Multi-installment Divisible Load Processing in Heterogeneous Systems with Limited Memory

M.Drozdowski[1]* and M.Lawenda[2]

[1] Institute of Computing Science, Poznań University of Technology, ul.Piotrowo 3A, 60-965 Poznań, and Faculty of Mathematics and Computer Science, Adam Mickiewicz University, ul. Umultowska 87, 61-614 Poznań, Poland
Maciej.Drozdowski@cs.put.poznan.pl

[2] Poznań Supercomputing and Networking Center, ul.Noskowskiego 10, 61-704 Poznań, Poland
Marcin.Lawenda@man.poznan.pl

Abstract. Optimum divisible load processing in heterogeneous star system with limited memory is studied. We propose two algorithms to find multi-installment load distribution: Exact branch-and-bound algorithm and a heuristic using genetic search method. Characteristic features of the solutions and the performance of the algorithms are examined in a set of computational experiments.

1 Introduction

Divisible load model represents computations with fine granularity, and negligible dependencies between the grains of computations. Consequently, the computations, or the load, can be divided into parts of arbitrary sizes, and these parts can be processed independently in parallel. Divisible load theory (DLT) proved to be a versatile vehicle in modeling distributed systems [3, 4, 10].

In this paper we assume a star communication topology. Heterogeneous processors (or workers, clients) receive load from a central server (or an originator, master) only. The load is sent in multiple small chunks, rather than in one long message. Since the amount of processor memory is limited, the accumulated load cannot exceed the memory limit. The problem is to find the sequence of processor communications, and the sizes of the load chunks such that the length of the schedule is minimum. We propose two algorithms for this problem. Exact branch and bound algorithm, and a heuristic based on genetic search.

Similar problems have already been studied. Multi-installment distribution was proposed in [2, 3], but the sequence of communications was fixed, and memory limits were not considered. Memory limitations and single installment communications were studied in [5–7, 9]. For a fixed communication sequence a fast heuristic was proposed in [9], and an optimization algorithm based on linear programming was given in [5]. A hierarchic memory system was studied in [6].

* This research has been partially supported by the Polish State Committee for Scientific Research.

A multi-installment load distribution with a fixed communication pattern was proposed to overcome out-of-core memory speed limitations. It was shown in [7] that finding optimum divisible load distribution in a system with limited memory sizes and affine communication delay is **NP**-hard. In [11] multi-installment divisible load processing with limited memory was studied, but the computer system was homogeneous and communication sequence was fixed. In this paper we assume that the sequence of communications can be arbitrary. The load is processed in multiple chunks by a heterogeneous system. To our best knowledge, none of the already published papers addressed the exact problem we study.

The rest of the paper is organized as follows. In Section 2 problem is formulated, in Section 3 solution methods are outlined, and results of computational experiments are reported in Section 4.

2 Problem Formulation

A set of processors $\{P_1, \ldots, P_m\}$ is connected to a central server P_0. By a processor we mean a processing element with CPU, memory, and communication hardware. Each processor P_i is defined by the following parameters: communication link startup time S_i, communication transfer rate C_i, processing rate A_i, and memory limit B_i. The time to transfer x load units from P_0 to P_i is $S_i + xC_i$. The time required to process the same amount of load is xA_i. Let n denote the number of load chunks sent by the originator to the processors. We will denote by α_j the size of chunk j. The total amount of load to process is V. Hence, $\sum_{j=1}^{n} \alpha_j = V$. The problem consists in finding the set of used processors, the sequence of their activation, and the sizes of the load chunks α_j such that schedule length C_{max}, including communication and computations, is the shortest possible. For the sake of conciseness we will mean both selecting the set of processors and their activation sequence while saying activation sequence. The optimum activation sequence must gear to the speeds of the processors, and available memory. Let d_j be the index of the destination processor for load chunk j, and $\overline{d} = (d_1, \ldots, d_n)$ a vector of processor destinations.

It is assumed that the amount of memory available at the processor is limited. If the new chunks arrive faster than the load is processed, then the load may accumulate in the processor memory. The method of memory management has influence on the conditions that must be met to satisfy memory limitations. Below we discuss some options. In all cases we assume that memory is allocated before the communication with the arriving load starts.

1) When load chunk j starts arriving, a memory block of size $\alpha_j \leq B_{d_j}$ is allocated. After processing chunk j the memory block is released to the operating system. This approach was assumed in the earlier papers [5–7, 9, 11]. Unfortunately, though each chunk uses less memory than B_{d_j}, the total accumulated memory consumption may be bigger. Hence, this method is not very effective in the case of multi-installment processing.

2) When load chunk j starts arriving, many small blocks of memory are allocated from the memory pool. The size of each small block is equal to the size

of the grain of parallelism. The total allocated memory size is α_j. As processing of the load progresses, the memory blocks are gradually released to the operating system. This method of memory management is illustrated in Fig.1a.

3) As in the first case, memory block of size α_j is allocated when chunk j starts arriving. It is released after processing chunk j. However, it is required that the total memory allocated on the processor never exceeds limit B_{d_j}. This method of memory management is illustrated in Fig.1b.

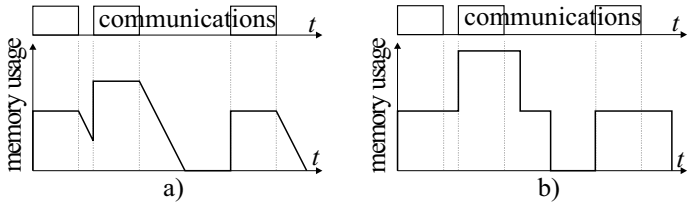

Fig. 1. Memory usage for a) management method 2, b) management method 3.

For the sake of simplicity of mathematical representation, in this work we use the second method of memory management. For the same reason we assume that: the time of returning the results of computations is negligible, and that processors cannot compute and communicate simultaneously. Consequently, computations are suspended by communications. We assume that the sequence of processor destinations $\overline{d}$ is given. Hence, we know number n_i of load chunks sent to processor P_i, and function $g(i,k) \in \{1,\ldots,n\}$ which is the global number of a chunk sent to processor P_i as k-th for $k = 1,\ldots,n_i$. Let t_j denote the time moment when sending load chunk j starts. We will denote by x_{ik} the amount of load that accumulated on processor P_i at the moment when communication k to P_i starts. The problem of the optimum chunk size selection can be formulated as the following linear program:

$$\min \quad C_{max}$$

$$t_j + S_{d_j} + \alpha_j C_{d_j} \leq t_{j+1} \qquad j = 1,\ldots,n-1 \tag{1}$$

$$x_{i,k-1} + \alpha_{g(i,k-1)} - \frac{t_{g(i,k)} - (t_{g(i,k-1)} + S_i + C_i\alpha_{g(i,k-1)})}{A_i} \leq x_{ik}$$
$$i = 1,\ldots,m, k = 2,\ldots,n_i \tag{2}$$

$$x_{ik} + \alpha_{g(i,k)} \leq B_i \qquad i = 1,\ldots,m, k = 1,\ldots,n_i \tag{3}$$

$$t_{g(i,n_i)} + S_i + C_i\alpha_{g(i,n_i)} + A_i(\alpha_{g(i,n_i)} + x_{in_i}) \leq C_{max} \quad i = 1,\ldots,m \tag{4}$$

$$\sum_{j=1}^{n} \alpha_j = V \tag{5}$$

$$x_{i1} = 0, x_{ik} \geq 0 \qquad i = 1,\ldots,m, k = 2,\ldots,n_i \tag{6}$$

$$t_j, \alpha_j \geq 0 \qquad j = 1,\ldots,n \tag{7}$$

In the above formulation inequality (1) guarantees that communications do not overlap. The amount of load accumulated on P_i at the moment when chunk k starts arriving must satisfy inequality (2). By inequality (3) memory limit is not exceeded. Computations finish before the end of the schedule by constraint (4), and the whole load is processed by equation (5). The above formulation can be adjusted to the case when simultaneous receiving a new load chunk and computation on the previous one is possible. In such a situation the following constraint should be added: $x_{ik} \geq \alpha_{g(i,k-1)} - \frac{1}{A_i}(t_{g(i,k)} - (t_{g(i,k-1)} + S_i + C_i\alpha_{g(i,k-1)}))$. We conclude that the optimum load distribution can be found provided that the sequence of processor communications $\overline{d}$ is given. In the next section we propose two methods of constructing $\overline{d}$.

3 Selecting Processor Activation Sequence

3.1 Branch and Bound Algorithm

Branch-and-bound (BB) algorithm is an enumerative method that generates and searches the space of possible solutions while eliminating these subsets of solutions which are infeasible or worse than some already known solution.

The search space consists of possible sequences $\overline{d}$. Solutions are generated expanding partial sequence $\overline{d(i)} = (d_1, \ldots, d_i)$, for $i < n$, by adding all possible destinations $d_{i+1} \in \{1, \ldots, m\}$, until obtaining complete sequences of length n. The generation of the sequences can be imagined as a construction of a tree with at most m^n leaves. Some branches of the tree can be pruned. Therefore, for each partial sequence $\overline{d(i)}$ a lower bound on the length of all $\overline{d(i)}$'s descendants is calculated. The lower bound is C_{max} obtained from linear program (1)-(7) by assuming that each load chunk $i+1, \ldots, n$ is sent to an ideal processor. An ideal processor P_{id} has all the best parameters in the processor set, i.e. $A_{id} = \min_{i=1}^{m}\{A_i\}, C_{id} = \min_{i=1}^{m}\{C_i\}, S_{id} = \min_{i=1}^{m}\{S_i\}, B_{id} = \max_{i=1}^{m}\{B_i\}$. If the lower bound is greater than or equal to the length of some already known solution then there is no hope that any of $\overline{d(i)}$ descendants will improve the schedule. If the resulting linear program (1)-(7) is infeasible then it means that volume V is greater than the available processor memory. In both cases $\overline{d(i)}$ is not expanded, and in this way the search tree is pruned.

3.2 Genetic Algorithm

Genetic algorithm (GA) is a randomized search method which implicitly discovers the optimum solution by randomly combining pieces of good solutions [8]. Here we present basics of our implementation of GA only.

A set of G solutions is a population. Solutions are encoded as strings $\overline{d} = (d_1, \ldots, d_n)$ of chunk destinations. The quality of solution $\overline{d}$ is the schedule length $C_{max}(\overline{d})$ obtained as a solution of the linear program (1)-(7) formulated for $\overline{d}$.

Solutions of the population are subject of genetic operators. We used three operators: crossover, mutation, and selection. Single-point crossover was applied.

The number of offspring generated by the crossover is Gp_C, where p_C is a tunable parameter which we will call crossover probability. Mutation changes Gnp_M randomly selected chunk destinations in the whole population. p_M is a tunable parameter which we call mutation probability. In the selection operation a new population is assembled. The best half of the old population solutions is always preserved. For the second half of the population some solution $\overline{d_j}$ is selected with probability $\frac{1}{C_{max}(\overline{d_j})} / \sum_{j=1}^{G} \frac{1}{C_{max}(\overline{d_j})}$. The populations are modified iteratively. The number of iterations is bounded by an upper limit on the total number of iterations, and an upper limit on the number of iterations without quality improvement.

4 Computational Experiments

4.1 Experiment Setting

BB and GA were implemented in Borland C++ 5.5 and tested in a set of computational experiments run on a PC computer with MS Windows XP. Linear programs were solved using simplex code derived from `lp_solve` [1]. Unless stated otherwise the instance parameters were generated with uniform distribution from ranges $[0, 2]$ for parameters A, C, S, and $[0, \frac{2V}{n}]$ for parameter B. Infeasible instances with $(n \max_{i=1}^{m}\{B_i\}) < V$ were discarded. We applied the following GA tuning procedure. 100 random instances with $m = 4$, and $n = 8$ were generated and solved by BB and GA. The average relative distance of GA solutions from the optimum calculated by BB was a measure of the tuning quality. First, population size $G = 40$ was selected as increasing G beyond 40 did not improve solution quality significantly. Second, crossover probability $p_C = 50\%$, and then mutation probability $p_M = 5\%$ were selected for which best quality was obtained in minimum number of iterations. Finally, the limits of iteration numbers 100 (without quality improvement), and 1000 (in total) were selected for which solution distance from optimum was better than 0.1%.

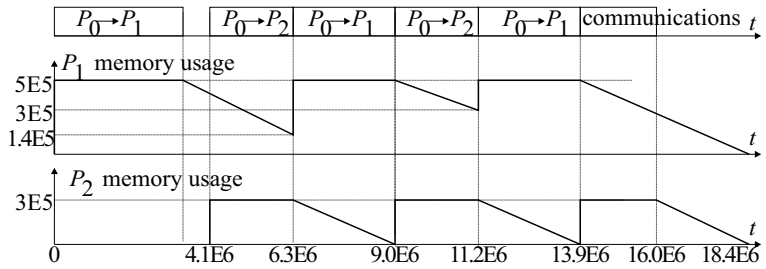

Fig. 2. Solution for $m = 2, n = 6, V$ =2E6, $A_1 = A_2 = 8.98, C_1 = C_2 = 7.39, S_1 = 2.01, S_2 = 3.02, B_1$ =5E5, B_2 =3E5.

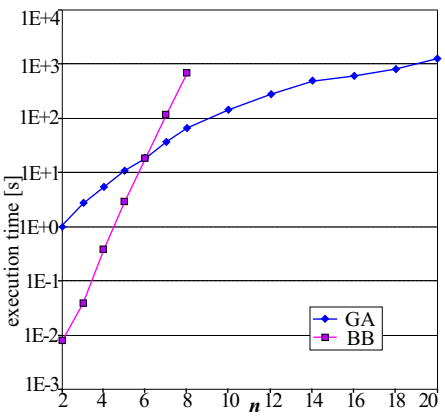

Fig. 3. Execution time vs. n, for $m = 4$.

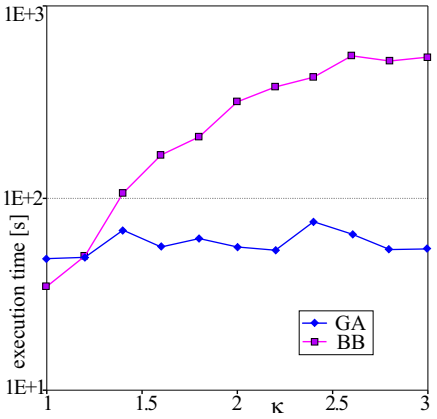

Fig. 4. Execution time vs. memory size, for $m = 4, n = 8, V =$ 1E6.

Now let us discuss features of the optimum solutions. In Fig.2 an optimum schedule constructed by BB algorithm is presented. It is a typical situation that memory buffers are filled to the maximum capacity. We observed that if the number of messages is small then memory buffers were empty when a new message arrived (i.e. $x_{ik} = 0$). With the increasing number of messages n we observed an increase in the number of the optimum solutions in which the old load is not completely processed on the arrival of the new load ($x_{ik} > 0$). Intuitively, this seems reasonable because when n is small each message must carry nearly a maximum load. If a message sent to processor P_i carries maximum load B_i, then the old load must be completely processed before receiving a new chunk.

4.2 Running Times

Dependence of the BB and GA execution times on n, B are shown in Fig.3, Fig.4, respectively. Each point in these diagrams is an average of at least ten instances. The worst case number of leaves visited in a BB search tree is m^n. Thus, the execution time of BB is exponential in n for fixed m (Fig.3). The execution time of GA grows with n because the length of the solution encoding and sizes of linear programs increase with n (Fig.3). GA running time dependence on m is weaker: 10-fold increase of m resulted in 60% increase of the execution time. In Fig.4 dependence of the running time on memory size is shown. For this diagram processor memory sizes (B_i) were generated with uniform distribution from range $[0, \kappa\frac{2V}{n}]$, where κ is shown along horizontal axis. With growing κ the size of available memory is growing on average. Consequently, more solutions are feasible, less branches can be cut in BB, and BB execution time grows. When κ is big, infeasibility of a solution becomes rare, and it is not limiting BB search tree. Hence, dependence of BB execution time on κ levels-off.

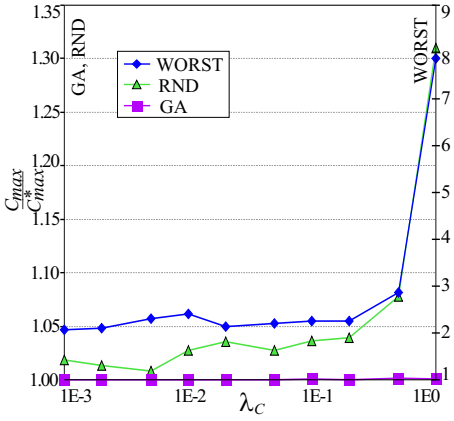

Fig. 5. Quality of the solutions vs. range of C, $m = 4, n = 8, V =$ 1E6.

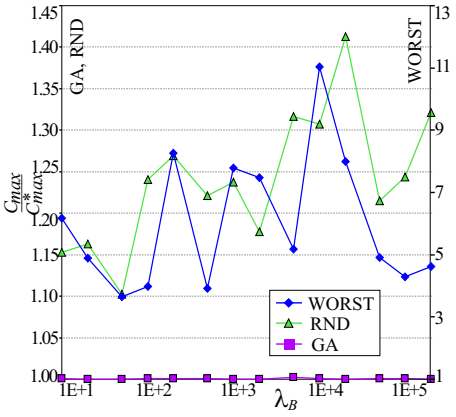

Fig. 6. Quality of the solutions vs. range of memory sizes, $m = 4, n = 8, V = 1E6$.

4.3 Quality of the Solutions

We observed that GA is very useful in deriving optimum, and near-optimum solutions. Over 55% of the instances were solved to the optimality by GA. The biggest observed relative distance from the optimum was 1.2%.

In Fig.5, Fig.6 we show dependence of the solutions quality on the range of C, B, respectively. Along the vertical axis a relative distance from the optimum is shown for three kinds of solutions: an average for the genetic algorithm (denoted GA), an average for a randomly selected sequence of destinations (RND), and for the worst case sequence ever observed (WORST). Load distributions for RND, WORST sequences were found from (1)-(7). A dedicated axis is used for the relation WORST. RND, and WORST are indicators of the characteristic features inherent in the problem itself. For Fig.5, the communication rates C_i were generated from range $[1-\lambda_C, 1+\lambda_C]$. The remaining parameters were generated as described above. Thus, with growing λ_C heterogeneity of the communication system was growing. Values of λ_C are shown along the horizontal axis. As it can be seen in Fig.5 with growing heterogeneity of parameter C the quality of both random and the worst case solutions is decreasing. Note, that this dependence is growing especially fast when C variation (λ_C) is big. For Fig.6 the memory sizes were generated from range $[\frac{2V}{n} - \lambda_B, \frac{2V}{n} + \lambda_B]$, for fixed value of V. The value of λ_B is shown along the horizontal axis. A trend of decreasing solution quality can be observed for growing λ_B. Note that in Fig.6 the distance from the optimum of RND, and WORST solutions is bigger than in Fig.5. Similar distance has been observed in the case of varying A. This demonstrates that narrowing the diversity of the system parameters simplifies obtaining a good solution, and communication rate C is a key parameter in performance optimization. Furthermore, the distance between WORST, RND, and GA or BB solutions can be used as an estimate of the gain from finding the optimum, or near-optimum, sequence

of processor activations. It can be inferred that this kind of gain is ≈ 10-40% on average (RND), and ≈ 10-fold in the worst case.

5 Conclusions

In this paper we studied optimum multi-installment divisible load processing in a heterogeneous distributed system with limited memory. A linear programming formulation has been proposed for a fixed processor activation sequence. Two algorithms were proposed to find an optimum, or near optimum processor activation sequences. The algorithm running times, and quality of the solutions were compared in a series of computational experiments. It turned out that the proposed genetic algorithm is very effective in finding near-optimum solutions. The impact of the system heterogeneity on the solution quality has been also studied. It appears that with growing system heterogeneity good solutions are harder to be found. Especially small communication speed diversity simplifies obtaining good solutions.

References

1. Berkelaar, M.: `lp_solve` - Mixed Integer Linear Program solver. `ftp://ftp.es.ele.tue.nl/pub/lp_solve` (1995)
2. Bharadwaj, V., Ghose, D., Mani, V.: Multi-installment Load Distribution in Tree Networks With Delays, IEEE Transactions on Aerospace and Electronic Systems **31** (1995) 555-567
3. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos CA (1996)
4. Drozdowski, M.: Selected problems of scheduling tasks in multiprocessor computer systems,. Poznań Univ. of Technology, Series: Monographs, No 321, Poznań (1997). Downloadable from `http://www.cs.put.poznan.pl/mdrozdowski/txt/h.ps`
5. Drozdowski, M., Wolniewicz, P.: Divisible load scheduling in systems with limited memory. Cluster Computing **6** (2003) 19-29
6. Drozdowski, M., Wolniewicz, P.: Out-of-Core Divisible Load Processing. IEEE Trans. on Parallel and Distributed Systems **14** (2003) 1048-1056
7. Drozdowski, M., Wolniewicz, P.: Optimum divisible load scheduling on heterogeneous stars with limited memory, accepted for publication in European Journal of Operational Research (2004)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989)
9. Li, X., Bharadwaj, V., Ko, C.C.: Optimal divisible task scheduling on single-level tree networks with buffer constraints. IEEE Trans. on Aerospace and Electronic Systems **36** (2000) 1298-1308
10. Robertazzi, T.: Ten reasons to use divisible load theory, IEEE Computer **36** (2003) 63-68
11. Wolniewicz, P., Drozdowski, M.: Processing Time and Memory Requirements for Multi-installment Divisible Job Processing. In R.Wyrzykowski (et al. eds.): Proceedings of 4th Int. Conf. PPAM 2001. Lecture Notes in Computer Science, Vol. 2328. Springer-Verlag, Berlin Heidelberg New York (2002) 125-133