Classification of optical orthogonal codes and spreads by backtrack search on a parallel computer

Tsonka Baicheva tsonka@math.bas.bg
Svetlana Topalova svetlana@math.bas.bg
Institute of Mathematics and Informatics, Bulgarian Academy of Sciences,
P.O.Box 323, 5000 V.Tarnovo, BULGARIA

Abstract. Backtrack search based algorithms are often used for the construction of different types of combinatorial structures, namely codes, designs, design resolutions, etc. In this paper we discuss on our implementation of backtrack search on a parallel computer for two different classification problems. We used this software on BlueGene/P for the classification of spreads of $PG(7,2)$ with definite properties and of optimal optical orthogonal $(v,5,2,1)$ codes.

1 Introduction

The software meant to work on a multiprocessor computer has to be designed in such a way that to make as small as possible both the time lost in communication among processors and the time when a processor has nothing to do (idle time). The speed-up $S(n)$ of a parallel algorithm is defined as

$$S(n) = \frac{t_s}{t_n}$$

where t_s is the time needed by the sequential algorithm, and t_n is the time the parallel algorithm works on n processors. Backtrack search can be successfully implemented on parallel computers with a speed-up within a small constant factor from optimal [1]. Two main types of parallel backtrack search (PBS) are considered in [1], namely randomised or with global control. In global control PBS one or several processors are only used to manage the communication, while in randomised PBS an idle processor asks for work a randomly chosen other processor.

Although the two problems and sequential programmes (about spreads and optical orthogonal codes (OOCs)) were very different, the changes we made to parallelize them were very similar. These changes are subject of the present note.

2 Parallelizing of the backtrack search

Consider search in depth implemented by a stack. By the sequential algorithm we always expand the upper node from the stack (namely we put it in the current solution, remove it from the stack, and insert its children in the stack). To parallelize the search we replace the stack with a *deque* – double ended queue. We perform the search in just the same manner. We always expand the last node from the deque. When we are asked to give work to an idle processor, we give the first node, and remove it from the deque. The other processor starts expanding the given node. If we omit the details, the principal $C++$ implementation of PBS looks like that:

```
#include <deque>

struct choose
{
   int num;
   int val;
};
deque<choose> D;
choose C, C1, C2, ...;
...
while(!D.empty())
{
   C = D.back( ); // take the last element of the deque
   D.pop_back(); // remove the last element
   if(C.num==N) WriteRes(); // write a result
   else
   {
      ... // find the possibilities C1, C2 ... for element C.num+1
      D.push_back(C1); //insert C1 in the deque
      D.push_back(C2); //insert C2 in the deque
      ...
   }
   ...
   if(flag) //must give work to another processor
   {
      Cf = D.front( ); //take the first element from the deque
      D.pop_front(); //remove the first element from the deque
      ... // donation: send Cf and the current solution to the idle processor
   }
}
```

We obtain solutions in lexicographic order. When expanding a node, we check the partial solutions for possible equivalence to lexicographically smaller ones, and thus add to the deque only children nodes of solutions for which such equivalence has not been established. The donation message contains the node to be expanded and the current partial solution. This is enough for the idle processor to start work again. We donate the node of the lowest possible level, which is likely to need the longest time to be fully worked out and will this way keep the other processor busy for long.

3 Global control PBS

In our Global control PBS implementation we choose the processor with number 0 to be the *manager*. The manager does not expand nodes.

The manager:

- Knows which processor has work of lowest level.
- When a processor finishes his work, the manager
  - tells him from which processor to receive work.
  - tells him to stop.

Each of the other processors:

- Sends a message to the manager on remaining without work.
- Sends a message to the manager on change of his first node level.
- Shares work with another processor if the manager tells him to.

4 Results management

We let each processor write the results in a separate file. We compress the files, copy them to the PC and then sort and summarize them on the PC. But it is convenient to know some of the main results right after the application finishes. So the processors send a summary of the results (the number of the constructed objects, etc.) to the manager before they terminate and the manager writes a small summary in a file.

For some parameters the programme might need more time than it is allowed. That is why from time to time the manager tells all the processors to write the current content of their deque in a file, so that next time the computation can start from that place on.

5 Initialization

In the spreads classification programme processor number 1 starts the search, and the rest are idle at the beginning. This adds a constant factor to the speed of the computation. In the OOCs classification we reduce this constant by first constructing on all processors all the possible first level nodes. Let their number be F. So each processor with number $p < F$ starts expanding the pth first level node.

We provide an option for continuation of a programme that has already run up to some state. In this case the initialization constant is very small, because each processor reads its deque elements from a file and starts work very fast.

6 Changes and additions to the sequential software

Most of the software remains unchanged. We add communication functions to the function implementing the backtrack search and change the stack by deque. We also add communication functions to *main*. The Message Passing Interface (MPI) is initialized there at the beginning, and finalized before exit. The other communication in *main* depends on the number of the processor, on which this copy of the programme runs. Small changes should also be made in the functions which write the results, and a function writing the current deque in a file might be added to enable continuation of the computation.

Acknowledgement. We are grateful to the Bulgarian National Supercomputing Centre for the permission to use its resources, and for the detailed instructions supplied by their team [2].

References

[1] Karp, R.M., Zhang, Y. Randomized parallel algorithms for backtrack search and branch-and-bound computation. J. Assoc. Comput. Mach. (USA), vol.40, (no.3), July 1993. p.765-89.

[2] http://www.scc.acad.bg/articles/bluegene-quick-guide.pdf