Divisible Loads Scheduling in Hierarchical Memory Systems with Time and Energy Constraints

Maciej Drozdowski(✉) and Jędrzej M. Marszałkowski

Institute of Computing Science, Poznań University of Technology,
Piotrowo 2, 60-965 Poznań, Poland
{Maciej.Drozdowski,Jedrzej.Marszalkowski}@cs.put.poznan.pl

Abstract. In this paper we consider scheduling distributed divisible computations in systems with hierarchical memory for energy and time performance criteria. Hierarchical memory allows to conduct computations on big data sets using out-of-core processing instead of coercing application data fit into core storage. However, out-of-core computations are more costly both in time and energy. A model for scheduling divisible loads under time and energy criteria is introduced. Two types of scheduling algorithms are proposed and evaluated: a single-installment algorithm which builds optimum schedules but may use out-of-core storage, and a set of multi-installment algorithms which use limited memory but require more communications.

Keywords: Scheduling · Divisible loads · Hierarchical memory · Energy efficiency · Performance evaluation

1 Introduction

Providing electricity and bearing its cost has become a key element in designing and running big data centers and supercomputing installations [9]. Dissipating heat generated in computations is currently one of the limitations to the further growth of the CPU speeds [8]. Hence, energy efficiency is a very active research area and recent advantages in this field are closely analyzed [14].

In this paper we study the trade-off between time performance and energy cost in processing divisible loads on systems with hierarchical memory. Divisible loads are data-parallel applications which can be divided into parts of arbitrary sizes, and the parts can be processed independently in parallel. Divisible load theory (DLT) has been proposed in [1,4] to analyze performance of distributed computations and schedule them accordingly. Thus, DLT provides methods of scheduling and analyzing performance of a broad class of distributed applications operating on big data volumes [2,3,5,12]. Contemporary computer systems have hierarchical memory organization. At the top of the hierarchy CPU registers have the shortest access time, but they are scarcest. Processor caches establish the next level of memory hierarchy. Main memory, here by convention referred to as RAM, has much bigger size but is again slower. The following levels of memory hierarchy are based on external and networked storage: HDDs, NAS, tapes, optical media, etc. In this study we reduce the above hierarchy to just two types of memory: *core* comprising registers, caches, RAM, and *out-of-core* memory comprising all types of external storage. This partitioning has a practical motivation. On the one hand, sizes of data (load) processed in big data applications far exceed size of CPU registers and caches. Hence, to a great extent, core is transparent for a developer of such applications. On the other hand, core accesses are managed by hardware, while out-of-core memory is accessed via software wrappers (virtual memory, (networked) file systems), and consequently, it is by orders of magnitude slower. Due to the limited core size a developer must undertake steps to fit data in core. Contrarily, out-of-core storage offers nearly unlimited storage but requires use of virtual memory or dedicated data management subsystem [11]. Consequently, on-core and out-of-core computations have different character both in the development and in performance.

Systems with hierarchical memory have been analyzed in DLT [7]. Energy may be considered a special type of cost in DLT. Scheduling with monetary cost has been considered in [13]. Energy in processing divisible loads on flat memory systems has been subject of [6]. In this paper we combine nonlinear energy consumption and computing time models specific for systems with hierarchical memory. We analyze two types of solutions: a single-installment method which sends load to processors once and multi-installment algorithms which send the load in many iterations. Since the problem is bicriterial the trade-off between time and energy will be analyzed.

Further organization of this work is as follows. In the next section we formulate the scheduling problem and provide timing and energy use models. In Sect. 3 algorithms solving the problem are proposed. Section 4 is dedicated to evaluation of the proposed methods. Section 5 summarizes results of this work.

2 Problem Formulation

It is assumed that computations are performed in a single-level tree system with root M_0 (a.k.a. master, server, originator) and machines (computers,processors) $M_1, \dots, M_m$ at the leaves. The machines can be in one of four states: (1) idle - consuming power P^I, (2) starting - which takes time S and power P^S, (3) networking - using power P^N, (4) computing. Busy-waiting is considered the same as networking state. Initially volume V of load is held by M_0, M_0 is in the networking state, $M_1, \dots, M_m$ are idle. M_0 activates $M_1, \dots, M_m$ which takes energy SP^S on each machine. Next load V is distributed in parts to machines $M_1, \dots, M_m$. Transferring α units of load to M_i takes time αC, where C is communication rate (in seconds per byte). M_0 sends the load to processors one after the other, i.e. load is distributed to slaves in the sequential manner. M_0 activates M_is just-in-time which means that completion of the starting procedure coincides with the beginning of receiving of the load to process. For simplicity of exposition we assume that the time of returning results from M_is to M_0 is very short compared to the whole schedule length T and can be neglected (this can be easily relaxed in DLT [3,5,12]). The duration and energy cost of sending the waking signal is negligible and starting some machine M_j can be performed in parallel with some other machine M_i communicating with M_0.

The time and energy of computations on load α depend on the load size, precisely whether the load part fits in the main memory [7,10]. It is assumed that the time of computing on load of size α is determined by a piecewise-linear function $\tau(\alpha) = \max\{a_1\alpha, a_2\alpha + b_2\}$. The first component of τ corresponds with computations in core with speed $1/a_1$, the second component represents out-of-core computations. Function τ has two properties: $\tau(0) = 0$ and $\tau(\rho) = a_1\rho = a_2\rho + b_2$, where ρ is the size of main memory (RAM) available to the application beyond which system starts using out-of-core memory. The energy consumed in the computations is determined by an analogous function $\varepsilon(\alpha) = \max\{k_1\alpha, k_2\alpha + l_2\}$ satisfying conditions $\varepsilon(0) = 0, \varepsilon(\rho) = k_1\rho = k_2\rho + l_2$. The problem considered here consists in constructing a schedule of minimum length T and energy E. Since this problem is effectively bicriterial we will be solving energy E minimization problem under constrained schedule length T.

3 Solution Methods

In this section we propose two strategies of load distribution. The first sends the load to machines once. Consequently, load parts can be big and out-of-core processing may be unavoidable. The second, iteratively distributes load chunks of small size in multiple communications.

3.1 Optimum Single-Installment

A schedule for the current method is shown in Fig. 1a. In the schedule M_0 busy-waits S units of time for M_1 initiation, then load V is distributed in parts $\alpha_1, \dots, \alpha_m$ to machines $M_1, \dots, M_m$, respectively. M_0 communicates continuously for time $C\sum_{i=1}^{m}\alpha_i = CV$ and switches off. Thus, in the schedule of length T, M_0 consumes energy

$$E_0 = P^N(CV + S) + P^I(T - CV - S).$$

Machine M_i remains idle until time $C\sum_{i=1}^{i-1}\alpha_i$ (where $\sum_{i=1}^{0}\alpha_i = 0$), starts in time S, receives its part of load in time $C\alpha_i$, computes it in time $\tau(\alpha_i)$ and switches off. Let us denote by $t_i = \tau(\alpha_i)$ the time of computations on M_i and by $e_i = \varepsilon(\alpha_i)$ the energy consumed in these computations. The duration of idle intervals on machine M_i is $T - S - C\alpha_i - t_i$. The energy consumed by M_i is

$$E_i = P^S S + P^N C\alpha_i + e_i + P^I(T - S - C\alpha_i - t_i)$$

The problem of minimizing energy consumption E under limited schedule length T can be formulated as a linear program:

$$\min \sum_{i=0}^{m} E_i \tag{1}$$

$$S + C\sum_{j=1}^{i} \alpha_j + t_i \le T \quad i = 1, \ldots, m \tag{2}$$

$$\max\{a_1\alpha_i, a_2\alpha_i + b_2\} = t_i \quad i = 1, \ldots, m \tag{3}$$

$$\max\{k_1\alpha_i, k_2\alpha_i + l_2\} = e_i \quad i = 1, \ldots, m \tag{4}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{5}$$

$$\alpha_i, t_i, e_i \ge 0 \quad i = 1, \ldots, m \tag{6}$$

In the above formulation inequality (2) guarantees feasibility of the schedule on each processor. Constraints (3), (4) instantiate functions $\tau(\alpha), \varepsilon(\alpha)$. We present constraints (3), (4) in a simplified form which is accepted by contemporary solvers (e.g. CPLEX), but it can be implemented in any LP solver by splitting the max function into two inequalities and adding cost of exceeding constraints. By Eq. (5) all work is executed.

3.2 Multi-Installment Methods

In the next three algorithms M_0 sends load chunks of equal size α. Actual methods of calculating α for each specific algorithm will be given in the following. The sequence of communications to $M_1, \ldots, M_m$ is repeated iteratively until exhausting the load. The number of communications may be indivisible by m and the size α^f of the last sent chunk may be smaller than α. It is assumed that computations on each of the machines $M_1, \ldots, M_m$ last longer than sending the load to the remaining $m-1$ machines. This imposes a requirement that $(m-1)C\alpha \le \tau(\alpha)$ which can be reformulated as $m \le a_1/C + 1$ for $\alpha \le \rho$ and $m \le a_2/C + 1 + b_2/(C\alpha)$ for $\alpha > \rho$. Thus, the number of processors which can be effectively exploited is limited and it is bigger when slower out-of-core processing takes place. Now we derive schedule length T and energy E used when chunks of size α are applied. For simplicity of exposition let $m > 1$.

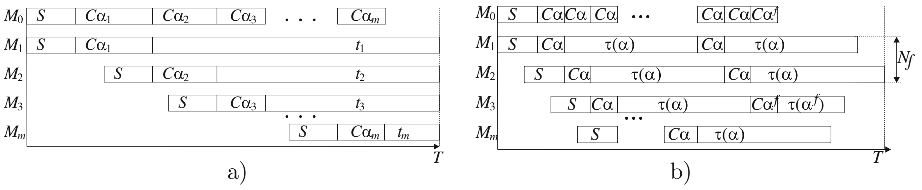

Fig. 1. (a) Single-installment schedule. (b) Multi-installment schedule.

The number of complete distribution iterations in which each of m machines obtain load α is $N_o = \lfloor \frac{V}{\alpha m} \rfloor$. The number of chunks of size α in the last incomplete iteration is $N_f = \lfloor (V - N_o m\alpha)/\alpha \rfloor$. Size of the last chunk is $\alpha^f = V - (mN_o + N_f)\alpha$. Then, the schedule length is (cf. Fig. 1b):

$$T = S + N_o(C\alpha + \tau(\alpha)) + \begin{cases} N_f C\alpha + \max\{\alpha^f C + \tau(\alpha^f), \tau(\alpha)\} & N_f > 0 \\ \max\{(m-1)C\alpha, \alpha^f C + \tau(\alpha^f)\} & N_f = 0 \end{cases}$$

Deriving energy consumption requires calculating idle time, computing and communication durations on $M_1, \ldots, M_m$. At the start of the schedule M_i is idle until time $C(i-1)\alpha$. Thus, total energy used before machines activation is $E_1^I = P^I \sum_{i=1}^{m}(i-1)C\alpha = P^I(m-1)m/2C\alpha$. Starting m machines consumes $E^S = P^S mS$ units of energy. Energy consumed on $M_1, \ldots, M_m$ in the computations and communications is $E^R = (N_o m + N_f)(P^N C\alpha + \varepsilon(\alpha)) + P^N C\alpha^f + \varepsilon(\alpha^f)$.

Let us assume that $\alpha^f C + \tau(\alpha^f) < \tau(\alpha)$, i.e., the schedule ends on the last machine receiving a chunk of size α (see Fig. 1b). The idle time on $M_i \in \{M_1, \ldots, M_{N_f}\}$ is $(N_f - i)C\alpha$, on M_{N_f+1} it is $\tau(\alpha) - C\alpha^f - \tau(\alpha^f)$, and on $M_i \in \{M_{N_f+2}, \ldots, M_m\}$ it is $\tau(\alpha) - (i - N_f - 1)C\alpha$. Thus, total idle time on $M_1, \ldots, M_m$ at the end of the schedule is

$$I = \sum_{i=1}^{N_f}(N_f - i)C\alpha + \tau(\alpha) - C\alpha^f - \tau(\alpha^f) + \sum_{i=N_f+2}^{m}(\tau(\alpha) - (i - N_f - 1)C\alpha).$$

Suppose that $\alpha^f C + \tau(\alpha^f) \geq \tau(\alpha)$, which means that M_{N_f+1} has no idle time. Idle time on machines $M_i \in \{M_1, \ldots, M_{N_f}\}$ is $(N_f - i)C\alpha + \tau(\alpha^f) + C\alpha^f - \tau(\alpha)$ and on $M_i \in \{M_{N_f+2}, \ldots, M_m\}$ it is $\tau(\alpha^f) + C\alpha^f - (i - N_f - 1)C\alpha$. Hence, total idle time on $M_1, \ldots, M_m$ at the end of the schedule is

$$I = \sum_{i=1}^{N_f}((N_f - i)C\alpha + \tau(\alpha^f) + C\alpha^f - \tau(\alpha)) + \sum_{i=N_f+2}^{m}(\tau(\alpha^f) + C\alpha^f - (i - N_f - 1)C\alpha).$$

Energy wasted in idle waiting at the end of the schedule is $E_2^I = P^I I$.

It remains to calculate the energy consumed by the originator. M_0 starts in networking state and then it is continuously communicating or busy-waiting until distributing the last piece of work. The idle time on M_0 is $\max\{\tau(\alpha) - C(\alpha^f), \tau(\alpha^f)\}$. Hence, the energy consumed on M_0 is $E_0 = P^N T + (P^I - P^N)\max\{\tau(\alpha) - C(\alpha^f), \tau(\alpha^f)\}$. Finally, total energy consumed by the methods using load chunks of fixed size α is

$$E = E_1^I + E^S + E^R + E_2^I + E_0.$$

Below we outline multi-installment scheduling algorithms with their specific ways of defining load chunk sizes α.

Simple Static Chunk (SSC) algorithm assumes that load chunk sizes are equal to the size of available RAM memory, i.e. $\alpha_{SSC} = \rho$. Thus, SSC avoids using out-of-core memory. A disadvantage of simple static chunk algorithm are the final outstanding load chunks. It means that if $q_1 = \lceil V/(\rho m) \rceil \neq \lfloor V/(\rho m) \rfloor = q_2$ then in the last iteration of load distribution many processors may remain idle.

Static Chunk with Underload (SCU) algorithm assumes $\alpha_{SCU} = V/(q_1 m)$. Thus, algorithm SCU sends load chunks of size at most ρ and avoids out-of-core processing at the cost of one more iteration.

Static Chunk with Overload (SCO) attempts to round the number of communication iterations down, at the cost of possibly using out-of-core processing. Hence, in SCO size of the load chunk is $\alpha_{SCO} = V/(m \max\{1, q_2\})$.

Guided Self-Scheduling (GSS) algorithm adapts the idea of the classic loop scheduling algorithm [5]. Let V' be the size of load remaining on M_0. Chunk sizes are calculated as $\alpha_{GSS} = \min\{V', \max\{1, \min\{V'/m, \rho\}\}\}$. For $V > \rho$, the algorithm starts with load chunk sizes of RAM size. When $V' < \rho$, GSS gradually decreases chunk sizes and thus minimizes the spread of machine completion times. GSS does not send load chunk sizes smaller than some fixed size which is denoted here as 1 by convention. This can be a result of data structures representing the solved problem or some size which sufficiently amortizes fixed overheads in processing one chunk. For $V \gg m\rho$ the maximum number of usable processors in GSS is the same as in the previous algorithms because initial load chunks have size ρ. However, if $V < m\rho$ GSS uses chunks smaller than ρ, chunk sizes decrease and communications are getting shorter. In such a situation GSS is able to start more machines than SSC, SCU, SCO without entailing idle time on $M_1, \dots, M_m$.

4 Performance Comparison

In this section we will analyze performance as consumed energy E vs schedule length T. We will also analyze sensitivity of the algorithms to changing problem size V and system parameters. Note, that only the single-installment (SI) method is capable of changing energy consumption E with changing T. A study of the E vs T trade-off computed by SI can be found in [10]. The multi-installment methods do not offer such a trade-off and the only parameter which can impact E and T, given computing system and problem size, is the number of machines m. Hence, in the following figures we study impact of m on E and T. In order to compare SI against multi-installment methods, the shortest schedules on the given number of machines m will be used for SI method. Unless stated to be otherwise the system and application parameters were the following: $V = 10\,\text{GB}$, $a_1 = 0.082\,\text{s/MB}$, $a_2 = 2.366\,\text{s/MB}$, $b_2 = -2274.9\,\text{s}$, $k_1 = 13\,\text{J/MB}$, $k_2 = 294\,\text{J/MB}$, $l_2 = -280\,\text{kJ}$, $C = 7.8\,\text{ms/MB}$, $S = 10\,\text{s}$, $P^I = 14\,\text{W}$, $P^N = 91\,\text{W}$, $P^S = 101\,\text{W}$, $\rho = 996\,\text{MB}$. It can be verified that processing out-of-core is roughly 28 times slower per MB than processing on-core. The energy consumption per MB is roughly 23 times higher out-of-core. Communication rate C corresponds with communication speed of $\approx 1\,\text{Gb}$ per second. P^I, S, P^S represent a very light-weight system which quite effectively switches from hibernation to the running state. The size of RAM accessible for storing data is $\rho = 996\,\text{MB}$. These values have been measured in a real system, for an application consisting in searching for patterns in a big data file [6, 10].

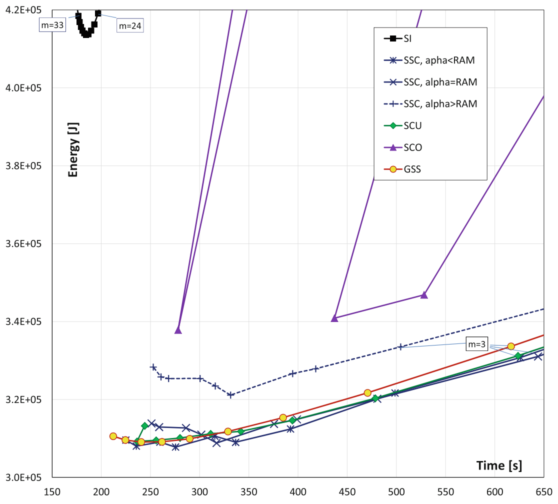

Fig. 2. Time-energy diagram for the default system.

We start with a time-energy chart in Fig. 2 for the above reference parameters, to introduce the phenomena guiding performance. The dependencies are shown only partially for better visibility (but will be shown in their entirety in the next figure). It can be observed that with growing m not only T decreases but so does used energy E. Hence, the smallest m is shown on the right-hand-side of the chart. Energy performance is ruled by the following effects. On the one hand, growing number of machines shortens the schedule and the root M_0 is using less energy. On the other hand, adding machines incurs energy cost. As a result, it can be observed that energy first decreases with shortening of the schedule, but then is starts to increase. This phenomenon can be observed in the following figures. The shortest schedules are built by the single installment method (SI, in the upper-left corner), but using $m = 24$ and more machines has big cost in energy needed to start them. At these values of m it is possible to fit the whole load V in core memories. Note that SI has apparent energy use minimum at $m \approx 28$. Big irregularities in time and energy can be observed in SCO. Since V is not always divisible by $m\rho$ and rounding chunk sizes up results in various values of the difference between α and ρ, even small excesses of chunk sizes above ρ escalate time and energy consumption. Consequently, SCO has big irregularity in performance and should be avoided. Results for the simple static chunk (SSC) algorithm are shown for three chunk sizes: 680 MB, 996 MB, 998 MB, where $\rho = 996$ MB. It can be seen that even small increase of the chunk size beyond ρ has bad impact on the energy use. Chunks smaller than ρ have advantage of shorter waiting time at the start of the schedule and better load balance at its end. Hence, a small dominance of SSC with $\alpha < \rho$ for the maximum usable number of machines. For the given parameters the maximum number of processors which can be applied without idle time is $m = 11$. Static chunk with underload (SCU), SSC with $\alpha < \rho$ and guided-self-scheduling (GSS) have very similar performance. Still, SCU suffers from minor irregularities in performance (T, E for $m = 11$ are bigger than for $m = 10$) which are results of uneven rounding of $V/(m\rho)$. Moreover, GSS is able to construct slightly shorter schedule due to decreasing chunk sizes and consequently smaller dispersion of processor completion times.

In Fig. 3a time-energy chart is shown for $V = 10\,\mathrm{G}$ and $V = 100\,\mathrm{G}$. The static chunk with overload (SCO) manifests great irregularities because T, E are not monotonic with growing m. Due to this adverse feature SCO will be omitted in the further discussion. The SI method greatly improves its performance with growing m because it is becoming able to shift the load from the out-of-core to the on-core processing for sufficiently big m. Finally, at $V = 10\,\mathrm{G}$ and $m > 11$ its performance becomes comparable with multi-installment methods. In Fig. 3b time-energy chart is shown for $\rho = 100\,\mathrm{MB}$ and $\rho = 10\,\mathrm{GB}$. For SI dependencies for $\rho = 1\,\mathrm{GB}$, $10\,\mathrm{GB}$ are shown because SI's results for $\rho = 100\,\mathrm{MB}$ are out of the range shown in Fig. 3b. It can be seen that SI method is competitive with the remaining algorithms only if the load is stored in core. What is more, under such circumstances SI is able to build the best energy schedules (lower-left part of the chart). SI is capable of constructing shorter schedules, but it activates new machines which brings energy costs bigger than in the other methods. SSC method for $\rho = 10\,\mathrm{G}$ uses just one load chunk, schedule length T is constant, and adding each new machine only increases energy costs. Surprisingly, energy performance of the multi-installment methods for small $\rho = 100\,\mathrm{MB}$ is better than for $\rho = 10\,\mathrm{GB}$ because small load chunks reduce initial and final idle times. It can be also observed that GSS for $\rho = 10\,\mathrm{GB}$ is capable of constructing shorter schedules than other multi-installment methods because by shrinking chunk sizes it is able to avoid idle times on processors and still activate more of them, though using more energy. Both GSS and SI approach the minimum

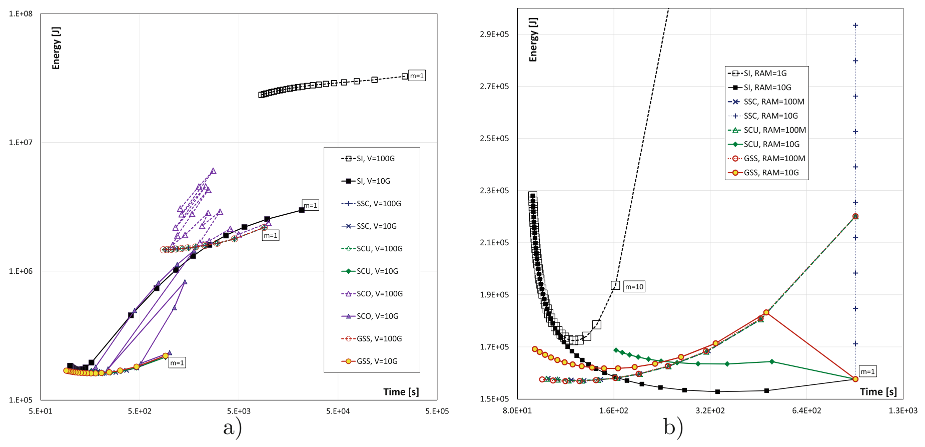

Fig. 3. Time-energy dependence (a) for $V = 10\,\mathrm{G}$ and $V = 100\,\mathrm{G}$. (b) for varying ρ.

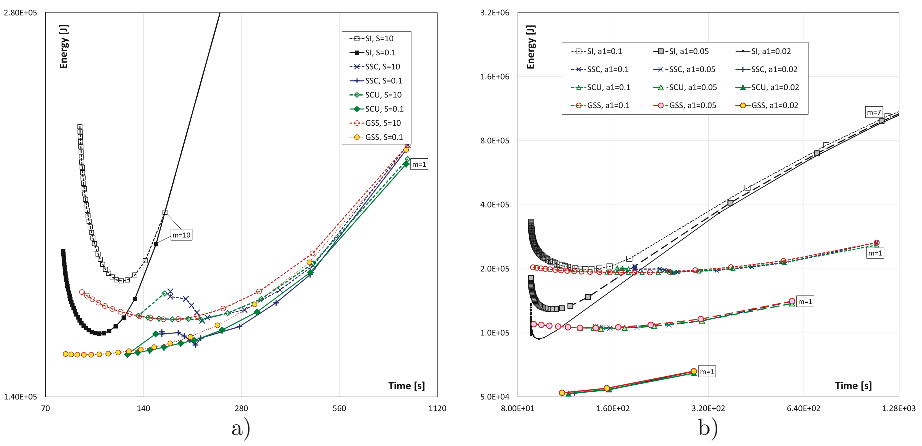

Fig. 4. Time-energy dependence (a) for $S = 10\,\mathrm{s}$ and $S = 0.1\,\mathrm{s}$, (b) for changing a_1.

schedule length determined by communication time: $S + CV$. However, GSS is more energy-efficient.

In Fig. 4a time-energy relation is shown for two values of the startup time $S = 0.1\,\mathrm{s}$ and $S = 10\,\mathrm{s}$. Two effects of reducing startup time can be observed. The schedules get shorter roughly by the startup time of the first processor, and energy consumption is decreased by the amount of energy saved in the startup of the machines. In Fig. 4b impact of changing processing rate a_1 is analyzed. The value of a_1 can be changed by designing a faster algorithm to solve the considered problem. Assuming, that this new application runs on the same computer, also k_1 must decrease proportionally. Three values of a_1 are shown: $a_1 = 0.1, 0.05, 0.02$ which corresponds with an algorithm twice and five time faster. The number of processors which can be activated by algorithms SSC, SCU decreases with increasing processing speed (a_1 decreases). Hence, this number decreases from $m = 13$ machines for $a_1 = 0.1$ to $m = 3$ for $a_1 = 0.02$. Though time- and energy-performance of all multi-installment algorithms is similar, GSS algorithm has an advantage of using more machines than SSC, SCU and consequently building shorter schedules though at higher energy costs. The SI method is able to construct schedules of comparable length but by using more machines and energy. The advantage in energy of multi-installment methods over SI grows with decreasing a_1 (i.e. increasing speed).

5 Conclusions

In this paper time- and energy-performance of scheduling algorithms for divisible computations in systems with hierarchical memory has been studied. The time- and energy-performance is determined by: (i) size of load chunks which regulates on-/out-of-core processing, (ii) number of usable processors which decide on minimum schedule length, (iii) amount of idle time which rule wasted energy. It turns out that intensive use of out-of-core computations is not a good idea and should be avoided as demonstrated by SCO method. Yet, it cannot be unanimously concluded that on-core processing is the only reasonable choice because in more complex applications the results obtained in small pieces still must be merged (which was not considered here). Hence, in such more complex applications, e.g. in sorting, some degree of out-of-core computations maybe acceptable. Algorithms SI and GSS are able to employ the biggest number of processors and hence build the shortest schedules. However, SI is energetically competitive only if whole load fits in core memories. Moreover, SI has much higher computational complexity and requires information on system parameters. GSS may perform quite many communications which in practice may be cumbersome and costly. Here communication costs were limited by bounding from below chunk size, but this may cripple performance. Thus, a more detailed model of communication cost can be a subject of the further work. Overall, GSS algorithm can be recommended as a good compromise of performance and implementation simplicity.

References

1. Agrawal, R., Jagadish, H.V.: Partitioning techniques for large-grained parallelism. IEEE Trans. Comput. **37**, 1627–1634 (1988)
2. Berlińska, J., Drozdowski, M.: Scheduling divisible MapReduce computations. J. Parallel Distrib. Comput. **71**, 450–459 (2011)
3. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling Divisible Loads in Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamitos (1996)
4. Cheng, Y.-C., Robertazzi, T.G.: Distributed computation with communication delay. IEEE Trans. Aerosp. Electron. Syst. **24**, 700–712 (1988)
5. Drozdowski, M.: Scheduling for Parallel Processing. Springer, London (2009)
6. Drozdowski, M., Marszałkowski, J.M., Marszałkowski, J.: Energy trade-offs analysis using equal-energy maps. Future Gener. Comput. Syst. **36**, 311–321 (2014)
7. Drozdowski, M., Wolniewicz, P.: Out-of-core divisible load processing. IEEE Trans. Parallel Distrib. Syst. **14**, 1048–1056 (2003)
8. Fuller, S.H., Millett, L.I.: Computing performance: game over or next level? Computer **41**, 31–38 (2011)
9. Katz, R.H.: Tech titans building boom. IEEE Spectr. **46**(INT), 36–49 (2009). http://www.spectrum.ieee.org/feb09/7327
10. Marszałkowski, J.M., Drozdowski, M., Marszałkowski, J.: Time and energy performance of parallel systems with hierarchical memory. J. Grid Comput. (2015, accepted). doi:10.1007/s10723-015-9345-8
11. Mills, R.T., Yue, C., Stathopoulos, A., Nikolopoulos, D.S.: Runtime and programming support for memory adaptation in scientific applications via local disk and remote memory. J. Grid Comput. **5**, 213–234 (2007)
12. Robertazzi, T.: Ten reasons to use divisible load theory. IEEE Comput. **36**, 63–68 (2003)
13. Sohn, J., Robertazzi, T.G., Luryi, S.: Optimizing computing costs using divisible load analysis. IEEE Trans. Parallel Distrib. Syst. **9**, 225–234 (1998)
14. The Green 500, November 2014. http://www.green500.org/