Isoefficiency Maps for Divisible Computations in Hierarchical Memory Systems

Maciej Drozdowski[1][0000−0001−9314−529X]✉, Gaurav Singh[2], and Jędrzej M.Marszałkowski[1]

[1] Institute of Computing Science, Poznań University of Technology, Poland
{Maciej.Drozdowski;Jedrzej.Marszalkowski}@cs.put.poznan.pl
[2] Technology Strategy and Innovation, BHP, Perth, 6000, Australia
Gaurav.Singh@bhp.com

Abstract. In this paper we analyze impact of memory hierarchy on divisible load processing. Current computer systems have hierarchical memory systems. The core memory is fast but small, the external memory is large but slow. It is possible to avoid using external memory by parallelizing computations or by processing smaller work chunks sequentially. We will analyze how a combination of these two options improves efficiency of the computations. For this purpose divisible load theory representing data-parallel applications is used. A mathematical model for scheduling divisible computations is formulated as a mixed integer linear program. The model allows for executing multiple load installments sequentially or in parallel. The efficiency of the schedule is analyzed with respect to the impact of load size and machines number. The results are visualized as isoefficiency maps.

Keywords: performance evaluation and prediction, hierarchical memory, divisible load theory, isoefficiency maps

1 Introduction

Current computer systems have hierarchical memory. At the top of the hierarchy, CPU registers are the fastest form of computer memory, but also available in the smallest amount. Further memory hierarchy levels include CPU caches, core memory commonly referred to as RAM, networked caches, SSDs, HDDs, tapes, etc. When going down the hierarchy, memory size increases but speed of data transfers decreases. The part of the hierarchy from CPU registers to RAM is managed by hardware. The lower levels are controlled by software stacks [16]. Consequently, computations exploiting external storage, e.g. in the form of virtual memory, can be by two orders of magnitude slower than using only the upper part of memory hierarchy [9, 14]. Therefore, for the purposes of this study, we will be distinguishing only two memory levels: CPU registers, caches and RAM by convention referred to as core (or main) memory, and out-of-core memory including all kinds of external storage.

One of the key applications of parallel systems is processing big volumes of data. Core memory size is a limitation in processing big volumes of data. Memory footprint can be reduced in at least two ways: (i) by parallelizing computations, and hence, reducing processor individual load shares, (ii) by multi-installment processing, i.e. dividing the load into multiple chunks and processing them sequentially. There is a need for understanding how the number of machines, number of installments, memory size, and problem size interplay in determining performance of processing big volumes of data. Thus, our goal in this paper is to analyze the impact of various system parameters on performance of processing big volumes of data in parallel. Most of the earlier approaches to performance analysis focus on scalability with machine number (speedup) but at fixed problem size, while the problem size vs machine number interrelationship is ignored. Our goal is to allow easily grasping the system parameter interrelationships, and in particular, between machine number and problem size. For this purposes divisible load theory (DLT) will provide mathematical model of data parallel application and isoefficiency maps will be used as a visual front-end.

Divisible load theory is a scheduling and performance model of data-parallel applications typical for big volumes of data. In the DLT it is assumed that the computation consists in processing large amount of similar data units, conventionally referred to as load. The load can be partitioned between remote computers and processed in parallel. The individual data units are small enough in relation to the total load size so it can be assumed, without great loss of accuracy, that the load is arbitrarily divisible. Accuracy of DLT in predicting performance and scheduling data-parallel applications such as text, image and video processing, file compression, linear algebra, sorting has been reported in [1, 11, 14]. Further introduction to the DLT can be found in surveys [4, 5, 10, 15]. The first DLT paper assuming flat memory model was [13]. A heuristic load distribution method has been proposed. In [9] hierarchical memory system has been analyzed and single-installment load distribution has been found by use of a linear program. In [3] data gathering networks with limited memory are studied. Time-energy trade-offs in divisible load processing have been studied in [6, 14]. A single-installment load distribution has be found by linear programming formulation. It was assumed that all available machines were always activate (powered on). In [6] the single-installment schedules built by the linear program have been compared with heuristic load distribution methods derived from loop scheduling (see e.g. [5] for loop scheduling). Isoefficiency maps borrow the performance visualization concept from other types of iso-lines (isotherms, isobars, isogones). Such visualizations proved very effective in building understanding of sensitivities and relationships of complex phenomena in other areas of science and technology

The contributions of this paper can be summarized as follows: (i) optimum divisible load scheduling formulation is proposed for systems with both hierarchical memory system, and multi-installment load distribution, while the individual machines are powered on only if needed; (ii) isoefficiency maps are proposed to analyze the impact of memory hierarchy on divisible load processing; (iii) com-

putationally hard mixed integer linear program model is used as performance oracle. Further organization of this paper is the following. In the next section the model of parallel computation is introduced, and the problem of planning optimum schedule for the computations is formulated as a mixed integer linear program. In Section 3 the method of drawing isoefficiency maps is explained. Section 4 is dedicated to a study of isoefficiency maps for divisible loads processing. We conclude in the last section.

2 Mathematical Model of Parallel Application

We will be assuming that execution of the data-parallel application is initiated by a root processor (a.k.a. master, originator) which starts worker machines, schedules communications and distributes the load. Computing environment is homogeneous and comprises m identical machines $M_1, \ldots, M_m$. The system interconnect is equivalent to a star (single level tree) and the originator communicates directly with worker processors. The machine starting process lasts for S time units and may include, e.g., cold-start, start from certain suspension mode, loading the application code and initializing data structures. A machine starting message is short and the delay it induces in the communication system may be neglected. Load of total size V is distributed to the worker processors in installments (messages, load chunks). Sending a load chunk of size α takes time $O + C\alpha$, where O is a fixed delay required to start the communication and C is communication rate (in seconds per byte). Only after receiving the whole load chunk can the worker machine start processing the load. A machine may receive more than one load chunk, but only after finishing computations on the previous one. Let n be the total number of load chunks distributed by the originator.

It has been experimentally demonstrated [9, 14] that the time of processing load of size α in a system with two memory levels can be represented by function

$$\tau(\alpha) = \max\{a_1\alpha, a_2\alpha + b_2\} \tag{1}$$

where a_1 is rate (seconds per byte) of processing in-core, a_2, b_2 are parameters of linear time function for computing out of core. Function $\tau(\alpha)$ has properties: (i) $\tau(0) = 0$, (ii) $0 < a_1 < a_2$, $b_2 < 0$, (iii) $\tau(\rho) = a_1\rho = a_2\rho + b_2$. The third property means that the two linear segments of τ in equation (1) intersect at load size ρ which is the core memory size available to application. The process of collecting results is not explicitly scheduled because, e.g., the size of results is small and their transfer time is very short, or the results are stored on the worker machines for further processing. The optimum schedule of the computations requires determining: (i) where to send the load (i.e. the sequence of load distributions to the processors), (ii) when to send the load, (iii) sizes of the sent load chunks.

Let x_{ij} be a binary variable equal to 1 if load chunk j is sent to machine M_i and equal to 0 otherwise. We will denote by α_{ij} the size of load chunk j sent to processor i. If the chunk is sent to some other processor, then $\alpha_{ij} = 0$. The moment when sending chunk j begins will by denoted by t_j. Let T be the

length of the schedule (makespan). We will use auxiliary variables $q_{ij} = t_j x_{ij}$ and $\tau_{ij} = \max\{a_1\alpha_{ij}, a_2\alpha_{ij} + b_2\}$. The problem of constructing an optimum computation schedule can be formulated as mixed integer linear program (MIP):

$$\text{minimize } T \tag{2}$$

subject to:

$$t_j + C\sum_{i=1}^{m}\alpha_{ij} + O \le t_{j+1} \quad j = 1,\ldots,n \tag{3}$$

$$\begin{gathered} q_{ij} + C\alpha_{ij} + Ox_{ij} + \tau_{ij} \le T \\ j = 1,\ldots,n \quad i = 1,\ldots,m \end{gathered} \tag{4}$$

$$\begin{gathered} q_{ij} + C\alpha_{ij} + Ox_{ij} + \tau_{ij} \le q_{il} + (1 - x_{il})Z \\ i = 1,\ldots,m \quad j = 1,\ldots,n-1 \quad l = j+1,\ldots,n \end{gathered} \tag{5}$$

$$S\sum_{i=1}^{m} x_{ij} \le t_j \qquad j = 1,\ldots,n \tag{6}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\alpha_{ij} \ge V \tag{7}$$

$$\alpha_{ij} \le Vx_{ij} \qquad i = 1,\ldots,m \quad j = 1,\ldots,n \tag{8}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1,\ldots,n \tag{9}$$

$$\begin{gathered} Zx_{ij} \ge q_{ij} \ge 0 \\ t_j \ge q_{ij} \ge t_j - Z(1 - x_{ij}) \\ i = 1,\ldots,m \quad j = 1,\ldots,n \end{gathered} \tag{10}$$

$$\begin{gathered} a_1\alpha_{ij} + Zu_{ij} \ge \tau_{ij} \ge a_1\alpha_{ij} \\ a_2\alpha_{ij} + b_2 + Z(1 - u_{ij}) \ge \tau_{ij} \ge a_2\alpha_{ij} + b_2 \\ i = 1,\ldots,m \quad j = 1,\ldots,n \end{gathered} \tag{11}$$

In the above formulation $x_{ij}, \alpha_{ij}, q_{ij}, t_j, T, \tau_{ij}, u_{ij}$ are decision variables. C, O, S, V, a_1, a_2, b_2, m, n are constants defined in the parallel application, while Z is a large number defined in the above MIP. Decision variables x_{ij} determine the sequence of communications and any n-message sequence to the m processors can be constructed. The purpose of constraint (3) is to guarantee that the jth message fits in interval $[t_j, t_{j+1}]$ and messages do not overlap in the communication channel. Inequalities (4) ensure that computations finish before the end of the schedule. Constraints (5) establish that if load chunks j, l are sent to processor i, then there is enough time to receive the jth chunk and process it before receiving the lth chunk starts. By (6) the processor which is receiving the jth load chunk is already started when sending the chunk begins. Inequality

(7) guarantees that the whole load is processed. Constraint (8) ensures that a processor that is not receiving the jth load chunks also receives no load in the jth communication. By (9) only one machine can receive the jth load chunk. Inequalities (10) ensure that the auxiliary variable q_{ij} is equal to $t_j x_{ij}$. Using product $t_j x_{ij}$ directly is not allowed because the formulation would become a quadratic mathematical program. Yet, it is possible to obtain the same value by use of linearizing constraints (10) and an additional variable q_{ij}. Inequalities (11) guarantee that $\tau_{ij} = \max\{a_1\alpha_{ij}, a_2\alpha_{ij} + b_2\}$. The trigger binary variable $u_{ij} = 0$ determines whether the first $(a_1\alpha_{ij})$ or the second component $(a_2\alpha_{ij} + b_2)$ in the max is active.

3 Isoefficiency Map Construction

In this section we introduce the concept of isoefficiency map, and then explain how such maps can be constructed for processing divisible loads in hierarchical memory systems. Performance of parallel computations is measured by two classic metrics: speedup $\mathcal{S}$ and efficiency $\mathcal{E}$:

$$\mathcal{S}(m) = \frac{T(1)}{T(m)} \qquad \mathcal{E}(m) = \frac{\mathcal{S}}{m} = \frac{T(1)}{mT(m)}, \tag{12}$$

where $T(i)$ is execution time on i machines. Speedup and efficiency measure scalability of the parallel application. $\mathcal{E}$ is often interpreted as the fraction of the processor set which really computes. In a well-designed application $\mathcal{S}$ should grow (preferably linearly) with m and $\mathcal{E}$ should be as close to 1 as possible. However, in most cases speedup saturates at certain number of machines and efficiency decreases with m. The location of the maximum speedup depends on the size of the solved problem. Bigger problems allow to exploit more processors while preserving certain efficiency level. In order to grasp this relationship a concept of isoefficiency function has been introduced [12]. Isoefficiency function $I(e, m)$ is size of the problem required to maintain efficiency $\mathcal{E}(m) = e$. Consider an example of finding a minimum spanning tree in a graph with n vertices. A straightforward parallel version of Prim's algorithm for this problem, has complexity $T(m) = c_1 n^2/m + c_2 n \log m$, where c_1, c_2 are constants (see e.g. [2], Section 10.6). Efficiency of this algorithm is $\mathcal{E}(m) = c_1 n^2/(c_1 n^2 + c_2 nm \log m)$. Hence, isoefficiency function for m machines and efficiency level $e < 1$ is $I(e, m) = c_2 em \log m/(c_1(1 - e))$. For a fixed value of e, function $I(e, m)$ can be viewed as a line of equal efficiency in the $m \times n$ space. Such a line of equal efficiency will be called an *isoefficiency line*. Thus, performance of parallel computations can be visualized as a set of isoefficiency lines in $m \times$ *problem size* space. Such a visualization will be called an *isoefficiency map* of a parallel computation. The idea of isoefficiency maps has been extended to other pairs of parallel computation efficiency parameters, such as speed of communication, speed of computation, etc. [7, 8]. Such visualizations are useful in comprehending the phenomena limiting performance of parallel processing.

Schedule length T calculated by solving (2)-(11) can be used in performance evaluation of data-parallel applications. Let $T(m,n,V)$ denote the value of T obtained for a particular number of machines m, communications n, and problem size V. The time of processing the same amount of load on a single machine is $T(1,1,V) = S+O+CV+\max\{a_1V, a_2V+b_2\}$. Thus, efficiency of the computation is $\mathcal{E}(m,n,V) = T(1,1,V)/(mT(m,n,V))$. The isoefficiency function for a given value of efficiency e can be defined as $I(e,m,n) = \{V : \mathcal{E}(m,n,V) = e\}$. Function $I(e,m,n)$ allows to draw one *isoefficiency line*, i.e. a line of efficiency e in the $m \times V$ space. The isoefficiency line depicts how problem size V should grow in order to maintain equal efficiency e with changing number of machines m.

Due to the complex nature of the formulation (2)-(11) it is not possible to derive a closed-form formula of $I(e,m,n)$. Therefore, $I(e,m,n)$ has been found numerically, using the following approach: It has been established that for fixed m,n, efficiency function $\mathcal{E}(m,n,V)$ has a single maximum $\mathcal{E}_{\max}(m,n)$ at load size $V_{\max}(m,n)$ and is monotonous on both sides of $V_{\max}(m,n)$. This will be further discussed in Section 4. A bisection search method has been used to find load sizes $V < V_{\max}(m,n)$ for which certain efficiency level $e < \mathcal{E}_{\max}(m,n)$ is achieved. Precisely, for a probe value V times $T(1,1,V)$ and $T(m,n,V)$ were calculated and if the resulting efficiency satisfied $T(1,1,V)/(mT(m,n,V)) < e$ then the probe load size was increased, respectively decreased in the opposite case. Analogous method has been applied to calculate $I(e,m,n)$ for load sizes greater than $V_{\max}(m,n)$. The values of $V_{\max}(m,n)$ and $\mathcal{E}_{\max}(m,n)$ have been found by a modification of the bisection method: Efficiency has been calculated for two probe values V_1, V_2 in some tested interval. Then the load size interval has been narrowed to V_1 or V_2, whichever resulted in the smaller efficiency. Both in the bisection search and in the search for the maximum efficiency the procedures have been stopped if the width of the searched V intervals dropped below 1MB.

As the MIP solver Gurobi 7.5.2 has been used. Observe that MIP is an **NP**-hard problem, and in the worst-case MIP solvers run in exponential time in the number of variables. In order to obtain solutions in acceptable time, the MIP solver run times have been limited to 300s on 6 CPU threads. Consequently, the obtained solutions mostly were not guaranteed optimum. Still, the solutions are always feasible and can be considered as approximations of the optimum solutions of (2)-(11).

4 Performance Modeling

In this section we present isoefficiency maps and discuss the performance phenomena they show. Unless stated to be otherwise the reference instance parameters were: for the computing time function $\tau(\alpha)$: $a_1 = 0.109$s/MB, $a_2 = 4.132$s/MB, $b_2 = -27109$s, for the communication delays $C = 5$ms/MB, $O = 75$ms, machine startup time $S = 25.4$s, and a limit of $n = 20$ load chunks. The a_1, a_2, b_2 parameters correspond with usable RAM size $\rho = 6739$MB. Since these parameters are machine- and application-dependent and can vary widely

Table 1. Maximum efficiency and corresponding load sizes vs m at $n = 20$ installments

m		2	3	4	5	6	7	8	9	10
$\mathcal{E}_{\max}(m,n)$		34.2	34.0	33.7	33.4	33.2	32.8	32.4	31.5	30.9
$V_{\max}(m,n)$ [MB]		134485	120827	123329	105097	119663	95748	106921	115514	75048
m	11	12	13	14	15	16	17	18	19	20
$\mathcal{E}_{\max}(m,n)$	30.4	29.6	28.7	27.7	26.7	25.7	24.8	23.8	22.9	22.1
$V_{\max}(m,n)$ [MB]	81394	86545	90457	94532	98591	99460	100506	100354	99708	102240

(cf. [7, 14]), we will concentrate on the frequent qualitative phenomena rather than on particular performance numbers. We will also attempt analytically explaining the relationships behind the observed shapes of the isoefficiency lines.

In Fig.1 isoefficiency map for the load sizes smaller than $V_{\max}(m,n)$ is shown, and in Fig.2 for the loads above $V_{\max}(m,n)$. For better clarity, maximum values of efficiency $\mathcal{E}_{\max}(m,n)$, and the corresponding load sizes $V_{\max}(m,n)$ are shown in Table 1. The line of maximum efficiency $\mathcal{E}_{\max}(m,n)$ is denoted as MAX in Figs 1,2 and the isolines are labeled with their efficiency levels. The efficiency for $m = 1$ is always 1, and no isolines for $m = 1$ are shown. Note that m, shown along the horizontal axis, is a discrete variable and consequently the isolines are step functions. It can be observed in both figures that efficiencies greater than 1 (consequently also super-linear speedups) are possible. Though such situation is rare in typical parallel applications, it is not unusual in the context of memory hierarchies. If only one machine is used (as in the calculation of $T(1,1,V)$) then for $V > \rho$ the processing rate tends asymptotically to a_2. Conversely, if the load is distributed between many processors then it can be processed in-core with rate a_1. In our case $a_2/a_1 \approx 37.9$ and efficiency levels close to 37 can be expected. The values in Table 1 are slightly smaller than a_2/a_1 which is a result of communication delays and machine startup times. The $\mathcal{E}_{\max}(m,n)$ line shows problem sizes V which achieve the best balance between the advantage of processing load in-core over single machine out-of-core processing, and the costs of starting the machines, communicating and avoiding idle time. MIP (2)-(11) is a discrete optimization problem and, e.g., there are fixed overheads S, O which can be switched on and off by the choice of the communication sequence. Furthermore, the best communication sequences are not always repetitive patterns. Consequently $\mathcal{E}_{\max}(m,n)$ is neither smooth nor does it show an obvious trend.

Let us consider the part of the isoefficiency map for problem sizes smaller than $V_{\max}(m,n)$ shown in Fig.1. For such load sizes machines in set $M_1, \ldots, M_m$ compute in-core, but if the same load were processed on just one machine then the load may spill to out-of-core. As it is not possible to derive a closed-form formula of the (2)-(11) solution, we will analyze range of $\mathcal{E}(m,n,V)$. The efficiency in this part of the isoefficiency map can be bounded in the ensuing range:

$$\frac{S+O+CV+a'V}{mS+nmO+mCV+a_1V} \le \mathcal{E}(m,n,V) \le \frac{S+O+CV+a'V}{mS+mO+a_1V}. \qquad (13)$$

In the numerator of (13) $a_1 \le a' \le a_2$ is an equivalent rate of processing on one machine. Product $mT(m,n,V)$ in denominator of (12) can be interpreted

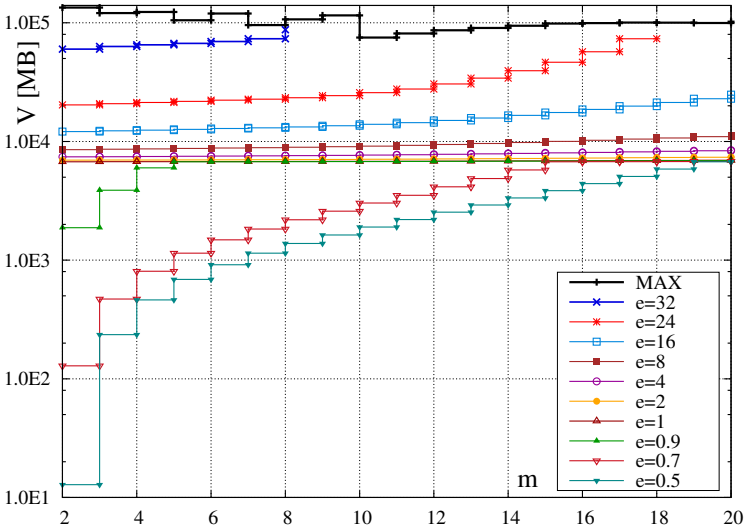

Fig. 1. Isoefficiency map for the load sizes V below maximum efficiency. Logarithmic vertical axis.

as area in $time \times m$ space which is easier to assess than the schedule length $T(m,n,V)$. The area of $mT(m,n,V)$ in (13) is bounded from below by $mS + mO + a_1V$ which is total machine startup time mS, minimum fixed overhead of communications mO and total work of the computations in core a_1V. For the upper bound of the area, $mnO + mCV$ is an upper bound of machine waiting during the communications. It can be verified that both bounds of $\mathcal{E}(m,n,V)$ are increasing with V, and value of V derived from the bound formulas increases with m for fixed efficiency e. Indeed, it can be seen in Fig.1 that problem sizes V must grow with the number of machines m to maintain some fixed level of efficiency. The isolines grow slightly faster than linearly with m because the total processor waiting time in the actual solutions increases faster than linearly with m. One more peculiarity can be seen in Fig.1: around $V = \rho = 6739$MB a bunch of isolines coalesce. This is a result of using out-of-core memory in $T(1,1,V)$ used in the efficiency formula. The single reference machine starts to use out-of-core memory at $V \approx \rho$, which extremely expands $T(1,1,V)$ and problem size expressed by $I(e,m,n)$ has to increase only marginally to attain the required efficiency level. Consider, e.g., the upper bound of (13). The size of the load required to attain efficiency e is $V = (S+O)(em-1)/(C+a'-a_1)$. When m grows also V grows, but the single machine must use out-of-core memory and a' tends to $a_2 \gg a_1$. As a result, the increase in the numerator $(S+O)(em-1)$ is intensively suppressed by a' growing in the denominator $(C+a'-a_1)$. Hence, isoline coalescing at $V \approx \rho$ can be observed.

In the part of the isoefficiency map above $V_{\max}(m,n)$ (see Fig.2) the single reference machine considered in $T(1,1,V)$ uses out-of-core memory while machines $M_1,\ldots,M_m$ use out-of-core memory at least partially. In the dominating

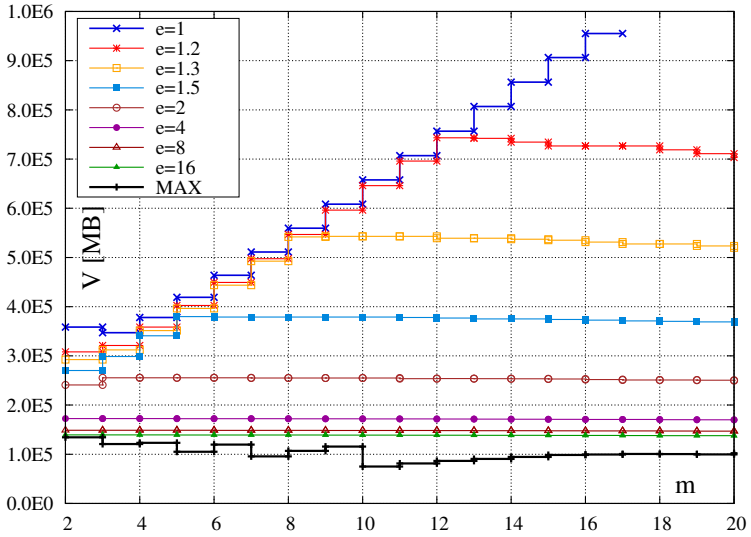

Fig. 2. Isoefficiency map for the load sizes V above maximum efficiency.

pattern of load distribution some part of the load is processed in load chunks of RAM ρ size while the remaining load is distributed to the machines in roughly equal sizes and processed out-of-core. Thus, for n installments and m machines, $n-m \geq 0$ load chunks have nearly RAM size, and the remaining chunks have size roughly $[V-(n-m)\rho]/m$. This load partitioning is intuitively effective because load as big as possible is processed in RAM, while the remaining load processed out-of-core on each machine is as small as possible. This load partitioning pattern results in the following efficiency formula

$$\mathcal{E}(m,n,V) \approx \frac{S+O+V(C+a_2)+b_2}{mS+nO+CV+(n-m)\rho a_1+m[(V-(n-m)\rho)/ma_2+b_2]}. \quad (14)$$

In the denominator of (14) area $mT(m,n,V)$ is calculated. It is assumed that data transfers to one machine overlap with other machines computations (latency hiding), and consequently, only CV area is used on communications in all machines. $(n-m)\rho a_1$ is the area of computing in core, and $m[(V-(n-m)\rho)/ma_2+b_2]$ out-of-core. From (14) estimation of the isoefficiency function can be derived:

$$I(e,m,n) \approx \frac{b_2(en-1)+S(em-1)+O(en-1)}{(C+a_2)(1-e)}. \quad (15)$$

In the derivation of (15) property (iii) $\rho = b_2/(a_1-a_2)$ of (1) has be used. Note that $b_2 < 0, e > 1, n > m, |b_2| \gg S \gg O$, and $I(e,m,n) > 0$. The load size necessary for certain efficiency e is almost independent of the number of machines m in (15). Thus, (15) represents well the lower-right part of Fig.2 where isoefficiency lines are nearly parallel to the horizontal axis. The top-left part of Fig.2 can be considered an artifact. Note that with growing V the time

of processing load out-of-core dominates in the computation time. As a result efficiency tends to $(Va_2+b_2)/(m[V/ma_2+b_2]) \approx 1$ with growing V and it is not possible to obtain schedules with efficiency significantly smaller than 1 without introducing artificial idle time. In other words, to construct a schedule with low efficiency, overheads are 'necessary' in the denominator of the efficiency equation like in (14). Yet, with decreasing m the amount of the overheads decreases in relation to growing out-of core computation cost, and it is becoming impossible to build a schedule with some low efficiency level unless some idle time is added. Consequently, we do not show isoefficiency lines for $e < 1$ in Fig.2 because the corresponding schedules require inserting unneeded idle time.

5 Conclusions

In this paper we studied time performance of divisible computations with non-linear processing time imposed by hierarchical memory. Efficiency of distributed computation has been estimated numerically by use of mixed integer linear programming. The performance has been visualized in the isoefficiency maps. It has been established that efficiency greater than 1 is possible as a result of memory hierarchy: parallel machines and multi-installment processing allow for computations in-core which is faster than if the same load were put on one machine, necessarily out-of-core. For problem sizes smaller than the maximum efficiency size, the efficiency decreases with increasing machine number. For problem sizes larger than the maximum efficiency size, the efficiency is almost independent of machine number. In the future study the idea of isoefficiency maps for systems with hierarchical memory can be extended to other pairs of system parameters than m and V.

References

1. Agrawal, R., Jagadish H.V.: Partitioning techniques for large-grained parallelism. IEEE Transactions on Computers **37**, 1627—1634 (1988).
2. Akl, S.G.: The Design and Analysis of Parallel Algorithms. Prentice-Hall Int. Inc., Englewood Cliffs New Jersey (1989)
3. Berlińska, J.: Communication scheduling in data gathering networks with limited memory. Applied Mathematics and Computation **235**, 530–537 (2014).
4. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos California (1996)
5. Drozdowski, M.: Scheduling for Parallel Processing. Springer-Verlag, London, (2009)
6. Drozdowski, M., Marszałkowski, J.M.: Divisible Loads Scheduling in Hierarchical Memory Systems with Time and Energy Constraints. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.), PPAM 2015. LNCS, vol. 9574, pp.111–120. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-32152-3_11

7. Drozdowski, M., Marszałkowski, J.M., Marszałkowski, J.: Energy trade-offs analysis using equal-energy maps. Future Generation Computer Systems **36**, 311–321 (2014).
8. Drozdowski, M., Wielebski, Ł.: Isoefficiency Maps for Divisible Computations. IEEE Transactions on Parallel and Distributed Systems **21**, 872–880 (2010).
9. Drozdowski, M., Wolniewicz, P.: Out-of-Core Divisible Load Processing. IEEE Transactions on Parallel and Distributed Systems **14**, 1048–1056 (2003).
10. Ghanbari, S., Othman, M.: Comprehensive Review on Divisible Load Theory: Concepts, Strategies, and Approaches, Mathematical Problems in Engineering Article ID 460354, 13 pages (2014) `http://dx.doi.org/10.1155/2014/460354`
11. Ghose, D., Kim, H.J., Kim, T.H.: Adaptive divisible load scheduling strategies for workstation clusters with unknown network resources. IEEE Transactions on Parallel and Distributed Systems **16**, 897–907 (2005).
12. Gupta, A., Kumar, V.: Performance properties of large scale parallel systems. Journal of Parallel and Distributed Computing **19**, 234–244 (1993).
13. Li, X., Bharadwaj, V., Ko, C.C.: Divisible load scheduling on single-level tree networks with buffer constraints. IEEE Transactions on Aerospace and Electronic Systems **36**, 1298–1308 (2000).
14. Marszałkowski, J.M., Drozdowski, M., Marszałkowski, J.: Time and Energy Performance of Parallel Systems with Hierarchical Memory. Journal of Grid Computing **14**, 153–170 (2016).
15. Robertazzi, T.: Ten reasons to use divisible load theory. IEEE Computer **36**, 63–68 (2003).
16. Swanson, S., Caulfield, A.M.: Refactor, Reduce, Recycle: Restructuring the I/O Stack for the Future of Storage. IEEE Computer **46**, 52–59 (2013).