# Improving Parallelism in Structural Data Mining

Min Cai[1], Istvan Jonyer[1], and Marcin Paprzycki[2]

[1] Department of Computer Science, Oklahoma State University, Stillwater, Oklahoma 74078, U.S.A.
cmin, jonyer@cs.okstate.edu
[2] Computer Science Institute, SWPS, 03-815 Warsaw, Poland
marcin.paprzycki@swps.edu.pl

**Abstract.** Large amount of data collected daily requires efficient algorithms for its processing. The SUBDUE data mining system discovers substructures in structurally complex data, based on the minimum description length principle. Its parallel implementation, MPI-SUBDUE, was created in 2001 to reduce computation time and/or to deal with larger datasets. In this paper, a new, more efficient implementation of MPI-SUBDUE is introduced. The experimental results show that, for the mutagenesis dataset, the new implementation outperforms the original one by up to 33% and that the performance gain increases with the number of processors used.

## 1 Introduction

Large amounts of data are collected on a daily basis and added to the existing repositories. The need to extract valuable information from this data challenges researchers to develop efficient techniques to discover and interpret interesting patterns in it. In this paper we are interested in discovering concepts in structural data, for which a number of algorithms have been proposed [1, 6, 8]. One of them, SUBDUE, discovers substructures on the basis of the minimum description length principle [8]. Working with graph-based data representation and utilizing graph-algorithms, when applied to real-world problems, SUBDUE takes a very long time to execute. In 2001, a parallel version of SUBDUE (MPI-SUBDUE) was implemented [1, 8]. MPI-SUBDUE partitions the graph representing the dataset into parts that are analyzed independently first, and then partial results are communicated so that the globally-best substructure is selected. Parallelization was achieved through MPI, while communication between processes consisted of a sequence of point-to-point messages. However, even a superficial analysis of MPI-SUBDUE indicated a number of possible inefficiencies, which not only could have resulted in poor performance, but also constitute a problem form the point of view of programming simplicity and expressiveness [7].

Since SUBDUE is one of the best existing graph-based data mining tools, we have decided to develop its more efficient version and in the process to improve it from the point of view of programming simplicity and expressiveness. To accomplish these goals, we have proceeded with the following modifications of the MPI-SUBDUE:

– the initial graph partitioning was parallelized,
– multiple point-to-point communications (MPI_Send and MPI_Recv) have been replaced by an MPI collective communication (MPI_Allgatherv),
– parallel summation has been applied to deciding the globally best discoveries.

In reporting our work we proceed as follows. In the next section we present an overview of the SUBDUE system. Both the original and the NEW-MPI-SUBDUE are described in Section 3. We follow with the description and analysis of experimental results of testing both algorithms on three datasets. In Section 5 we summarize our results and outline our future work.

## 2 SUBDUE

SUBDUE is a data mining system working through substructure discovery aimed at finding interesting and recurring subgraphs in a labeled graph. This goal is achieved by applying the minimum description length principle (MDL). SUBDUE has been successfully applied to molecular biology, image analysis and computer-aided design, as well as other domains [8]. During SUBDUE's execution two basic steps (1) substructure discovery and (2) replacement are performed. Structural data (input for SUBDUE) is represented as a labeled graph. Objects in the data correspond to vertices, while relationships between them correspond to edges. A substructure is a connected subgraph (a subset of vertices and edges from the input graph) and represents a structural property of the dataset. In the first step SUBDUE discovers the best (according to the MDL principle) substructure in the complete data-graph. This substructure could be the final answer, but it is also possible to repeat the process to discover a hierarchy of important structures. To achieve this goal, all instances of the best substructure at a given level are compressed to a single vertex and SUBDUE is invoked on the "reduced graph". Hierarchy of substructures obtained in this way is then used for various levels of interpretation; depending on goals of data analysis (see [1, 6, 8] for a complete description of SUBDUE and examples of its application to real-life problems).

## 3 Parallel SUBDUE

Data parallel approach and functional (or control) parallelization are the two main ways of achieving algorithm parallelization. Both approaches can be used to parallelize SUBDUE. Functional parallel SUBDUE has been introduced in [1]. In this paper, we focus on the data parallel approach. In parallel SUBDUE, the input graph is first partitioned into $n$ partitions that are distributed among $n$ processors. Each processor finds the best substructure in its partition and communicates it to all other processors. Each processor uses these substructures to compare them with its own structures, using the MDL principle (structure that is very good for the data in partition $k$ may be bad in partition $l$ and thus all "local champions" must be compared vis-*á*-vis structures in each partition).

Results of local comparison are then combined and exchanged between processors and as a result, the "globally best" substructure is found. This process can be repeated to obtain a hierarchical decomposition [1, 6, 8].

Let us note that this approach to algorithm parallelization may come at a price. Since the original graph is divided into subgraphs it is possible that some structures that existed in the original dataset will be lost. This happens when the input is partitioned by removing some edges, thus potentially cutting important substructures and dividing them between multiple processors. To maximally offset this problem a high-quality graph partition program (METIS) is used, which eliminates only a minimal number of edges; thus reducing possibility of splitting important substructures.

### 3.1 MPI-SUBDUE

In the original version of MPI-SUBDUE [8], the input graph was partitioned into $n$ subgraphs using the METIS package [2]. In METIS 4.0, two partitioning programs *pmetis* and *kmetis* can be used to partition a graph. The *pmetis* is based on multilevel recursive bisection [5], whereas the *kmetis* is based on multilevel $k$-way partitioning [4]. Both routines produce high quality partitions. However, as specified in [2], *kmetis* is expected to be considerably faster than *pmetis* when the number of partitions is larger than $n = 8$.

A series of point-to-point communications (MPI_Send and MPI_Recv) were used to exchange information between processors. There is a total of three situations in MPI-SUBDUE, when inter-processor communication takes place: (1) when the best local substructures are exchanged, (2) when the results of local comparisons are propagated, and (3) when the final best substructure is established (see above and [8]).

### 3.2 NEW-MPI-SUBDUE

We made three improvements to the original version of MPI-SUBDUE. First, we have explored parallelism in the graph partitioning. The input graph is now partitioned using the PARMETIS (version 3.1) graph partitioning package [3]. Second, instead of using point-to-point communications (MPI_Send and MPI_Recv routines), the MPI collective communication (MPI_Allgatherv routine) is used. Obviously, point-to-point communication involves a pair of processors and use of MPI_Send and MPI_Recv puts excessive communication demands on the system, which deteriorates the program's performance. Collective communication, on the other hand, involves every process in a group, and allows all machine resources to be exploited more efficiently. Furthermore in many cases, vendors offer machine-tuned, efficient implementations of collective communication [7]. Third, hierarchical (binary-tree based) summation is used in finding globally best substructures. After being evaluated on all partitions, the "scores" of substructures are propagated up the hierarchy, where at each internal node of the tree these "scores" are added for substructure reported from two different partitions, and their sum is passed further up the hierarchy. The algorithm of the

NEW-MPI-SUBDUE is summarized in Figure 1. Interestingly, as we will see in

```
NEW-MPI-SUBDUE(Graph)
Global variables: n, j, Value[0…(n-1)], Pos_Instances[0…(n-1)], Neg_Instances[0…(n-1)]
begin
  spawn(P0, P1, P2, P3, …, Pn);
  apply PARMETIS to partition the graph into n partitions;
  for all Pi where 1 ≤ i ≤ n do
    each processor discovers the best substructure in its graph partition;
    each processor broadcasts its best substructure to all other processors;
    each processor evaluates its best substructure and broadcasts the results to all other processors;
    each processor stores the value, number of positive instances, number of negative instances of
      the best substructures in Value[0…(n-1)],Pos_Instances[0…(n-1)],
      and Neg_Instances[0…(n-1)], respectively;
    for j ← 0 to log(n) do
      if (i-1) mod 2^j = 0 and i-1+2^j < n then
        Value[i-1] ← Value[i-1] + Value[i-1 + 2^j];
        Pos_Instances[i-1] ← Pos_Instances[i-1] + Pos_Instances[i-1+2^j ];
        Neg_Instances[i-1] ← Neg_Instances[i-1] + Neg_Instances[i-1+2^j ];
        each processor broadcasts Value[i-1], Pos_Instances[i-1], Neg_Instances[i-1]
          to all other processors;
      endif
    endfor
    each processor updates the values of its best substructure with its corresponding elements
      in Value[0…(n-1)], Pos_Instances[0…(n-1)], and Neg_Instances[0…(n-1)] and
      sends the updated best substructures to P0;
    endfor
    P0 finds and outputs the global best substructures
end
```

**Fig. 1.** The NEW-MPI-SUBDUE algorithm

the next section, set of changes that could be considered relatively insignificant, resulted in a very significant performance improvement.

## 4 Experimental Results

Let us now illustrate the performance of the NEW-MPI-SUBDUE in experiments performed on three input graphs that represent mutagenesis data and are derived from OxUni [9]. These datasets were collected in order to predict

the mutagenicity of aromatic and heteroaromatic nitro compounds. Here, mutagenicity is denoted by positive or negative real numbers. There are four levels of background knowledge in the database: atoms in the molecules, bonds in the molecules, and two attributes describing the molecule as a whole and higher level submolecular structures. The atom and bond structures were obtained from a standard molecular modeling package called Quanta [9]. In our current experiments, we were concerned only with performance improvements of the NEW-MPI-SUBDUE, and due to space limitations and since such results are outside of scope of this paper, we will not report on patterns found in the data.

First, we experimented with two small data graphs: Graph 1 had 2844 vertices and 2883 edges, while Graph 2 had 2896 vertices and 2934 edges. These two graphs are similar in size to these used in the original report [8] and we have found that the computer hardware has progressed so fast, that problems that were computationally challenging in 2000 are now much too small for an interesting comparison. We have therefore moved to the complete mutagenesis dataset (Graph 3 had 22268 vertices and 22823 edges), where the real performance gain could be observed.

Our experiments were performed on a cluster composed of 20 compute nodes, each with two AMD Athlon MP 1800+ (1.6GHz) CPUs. Each compute node has 2 GB of DDR SDRAM, and switching between compute nodes is provided by a 24-port full-backplane Gigabit Ethernet switch. All nodes run RedHat 9.0 operating system. In our work we used Portland Group C compiler version 5.0-2 and MPICH. All experiments were performed on an empty or lightly loaded machine. Results reported here, for all numbers of processors, represent best times obtained for multiple runs.

Since the available cluster had 40 processors and since we have used a binary summation algorithm, it was natural to utilize up to $2^5 = 32$ processors. Since the NEW-MPI-SUBDUE was implemented on the basis of the MPI-SUBDUE, we used the same master-slave approach and processor $P_0$ coordinated the execution of the program. Therefore, a total of 33 processors were used in the largest experiment. To obtain a complete performance picture each input data-graph was partitioned into $2, \ldots, 32$ partitions. First, in Tables 1-2 we present the time of graph partition of the original MPI-SUBDUE (sequential partition) and the NEW-MPI-SUBDUE (parallel partition). When experimenting the original MPI-SUBDUE, since there are two programs in the sequential partition, *pmetis* and *kmetis*, and *kmetis* is considerably faster than *pmetis* when the number of partitions is larger than 8 [2], we used *kmetis* for partitioning the graph into more than 8 subgraphs and *pmetis* for partitioning the graph into less than or equal to 8 subgraphs. In the NEW-MPI-SUBDUE, we used PARMETIS for graph partitioning.

Two observations can be made. First, as expected, parallel performance of PARMETIS is not very impressive, as we observe almost no speedup. However, thanks to its parallelism the partition time remains almost flat, whereas the time for the sequential partition increases with the number of partitions required. Therefore, second, when the total execution time of parallel SUBDUE is the

**Table 1.** Partition time for Graphs 1 and 2

| | Graph 1 | | Graph 2 | |
|---|---|---|---|---|
| N | Execution time METIS (seconds) | Execution time PARMETIS (seconds) | Execution time METIS (seconds) | Execution time PARMETIS (seconds) |
| 2 | 0.145 | 0.081 | 0.147 | 0.082 |
| 4 | 0.164 | 0.053 | 0.169 | 0.073 |
| 8 | 0.164 | 0.076 | 0.172 | 0.071 |
| 16 | 0.182 | 0.075 | 0.204 | 0.070 |

**Table 2.** Partition time for Graph 3

| N | Execution time METIS (seconds) | Execution time PARMETIS (seconds) |
|---|---|---|
| 2 | 1.009 | 0.700 |
| 4 | 1.268 | 0.621 |
| 8 | 1.269 | 0.597 |
| 16 | 1.425 | 0.599 |
| 32 | 1.626 | 0.601 |

shortest (for the largest number of processors), the sequential partition time is the longest and thus its effect (approached from Amdahl's Law perspective) is the most significant. However, third, the overall partition time is very small in comparison with the total execution time.

In Tables 3-5 we depict performance of the complete execution of the two versions of parallel SUBDUE obtained for running 1 iterations (finding the best substructure only). We also present the sizes of the largest and smallest partition and the actual performance gain.

**Table 3.** Experimental results for Graph 1 for MPI-SUBDUE & NEW-MPI-SUBDUE

| | | Number of vertices | | Execution time | | |
|---|---|---|---|---|---|---|
| N | P | Minimum | Maximum | MPI-SUBDUE (seconds) | NEW-MPI-SUBDUE (seconds) | Improvement |
| 2 | 3 | 1376 | 1468 | 28 | 9 | 68% |
| 4 | 5 | 586 | 790 | 6 | 4 | 33% |
| 8 | 9 | 244 | 412 | 2 | 2 | 0% |
| 16 | 17 | 96 | 236 | 1 | 1 | 0% |

The performance improvement of NEW-MPI-SUBDUE over the original MPI-SUBDUE is significant. While the performance gains of up to 68% in Graph 1

**Table 4.** Experimental results for Graph 2 for MPI-SUBDUE & NEW-MPI-SUBDUE

| | | Number of vertices | | Execution time | | |
|---|---|---|---|---|---|---|
| N | P | Minimum | Maximum | MPI-SUBDUE (seconds) | NEW-MPI-SUBDUE (seconds) | Improvement |
| 2 | 3 | 1404 | 1492 | 21 | 7 | 67% |
| 4 | 5 | 614 | 790 | 6 | 5 | 17% |
| 8 | 9 | 218 | 496 | 2 | 1 | 50% |
| 16 | 17 | 84 | 248 | 1 | 1 | 0% |

**Table 5.** Experimental results for Graph 3 for MPI-SUBDUE & NEW-MPI-SUBDUE

| | | Number of vertices | | Execution time | | |
|---|---|---|---|---|---|---|
| N | P | Minimum | Maximum | MPI-SUBDUE (seconds) | NEW-MPI-SUBDUE (seconds) | Improvement |
| 2 | 3 | 10914 | 11354 | 653 | 514 | 21% |
| 4 | 5 | 5362 | 5756 | 178 | 132 | 26% |
| 8 | 9 | 2580 | 2932 | 58 | 45 | 22% |
| 16 | 17 | 1218 | 1560 | 23 | 14 | 39% |
| 32 | 33 | 458 | 870 | 12 | 8 | 33% |

and 67% in Graph 2 were obtained, they are not very significant due to the short total execution time of the code. The realistic picture of performance gains is illustrated in the case of the large Graph 3. Here the gain ranges between 21% and 39% and slowly increases with the number of processors used.

The single-processor execution time of SUBDUE applied to Graph 3 is 2149 seconds and this means that a total speedup of approximately 268 is obtained on 32 processors. This results is easily explained when one considers the fact that when the data-graph is divided, then each processor operates only on a subgraph. Since formulas that express complexity of SUBDUE, while known to be polynomial in terms of the number of vertices and edges, depend also on the density of the graph, and thus we cannot easily utilize them in the case of our graph. However, we can see in our final experiment (Graph 3) that increase of the number of processors by a factor of 2 results in each case in decrease of time by a factor of 4 and this gives us some indication about the complexity of the SUBDUE when applied to this particular data-graph. One could therefore suggest that even when 4 processors are used, graph could be divided into 32 partitions (8 partitions per processor) and in this way the total execution time reduced. Unfortunately, this approach may be counter-productive as it may lead to problems described above as by splitting the data-graph into unnecessary sub-graphs we may be loosing information about important substructures.

It is worth noting that the even though the mutagenesis dataset leads to a very sparse graph the results of applying PARMETIS are not very good (especially for large number of processors). Let us consider Graph 3 and $n = 32$

processors. In this case an optimal partition should be into subgraphs of size 696, whereas the actual minimum partition size was 458, while the maximum 870. This immediately points to a serious workload imbalance. If parallel SUBDUE is to be successful for even larger datasets, balancing sizes of subgraphs needs to become one of research priorities.

## 5 Conclusions

In this paper, by exploring data parallelisms and applying MPI collective communication, a NEW-MPI-SUBDUE algorithm was implemented and tested with three mutagenesis datasets (two small ones and one large). The experimental results show that the NEW-MPI-SUBDUE algorithm provides performance that is approximately 20-30% better that the original MPI-SUBDUE.

## References


1. Cook, D.J., Holder, L.B., Galal, G., Maglothin, R.: Approaches to Parallel Graph-Based Knowledge Discovery. Journal of Parallel and Distributed Computing, 61(3) (2001) 427-446
2. Karypis, G., Kumar, V.: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0. University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN (1998)
3. Karypis, G., Schloegel, K., Kumar, V.: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Ver. 3.1. University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN (2003)
4. Karypis, G., Kumar, V.: Multilevel K-way Partitioning Scheme for Irregular Graphs. Journal of Parallel and Distributed Computing, 48(1) (1998) 96-129
5. Karypis, G., Kumar, V.: A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing (1998)
6. Galal, G.M., Cook, D.J., Holder, L.B.: Improving Scalability in a Knowledge Discovery System by Exploiting Parallelism In the Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (1997) 171-174
7. Gorlatch, S.: Send-Receive Considered Harmful: Myth and Realities of Message Passing. ACM Transaction on Programming Languages and Systems, Vol. 26, No. 1. (January 2004)
8. http://cygnus.uta.edu/subdue/
9. http://www-ai.ijs.si/∼ilpnet2/apps/pm.html