which component parts participate in the production of each product. This process begins with the end products and finishes with purchased parts (they may be raw materials or manufactured parts that are being purchased of a particular enterprise). The above consideration suggests a tree structure for the representation of BOM data, more specifically a forest of trees, where a single tree is used to describe the structure of each end product.
Moreover the BOM structure should :

- be organized in an efficient way, in terms of fast data retrieval, as it is exploded many times during MRP calculation
- have the least possible storage requirements, because it is kept in RAM.

The use of tree structure for BOM representation does not fully meet these requirements since :

- a part may participate in the production of more than one end products (figure 3)
- A part may participate more than one times in the production of the same end product (figure 4a).

Figure 3

In both of the above cases, we have information redundancy resulting in wasting RAM space. Additionaly, the tree structure does not provide much efficiency in accessing the stored data.

In order to deal with these problems, we propose the Directed Acyclic Graph (DAG) structure [4] for BOM representation, described below.

Assuming that the structure of a part is uniquely defined we can represent a part structure by assigning nodes to parts and arrows to "parent part - component part" relationships. Thus, we can represent the BOM structure with a directed graph where each part corresponds to a unique node.

For each part, except for the *part id* field which identifies it, we must store additional information necessary for MRP calculations. This information concerns a) the time needed for a part to be produced from its component parts (lead time) and b) the quantity with which each component part participates in the production of the parent part. Lead time is unique for every part, so it is stored in the corresponding node. On the contrary, the quantity that each part contributes in the production of the parent is stored in the arrow indicating their relationship, since a part may participate in the production of many other parts (figure 4b).

It is clear that such a graph can not have circles, because the existence of a circle would mean that a part could participate in its own production. We also observe that the nodes with in-degree equal to zero, represent end products and nodes with out-degree equal to zero, represent purchased parts.

The DAG structure can be implemented efficiently using linked lists [4]. Particularly, we use a linked list, called MAIN LIST, which keeps the parts in increasing order of *part id*. Moreover, for each element of the MAIN LIST, there exist two additional lists. The first, called USE_WHAT LIST, keeps all the component parts of the corresponding part. The second, called WHERE_USE LIST, includes all the parents of the particular part (fig. 5).

(a) Tree sructure (b) DAG structure

Figure 4

Figure 5 : The BOM structure

Every node of the DAG structure contains the following information, necessary for MRP calculations :

- PART_ID: for the identification of the particular part.
- LT (lead time): the time needed for a part to be produced from its component parts.
- LOWLEVC (low level code): the length (number of edges) of the longest path, starting from an end product and finishing at the specific part, in the directed acyclic graph.
- USE_WHAT: a pointer to the list containg the component parts of the particular part. Each element of this list constists of:
  - NEXT_EL: a pointer to the next element of the list,
  - WU_MRPREC: a pointer to the corresponding part of MAIN LIST and
  - QTY: the quantity with which this part (USE_WHAT LIST element) contibutes in the production of its parent (MAIN LIST element).
- WHERE_USE: a pointer to the list containing the parents of the particular part. Each element of this list constists of:
  - NEXT_EL: a pointer to the next element of the list,
  - WU_MRPREC: a pointer to the corresponding part of MAIN LIST and
  - QTY: the quantity with which the current element of MAIN LIST contibutes in the production of its parent (WHERE_USE LIST element).
- NEXT_EL: a pointer to the next (in increasing order) element of the list.

The use of DAG structure for BOM representation, has the following advantages:

- Reduction in memory space, since each part is stored once and the necessary relationships are achieved with the use of pointers (when a part participates in the production of many other parts or many times in the production of the same part).
- Direct O(1) time access to all the component parts of a particular part (explosion).
- Direct O(1) time access to all parents of a particular part (pegging).

The programming environment of e.g. the C language has powerful capabilities in creating and managing linked lists, using pointer operations [11].

3.2 BOM operations

The operations performed on the proposed BOM structure are the folowing:

- **bom_init_list :** creates the structure and loads the necessary information from the relational tables of the database.
- **bom_ins_list :** inserts a new part in the structure in a way that MAIN LIST remains in increasing order of low level code.
- **bom_upd_list :** updates information related to a part structure.
- **bom_del_list :** deletes a part structure.
- **search_bom_list :** searches, using the binary search technique, the MAIN LIST of BOM structure to find a specific part.
- **bomsex :** performs the single explosion function that gives the component parts of the next lower BOM level, for a specific part.
- **bommex :** performs the multiple explosion function that gives the complete structure of a specific part (up to raw materials).
- **bomsim :** performs the single implosion function that gives the parent parts of the next higher BOM level, for a specific part.
- **bommim :** performs the multiple implosion function that gives the paths from a specific part to all its parents (up to end products).

The first operation is performed every time a user enters the application. The operations of insert, update and delete a part structure, are rarely performed (when the process plans are modified) and they affect both the database and the cache memory structure. The remaining operations do not affect or change BOM. On the contrary, they refer to data retrievals that are very useful in MRP processing and performed in the cache memory. They are generaly time consuming operations, especially the multiple-level functions, but the the BOM structure and the fact that it is stored in RAM, considerably reduce the running time.

4 Material Requirements Plan (MRP)

MRP logic is based on the observation that in order to have a product available in a particular date, the quantities of the parts needed for its production (component parts) should be available in a time that is at least equal to the lead time of the product [7]. Consequently, demand for end products generates demand for a number of other products and this, in turn, demand for new products, and so forth raw materials. Due to the fact that the dependent demand for a part may come from more than one parent, the situation gets complex enough and leads to the conclusion that an order between parts must be defined. This order is achieved with the use of low level code [1]. Low level code is the length of the longest path, from a node with in-degree equal to zero (end product) to the node corresponding to the particular part and is kept in the BOM structure.

The use of low level code comes from the fact that in order to calculate demand for a part, the demand for all the parts that have this particular part as their component part must have been calculated. Hence, low level code defines an order between parts and organizes them in levels. According to this, MRP processing is executed in levels starting from end products (top) and ending to raw materials (bottom). Each level, generates "material requirements", an information which is dynamic and is stored in a structure, called current requirements structure which is described below.

Figure 6 : Current requirements structure

4.1 Current requirements structure

As described above, MRP processing needs to keep the current requirements that the products generate for their component parts on each level of BOM structure. These requirements are stored in the current requirements structure (figure 6) which consists of two types of linked lists.

Each element of the first list, called MAIN LIST, corresponds to a part with low level code greater than or equal to the current BOM level. The elements of this list are ordered according to the *part id*. Each of these elements consists of the following fields :

- PART_ID
- NEXT_EL : a pointer to the next element of MAIN LIST
- DEM_LIST : a pointer to the second list called DEMAND LIST.

DEMAND LIST holds all the requirements (expressed by date and quantity) of the current MAIN LIST element, which have been produced as a result of its parents demand. DEMAND LIST elements are order according to increasing demand date and consist of the following attributes :

- DEM_QTY : the quantity of a single requirement,
- DEM_DATE : the date that the specific requirement should be satisfied and
- NEXT_EL : a pointer to the next requirement of the current part.

Current requirements structure is a dynamic structure since in every level the requirements of each product are deleted at the moment they complete the generation of the requirements for its component parts. Current requirements structure is created, updated and deleted (finaly) during the MRP processing.

4.2 Current requirements structure operations

The operations performed on current requirements structure are the following:

- **mrp_init_list :** generates requirements for end products. This means creation of both MAIN LIST and DEMAND LISTs.
- **create_new_requirements :** insertion of a MAIN LIST element (new part) or a DEMAND LIST element (new requirement) for some part that already exists in MAIN LIST.
- **mrp_del_list :** each requirement of a MAIN LIST element generates requirements for its component parts. As soon as this process is completed for all the components, the part and its own requirements are deleted from the current requirements structure.

4.3 The MRP algorithm

Let n be the current level. The MRP algorithm has the following steps :

- **Step 1 :** Determine the gross requirements [7] for all products of the current BOM level per time period. This information exists in current requirements structure.
- **Step 2 :** Determine net requirements N(t) for the parts of level n at time period t. This comes from the relationship

$$N(t) = G(t) - S(t) - H(t-1)$$

where G(t) are the gross requirements at the current time period t, S(t) are the scheduled receipts of the current time period and H(t-1) is the existing inventory of the previous time period. If N(t) ≤ 0, set $N(t) = 0$.

- **Step 3 :** According to the lot policy [7] generate the production requirements (orders) P(t) (quantity and date) for all the products of the current BOM level for the specific time period t. If $N(t) = 0$ then $P(t) = 0$. Otherwise the quantity and the date of the order are determined from the lot policy.

- **Step 4 :** Calculate the current inventories H(t) for the products of level n at time period t, using the formula

$$H(t) = S(t) + P(t) + H(t-1) - G(t)$$

- **Step 5 :** If all the time periods are completed go to step 6. Otherwise continue to the next time period (set $t = t + 1$) and go to step 1.

- **Step 6 :** Determine scheduled releases R(t) of all production orders of step 3 for all products of the current level. If the lead time of the product is L, then $R(t - L) = P(t)$.

- **Step 7 :** If the requirements of the products of all levels have been determined, then stop. Otherwise, continue to step 8.

- **Step 8 :** Determine requirements of all products of the next BOM level. This is the outcome of the single level explosion of BOM structure and inserts new products in the current requirements structure or new requirements for already existing products.

- **Step 9 :** Continue to next BOM level, setting $n = n + 1$, and go to step 1 again.

5 Memory considerations

This chapter gives an examble of the estimation of the required memory for MRP processing of a typical enterprise, to demonstrate that the memory overhead is not very large and thus is trade off by the reduction of processing time.
Consider an enterprise with :

- 200 end products which need for their production 10000 component parts

- every part having (an average of) 8 component parts
- every part having (an average of) 4 parents
- the last BOM level corresponding to (an average of) 800 purchased parts

Each MAIN LIST element (of BOM structure) has the following space requirements:

part_id	16 *Bytes*
lt	4 *Bytes*
lowlevc	4 *Bytes*
next_el	4 *Bytes*
use_what	4 *Bytes*
where_use	4 *Bytes*
Total	36 *Bytes*

Each USE_WHAT LIST element has the following memory requirements :

qty	8 *Bytes*
uw_mrprec	4 *Bytes*
next_el	4 *Bytes*
Total	16 *Bytes*

Each WHERE_USE LIST element has the following memory requirements :

qty	8 *Bytes*
wu_mrprec	4 *Bytes*
next_el	4 *Bytes*
Total	16 *Bytes*

According to this, every part of the BOM structure occupies an average of

$$36 + 8 * 16 + 4 * 16 = 228 Bytes. \tag{1}$$

For a number of $10000 + 200$ parts we need $228 * 10200 = 2325600$ *Bytes* for the BOM structure.

Besides the BOM structure, an amount of memory is needed for the current requirements structure. This structure is dynamic; it changes during MRP processing and every time it keeps the requirements of a single BOM level. In the worst case, current requirements structure keeps requirements for the last BOM level (purchased parts). This is the case when we have a *pyramid BOM*. It could be the first BOM level (end items) for an *inverted-pyramid BOM*, etc. (see [7]). In any case the maximum length of current requirements structure is known and is determined by the BOM structure.
Each MAIN LIST element of the current requirements structure occupies :

part_id	16 *Bytes*
dem_list	4 *Bytes*
next_el	4 *Bytes*
Total	24 *Bytes*

Each MAIN LIST element (part) corresponds to a DEMAND LIST which occupies memory propotional to the number of orders for the specific part, within planning period.
Each DEMAND LIST element occupies :

dem_qty	4 *Bytes*
dem_date	4 *Bytes*
next_el	4 *Bytes*
Total	12 *Bytes*

According to the above we need $12 + 24 = 36\ Bytes$ (2)
for each part of the current requirements structure. Assuming that the average number of planned orders for a part in a six month period is 20, we need

$$36 * 20 * 800 = 576000 Bytes$$

to store current requirements structure. Consequently, we need less than 3 *MBytes* of memory for MRP processing. We consider this amount of memory reasonable, compared to the significant reduction of processing time.
In any case, the size of memory required is predictable and equal to :

$228 * E * C\ Bytes$	for the BOM structure and
$36 * P * M\ Bytes$	for the current requirements sructure

where

228	:	size of each element of the BOM structure (see eq. 1)
E	:	number of end products
C	:	number of component parts
36	:	space occupied by each part of the current requirements structure (see eq. 2)
P	:	average number of planned orders in a six month planning horizon
M	:	number of parts corresponding to the last BOM level

6 Conclusions - Open problems

The MRP system described above is implemented with the following restrictions:

The use of DAG is based on the assumption that the structure of all products is uniquely defined. If this is not true, we can assume the parts to be separate with each corresponding to a different structure.

We also assume that the memory space is enough to accomodate both structures (BOM and the current requirements structure). Otherwise, some swapping mechanism has to be implemented at a lower level (e.g. through appropriate system calls to the operating system).

In the case that concurrent updates on the BOM structure occur, a locking mechanism is needed. Such a mechanism can be implemented using the RDBMS concurrency control facilities.

The described MRP system has been implemented on the INGRES RDBMS as a part of an integrated PPCS in the context of the TEXTILE/STRIDE project (see [10]). The following open questions could lead to further improvements: How good is the linked list data structure for the representation of BOM objects ? Or, how can someone model product data so that the resulting BOM structure should be more efficient from both the designer (production manager) and the developer point of view ? In our test case, the MRP system takes the BOM objects as given and fixed (several trees with different levels and fan-outs) and does not attempt to optimize their structural properties. An interesting extention concerns application of the

cache memory architecture to cover other PPC applications too, e.g. bill of operations applications.

References

[1] J. BROWNE, J. HARHEN, J. SHIRMAN, "PRODUCTION MANAGEMENT SYSTEMS", 1988, *ADDISON-WESLEY*.

[2] Y. CHUNG, G. W. FISCHER, "ILLUSTRATION OF OBJECT-ORIENTED DATABASES FOR THE STRUCTURE OF A BILL OF MATERIALS", COMPUTERS IN INDUSTRY 19, 1992, 257-270, *ELSEVIER*.

[3] R. COMPANYS, P. FALSTER, J. BURBIDGE, "DATABASES FOR PRODUCTION MANAGMENT", 1990, *NORTH-HOLLAND*.

[4] K. MEHLHORN, "GRAPH ALGORITHMS AND NP-COMPLETENESS", 1984, *SPRINGER-VERLAG*.

[5] E. KORTH and A.SILBERSCHATZ, "DATABASE SYSTEM CONCEPTS", 1986, *MCGRAW HILL*.

[6] J. A. ORLIKY, "MATERIAL REQUIREMENTS PLANNING",1975, *MCGRAW HILL*.

[7] S. SMITH, "COMPUTER BASED PRODUCTION AND INVENTORY CONTROL", 1988, *PRENTICE HALL*.

[8] F. STEYER, "SYSTEM ARCHITECTURE AND SPECIFICATION OF A FAST BOM OBJECT PROCESSOR USING A STANDARD RELATIONAL DATABASE MANAGMENT SYSTEM AND A MAIN MEMORY CACHE", DEXA PROCEEDINGS, 1991, PP 487-490.

[9] D. THILIKOS, D. BLONIS, S. NIKOLETSEAS, D. SOFOTASSIOS, A. SPILIOU, P. SPIRAKIS, V. TRIANTAFILLOU, "IMPLEMENTATION OF A FAST MATERIAL REQUIREMENTS PLANNING PROCESSOR USING CACHE MEMORY", TR 93.07.29, *CTI PATRAS*.

[10] D. THILIKOS, D. BLONIS, S. NIKOLETSEAS, D. SOFOTASSIOS, A. SPILIOU, P. SPIRAKIS, V. TRIANTAFILLOU, "FINAL REPORT FOR THE PRODUCTION PLANNING AND CONTROL SYSTEM", *CTI PATRAS*, INTERNAL REPORT.

[11] C. VAN WYK, “DATA STRUCTURES AND C PROGRAMMS”, 1988, *ADDISON-WESLEY*.

[12] E. A. VAN VEEN, “MODELING PRODUCT STRUCTURES BY GENERIC BILLS-OF-MATERIALS”, 1992, *ELSEVIER*.