Parallel Mixed Integer Programming - A Status Review

V. Nwana and G. Mitra
Department of Mathematical Sciences,
Brunel University, West London,
UB8 3PH, UK
Email: mapgvln@brunel.ac.uk, gautam.mitra@brunel.ac.uk

Abstract

Integer programs (IP) and mixed integer programs (MIP) are computationally intractable (*NP-Hard*). Considerable theoretical and applied research has been expended to design progressively superior solution methods for processing this class of problems. In addition to theoretical progress, algorithms and solution techniques that exploit novel computing technologies also achieve improvements in computational performance. Maturing of parallel computing hardware platforms and software tools have motivated many researchers to develop algorithms which achieve processing speed-up and robust fault tolerant performance. In this paper, we review the leading issues in designing and implementing such solution methods. In particular, we discuss parallel architectures that have been chosen to implement branch and bound, the de facto algorithm for solving integer and mixed integer problems. We also review the current state-of-the-art of MIP parallel solvers by examining critically the parallelisation strategies used and assess how well these strategies capture the structure of the MIP solution algorithm.

KEYWORDS: Mixed Integer Programming, Branch and bound, Parallel computing

Contents

1 Introduction and overview

There are numerous practical problems that are modelled and solved as integer programs (IP) and mixed integer programs (MIP) and the range of applications continue to grow. Vehicle and crew scheduling, VLSI routing, graph theoretic problems, production planning and several combinatorial optimisation problems are typical examples of such applications. The context of these problems sometimes requires real-time solutions, hence there is a need to solve the problems within acceptable time frame. There is also a need to solve progressively larger MIP models that arise in industrial settings.

Unfortunately, MIPs are extremely difficult to solve and even small models can require vast computational resources, commensurate with the theoretical *NP-Hard* [25] classification of these problems. While many algorithms have been successfully developed for particular instances of MIPs, the development of a generic algorithm that efficiently tackles the general MIP problem remains the 'holy grail' of optimisation research. Most of the solution methods developed for the general MIP are based on the branch and bound (B&B) algorithm – a 'divide and conquer' technique which progressively searches for an integer feasible solution in successively smaller subspaces of the entire search space defined by the MIP.

Two problems plague the applicability of B&B in solving MIPs. Firstly, although there have been considerable algorithmic advances (such as introduction of cuts, intelligent heuristics for variable and node choices, better bounding techniques), the overall usefulness of the algorithm may sometimes be limited by unrealistic solution times. Secondly, branch and bound proves optimality by a complete (albeit implicit) enumeration of the search space. Quite often, proving optimality becomes intractable because of the sheer size of the problem and one is more often than not, content with a 'good' solution in place of an 'optimal' one. In this case, B&B behaves as a heuristic.

Parallel computing has been put forward as a method of overcoming the limitations of the serial B&B algorithm. [27][59][3]. Parallel computing technologies have continued to advance and their benefits have been noted in other scientific fields such as digital image processing, pattern recognition, and distributed artificial intelligence (DAI). The solution techniques for these applications share a common characteristic: they consist of several demanding loosely-coupled independent processing operations which can be naturally executed on multiple processors concurrently. The tree structure of B&B for MIP, with its independent nodes, each requiring a substantial amount of processing, makes it a natural candidate for parallel computing.

Most of the established methods for solving the general MIP rely upon the B&B algorithm structure. It can therefore be said that 'parallel MIP solution techniques' is tantamount to 'parallel branch and bound for MIP'. In this paper, our main goal is to review the broad area of parallel mixed integer programming. We position MIP optimisation within the context of parallel computing and appraise the rationales, design objectives and challenges of the resulting algorithms. We also evaluate some of the state-of-the-art implementations in academic and industrial research settings.

The contents of the paper are organised in the following way. In section 2, we present the MIP model definition and outline the essential aspects of B&B – the *de facto* solution technique for MIP. We also provide an insight into those features of the B&B algorithm that can be fine-tuned to improve its efficiency. Over the last two decades, research in High Performance Computing has led to alternative models of parallel computing - each with its strengths and weaknesses. In section 3, we provide a brief overview of parallel computing and highlight the concepts therein that may be useful in parallelising B&B. We also investigate the alternative parallel platforms and parallelisation strategies with a view to assessing their suitability for parallelising B&B for MIP. We also discuss the assessment criteria for measuring the performance of the resultant parallel MIP algorithms, as well as issues pertaining to the effective use of the parallel architecture in implementing the algorithms. In section 4 we discuss the computational structure of parallel B&B (PB&B) as well as its extension, parallel branch and cut (PB&C). In Section 5, we review and critically analyse the leading implementations of parallel MIP reported in the literature. We conclude and suggest possible areas of extension of research into parallel mixed integer programming.

2 MIP Models and Algorithms

2.1 The MIP Model Definition

We define the MIP reference model using the following *index sets* and *model coefficients*.

Index sets

$B = \{1, \cdots, \lvert B\rvert\}$,	index set for binary variables;
$I = \{\lvert B\rvert + 1, \cdots, \lvert B\rvert + \lvert I\rvert\}$,	index set for integer variables;
$C = \{\lvert B\rvert + \lvert I\rvert + 1, \cdots, \lvert B\rvert + \lvert I\rvert + \lvert C\rvert\}$,	index set for continuous variables;
$N = B \cup I \cup C$,	index set for all variables;
$M = \{1, \cdots, \lvert M\rvert\}$.	index set for all the constraints

Model coefficients

$l_j, u_j, c_j, a_{ij}, b_i, (i \in M, j \in N)$ are given values of the vector and matrix coefficients used to describe the problem.

The general MIP is defined compactly as

$$Min \qquad Z_{MIP} = \sum_{j \in N} c_j x_j \tag{1}$$

subject to

$$\sum_{j \in N} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, |M| \tag{2}$$

$$l_j \leq x_j \leq u_j \tag{3}$$

In an expanded form, the MIP can be defined as

$$Min \qquad Z_{MIP} = \sum_{j \in B} c_j x_j + \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j \tag{4}$$

subject to

$$\sum_{j \in B} a_{ij} x_j + \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, |M| \tag{5}$$

$$l_j \leq x_j \leq u_j \tag{6}$$

If $B, I = \emptyset$ then $N = C$. The problem simplifies to a linear program (LP).
If $I, C = \oslash$ then $N = B$. This problem is called a Binary Integer Program (BIP) or Pure Zero-One Integer Problem (PZIP).
If $C = \oslash$ then $N = B \cup I$. This problem defines a Pure Integer Program (PIP).
If $I = \oslash$ then $N = B \cup C$. This problem is called a Mixed Binary Integer Program (MBIP). This class of IP can also be referred to as Mixed Zero-One Integer Program (MZIP).

For further discussions on the model definition and problem classification, the reader is referred to Nemhauser and Wolsey [51].

2.2 Serial B&B algorithm

The term *branch and bound* (B&B) refers to an enumerative technique which was initially used by Little *et al* [41] for solving the Travelling Saleman Problem (TSP). It was first used by Land and Doig [37] to solve the general MIP problems. A B&B algorithm follows the familiar idea of divide and conquer. In other words, if it is too difficult to consider the set of all integer

feasible points at once, the solution space is partitioned and the problem is solved by optimising severally over smaller sets and then putting the results together. The reader is referred to [47][4][5][41] for the original ideas in B&B.

The B&B procedure in its simplest form can be described as a *tree search* which is characterised by the rules which perform the *branching* and *bounding* of the solution space. The tree to be searched is composed of *nodes*, representing subproblems, and *branches* on variables on which lower and upper bounds are imposed. A node corresponds to a subproblem which is created from its parent (previously created subproblem). A branch links two nodes (parent and son) where the latter is derived from the former by setting a new integer bound on one of its integer variables with non-integer value in the parent's solution. The objective function value of the parent is the lower bound on the objective function of its son. At each node of the tree, a relaxation to the MIP, usually the *linear programming relaxation* (LPR), which drops all integrality constraints, is solved. If there is no feasible solution to the LPR of a sub-problem/node then this node is terminated. If the solution of the LPR of a subproblem has no fractional value for its integer variables then the objective function value of this subproblem is set as an upper bound for all remaining subproblems (waiting nodes). The subproblem/node is then called the *incumbent node* and its optimum objective function value is called the *incumbent value*. After each branching, those subproblems with an objective function lower bound that exceeds the incumbent value are excluded from further branching. The branching continues until the best integer feasible solution is found and its optimality is proven by examining all the eligible nodes in the search tree.

The efficiency of B&B lies in the fact that the algorithm does not require a complete enumeration of all feasible solutions in the search space. In fact, a reasonable way of assessing how well B&B performs in arriving at a given feasible solution within a given time frame is by considering the number of nodes in the B&B tree. If a complete enumeration of the search space defined by an MIP model were to be carried out, the B&B procedure will have 2^n nodes (assuming a tree which, when fathomed has a depth of n branches), each of the same complexity as the LP. It is essential therefore to exploit the advances in LP research so as to reduce the total amount of processing time taken to solve the LPs within B&B. An analysis of the computational behaviour of LPs is given in appendix 1.

Since a large proportion of the time and computational effort in B&B is spent on solving LPs, a logical way of improving the speed and efficiency of the algorithm is to limit the number of nodes that are investigated by B&B. Limiting the number of nodes in a B&B tree is tantamount to avoiding un-

necessary branching. A large amount of branching can be forestalled by obtaining a good feasible solution early-on in the search. This effectively excludes portions of the search space that would otherwise have been searched. A second approach to curtailing the size of the tree is by dynamically adding *cutting planes* to the *subproblems/nodes* in the B&B tree. This has the effect of 'cutting off' portions of the search space, thereby improving the linear programming relaxation polytope for the MIP.

Inherent within the branching operation are two choices that can influence the level of success of B&B for MIP. First, is the choice of node to branch off from and second a non-integer (fractional) variable to perform branching on. The theory that guides these choices is beyond the scope of this paper. Some techniques for choosing a branching variable include *pseudocosts* or *priority* ordering (based on some a priori knowledge of the problem), while commonly used node choice strategies are *depth-first*, *breadth-first* and *best-first*. The reader is referred to Nemhauser and Wolsey [51], Johnson *et al* [31], Linderoth and Savelsbergh [39] and Mitra [48] for a discussion of these issues. The combination a variable choice technique and node choice technique is referred to as a *search heuristic* and a good choice of a search heuristic may result in smaller trees.

2.3 Computational behaviour of B&B

MIP problems are usually processed by first computing their LP relaxed solutions which provide immediate lower bounds. For large LPs, *interior point methods* (IPM) have proved to be most successful [45]. However, in the context of B&B, when a family of LP relaxed subproblems are considered, IPM loses out to SSX. This is because it is not possible to make good use of optimum solution values for a given LP to compute the optimum solution to another "neighbouring" LP with slightly revised model (bounds) parameters. The process of solving slightly modified problems using previously known optimum solution of the original problem is referred to as *warm start*. It has been well known for some time that the SSX possesses very good warm-start properties. Thus simplex solvers are well positioned to exploit the B&B hierachical tree structure of the family of subproblems. This is because the use of the stored optimum basis of the parent subproblem and applying the dual algorithm provides an "advanced starting point" in the solution of the subproblem. For a detailed discussion of issues pertaining to SSX, the reader is referred to Maros and Mitra [46].

However, as shown in Figure 1, there are several alternative ways of applying the simplex method to the subproblems. SSX can be used to solve the LPs at each node in one of the following ways:

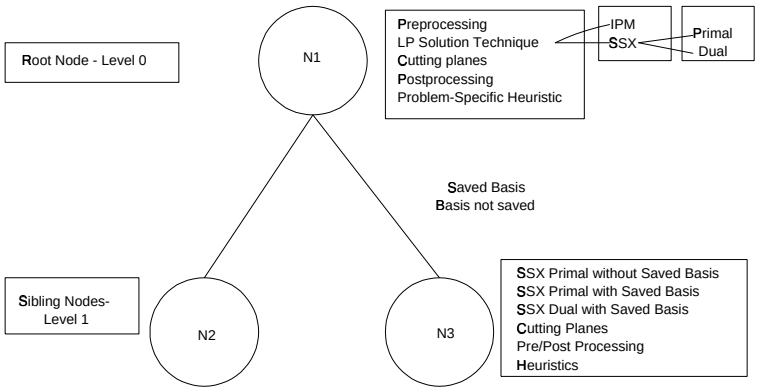

Figure 1: Alternatives at nodes of B&B tree

1. Each subproblem is solved 'afresh', not taking into account the optimal basis of the parent node and ignoring the hierarchical relationships expressed in the tree. This is of course a 'naive' approach.

2. The hierarchical tree relationship is taken into account by saving the basis of the optimal solution of a parent node. The saved optimal basis of the parent node is then used as a starting basis for the phase 1 primal SSX, in solving the sibling node.

3. Given the optimal basis of a parent problem, each of that node's subproblems is dual feasible and primal infeasible (bound violated). It is then meaningful to apply a dual or parametric algorithm that takes advantage of this property.

4. Recently, methods of pre-analysis of LP problems prior to applying the simplex algorithm have been introduced within the B&B search procedure. In this approach, both the relaxed LP and the IP problem are analysed with a view to reduction, simplification and inferencing. This is usually done by applying a bound analysis procedure now well known as *presolve* or *preprocessing* due to Brearley *et al* [1]. Further work on preprocessing has been reported by Savelsbergh [60] and Kularajan *et al* [34]. LP and IP preprocessing may be applied at the root node and at any or all of the other nodes in the B&B search tree.

These four alternatives represent a progressively sophisticated approach which encapsulates the structure of SSX algorithm. The results of a simple experiment illustrating the usefulness of capturing the hierachical relationship in the tree (1, 2, 3) and the structure of the model (4) is reported in appendix 1 and [53].

3 Parallel computing - The relevant issues

3.1 Motivation

There have been meteoric improvements in the performance of *Von neumann* single-processor computers. Even with these improvements some MIPs cannot be solved in an acceptable time. A newer model of computation, *parallel computing*, is designed so that several processing elements combine and cooperate in solving a single large task. Whereas in the traditional model a large task is performed serially, one step followed by the other, the parallel model decomposes a task into smaller sub-tasks that can be performed concurrently, that is, in parallel, with some degree of coordination. To identify a task for parallel implementation, one may either decompose the series of operations inherent in the task, or partition the data that is processed by the same set of operations. The former is referred to as *functional* or *control* decomposition, and the latter as *domain* or *data* decomposition of the problem. Kumar *et al* [35] outline issues which need to be examined in order to use parallel computing effectively. These include the design and choice of parallel computers, languages and software tools, which we classify as *hardware and system-software* issues and, the design of efficient and portable parallel algorithms as well as methods for evaluating them are classified as *software* issues.

As far as the implementation of parallel B&B for MIP is concerned, there is less versatility in the hardware and system software issues. The amount of control we have in these issues is limited to choice of the best parallel computer, language and computer tools, respectively. We have more control, however, in the software issues – we can design PB&B algorithms efficiently to exploit the chosen platform, implement programs that are portable from one platform to another and/or devise different methods for evaluating algorithms. This section discusses issues pertaining to "hardware and system-software" issues and the following section addresses specific issues in developing parallel software based on branch and bound for MIP.

3.2 Parallel architectures

There are different ways of classifying computer architectures. One of the most used classifications [49][13][27] in parallel computing literature was proposed by Flynn [24]. Flynn's taxonomy captures the essential aspects of the parallel computing model and is based on the relationship between the instruction and data streams of the processor(s), sometimes referred to as the *control parameter*. The four classes that constitute the classification are: single instruction stream single data stream(SISD), single instruction

stream multiple data stream (SIMD), multiple instruction stream multiple data stream (MIMD) and multiple instruction stream single data stream (MISD) models. In the *SISD* model, individual processing units perform a similar set of instructions simultaneously on a single data set. The classic von neumann computer is an example of such an architecture. The *SIMD* model represents a very specialized kind of computers in which multiple processors carry out the same instruction on different pieces of data. This model, while appealing because of its potential reduction in hardware and software complexities, can only be applied to problems characterised by a high degree of regularity. For example, they can be used in image processing. Examples of such computers are the MasPar MP-1, MasPar MP-2, CM-2 and MPP.

In *MIMD*, each of the multiple processors has a control unit that acts independently on the data provided. This gives a programmer sufficient versatility to run different sections of the program in parallel. *SIMD* and *MIMD* architectures are bespoke to the *SPMD* (Single Program, Multiple Data) parallel programming model because of their ability to handle multiple data simultaneously. The multiple processors that act on the multiple data sets are usually coupled via shared memory or communication links. Shared memory is characteristic of a *multiprocessor* computer while communication links usually involve a *multicomputer* network with distributed memory. In the former, processors may read and write to a common accessible memory location, while in the latter, communication between the processors is via *message passing*. In message passing systems, the network topology determines how the communication is achieved. The topology can either be *farm*, *ring*, *mesh* or *tree* [8].

The strengths and weaknesses of shared memory and distributed memory MIMD computers are as a consequence of their respective design philosophies. In the former, there are few communication issues to be considered; thus simplifying its programming. However, the concept of processors reading and writing to a common memory in an asynchronous way is highly theoretical as the limited bandwidth of the interconnection network (bus) limits the speed of access of information from the common memory. Moreover, a memory hierarchy is usually used to prioritise access of information from the shared address. The main advantage in a distributed memory model is scalability; meaning that there is no theoretical limit to the number of processors that comprise the parallel machine. The programming of a distributed memory computer is considerably more complicated than a shared memory computer because the programmer needs to control the communication between the processors. However, several message passing systems have been developed to aid the programmer with communication issues and this model of parallel computing is arguably the most widely used model. Consequently,

it is worth identifying the communication issues involved in using a MIMD model and how best to deal with them.

Communication bottlenecks in distributed MIMD emerge from three sources. First, by definition, there are communication overheads inherent in the network topology. Secondly, the message passing systems usually incur communication overheads in packing, sending and unpacking messages(instructions) and data over a network. Finally, the amount of useful information shared over the network is the programmer's responsibility and can be instrumental in designing a good parallel algorithm. The size of data handled by each processor is referred to as the granularity of the algorithm. Fine-grain algorithms handle small amounts of data while coarse grain algorithms deal with larger data sizes.

Several system software exist to harness the computing power of a distributed network (or distributed networks) of workstations/Pcs. *Parallel Virtual Machine* (PVM)[26] and *Message Passing Interface* (MPI)[62] are some of the most widely used message passing systems. MPI and PVM share some similar goals; MPI however strives to define syntax and semantics of its message passing library in a bid to establish a message-passing standard which ensures portability. PVM has been used in a majority of parallelisations of the general parallel B&B as well as parallel B&B for MIP [49] [30] [15] [29] [17]. Other parallel software platforms that have been used for parallel MIP include TreadMarks [33] by Bixby *et al.*

3.3 Choice of platform for PB&B for MIP

Linderoth [40] notes that the current trend in implementing PB&B algorithms is skewed towards a distributed memory architecture. He advances two main reasons for this. First, since dedicated parallel computers are very expensive, a network of personal computers presents a rational as well as an economic way of achieving parallel processing capabilities. Second, there are no physical limits to the number of processors that can be used within a distributed network system to form a *massively parallel* machine. Theoretically at least, if we use this model of parallel computing, computational power may be appended when needed for larger models. In practice however, the communication costs incured by having a large number of processors inhibits the use of too many processors. Moreover, a distributed memory machine with a large number of processors requires very *fine-grain* implementations of algorithms. This sets a dangerous precedence as the implementer may be faced with a situation in which it takes longer to instruct the worker (node) what to do than how long it takes to do it. Laundy [38] is more ambitious in his choice of platform for implementing PB&B. He advocates the develop-

ment of a system which runs well on both a dedicated parallel computer and on heterogeneous networks of PCs and workstations. In our opinion, the advantages of low cost and scalability make a distributed memory architecture the most viable type of architecture for parallelising MIP algorithms. In fact, as seen in section 5, most of the state-of-the-art implementations of parallel B&B use distributed MIMD as the parallel implementation platform.

3.4 Performance Measures of Parallel Algorithms for MIP

The rationale for applying parallel computing techniques to MIP is to improve solution times of problems as well as to solve larger problems in acceptable time. *Speed-up* and *scale-up* defined contextually below are two measures that have been proposed to capture the extent to which a parallel MIP algorithm achieves one or both of the above objectives.

Speed-up

If we are faced with solving an MIP and suppose that the 'best' time taken by a serial algorithm to solve the problem is t_s. Assuming that the same amount of work is done by a parallel algorithm in solving the same problem using n processors in t_p seconds. We define *speed-up*, μ as:

$$\mu = \frac{t_s}{t_p} \tag{7}$$

, the ratio of wall clock times for the serial and parallel executions of the algorithm. We ideally want a linear speed-up but, Amdahl's law [2] implies that speed-up in a program that consists of sequential and parallel computational components is bounded. The reader is referred to [2],[35] and [8] for the implications of amdahl's law.

Scale-up

Scale-up in the context of an MIP problem makes it a more realistic and challenging design goal of a parallel algorithm which may be used to solve it [49]. Its superiority as a parallel algorithm evaluator lies in the fact that unlike *speed-up*, *scale-up* does not consider the sequential parts of the program. Mitra [49] contextually describe *scale-up* for as follows:

> "...Since this problem simply cannot be solved on a serial machine within a reasonable time, a much simpler instance of the same problem for the same wall clock solution time is considered. The ratio of the problem sizes is defined as the potential scale-up of the parallel algorithm."

Scale-up viewed from another dimension may be considered as the number of processors needed to work together to solve a problem that was hitherto,

intractable. In such a case we assume that the intractability of the problem is due to insufficient computational power for examining the search space. For a more indepth study of perfomance measures for parallel algorithms, the reader is referred to [49] [17] [29].

3.5 Efficiency in Processor-Resource Utilisation

Load balancing is a technique of evenly spreading the amount of useful work throughout the network of processors during execution and keeping the processors *nearly* equally busy. We therefore aim to achieve a scenario which guarantees that all available processors be in use so far as there is any pending work. In order to obtain the advantages of speed-up and scale-up for MIP in a parallel framework, the algorithm implementor has to make sure that the search space is efficiently divided amongst all available processors. A monitoring process is usually set up which detects an idle processor and assigns part of the search tree to it. In designing load balancing strategies, one strives to assign the most promising unexplored portion of the search tree (judged by some user-set criteria) to the idle processor. In so doing there are improved chances of finding good solutions quickly as well as making sure that the idle processor is assigned useful work for processing. Use of efficient load balancing strategies improves the *efficiency* of a parallel MIP algorithm. Some efficiency considerations for PB&B are reported in [58]. Load balancing is discussed in detail in Ma *et al* [44], Pardalos *et al* [54] [18] and Lüling *et al* [43].

3.6 Fault Tolerance

In many critical applications, fault tolerant hardware that employed the use of multiple processors were designed. Fault tolerance was achieved using these computers by running multiple copies of the exact program image on all the processors whereby, failure of one processor still allowed for computation to pursue on the other processors. Achieving fault tolerance for MIP in this way is not a viable way as it hinders both speed-up and scale-up. Fault tolerance, with respect to MIP parallel algorithms, pertains to the instituting measures to ensure that the robustness of the entire algorithm is preserved in the event of sudden availability of one or more processors. Fault Tolerance as highlighted by Chen and Ferris [15] and Laundy [38] is of prime importance within a distributed processing environment. In practical situations, a parallel B&B algorithm run over a distributed network of processors in use by other persons and consequently, subject to abrupt shutdown. The parallel B&B algorithm has to take this into account and make "contingency" plans either to reconstruct lost sections of the tree or to transfer work load of the exiting processor amongst the others that form the parallel machine.

4 Computational structure of MIP parallel algorithms

4.1 Challenges in implementing PB&B for MIP

Our main goal in employing parallelised algorithms in solving MIP problems is to achieve *speed-up* and *scale-up*. However, many implementations of PB&B for MIP exhibit some anomalous behaviour in the solution times. This behaviour is usually due to a degree of non-determism that PB&B inherits from the serial B&B algorithm, whereby different implementations of the same algorithm may vary greatly in their processing time [49].
Anomalous behaviour is one of the major problems in converting serial algorithms to a parallel environment. It makes predictability of the possible advantages of a parallel implementation over its serial counterpart difficult to perceive. These anomalies are classified into three categories, namely, *acceleration, deceleration* and *detrimental* anomalies. Ideally, if T_n was the amount of time taken by n processors to solve an MIP which was solved in T_s on a sequential machine, we would expect

$$T_n = \frac{T_s}{n} \tag{8}$$

to hold.
However, in practice, one of the following may occur.

$$T_n < \frac{T_s}{n} \tag{9}$$

,

$$T_n \geq \frac{T_s}{n} \tag{10}$$

Deceleration anomalies are observed when the time taken to solve the MIP by the parallel algorithm is greater than that taken by the serial algorithm as depicted in (9). Acceleration anomalies on the other hand is depicted by (10) in which the time taken by the serial algorithm is smaller than the expected time T_n. A more extreme and undesirable anomaly is *detrimental* anomalies in which the parallel execution time is more than the sequential one. One of the key ways of avoiding deceleration and detrimental anomalies is by achieving good *load balancing* (see section 3.4) schemes. For further discussions on the anomalous behaviour of PB&B algorithms see Lai & Sahni [36] and de Bruin *et al* [17]

Many parallel algorithms which have been developed both for MIP and for other problems have not stood the test of time. The current trend in both industry and academia is to develop algorithms that can work on a single

serial processor and if need be, can be ported over a cluster of processors to achieve *speed-up* and *scale-up*. Achieving this goal for MIP can be a challenging problem for the following reason – in achieving MIP parallelism, we have to capture both the entire structure of the branch and bound algorithm as well as information stored at each node explored in the sub-problems. Typically, the factorisation of the basis matrix in an *linear programming relaxation solution* and the status of all variables (*lower and upper bounds*) have to be stored separately for each sub-problem. In addition, the structure of the subtrees has to be stored in such a way that the union of the subtrees gives the full branch and bound tree. When a B&B algorithm for MIP is designed and implemented with the above considerations, the resulting parallel algorithm, barring anomalous behaviour, will solve benchmark problems to achieve both speed-up and scale-up. We refer to any MIP parallel algorithm that conforms to this criterion as being *serial to parallel robust*.

Finally, Linderoth [40] notes a dilemma that faces the implementor of parallel B&B - sharing globally useful information that the algorithm generates. It is easily seen that there is a tradeoff between complete sharing of information between processors, requiring potentially significant overheads, and little sharing of information, where potentially valuable information is lost.

4.2 Classification and types of PB&B Algorithms

The classification of PB&B algorithms of Gendron and Crainic [27] provides a good framework for classifying MIP parallel algorithms. In section 5, we have used it as one of the criteria for comparing and analysing alternative implementations of MIP parallel algorithms. Gendron and Crainic [27]identified three design approaches to PB&B, namely:

Parallelism of Type 1

This class of parallelism is akin to *low level parallelism* in which the independent operations (that can be parallelised) within serial B&B are parallelised. The basic structure of serial B&B is maintained with the innermost cycles of the algorithm implemented in parallel. The bounding operation or the node selection procedure may be implemented in parallel. The philosophy of this approach is that the sum of the gains in the independent parallelised parts of the algorithm is likely to improve on the speed and scalability of the entire algorithm.

Parallelism of Type 2

This approach is diametrically opposed in philosophy to *parallelism of type 1*. Whereas in the former, parallelism is applied within independent subproblems in a B&B tree, in *parallelism of type 2*, several sub-problems of a branch and bound tree are explored simultaneously. This kind of parallelism can also be referred to as *high level parallelism*.

Parallelism of Type 3
This class of parallelism is based on the fact that B&B has parameters which can be altered. Different *search parameters* result in different B&B trees. *Parallelism of type 3* is achieved when several distinct B&B trees are explored simultaneously.

4.3 Branch and cut (B&C) extensions

Through the addition of cuts at each node, the B&C approach (exploiting the B&B algorithm structure) finds better lower bounds as it tightens the convex hull of the integer feasible solution space. Whereas the computation of a B&C node is more time consuming than a B&B node, the number of nodes in a B&C tree are normally smaller than the number of the nodes in the B&B tree.

Implementing parallel branch and cut (PB&C) is significantly more complex, depending on the parallelisation strategy employed than PB&B. Ralphs [56] outlines three ways of parallelising B&C together with their advantages and drawbacks. The first approach is processing several subproblems in parallel. In this mode, cuts are generated and applied at the subproblems as part of the LP solution process. This method can lead to both faster enumeration and large search trees. Secondly, parallelism could be applied at the level of a single subproblem with one processor solving the LPR and another generating the cuts in parallel. In this instance, cut generation could be outphased by the quick and frequent reoptimisation of the LPR. Finally, in a *parallelism of type 3*, multiple search trees can be investigated and the cuts found on one tree is used in another tree. Irrespective of the stretegy used to parallelise B&C, it is evident that managing the generation and application of cuts is of prime importance. This is usually referred to as *cut management* and as noted by Linderoth [40], has not been extensive investigated – leaving room for further research.

Addition of cuts increase the LPR solution time and as a consequence, cuts are not usually generated at every node of the search tree in serial B&B. Linderoth [40] also considers two cut-assessment measures to evaluate the effectiveness of the cut with respect to the overall effectiveness of the algorithm. The importance therefore of an effective cut management strategy which controls the generation, application and deleting of cutting planes from the *cut pool* cannot be overemphasized. It becomes even more important in PB&B since cuts generated by different processors solving different LPs can be shared and used by other processors. Issues pertaining to cut management within PB&C are discussed in Linderoth [40].

4.4 Information sharing across processors and subproblems

It has been well known to designers of B&B algortims for MIP [7] [5] [48] that strategic use of certain model and solution information (bounds, optimal basis, pseudocosts and global cuts amongst others) dramatically enhance the performance of the solution algorithm. The importance of sharing such information in a multiprocessor environment is both natural and important. Assuming a distributed MIMD architecture for implementing PB&B, all processors are furnished with the model information and solution parameters at the start of the execution. However, it is more important in PB&B algorithms to study how information generated at one processor may be beneficial to the computations on another processor.

Information that is dynamically shared during the execution of PB&B include bounds (lower and upper bounds), optimal basis, global cuts from the cut pool(s), pseudocost information and the structure of the B&B tree. The structure of the B&B tree is useful for the purposes of (quality) load balancing while the other shared information types influence the structure of the B&B algorithm. In an ideal setting, we would like to broadcast all information found on one processor to all other processors that constitute the "parallel machine". However, this will increase the complexity and usually, there is a trade-off between excessive exchange of information and communication overheads. It is important therefore to identify the information that should be shared amongst the processors and the possible advantages or otherwise, of sharing the information. Linderoth [40] identifies four goals of an information sharing system. These include maximising the sharing of "useful" information, minimising latency of access to information, minimising the number of messages passed, and minimising memory use. An in-depth study of information sharing is contained in Trienekens [63] and Linderoth[40].

Bounds

The most important piece of information that is shared is the upper bound (for a minimisation problem). It is well known that these bounds are very useful in pruning the B&B search tree. Sharing such bounds is even more important therefore in a parallel environment as bounds found on one processor may affect the tree development in another. The efficiency of a PB&B algorithm may also be affected by the mechanisms through which critical information is evaluated and shared. The following two ways of evaluating and sharing bounds illustrates this point.

1. Every processor stores the best global upper bound found so far. If a processor finds a feasible integer solution, it *tests* its value against

the value of the best global upper bound. If the new bound is better, it updates the bound and broadcasts the value of the new best global upper bound. Otherwise, the new bound is ignored, thus necessitating no communication. We refer to this as *type 1 bound analysis.*

2. The second option is for the processor to immediately broadcast the bound each time a feasible solution is found – thus leaving the *testing* procedure (to check whether the bound received is better than the existing best global bound) to the individual processors. This approach is used by Linderoth [40], and Eckstein [20]. We refer to this approach as *type 2 bound analysis.*

Communicating a bound, a scalar, is not very computationally expensive and either of the approaches outlined above could be used. However, there are potentially higher testing iterations in the latter approach. Moreover, using the latter approach may result in redundant bounds being broadcast. Sharing bounds is important if the PB&B algorithm is using parallelism of types 2 or 3 strategies. Best lower bounds may also be shared periodically to reduce the duality gap.

Variable choice information

The strategies for choice of branching variable within serial branch and bound are briefly discussed, and references provided in section 2 of this paper. In some implementations of PB&B, a strategy is specified at the start of the search. In case of parallelism of type 3, use of different variable choice strategies (possibly in conjunction with node choice strategies) result in different trees being explored by different processors [49]. In Parallelism of type 2, a fixed user-specified variable choice rule may be used. Sharing pseudocosts (the most widely used variable choice technique) within a PB&B environment is very important as indicated by Linderoth [40]. He asserts that while good initialisation of psuedocosts is essential in serial B&B, it is even more important in PB&B as choice of a "bad" branching variable may lead to useless work on some or all of the processors of the parallel machine. He uses the following example to illustrate his point: if a variable x_b is established to be a bad branching variable on one processor and this information is not made known to the other processors, then the these processors may carry on their search by branching on this variable, thus not benefitting from the knowledge found on one of the processors in the parallel machine.

Cutting planes

One of the advantages of serial B&C over B&B is a reduction in the number of nodes investigated for a given model. Effective sharing of cuts in a distributed enviromnent may further improve the number of nodes investigated. There are two possible reasons for this. First, since cuts generated

at different processors are generated for different LP solutions occuring at possibly distant locations in the B&B tree, sharing cuts among processors may have the effect of building a "complementary" set of cuts to tighten the LPR. Second, a cut found at one processor may cut off the fractional solution at another processor. Since many cuts may be generated in the course of the algorithm, it is imperative to mimimise the amount of cut information shared by employing effective cut assessment procedures. Cut sharing also depends on the way in which cuts are generated and stored. On the one hand, a single cut pool may be employed in which cuts are generated at the individual processors are stored in the global cut pool. Alternatively, there may be multiple cut pools, typically one pool per processor in which each processor stores its generated cuts. Assuming a master-slave paradigm, cut management in single cut-pool algorithms is usually performed by the master processor. In multiple cut-pool algorithms, however, each processor has to do local cut management and exchange of cuts is usually more computationally expensive than in single cut-pool algorithms.

Tree information

Finally there is information about the actual search tree that is transferred from one processor to another. This information usually comprises the unexplored nodes on one processor. When a load balancing procedure is initiated, there is a need to transfer a section of the search tree together with optimal basis, bounds, psuedocosts and cuts (discussed in the previous section). The amount of data involved in transferring a section of a tree from one processor to another is larger than in all the previously identified items of shared information and as a result demands higher memory and storage requirements. Moreover, the information has to be shared in such a way as to preserve the overall B&B tree structure. It is therefore important to design effective data structures that store the tree as compactly as possible. Effective storage of tree information also helps in analysis prior to load balancing in the following way. In load balancing, the aim is to make sure that the node that is transferred to the idle processor is a "promising" node thus ensuring that all the processors are not only busy, but also are busy searching useful (promising) regions. Linderoth [40] refers to this approach of transferring the "optimal" section of the tree to an idle processor as *quality load balancing.*

Factors affecting information sharing

Information sharing depends to a large extent on the storage of the items of information to be shared. Gendron and Crainic [27] use a notion of a *work pool* – the location where processors find and store their units of work (generated subproblems) to classify parallel algorithms. When there is a centralised memory location to which the work units are stored, the algorithm is said to be a *single pool* algorithm, as opposed to a *multiple pool algorithm*

in which there are several memory locations (typically one per processor) for storing the units of work. Single pool algorithms can be achieved within a distributed memory platform via the master-slave paradigm, in which one processor, the *master* manages and allocates the work units to the other processors, the *slaves*. The slaves in turn send back their results to populate the work pool. Single and multiple pool algorithms have their drawbacks as far as communication is concerned. In single pool algorithms for example, running via the master/slave paradigm, there is a possibility of the master being "overwhelmed" with information such that there may be delayed action for a received message, thereby reducing the efficiency of the parallel algorithm. Secondly, over-reliance on a master is an obvious drawback as any physical problems with the master processor may destabilise the entire algorithm. Here again, fault tolerant measures [38][15] should be put in place. However, Goux *et al* [28] have shown that the master-slave paradigm can be used to circumnavigate implementation difficulties encountered in implementing distributed computations. On the other hand, a multiple pool algorithm, the individual processors assume a pseudo-master role, maintaining the internal mechanisms of their search and broadcasting "useful" information when they become available.

5 A Review and Critique of Existing MIP Parallel Algorithms

In this section we first provide a brief literature review of the implementations of PB&B for MIP and identify those algorithms which are considered for a critical review. In section 5.2, we intoduce a framework for comparing and analysing these algorithms and present an analysis of each implementation in tabular form, followed by a brief discussion.

5.1 A Short Literature Review

Branch and bound fits naturally with the parallel computing paradigm and there is a wealth of literature on approaches and experiences with parallel branch and bound. However, given the vast amount of research done on parallel branch and bound, there is relatively little material that focuses specifically on the general MIP.

Boehning *et al* [11] present an implementation of a simple linear programming based branch and bound algorithm for solving MIP, but incorporate no 'advanced' techniques into the procedure.

Cannon and Hoffman [12] were the first to report results of a parallel branch and cut implementation. Their code performed most of the 'advanced' techniques in integer programming. However, because parallel computing tools were not as developed as today, Cannon and Hoffman resorted to operating system constructs in order to perform many of the required parallelisation tasks. They report results on a small suite of problems of up to eight processors.
Ashford *et al* [3] present a more sophisticated branch and bound algorithm for a transputer network of up to eight nodes.
Eckstein [20] developed a PB&B code for a specific dedicated parallel computer, Thinking Machines CM-5. Some advanced features of his algorithm were reduced cost fixing, a primal heuristic, and pseudocost branching.
Bixby *et al* [9] use a parallel software platform called TreadMarks [33] in order to implement a parallel branch and cut algorithm. Computational results and detailed analyses are reported on a parallel system of eight workstations.
Junger and Stormer [32] describe an implementation of a parallel branch and cut for the Travelling Salesman Problem (TSP). Their algorithm follows a novel parallelisation approach whereby a number of processors are dedicated to tasks such as performing heuristics and managing cutting planes in addition to the processors exploring the branch and bound search tree.
Laundy [38] has described the parallel implementation of the XPRESS-MP commercial IP solver. A main emphasis of this work is to build a fault-tolerant system – a system that works if some of the processors become unavailable during the algorithm's execution. In order to build fault-tolerance into the system, the parallelisation strategy used is somewhat simple. Experimental results on a network of up to four workstations are reported.
Chen and Ferris [15] also implement a fault tolerant parallel MIP system that incorporates some of the more advanced branch and bound enhancements.
Homeister [30] gives a relatively simple implementation of the branch and cut algorithm.
Mitra *et al* [49] describe a novel, two stage parallel scheme for solving MIP. In the first stage, all processors begin searching in (possibly overlapping) portions of the search space. The second stage is the more classical branch and bound approach. Limited results on a configuration of up to thirty workstations are reported.
PARINO [40] is also designed to be a general purpose parallel branch and cut solver which includes a most of the advanced MIP solution techniques.
Ralphs and Landanyi [57] and Eckstein al [23] both provide a framework for PB&B algorithms for MIP. SYMPHONY [57] is a modular extensible library generic subroutines while PICO [23] seeks greater versatility in hardware platforms.

Finally, there are tools such as PPBB-Lib [64], BoB [6] and PUBB [61],

emanating mainly from the computer science community which can be used for implementing the general branch and bound algorithm.

5.2 A Critical Analysis of Some PB&B Implementations

In order to compare these algorithms, we have introduced a number of parameters, which, taken together, provide a qualitative framework for analysing the five chosen state-of-the-art implementations. These algorithm parameters are based on earlier analyses in this paper of issues pertaining to parallelism and the B&B algorithm structure and are set out below.

Parallel Architecture – The particular hardware architecture(s) on which the implementaion was made and tested. See section 3.2

Parallel software tools employed – Specialized parallel programming software tools (such as PVM, TreadMarks e.t.c) employed by the implementaion. See section 3.2

Type(s) of parallelism employed – Type 1, Type 2 or Type 3 parallelism. See section 4.2

"Basic" design – Pertains to the parallel programming paradigm that is employed e.g. master-slave, or distributed programming paradigm. See sections 3.2 and 3.3

Program control and node list management – Addresses node storage mechanism (centralised or distributed) and consequently node management for the B&B tree. See section 4.4

Incumbent value sharing mechanism – Ways in which the different B&B-sensitive information most notably, bounds are stored and shared. See section 4.4

IP techniques – Any advanced IP techniques employed such as preprocessing, cutting planes, heuristics, advanced branching rules e.t.c. See sections 2.2 and 2.3

Fault tolerance mechanisms Implicit or explicit fault tolerant mechanisms considered. See section 3.6

Bixby *et al* [9], *1997*	**Parallel Mixed Integer Programming**
Algorithm Parameter	**Description**
Parallel Architecture	Network of workstations consisting of up to 8 SPARC20s'
Parallel Software Tools employed	TreadMarks
Types of Parallelism	Type 2 Parallelism – High level parallelism
Basic Design	"Emulating" shared memory design – Shared memory, but using message passing
Program Control and Node List Management	Centralised control – Global list of active nodes accessed by individual processors using best bound strategy.
Incumbent Value Sharing Mechanism	Best bound is stored in global shared memory
Advanced IP Techniques	Preprocessing, cutting planes and use of specialised heuristic.
Fault Tolerance Mechanisms	No explicit fault tolerance mechanisms used

Analysis of Implementation

The 'start-up' phase of the algorithm exploits 'advanced' IP techniques to tighten the IP feasible region prior to the actual start of parallel B&B. The implementation benefits from the advantages of shared memory architectures as discussed in section 3.2. Since the best bound is only obtained by processors after fetching a new node, there is a chance of wasted computation that could adversely affect speed-up. All parallel 'administrative' duties (such as locking mechanisms to prevent jams during simultaneous access of global information) are acheived through the use of TreadMarks. However, the overheads incurred by using TreadMarks were not investigated.

The authors performed experiments on the complete test suite of MIPLIB [10] benchmark problems, albeit using up to eight processors. Although linear speed-up was not always achieved, the parallelisation strategies employed clearly justify the use of parallel computing in solving MIP problems.

Mitra *et al* [49], *1997*	**A distributed processing algorithm for solving IPs**
Algorithm Parameter	**Description**
Parallel Architecture	Cluster of up to 32 SUN-IPC work-stations
Parallel Software Tools employed	Parallel Virtual Machine (PVM)
Types of Parallelism	Parallelism of Type 3 in stage 1 and Parallelism of Type 2 in stage 2
Basic Design	Master-slave paradigm is used in the stage 2 of the algorithm.
Program Control and Node List Management	The master processor stores a 'master list' list of nodes. The slave processors independently store their individual node lists. The master's node list is used for load balancing procedures - itselects nodes from overloaded processors, stores the 'relieved nodes' in its lists (together with their priority order) and distributes them to less busy processors
Incumbent Value Sharing Mechanism	New bounds are immediately broadcast and stored locally by individual processors in both stages of the algorithm
Advanced IP Techniques	Use of the standard IP techniques in the FortMP solver
Fault Tolerance Mechanisms	While fault tolerance was not explicitly stated a s a design objective, Stage 1 of the algorithm has inherrent fault tolerance since the search trees being examined by processors are independent

Analysis of Implementation

While the implementation did not fully exploit some integer programming enhancements like preprocessing and cutting planes, the nature of the implementation – retaining the serial branch and bound code as much as possible, makes the incorporation of these enhancement features easy [52] . The rationale behind the implementation is novel and promises to be worthwhile, and to our knowledge, it is the only parallel branch and bound implementation of MIP that exploits Parallelism of type 3. Stage 1 of the algorithm is, in effect, a number of independent serial searches run in parallel and enhanced by sharing bounds. The use of co-operating, processing multiple branch and bound trees in stage 1 presents a higher chance of finding good feasible solutions. The transition between stage 1 and stage 2 of the algorithm was controlled by static criteria based on a cut off on time or number of nodes. However, there

is potential to do a more robust analysis of the trees generated from stage 1, in order to choose the potential 'best' search tree to be investigated in stage 2.

Experimental results are reported in this paper are restricted to a small set of MIPLIB problems and show near-linear speedups when the code is tested on some MIPLIB models.

Laundy [38], *1998*	**Implementation of PB&B Algorithms in XPRESS-MP**
Algorithm Parameter	**Description**
Parallel Architecture	Network of PC's consisting of up to four 200 MHz COMPAQ Pentium Pros and one 133 MHz
Parallel Software Tools employed	HPPVM - The high performance version of PVM
Types of Parallelism	Type 2 parallelism and to a limited extent, type 1 parallelism in order to exploit the advantages of the search strategies of the serial algorithm
Basic Design	The master-slave (farming) paradigm is used in the implementation
Program Control and Node List Management	The approach to parallelism is based on a centralised control in which the master processor holds the list of unsolved nodes which are distributed to slave processors
Incumbent Value Sharing Mechanism	The author does not explicitly report the way in which the best bounds are shared
Advanced IP Techniques	XPRESS-MP is a commercial solver which has mature and well tested heuristics and search strategies embedded in the serial B&B code. Consequently, one of the key design objectives of the parallel code was to include insofar as possible all the advanced IP features in the serial code.
Fault Tolerance Mechanisms	Fault tolerance is one of the stated design goals of the implementation and processor failures are detected by using PVM calls to check their status. If there is an processor error in the master, then the failure is considered as unrecoverable. However, if failure occurs at a slave processor, the implementation ensures that the master re-schedules the destroyed node to another processor for re-exploration. The rescheduling of nodes is easily achieved since the master keeps a record of all jobs that are sent to slave processors.

Analysis of Implementation

In this paper, the author describes the implemention of parallel B&B for a leading commercial solver, XPRESS-MP, and as such pays particular importance to fault-tolerance – ensuring that the software is robust for running in a volatile office environment in which machines can easily crash or are turned off by their owners. The approach in achieving fault tolerance puts the onus on the user of the system to ensure that they possess control of the master processor, thereby minimising the risk of unrecoverable failures.

As discussed in section 4.1, it is desireable to design algorithms for MIP that can run on a single processor, as well as on several processors (if need be). This implementation captures that model especially because the parallel implementation seeks to benefit from the hybrid depth-first/best-first strategy (as well as other advanced heuristics) that have been developed to solve MIP using XPRESS-MP on a single processor.

Another key feature described in the implementation is the portability of the code. Their overriding design philosophy is "to develop a system which runs well both on dedicated parallel computers and on heterogeneous network of PC's and workstations."

Since the implementation is centralised, it suffers from the drawbacks (as discussed in section 3) of such an approach. However, the author also suggests a multi-master/multi-slave paradigm that could be implemented on shared-memory machines with many processors and fast communication links. Such an implementation will require considerable load-balancing measures which are not neccessary in the implementation described in the paper.

Breadth-wise experimental results are reported for a small subset of "harder" MIPLIB [10] models. Comparisons are made between serial execution times and parallel execution times based on different number of slave processors. In general, linear speed-ups are obtained in a comparible number of nodes with the serial algorithm.

Chen *et al* [15][14],*1998, 1999*	**FATCOP 1.0 and FATCOP 2.0**
Algorithm Parameter	**Description**
Parallel Architecture	Network of up to 100 (heterogeneous) workstations
Parallel Software Tools employed	Condor-PVM [55] – an extension to the regular PVM that may obtain more computational power in an opportunistic way
Types of Parallelism	Type 2 parallelism – High level parallelism
Basic Design	Two separate programs are used in a master-slave paradigm. The master farms out subproblems to slave processors.
Program Control and Node List Management	There is a one central work pool consisting of uninvestigated subproblems. Subproblems are stored in a multi-indexed data structure to facilitate the searching rules.
Incumbent Value Sharing Mechanism	The master process updates the best bound information in the work pool. This new bound is used when new subproblems are being farmed out to slave processors.
Advanced IP Techniques	Preprocessing is used in FATCOP 1.0 and global cuts and global pseudocosts are used in FATCOP 2.0
Fault Tolerance Mechanisms	Fault tolerance is one of FATCOP's main design objectives. Consequently, rigourous fault tolerance mechanisms are employed

Analysis of Implementation

FATCOP 1.0 [15] and to a greater extent FATCOP 2.0 [14] combine the goals of using freely available distributed computational power – typical of office networks, with a B&B code which incorporates many advanced solution features. Unlike the all the other implementations of PB&B which have a pre-determined number of parallel processors, FATCOP can dynamically add and delete hosts depending on their availability. It achieves this by using a Condor-PVM environment which extends the functionality of the regular PVM. Greedy acquisition of processors to populate the Condor pool has the potential advantage of providing scalability on the fly *if* required by larger MIP models.

FATCOP 2.0 improves on FATCOP 1.0 both at a parallelisation and B&B code level. In FATCOP 2.0, a more coarse grain implementation is sort in order to limit contention effects at the master. In addition, more advanced integer programming features, namely, global cuts and global pseudocosts are appended to improve on the implementation. Rigid fault tolerance measures are employed such that there is at least a partial recovery of search

process even in the extreme and unlikely case of unavailability of the master processor. Extensive results and analysis are carried out for a subset of MIPLIB [10] models and the implementation shows the benefits of using their approach.

Eckstein *et al 2000*	**PICO: An Object-Oriented Framework for Parallel Branch and Bound**
Parameter	**Description**
Parallel Architecture	Aims to work on a variety of parallel architectures. It is however designed using a distributed model using message passing.
Parallel Software Tools employed	Message Passing Interface (MPI)
Types of Parallelism	Framework approach gives user the flexibility to choose the type of parallelism to implement
Basic Design	A flexible master-slave paradigm is employed. The master can also function as a worker
Program Control and Node List Management	Each of the slave processors hold their own pool of active nodes – hence a distributed strategy
Incumbent Value Sharing Mechanism	Best bounds found at individual processors are immediately broadcast. Each receiving processor performs bound analysis of type 2
Advanced IP Techniques	Preprocessing and pseudocosts
Fault Tolerance Mechanisms	No explicit fault tolerance mechanisms are discussed

Analysis of Implementation

The design philosophy of PB&B algorithms in this work is different from all the other works described. The authors use a framework approach – an object oriented approach which provides a set of classes (objects that share a common structure and behaviour) that embodies an abstract design for solutions to a number of related problems. The implementation provides a hierachical class of serial and parallel layers thereby making it very modular and flexible. The classes of the serial layer store and manipulate data pertaining to the nodes of the B&B tree, while the corresponding classes of the parallel layer behave similarly for a PB&B tree.

The implementation of PICO is largely based on CMMIP [20][19][22][21]. However, the implementation extends CMMIP by being more versatile with respect to the parallel architecture employed – the system can be configured at run-time to suit shared memory or distributed memory architectures. A "pseudo" master-slave paradigm may also be employed in which the master

performs the administrative duties of the code, and simultaneously acts as a slave by solving subproblems.

Results are reported for a small subset of MIPLIB [10] models and the implementation is shown to achieve linear speed-ups for between 32-48 processors. PICO does not include some "advanced IP techniques" such as cutting planes or node-level preprocessing.

6 Conclusions and directions for future work

Our investigation of the recent research and development in parallel MIP algorithms shows that there is a growing interest and steady progress in both academic and industrial settings. The main motivation for using parallel algorithms for MIP is to solve challenging industrial problems which are otherwise intractable. For the end user it is also desirable that the parallel MIP software bears close resemblance to its serial counterpart; the main difference being the possibility of specifying the use of multiple processors. In this paper, we have examined some of the leading design issues pertaining to exploiting parallel platforms for branch and bound. We have highlighted the algorithmic components in branch and bound that benefit from parallel computation and discussed aspects of the parallel architectures that are now used by the leading of MIP parallel algorithms. Finally, we have critically examined the parallelisation strategies employed in five major PB&B implementations. In conclusion, we summarise the current trends in parallel MIP algorithms and identify some areas for further research.

6.1 Current trends

The following four aspects summarise the current trend in parallel MIP algorithms.

1. It is well known that serial B&B algorithms can be enhanced by introducing suitable preprocessing, use of global and local cuts, and computing pseudocosts. To gain full advantages in performance, algorithm designers are introducing these features intelligently into PB&B algorithms.

2. A distributed architecture, consisting of loosely-coupled workstations and PCs has become the *de facto* platform for implementing MIP parallel algorithms. This architecture provides an inexpensive and readily available platform for reaping the benefits of parallel computing and has been adopted by many implementors.

3. One or more computers within a distributed network (usually of office computers) are prone to sudden unavailability. There is therefore an emphasis on developing fault-tolerant solver software so that the effect of losing a processor during a run has minimal impact on the PB&B algorithm.

4. Most of the parallel MIP implementations show a near-linear speed-up – somewhat emphasising the fact that the B&B tree is "embarassingly bespoke to parallel computing." Consequently, at least three industrial strength MIP solvers have their parallel versions available on a commercial basis [50] [38] [16]

6.2 Directions for future research

1. The employment of cutting planes within serial B&B is arguably an art in which success is determined by the type of cuts generated, their storage and their application at different levels of the B&B tree. In PB&B there is a need for efficient cut management strategies, in which cuts generated by different processors can be evaluated with a view to always having an optimal set of cuts for application throughout the parallel algorithm. Experiments with single and multiple cut-pool algorithms can also give varying performances in PB&B implementations. There is also scope for using processors to carry out dedicated tasks such as generating and managing cuts or pseudocosts. Junger and Stormer [32] investigated this approach with some success for TSP. Finally, natural extension for consideration in PB&B algorithms will be the branch and price algorithm.

2. In our opinion, parallelism of type 3 as applied to MIP parallel algorithms deserves more investigation. Mitra al [49] showed promising results by investigating different branch and bound trees in parallel. Systems like Condor [42] provide extensive computational resources to experiment with several different trees in the hope of finding good integer solutions.

3. Taking into account *parallelism of type 3*, we can further combine the branch and bound approach with special-purposed and meta heuristics such as simulated annealing and tabu search.

4. The availibility of potentially massive ditributed systems such as Condor [42] provides a platform for testing current implementations on larger systems so as to investigate the number of (heteregeneous) processors needed to obtain maximum speed-up and scale-up in a PB&B algorithm. Such investigations will undoubtedly provide more insight

into the best way of allocating processes in a PB&B algorithm. The possibility of using new processors which are added on the fly to a parallel machine also present enhanced searching opportunities in a PB&B algorithm. It is therefore desirable to design algorithms that can dynamically reconfigure themselves – adding or removing computing resources during the course of a run.

5. There is an emergence of computers with multi (two or four) symmetric processors that use a shared memory architecture. Set against this background, there exists the possibility of combining message passing and shared memory configurations within MIP parallel algorithms. Such an example may be particularly useful, for problems whose LPRs are difficult to solve. A parallel dual-simplex algorithm may be used on the symmetric multiprocessor machine for solving LPRs whereas the PB&B algorithm may run on a distributed network of such workstations.

The main objective of this paper has been to review the current state-of-the-art and issues pertaining to the design, implementation and evaluation of PB&B algorithms for MIP. We hope that our analyses have made a useful contribution which the designers of parallel algorithms for MIP will find valuable in their further work.

References

[1] Brearley A.L., G. Mitra, and H.P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming*, 8:54–83, 1975.

[2] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conf. Proc.*, 30:483–485, 1967.

[3] R.W. Ashford, P. Connard, and R. Daniel. Experiments in solving mixed integer programming problems on a small array of transputers. *Journal of the Operational Research Society*, 43:519–531, 1992.

[4] M. Balinski. Integer programming: Methods, Uses, Computation. *Management Sci*, 12:253–313, 1965.

[5] E.M.L. Beale. Branch and bound methods for mathematical programming systems. *Annals of Discrete Mathematics*, 5:201–219, 1969.

[6] M. Benachouche, V-D. Cung, S. Dowaji, B. Le Cun, T. Mautor, and C. Roucairol. Building a parallel branch and bound library. In *Solving combinatorial optimisation problems in parallel, Lecture Notes in Computer Science*, pages 201–231. Springer, Berlin, 1996.

[7] M. Benichou, J.M. Gauthier, P. Girodet, G. Hehntges, G. Ribiere, and O. Vincent. Experiments in mixed integer linear programming. *Mathematical Programming*, 1:76–94, 1971.

[8] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and distributed computation, Numerical Methods.* Prentice-Hall, 1989.

[9] R. Bixby, W. Cook, A. Cox, and Lee E. Parallel mixed integer programming. Technical Report CRPC-TR95554, Rice University, Center for Parallel Computation, 1995.

[10] R. E. Bixby, S. Ceria, C. M. McZeal, and M.W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[11] R.L. Boehning, R.M. Butler, and B. E. Gillett. A parallel integer linear programming algorithm. *European Journal of Operational Research*, 34:393–398, 1988.

[12] T.L. Cannon and K.L. Hoffman. Large scale 0-1 linear programming on distributed workstations. *Annals of Operations Research*, 22:181–271, 1990.

[13] Y. Censor and S. A. Zenios. *Parallel Optimization: Theory, Algorithms and Applications.* Oxford University Press, 1997.

[14] M. Chen, Q. Ferris and J. Linderoth. Fatcop 2.0: Advanced features in an opportinistic mixed integer programming solver. Technical report, University of Wisconsin, Madison, Department of Computer Sciences, Madison, WI 53706, December 1999.

[15] Q. Chen and M. Ferris. A fault tolerant condor-pvm mixed integer program solver. Technical report, University of Wisconsin-Madison, Department of Computer Sciences, Madison, WI 53706, 1999.

[16] Inc. CPLEX Optimization. *Using the CPLEX Callable Library.* Using the CPLEX Callable Library, 1995.

[17] A. de Bruin, G.A.P Kindervater, and H.W.J.M. Trienekens. Assynchronous parallel branch and bound and anomalies. In *IRREGULAR'95, LNCS*, volume 980, pages 363–377, 1995.

[18] C.G. Diderich and M. Gengler. Some strategies for load balancing. In *IRREGULAR'94*, pages 395–409. Kluwer, 1995.

[19] J. Eckstein. Control strategies for parallel mixed integer branch and bound. In *Proceedings of Supercomputing '94*, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[20] J. Eckstein. Parallel branch-and-bound algorithms for general mixed integer programming on CM-5. *SIAM Journal on Optimization*, 4:794–814, 1994.

[21] J. Eckstein. Distributed versus centralized storage and control for parallel branch and bound: Mixed integer programming on CM-5. *Computational Optimization and Applications*, 7:199–220, 1997.

[22] J. Eckstein. How much communication does parallel branch and bound need? *INFORMS Journal on Computing*, 9:15–29, 1997.

[23] J. Eckstein, C.A. Phillips, and W. Hart. Pico: An object-oriented framework for parallel branch and bound. Technical Report RRR 40-2000, Rutgers University, Rutgers University, New Jersey, August 2000.

[24] M.J. Flynn. Some computer organisations and their effectiveness. *IEEE Transactions on Computers*, 21:948–960, 1972.

[25] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co, 1979.

[26] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual*. Oak Ridge National Laboratory, Oak Ridge, TN 37831-6367, 1994.

[27] B. Gendron and T. G. Crainic. Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, 42(6):1042–1066, 1994.

[28] J. Goux, J. Linderoth, and M. Yoder. Metacomputing and the master-worker paradigm. *Technical Report, Argonne National Laboratory, Available from http://www.cs.wisc.edu/condor/mw*.

[29] I. Hai. Integer programming on parallel computers. Master's thesis, Brunel University, 1994.

[30] D. Homeister. Efficient implementation of parallel branch and cut. Presented at the Fifth SIAM Conference on Optimization, Victoria, B.C., Canada, 1996.

[31] E.L. Johnson, G.L. Nemhauser, and M.W.P. Savelsbergh. Progress in linear programming-based algorithms for integer programming. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

[32] M. Jünger and P. Stormer. Solving large scale travelling salesman problems with parallel branch-and-cut. Technical Report 95.191, Universitat zu Koln, Zentrum fur Paralleles Rechnen, 1995.

[33] P. Keheler, A. Cox, S. Dwarkadas, and W. Zwaenepoel. Treadmarks: Distributed memory on standard workstations and operationg systems. In *Proceedings of the 1994 Winter Usenix Conference*, pages 115–131, 1994.

[34] K. Kularajan, G. Mitra, F. Ellison, and B. Nygreen. Constraint classification, preprocessing and a branch and relax approach to solving mixed integer programming models. *International Journal of Mathematical Algorithms*, 2, 2000.

[35] V. Kumar and A. Grama. *Introduction to parallel computing – design and analysis of algorithms*. Benjamin/Cummings, 1994.

[36] T. H. Lai and S. Sahni. Anomalies in parallel b&b algorithms. *Res. Contrib*, 27:594–602, 1984.

[37] A. Land and A. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[38] R. S. Laundy. *Implementation of branch and bound algorithms in XPRESS-MP.* in Operational Research in Industry, T. Ciriani et al. eds. Macmillan Press, 1999.

[39] J.T. Linderoth and M.W.P Savelsbergh. A computational study of search strategies in mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[40] J.T. Lindertoth. *Topics in Parallel Integer Optimization.* PhD thesis, Georgia Institute of Technology, 1998.

[41] J.D.C Little, K.G. Murty, D.W. Sweeney, and C. Karel. An algorithm for the travelling salesman problem. *Operations Research*, 11:972–989, 1963.

[42] M. Litzkov, M.J. Livny and M. W. Mutka. Condor: A hunter of idle workstations. In *Proceedings of 8th International Conference on Distributed Computing Systems*, pages 104–111, June 1988.

[43] R. Lüling and B. Monien. Load balancing for distributed branch and bound algorithms. In *Proceedings of the international Parallel Computing Symposium*, pages 543–549. IEEE Computer Society Press, Los Angeles, CA, 1992.

[44] R.P. Ma, F. Tsung, and M. Ma. A load balancer for a parallel branch and bound algorithm. In *Proceedings of 3rd Conference on Hypercube Concurrent Computers and applications*, 1988.

[45] I. Maros and G. Mitra. *Simplex algorithms for linear programming*, chapter 1. in Advances in Linear and integer programming, J. Beasley. Oxford University Press, 1996.

[46] I. Maros and G. Mitra. Investigating the sparse simplex algorithm on a distributed memory multiprocessor. *Parallel Computing*, 26:151–170, 2000.

[47] L. G. Millen. Branch and bound methods: General formulations and properties. *Operations Research*, 18:24–34, 1970.

[48] G. Mitra. Investigation of some branch and bound strategies for the solution of mixed integer linear programs. *Mathematical Programming*, 4:155–170, 1973.

[49] G. Mitra. A distributed processing algorithm for solving integer programs using a cluster of workstations. *Parallel Computing*, 23:733–753, 1997.

[50] NAG Ltd. *FORTMP Reference Manual.*

[51] G.L Nemhauser and L.A Wolsey. *Integer and Combinatorial Optimisation.* Wiley, 1989.

[52] V. Nwana, E.F. Ellison, G. Mitra, and K. Kularajan. A two-stage parallel branch and cut algorithm for mixed integer programs. Document in preparation, July 2000.

[53] V. Nwana and G. Mitra. Parallel mixed integer programming: A status review. Technical Report TR/02/00, Brunel University, Department of Mathematical Sciences, January 2000.

[54] P. Pardalos and X. Li. Parallel branch and bound algorithms for combinatorial optimisation. Technical report, The Pennsylvania State University, Department of Computer Science, 1990.

[55] J. Pruyne and M. Livny. Interfacing Condor and PVM to harness the cycles of workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[56] T.K. Ralphs. Computational experience with generic parallel branch and cut. Presented at the Institute of Operations Research and Management Science Conference, Seattle, WA, October 1998, October 1998.

[57] T.K. Ralphs and L. Landanyi. Symphony: A parallel framework for branch, cut and price. Technical report, Rice University, Department of Computational and Applied Mathematics, Rice University, Houston, Texas 88005, December 1999.

[58] V.J. Rayward Smith, S.A. Rush, and G.P. Mckeown. Efficiency considerations in the implementation of branch and bound algorithms. *Annals of Operations Research*, 43:123–145, 1993.

[59] C. Roucairol. *Parallel Branch and bound algorithms; An Overview*, pages 153–163. Parallel and Distributed Algorithms. Elsevier Science Publishers, 1988.

[60] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming models. *ORSA Journal on Computing*, 6:445–454, 1994.

[61] Y. Shinano, M. Higaki, and R. Hirabayashi. Generalized utility for parallel branch and bound algorithms. In *Proceedings of the 1995 Seventh IEEE Symposium on Parallel and Distributed Processing*, pages 382–401. IEEE Computer Society Press, Los Angeles, CA, 1995.

[62] M. Snir, S.W. Otto, D.W. Walker, and J. Dongarra. *MPI: The Complete Reference.* MIT Press, Cambridge, MA, 1996.

[63] H.W.J.M Trienekens and A. de Bruin. Towards a taxonomy of parallel branch and bound algorithms. Technical Report EUR-CS-92-01, Erasmus University, Rotterdam, 1992.

[64] S. Tschöke and T. Polzer. *Portable Parallel Branch-and-Bound Library PPBB-Lib User Manual, Library Version 2.0.* Department of Computer Science, University of Paderborn, 1996.

7 Appendix 1: An insight into the computational structure of B&B

This section highlights aspects of the key information shared across the nodes of the B&B tree in order to construct a serial as well as parallel B&B algorithm. The aspects considered here include:

1. Sharing of improving lower and upper bounds
2. Use of optimal basis
3. Choice of algorithm for subproblem solution
4. Integer preprocessing

To illustrate these points, the following experiment was performed for five MIPLIB [10] models. The aim of the experiment was to investigate the improvement in the computational perfomance of a simplex-based B&B solution algorithm. By progressively refining the "solution processing" of the subproblems reflecting the choices highlighted above, we notice a commensurate performance gain. The same set of node and variable choice methods were used in solving five instances of MIPLIB [10] problems. The time limit for running B&B was set for two hours in all cases. *IP(opt)* refers to the optimal or best integer feasible solution obtained by the search while *T(best)* is the time taken to obtain the best integer solution. *T(total)* is the total time taken to complete the search.The node and variable choice strategies used in solving all the models are not necessarily the best for each problem.

Table 1. shows the computational behaviour of B&B when the alternative ways of applying SSX in solving subproblems are used. *Code* refers to the mode in which B&B was run. In *B&B(Crash)*, each node is solved independently, that is, no starting basis is used. In *B&B(Primal)*, the primal simplex algorithm used the basis of the optimal solution of a parent node to

solve the child node, while in *B&B(Dual)*, the optimal basis of the parent is used with the dual simplex method. The column *IP(opt)* contains the best or optimal solution achieved by the B&B code and *T(total)* represents the time taken to achieve the solution. *Iterations* is the number of iterations performed by the code, while *nodes* is the number of nodes investigated by the code. *B&B(Prep)* refers to the B&B code run with similar settings as B&B(Dual), with preprocessing applied at the root node (only). Similarly,in B&B*(Prepall)* preprocessing was applied to all nodes in the tree.

Model	Code	IP(opt)	T(total)	Iterations	Nodes
10TEAMS	B&B(Crash)	948*	> 7200	344817	874
	B&B(Primal)	924	2978.56	12848	1456
	B&B(Dual)	924	2698.43	10653	1258
	B&B(Prep)	924	2098.67	9846	897
	B&B(Prepall)	924	2045.67	9785	899
P0201	BB(Crash)	7615	110.38	473080	987
	B&B(Primal)	7615	21.77	65583	1792
	B&B(Dual)	7615	13.71	26243	3323
	B&B(Prep)	7615	7.54	13783	1648
	B&B(Prepall)	7615	6.34	11324	1265
BLEND2	B&B(Crash)	7.5989850	6295.65	25856468	204250
	B&B(Primal)	7.5989850	1066.67	2270419	212940
	B&B(Dual)	7.5989850	787.28	302585	208210
	B&B(Prep)	7.5989850	179.02	171759	112262
	B&B(Prepall)	7.598950	165.43	141324	90267
NWO4	B&B(Crash)	16862*	> 7200	229954	477
	B&B(Primal)	16862	4321.13	45425	2987
	B&B(Dual)	16862	3973.23	42073	2635
	B&B(Prep)	16862	3999.14	42073	2635
	B&B(Prepall)	16862	3654.09	41324	2465
L152LAV	B&B(Crash)	4769*	> 7200	30975852	89067
	B&B(Primal)	4722	3867.65	2340876	147687
	B&B(Dual)	4722	2876.98	1825444	136754
	B&B(Prep)	4722	2878.59	1825444	136754
	B&B(Prepall)	4722	2566.30	1756234	126565

'*' means tree search was not completed while '> 7200' means the time limit was exceeded.

Table 1: Results of alternative LP-solution strategies

It can be seen that the number of iterations, as well as ratio of iterations to nodes are very high for B&B(Crash)in all five models. This can be attributed to the excessive work done in re-solving each node from 'scratch'.

B&B crash also has undesirable effects on the time taken to solve the problems and may result in sub-optimal solutions. B&B(Dual) is shown in the models to be of superior quality to B&B(Primal).