Proceedings, EuroPVM/MPI 2005, Sep. 18-21, Sorrento, Italy, LNCS, Springer-Verlag, 2005.

Nesting OpenMP in MPI to Implement a Hybrid Communication Method of Parallel Simulated Annealing on a Cluster of SMP Nodes

Agnieszka Debudaj-Grabysz[1] and Rolf Rabenseifner[2]

[1] Silesian University of Technology, Department of Computer Science
Akademicka 16, 44-100 Gliwice, Poland
agrabysz@star.iinf.polsl.gliwice.pl
[2] High-Performance Computing-Center (HLRS), University of Stuttgart
Nobelstr. 19, D-70550 Stuttgart, Germany
rabenseifner@hlrs.de, www.hlrs.de/people/rabenseifner

Abstract. Concurrent computing can be applied to heuristic methods for combinatorial optimization to shorten computation time, or equivalently, to improve the solution when time is fixed. This paper presents several communication schemes for parallel simulated annealing, focusing on a combination of OpenMP nested in MPI. Strikingly, even though many publications devoted to either intensive or sparse communication methods in parallel simulated annealing exist, only a few comparisons of methods from these two distinctive families have been published; the present paper aspires to partially fill this gap. Implementation for VRPTW—a generally accepted benchmark problem—is used to illustrate the advantages of the hybrid method over others tested.

Key words: parallel processing, MPI, OpenMP, communication, simulated annealing

1 Introduction

The paper presents a new algorithm for parallel simulated annealing—a heuristic method of optimization—that uses both MPI [9] and OpenMP [12] to achieve significantly better performance than a pure MPI implementation. This new hybrid method is compared to other versions of parallel simulated annealing, distinguished by varying level of inter-process communication intensity. Defining the problem as searching for the optimal solution given a pool of processors available for a specified period of time, the hybrid method yields distinctively better optima as compared to other parallel methods. The general reader (i.e., not familiar with simulated annealing) will find the paper interesting as it refers to a practical parallel application run on a cluster of SMPs with the number of processors ranging into hundreds.

Simulated annealing (SA) is a heuristic optimization method used when the solution space is too large to explore all possibilities within a reasonable amount

of time. The vehicle routing problem with time windows (VRPTW) is an example of such a problem. Other examples of VRPTW are school bus routing, newspaper and mail distribution or delivery of goods to department stores. Optimization of routing lowers distribution costs and parallelization allows to find a better route within the given time constraints.

The SA bibliography focuses on the sequential version of the algorithm (e.g., [2, 15]), however parallel versions are investigated too, as the sequential method is considered to be slow when compared with other heuristics [16]. In [1, 3, 8, 10, 17] and many others, directional recommendations for parallelization of SA can be found. The only known detailed performance analyses of intensive versus sparse communication algorithms are in [4, 11, 13].

VRPTW—formally formulated by Solomon [14], who also proposed a suite of tests for benchmarking, has a rich bibliography as well (e.g., [16]). Nevertheless, parallel SA to solve the VRPTW is discussed only in [4, 6, 7].

The parallel implementation of SA presented in this paper had to overcome many practical issues in order to achieve good parallel speedups and efficiency. Tuning of the algorithms for distributed as well as for shared memory environment was conducted.

The plan of the paper is as follows: Section 2 presents the theoretical basis of the sequential and parallel SA algorithm. Section 3 describes how the MPI and OpenMP parallelization was done, while Section 4 presents the results of the experiments. Conclusions follows.

2 Parallel simulated annealing

In simulated annealing, one searches for the optimal state, i.e., the state that gives either the minimum or maximum value of the *cost function*. It is achieved by comparing the current solution with a random solution from a specific *neighborhood*. With some probability, worse solutions could be accepted as well, which can prevent convergence to local optima. However, the probability of accepting a worse solution decreases during the process of annealing, in sync with the parameter called *temperature*. An outline of the SA algorithm is presented in Figure 1, where a single execution of the innermost loop step is called a *trial*. The sequence of all trials within a temperature level forms a *chain*. The returned final solution is the best one ever found.

2.1 Decomposition and communication

Although SA is often considered to be an inherently sequential process since each new state contains modifications to the previous state, one can isolate *serialisable sets* [8]—a collection of rejected trials which can be executed in any order, and the result will be the same (starting state). Independence of searches within a serialisable set makes the algorithm suitable for parallelization, where the creation of random solutions is decomposed among processors. From the communication point of view SA may require broadcasting when an acceptable

```
01  S ← GetInitialSolution();
02  T ← InitialTemperature;
03  for i ← 1 to NumberOfTemperatureReduction do
04      for j ← 1 to EpochLength do
05          S′ ← GetSolutionFromNeighborhood();
06          ΔC ← CostFunction(S′) - CostFunction(S);
07          if (ΔC < 0 or AcceptWithProbabilityP(ΔC, T))
08              S ← S′;     {i.e., the trial is accepted}
09          end if;
10      end for;
11      T ← λT;     {with λ < 1}
12  end for;
```

Fig. 1. SA algorithm

solution is found. This communication requirement suggests message passing as the suitable paradigm of communication, particularly if intended to run on a cluster.

2.2 Possible intensity of communication in parallel simulated annealing

Selection of both decomposition and communication paradigms seems to be naturally driven by the nature of the problem, but setting the right intensity of communication is not a trivial task. The universe of possible solutions is spanned by two extremes: communicating each event, where *event* means an *accepted trial*, and, independent runs method, where no event is communicated. The former method results in the *single chain algorithm*—only a single path in the search space is carried out, while the latter results in the *multiple chains algorithm*—several different paths are evaluated simultaneously (see Figure 2). The location of starting points depends on implementation.

Intensive communication algorithm—the time stamp method. In current research the intensive communication algorithm is represented by its speed-up optimized version called the *time stamp method*. The communication model with synchronization at solution acceptance events proposed in [7] was the starting point. The main modification, made for efficiency reasons, is to let processes work in an asynchronous way, instead of frequent computation interruptions by synchronization requests that resulted in idle time. After finding an accepted trial, the process announces the event and continues its computation without any synchronization. In the absence of the mechanism which ensures that all processes are aware of the same, global state and choose the same, accepted solution, a single process can decide only locally, based on its own state and information included in received messages. Information about the real time when the accepted solution was found—the *time stamp*—is used as the criterion for

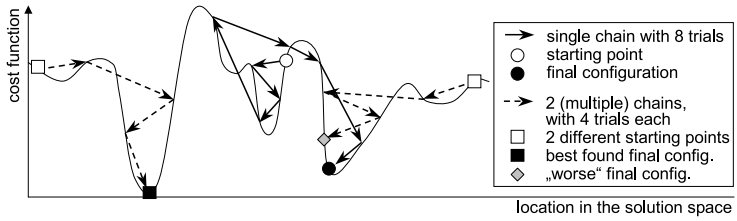

Fig. 2. One single chain versus multiple chains.

choosing among a few acceptable solutions (known locally). The solution with the most recent time stamp is accepted, while older ones are rejected. From the global point of view the same solutions will be preferred.

Generally, the single chain approach is believed to have two main drawbacks: only limited parallelism is exploited due to the reduction to a single search path and noticeable communication overhead. The second drawback especially reduces the application of this method to a small number of engaged processes.

Non-communication algorithm—independent runs. The main assumptions for independent runs were formulated in [2], where the *division algorithm* is proposed. The method uses all available processors to run basically sequential algorithms, where the original chain is split into subchains of *EpochLength* (see Figure 1) divided by the number of processes. At the end, the best solution found is picked up as the final one; thus the communication is limited to merely one reduction operation.

Although the search space is exploited in a better way than in the approach described previously, very likely only a few processes work in the "right" areas while the rest perform useless computation. Additionally, excessive shortening of the chain length negatively affects the quality of results, so application of this method is not suitable for a great number (e.g., hundreds) of engaged processes.

Lightweight communication—periodically interacting searches. Recognizing the extreme character of the independent runs method, especially when using a large number of processes, one is tempted to look for the golden mean in the form of periodic communication. The idea was fully developed in [11]. In that approach processes communicate after performing a subchain called a *segment*, and the best solution is selected and mandated for all of them. In this study a segment length is defined by a number of temperature decreases. As suggested in [11] to prevent search paths from being trapped in local minima areas as a result of communication, the period of the information exchange needs to be carefully selected. Additionally, the influence of the periodic exchange doesn't always result in a positive effect and varies according to the optimized problem.

Hybrid communication method—nesting OpenMP in MPI. In this study a new approach is proposed, which tries to adopt the advantages of the methods mentioned above while minimizing their disadvantages. In contrast with

these methods, this implementation is intended to run on modern clusters of SMP nodes. The parallelization is accomplished using two levels: the outer parallelization which uses MPI to communicate between SMP nodes, and the inner parallelization which uses OpenMP for shared memory parallelization within nodes.

Outer-level parallelization. It can be assumed that the choice of an appropriate algorithm should be made between independent runs or periodically interacting searches, as they are more suitable for more than few processes. The maximal number of engaged nodes is limited by reasonable shortening of the chain length, to preserve an acceptable quality of results.

Inner-level parallelization. Within a node a few threads can build one subchain of a length determined at the outer-level. Negligible deterioration of quality is a key requirement. If this requirement is met, the limit on the total number of processors to achieve both speed-up and preserve quality is determined by the product of the processes number limit at the outer level and the threads number limit at the inner level. An efficient implementation can also take advantage of the fact that CPUs on SMP nodes communicate by fast shared memory and communication overhead should be minimal relative to that between nodes. In this study a modified version of the *simple serialisable set* algorithm [8] was applied (see Section 3). For a small number of processors (i.e., 2 to 8), apart from preserving the quality of solutions, it should provide speed-up.

3 Implementation of communication with MPI and OpenMP

3.1 Intensive communication algorithm

Every message contains a solution together with its time stamp. As the assumption was to let the processes work asynchronously polling is applied to detect moments when data is to be received. An outline of the algorithm is presented in Figure 3.

In practice, as described in [7], the implementation underwent a few stages of improvement to yield acceptable speed-up. Among others: a long message containing a solution was split into two, to test the differences in performance when sending different types of data, data structure was reorganized—an array of structures was substituted by a structure of arrays, MPICH2 was used since there was a bug in MPICH that prevented the program from running.

3.2 Non– and lightweight communication algorithms

In case of both independent runs and periodically interacting searches methods, MPI reduction instructions (MPI_Bcast, MPI_Allreduce) are the best tools for exchanging the data.

```
01 MyData.TimeStamp ← 0;
02 do in parallel
03    do
04       MPI_Iprobe();    { check for incoming messages }
05       if (there is a message to receive)
06          MPI_Recv(ReceivedData, ...);
07          if (MyData.TimeStamp < ReceivedData.TimeStamp)
08             update MyData and current TimeStamp;
09          end if;
10       end if;
11    while (there is any message to receive);
12    performTrial();
13    if (an acceptable solution was found, placed in MyData.Solution)
14       MyData.TimeStamp ← MPI_Wtime();
15       for all cooperating processors do
16          MPI_Send(MyData, ...);
17       end for;
18    end if;
19 while (not Finish);
```

Fig. 3. The outline of the intensive communication algorithm

3.3 Hybrid communication method

The duality of the method is extended to its communication environment: MPI is used for communication between the nodes and OpenMP for communication among processors within a single node. The former algorithm is implemented as described in the previous section (3.2), whereas an outline of the latter one is presented in Figure 4.

At the inner-level, the total number of trials (*EpochLength* from the outer level) in each temperature step is divided into short sets of trials All trials in such a set are done independently. This modification is the basis for the OpenMP parallelization with loop worksharing. To achieve an acceptable speed-up, the following optimizations are necessary:

- The parallel threads must not be forked and joined for each inner loop because the total execution time for a set of trials can be too short, compared to the OpenMP fork-join overhead;
- The size of such a set must be larger than the number of threads to minimize the load imbalance due to the potentially extremely varying execution time for each trial. Nevertheless, for keeping quality, the size of the set of trials should be as short as possible to minimize the number of accepted but unused trials;
- Each thread has to use its own independent random number generator to minimize OpenMP synchronization points.

```
01  for i ← 1 to NumberOfTemperatureReduction do
02      {entering OpenMP parallel region}
03      for j ← 1 to EpochLength do
04          {OpenMP parallel for loop worksharing}
05          for i ← 0 to set_of_trials_size do
06              performTrial();
07          end for;
08          {OpenMP entering master section}
09          select one solution, common for all threads, from all
            accepted ones, based on ΔC < 0 or AcceptWithProbabilityP(ΔC, T)
10          j ← j + set_of_trials_size;
11          {OpenMP end of master section}
12      end for;
13      {end of OpenMP parallel region}
14      T ← λT;
15  end for;
```

Fig. 4. Parallel SA algorithm within a single node

4 Experimental results

In the vehicle routing problem with time windows it is assumed that there is a warehouse, centrally located to n customers. The objective is to supply goods to all customers at the minimum cost. The solution with lesser number of route legs (the first goal of optimization) is better then a solution with smaller total distance traveled (the second goal of optimization). Each customer as well as the warehouse has a time window. Each customer has its own demand level and should be visited only once. Each route must start and terminate at the warehouse and should preserve maximum vehicle capacity. The sequential algorithm from [5] was the basis for parallelization.

Experiments were carried out on NEC Xeon EM64T Cluster installed at the High Performance Computing Center Stuttgart (HLRS). Additionally, for tests of the OpenMP algorithm, NEC TX-7 (ccNUMA) system was used. The numerical data were obtained by running the program 100 times for Solomon's [14] R108 set with 100 customers and the same set of parameters.

The quality of results, namely the number of final solutions with the minimal number of route legs generated by pure MPI-based algorithms in 100 experiments is shown in Table 1. Experiments stopped after 30 consecutive temperature decreases without improving the best solution. As can be seen in the table, the intensive communication method gives acceptable results only for a small number of cooperating processes. Secondly, excessively frequent periodical communication hampers the annealing process and deteriorates the convergence. The best algorithm for the investigated problem on a large number of CPUs, as far as the quality of results is concerned, is the algorithm of independent runs, so this one was chosen for the development of the hybrid method.

Table 1. Comparison of quality results for MPI based methods

No. of processes	Percentage of solutions with minimal no. of route legs						
	Non-comm.	Periodic communication with the period of					Intensive comm.
		1	5	10	20	30	
seq	94	N/A	N/A	N/A	N/A	N/A	N/A
2	97	95.6	96.2	96.8	94.2	96	91
4	95	93	96	93	93	96	96
8	91	91	82	86	90	91	93
10	94	85	88	85	88	96	82
20	84	70	77	77	74	89	69
40	85	56	60	63	71	74	N/A
60	76	30	46	55	60	68	N/A
100	60	32	38	35	44	55	N/A
200	35	12	23	30	38	37	N/A

The results generated by the hybrid algorithm are shown in Table 2. It compares two methods using the same number of processors, e.g., 20 processor independent runs (MPI parallelization) versus computation on 10 nodes with 2 CPUs each or on 5 nodes with 4 CPUs each (MPI/OMP parallelization). For better comparison, a real time limit was fixed for each independent set of processors. The time limit is the average time needed by the sequential algorithm to find the minimal-leg solution divided by the number of processors. The hybrid version of the algorithm with 2 OMP threads per each node ran on NEC Xeon Cluster, while the usage of 4 OMP threads per node was emulated, due to the lack of access to the desired machine. Because a separate set of experiments with 4 OMP threads demonstrated the speed-up of 2.7, then the emulation was carried out by multiplying the applied time limit by the this factor, as if undoing the speed-up to be observed on a real cluster of SMP nodes. The accumulated CPU time of a real experiment would be shortened by the factor $2.7/4 = 0.67$.

It should be noted that both variants of the hybrid method give a distinctively greater number of solutions with the minimal number of route legs if one uses 32 or more CPUs. Additionally, for smaller number of CPUs, the 4 OMP threads version could be competitive as well (up to 40 CPUs), despite the loss of CPU-time due to the limited efficiency of the parallelization inside of each SMP node. If one can accept a reduced quality, e.g. 85%, then only a speed-up of 40 can be achieved without SMP parallelization. With hybrid parallelization, the speed-up can be raised to 60 (with 2 threads) and 120 (with 4 threads), i.e., an interactive time-scale of about 15 sec can be reached.

5 Conclusions and future work

In this study a new implementation of the multiple chain parallel SA that uses OpenMP with MPI was developed. Additionally, within the framework of the

Table 2. Comparison of quality results for hybrid and independent runs methods

			Hyb., 2 OMP threads		Hyb., 4 OMP threads		Non-comm.
Total no. of used processors	Used time limit [s]	Speed -up	No. of MPI processes	No. of sol. with min. no. of route legs	No. of MPI processes	No. of sol. with min. no. of route legs	No. of sol. with min. no. of route legs
1	1830.0	N/A	N/A	N/A	N/A	N/A	97
4	457.5	4	2	92	1	96	95
16	114.4	16	8	93	4	97	93
20	91.5	20	10	93	5	93	94
32	57.2	32	16	90	8	90	85
40	45.8	40	20	92	10	93	85
60	30.5	60	30	86	15	91	76
80	22.9	80	40	78	20	88	69
100	18.3	100	50	87	25	87	64
120	15.3	120	60	67	30	85	62
200	9.2	200	100	55	50	78	34
400	4.6	400	200	27	100	57	9
600	3.1	600	300	9	150	31	0
800	2.3	800	400	N/A	200	13	0

single chain parallel SA, a time-stamp method was proposed. Based on experimental results the following conclusions may be drawn:

- Multiple chain methods outperform single chain algorithms, as the latter lead to a faster worsening of results quality and are not scalable. Single chain methods could be used only in environments with a few processors;
- The periodically interacting searches method prevails only in some specific situations; generally the independent runs method achieves better results;
- The hybrid method is very promising, as it gives distinctively better results than other tested algorithms and satisfactory speed-up;
- Emulated results shown need verification on a cluster of SMPs with 4 CPUs on a single node.

Specifying the time limit for the computation, by measurements of the elapsed time, gives a new opportunity to determine the exact moment to exchange data. Such a time-based scheduling could result in much better balancing than the investigated temperature-decreases-based one (used within the periodically interacting searches method). The former could minimize idle times, as well as enables setting the number of data exchanges. Therefore, future work will focus on forcing a data exchange (e.g., after 90% of specified limit time), when—very likely—the number of route legs was finally minimized (first goal of optimization). Then, after selecting the best solution found so far, all working processes—instead of only one—could minimize the total distance (the second goal of optimization), leading to significant improvement of the quality of results.

Acknowledgment

This work was supported by the EC-funded project HPC-Europa. Computing time was also provided within the framework of the HLRS-NEC cooperation.

References

1. Aarts, E., de Bont, F., Habers, J., van Laarhoven, P.: Parallel implementations of the statistical cooling algorithm. Integration, the VLSI journal (1986) 209–238
2. Aarts, E., Korst, J.: Simulated Annealing and Boltzman Machines, John Wiley & Sons (1989)
3. Azencott, R. (ed): Simulated Annealing Parallelization Techniques. John Wiley & Sons, New York (1992)
4. Arbelaitz, O., Rodriguez, C., Zamakola, I.: Low Cost Parallel Solutions for the VRPTW Optimization Problem, Proceedings of the International Conference on Parallel Processing Workshops, IEEE Computer Society, Valencia–Spain, (2001) 176–181
5. Czarnas, P.: Traveling Salesman Problem With Time Windows. Solution by Simulated Annealing. MSc thesis (in Polish), Uniwersytet Wrocławski, Wrocław (2001)
6. Czech, Z.J., Czarnas, P.: Parallel simulated annealing for the vehicle routing problem with time windows. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands–Spain, (2002) 376–383
7. Debudaj-Grabysz, A., Czech, Z.J.: A concurrent implementation of simulated annealing and its application to the VRPTW optimization problem, in Juhasz Z., Kacsuk P., Kranzlmuller D. (ed), Distributed and Parallel Systems. Cluster and Grid Computing. Kluwer International Series in Engineering and Computer Science, Vol. 777 (2004) 201–209
8. Greening, D.R.: Parallel Simulated Annealing Techniques. Physica D, 42, (1990) 293–306
9. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing 22(6) (1996) 789–828
10. Lee, F.A.: Parallel Simulated Annealing on a Message-Passing Multi-Computer. PhD thesis, Utah State University (1995)
11. Lee, K.–G., Lee, S.–Y.: Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains, IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 10 (1996) 993–1008
12. OpenMP C and C++ API 2.0 Specification, from www.openmp.org/specs/
13. Onbaoglu, E., Özdamar, L.: Parallel Simulated Annealing Algorithms in Global Optimization, Journal of Global Optimization, Vol. 19, Issue 1 (2001) 27–50
14. Solomon, M.: Algorithms for the vehicle routing and scheduling problem with time windows constraints, Operation Research 35 (1987) 254–265, see also http://w.cba.neu.edu/˜msolomon/problems.htm
15. Salamon, P., Sibani, P., and Frost, R.: Facts, Conjectures and Improvements for Simulated Annealing, SIAM (2002)
16. Tan, K.C, Lee, L.H., Zhu, Q.L., Ou, K.: Heuristic methods for vehicle routing problem with time windows. Artificial Intelligent in Engineering, Elsevier (2001) 281–295
17. Zomaya, A.Y., Kazman, R.: Simulated Annealing Techniques, in Algorithms and Theory of Computation Handbook, CRC Press LLC, (1999)