Poznań University of Technology
Institute of Computing Science

Scheduling divisible computations with energy constraints

Doctoral thesis

Jędrzej Marszałkowski

Supervisor:
prof. dr hab. eng. Maciej Drozdowski

Poznań, 2020

Abstract

In this thesis scheduling and performance of data-parallel computations are studied. Data-parallel computations consist in processing data objects of similar nature in distributed computing systems. Important features of data-parallel applications are that the data objects are small in relation to the whole size of processed information and can be processed independently. Since the volumes of data objects are great usually, the time of distributing them for remote processing must be taken into account. Furthermore, memory sizes of computer systems are too small to process significant parts of load in core (RAM) and this limitation should be also taken into account when planning a schedule for distributed data-parallel computation. The schedule must be effective in two key criteria: time and energy. Divisible load theory is used as a general framework for the analysis of the considered parallel processing problems. Two key assumptions of divisible load theory are that parts of the load can be processed independently in parallel and that these parts can be flexibly sized as if the load were arbitrarily divisible. These two assumptions suit well data-parallel computations. Furthermore, the two assumptions allowed to formulate scheduling models which can be solved by computationally tractable methods, in some cases to optimality. The scheduling algorithms optimizing time and energy performance are used not only to effectively arrange communications and computations in time and

space, but also to predict the performance. Since the performance of parallel computation is ruled by many mutually dependent factors, isoefficiency and isoenergy maps were applied as visual aids supporting performance analysis and building understanding of the phenomena determining the performance. Isoefficiency and isoenergy maps are two-dimensional depictions of system parameter values giving constant time- and energy-performance, respectively. The impact of memory hierarchy and system heterogenity is studied. The trade-off between solution quality and computational cost of scheduling algorithms is also examined.

Contents

1 Introduction

1.1 Motivation

Processing big volumes of data is a challenge resting at the core of contemporary science and industry. For the purpose of dealing with the challenges of big data processing many new technologies have been developed: data-processing platforms and frameworks [8, 12, 27], programming libraries [10, 11], database management systems [7, 9, 65]. Another concern is energy consumption of the computing equipment. It was established quite early [41, 49, 51] that powering the data centers driving the Internet economy is very costly and energy is an important component of the cost of ownership. Moreover, huge data centers strain power grids. Thus, energy supply and cost impose limits on further growth of data centers and supercomputers. Available energy limitations are an important issue also in the sensor networks, aerospace, Internet of things applications. This thesis is dedicated to the analysis and optimization of time- and energy-performance in processing big data volumes. Effective scheduling of parallel application execution will be the way of optimizing its performance.

Computing platforms, especially in the context of big data volumes, impose a number of limitations. For example, distributing data for remote processing is not instantaneous and communication delays are significant part of the application execution time. Hence, communication delays must be taken into account when planning execution of a parallel application. Present-day computers

have hierarchical memory structure ranging from CPU registers, through CPU caches, core memory a.k.a. RAM, to the external storage (networked caches, SDDs, HDDs). The size of memory is growing, but access speed is decreasing when going to lower memory levels. Also energy intensity of applications operating on different memory levels differs extensively. Data volumes which must be processed in the current applications easily exceed core memory size of contemporary computers. Therefore, using out-of-core memory should be avoided or the drop in processing speed resulting from using lower memory levels must be accounted for. Computer energy consumption is reduced by applying various energy saving modes accessible, e.g., by ACPI [92] standard. A practical plan for parallel application execution should exploit this option of energy saving, e.g., by switching on a necessary computer when needed only. Finally, heterogeneity of computing platforms, be it in the form of mixing CPU with GPU computing, applying various compute instances like Amazon EC2 [1], or central processing servers with remote sensors, is one more reality of the current applications which should represented in the scheduling methods for the present computing. In this thesis we intend to propose methods for scheduling parallel applications taking into account communication delays, hierarchical memory levels, existence of energy saving modes and system heterogeneity.

1.2 Goals and Scope

A high-level goal of this thesis is to advance energy efficiency by better management of the parallel application execution and system resources. Another high-level goal is to foster understanding of time and energy performance relationships and limitations in parallel processing. A lower level goal of this thesis is to represent distributed computing platforms and applications in scheduling and performance models. The next goal is to propose algorithms solving these models, and as a result to be able to construct effective schedules for parallel

applications and to predict their performance. By virtue of analyzing quality criteria of the constructed schedules, understanding of the performance determinants and relationships will be built.

In order to accomplish the above goals, models of computation runtime and energy consumption on a single computer vs size of the processed data for data-parallel applications will be developed and validated. On the basis of the individual computer runtime and energy models, problems of constructing schedules for parallel application on systems of distributed machines will be expressed as optimization problems with the criteria of time and energy. The optimization formulations will cover systems with unlimited memory, homogeneous systems with hierarchical memory, and heterogeneous systems with hierarchical memory. Since the formulated optimization problems must be solved, algorithms for this purpose will be developed and their computational costs will be also assessed. Finally, the impact of the computing platform and application parameters on the time and energy criteria will be examined to depict performance phenomena. That is to say that visual tools will be used to uncover time and energy performance phenomena.

1.3 Methodology

Data-parallel computations consist in processing objects of similar nature in distributed computing systems. The data objects are usually small in relation to the whole size of processed information and can be processed independently. Data-parallel computations are quite common in processing big volumes of data. Divisible load theory (DLT) is a general framework for scheduling and performance analysis of parallel applications. DLT conventionally refers to the processed data as to *load*. Two key assumptions of divisible load theory are that parts of the load can be processed independently in parallel and that these parts can be flexibly sized as if the load were arbitrarily divisible. These two assumptions suit well data-parallel computations, or computations

on large volumes of data. Furthermore, the two assumptions allowed in the past to formulate scheduling models for various parallel processing problems, and these formulations could be solved by computationally tractable methods. Scheduling data-parallel computations is the subject of this thesis and divisible load theory will provide the analysis framework. Divisible load theory assumes linear dependence of computation time on the size of processed data. This assumption will be validated experimentally both for the computation time and for the consumed energy.

Going further into the technical details of the applied methodologies, mathematical scheduling problems will be formulated as optimization problems in the framework mathematical programming, and in particular, of mixed integer linear programming. These formulations need solving algorithms, while algorithms may differ in the sense of computational cost and solution quality. The algorithms will be assessed analytically according to the methodology of computational complexity theory [42] and experimentally in a series of computational experiments, hence data analysis methods will be used.

Performance of parallel computation is ruled by many mutually dependent factors. Though mathematical formulations of scheduling problems as optimization problems provide analytical models of performance, understanding the performance sensitivities and their relationships with the platform is not easy. Therefore, isoefficiency and isoenergy maps will be applied as visual aids supporting performance analysis and building understanding of the phenomena determining the performance. Isoefficiency maps and isoenergy maps, respectively, are two-dimensional depictions of system parameter values giving constant time- and energy-performance.

1.4 Outline of the thesis

Further organization of the thesis is the following. In the next section a short outline of the state-of-the-art on the approaches to energy optimization in computer systems and on divisible load theory aspects related to this thesis will be given. In Chapter 3 results of the measurements validating the model of time and energy dependence on the size of processed load are presented. Chapter 4 is dedicated to performance visualization by use of isoenergy maps. Energy consumption in homogeneous systems with hierarchical memory is studied in Chapter 5. Chapter 6 introduces the most general problem of scheduling divisible computations on heterogeneous systems with hierarchical memory and multi-installment load distribution. The last section is dedicated to conclusions. Key notations used throughout the thesis are summarized in Appendix A. However, each chapter will use also its own local notation for the sake of simplifying the notations where possible.

2 Related Work

In this section we provide a short introduction to the fields related to the thesis. In particular a short outline of the studies on energy consumption in parallel systems will be given. Divisible load theory with its basic assumptions, and its classic scheduling formulations for star network with and without memory limitations will be given. Finally, the idea of performance visualization by maps comprising isolines will be introduced.

2.1 Energy Consumption in Computer Systems

Time and energy efficiency in parallel processing is intensively studied and a wealth of results on this subject exists, see the surveys [50, 62, 68, 72, 80, 88, 101]. The problem of attaining time-energy efficiency has been attacked on several, not mutually exclusive, levels of abstraction: (i) on hardware of computing platform level [13, 59, 70, 77], (ii) algorithm level [23, 53, 97], (iii) runtime environment level [20, 54, 60, 99], (iv) scheduling and management at a data center level [14, 39, 40].

For example, progresses in CPU hardware can be illustrated by the results from [66]. In [66] authors evaluated 64 CPUs with respect to their speed in specpower_ssj2008 benchmark [83] transactional workload processing and energy consumption when all the processor cores are used. We arranged these results in Fig. 2.1 and added years of releasing certain processor models as color shades.

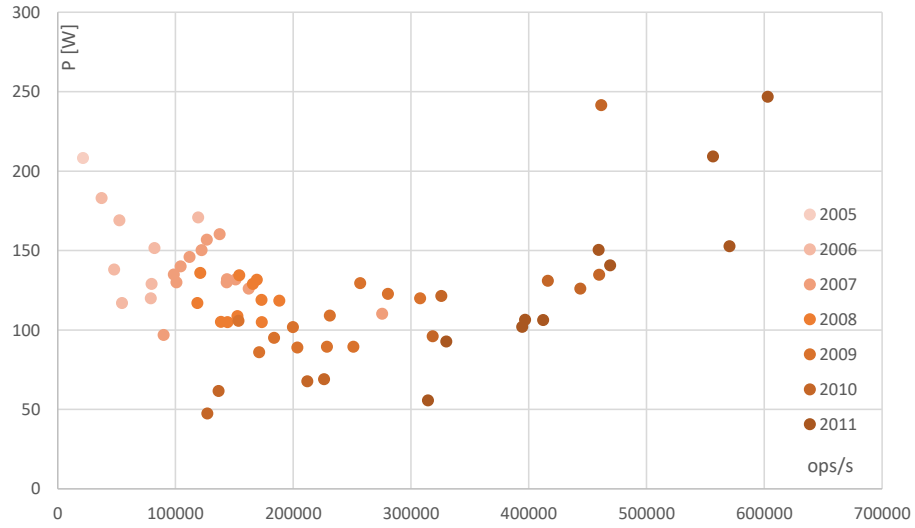


Figure 2.1: CPU Power vs Speed. On the basis of [66]. Shades of color indicate year of introducing the CPU.

The lower-right corner in Fig. 2.1 is the ideal position with the highest speed and the lowest power consumption. It can be observed that recent CPU generations align along a kind of a Pareto-front because higher speed requires higher power usage. Conversely, the older CPU generations (on the left side of Fig2.1 expose both lower speed and higher power consumption. The progresses in CPU time and energy efficiency are even better illustrated in Fig. 2.2 which is showing operations per second and operations per Watt. It can be seen that newer generation CPUs are closer to the ideal point with low energy consumption and short processing time, than the older CPU generations.

Construction of efficient interconnections for big datacenters is studied in [4, 38]. It appears that not only computers, but also networking consumes considerable power. It is observed that communication equipment can easily consume power in the range of hundreds of kW.

The problem of effective algorithms, hardware, and their co-design (level (ii)) has been studied, e.g., in [4, 15, 18, 24, 38, 52]. The issue of energy-efficient algorithms can be illustrated with the example of data compression. It is believed that compression may provide performance benefits, and energy

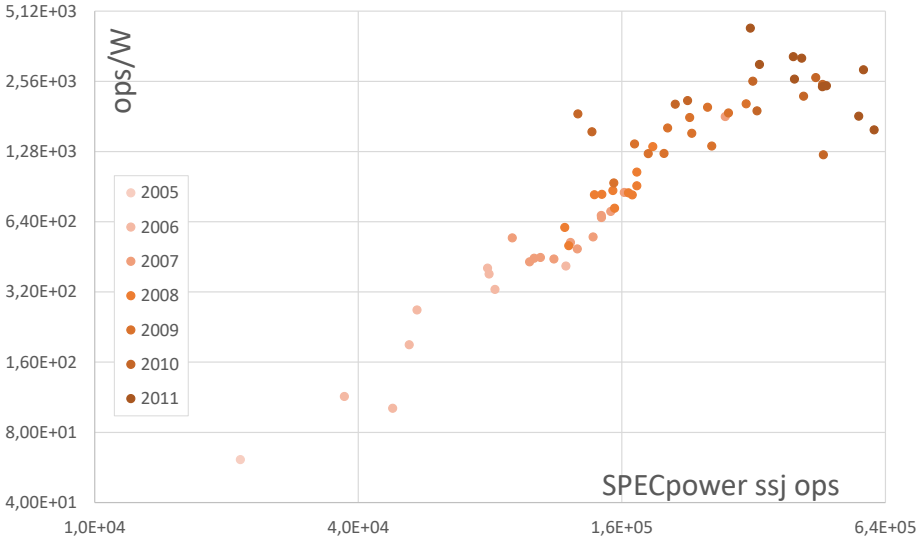


Figure 2.2: CPU operations per Watt vs operations per second. On the basis of [66]. Color shade indicates year of introducing the CPU.

savings when transferring data between remote computers or different levels of memory hierarchy. However, it is demonstrated in [15, 52] that the real picture is much more complicated. Only some compression algorithms, for some types of data give any gain in energy.

Publications [69, 82, 84, 85, 95] serve as examples of the application-level energy usage optimization. The energy cost and performance loss of parallel applications executed at different energy gears are empirically studied in [69]. An energy gear is voltage-frequency combination of a CPU. In [85] multi-variable linear regression is used to model execution time and energy consumption of the high-performance Linpack benchmark. In [84] a problem of constructing the shortest schedule for multiphase parallel computation, meeting energy limit is considered. The energy use model distinguishes energy used in communication and in computation. The model is experimentally validated. An index of iso-energy-efficiency is introduced in [82], as the ratio of the energy consumed in sequential computation to the energy consumed in parallel computation. Let us observe, that despite similarity of the name the iso-energy-efficiency of [82] is conceptually different than the isoefficiency in [33, 44, 45] and isoenergy maps

in this work. Analogously to [33, 44, 45], we consider isolines as relations (in the mathematical sense) linking system and application parameters such that energy needed for the computation is constant. In [95] Amdahl's law is used to construct general analytical model of energy consumption in multicore processors.

Energy use in DLT applications has been considered, e.g., in [78] to assign measurement workload in a wireless sensor network. The residual battery energies were used to determine workload partition, resulting in longer lifetime of the whole network.

Publications [5, 96] serve as examples of dynamic voltage and frequency scaling (DVFS) which can be applied at the runtime environment, or operating system levels, to optimize energy usage and respect application timing constraints (deadlines).

The dedicated cluster level management of distributed application was studied, e.g., in [84, 91]. Energy usage optimization as an issue of scheduling and management at the data center level has been analyzed, e.g., in [24]. In [24] recommendations for energy optimization in datacenters are given: proportionality between energy and computation, frugal use of resources, robust components which can tolerate mutual dis-synchronizations and delays caused by energy-saving modes. Managing data centers for energy efficiency and profit, even at global scale, was considered in [43, 71].

Now let us locate the contribution of this thesis in the relation to the above literature. In this work we assume application level scheduling by the runtime environment in a close cooperation with the computing platform. On the one hand, our results give hints to the platform on the set of active machines and on applying energy-saving modes. On the other hand, the algorithms in runtime environment do their best with the provided heterogeneous computing platform.

In [18] techniques of energy-efficient hardware and software design are reviewed. Authors distinguish three phases of system design: 1) modeling and conceptualization, 2) design and implementation, 3) runtime operation. Ac-

cording to the distinction made in [18] we introduce a modeling and conceptualization method which can be applied at the design phase of a datacenter development. As in [4, 38] we consider networking as an important component of the overall power consumption. Many of the above papers introduce models of energy consumption. Some of the models are very detailed and tailored to a single algorithm. Also in this thesis energy consumption models will be proposed, validated and used.

2.2 Divisible Load Theory

Energy consumption models considered in this work are built on the divisible load theory (DLT). DLT assumes that computation consists in processing big amounts of data (the load) on remote computers. The data granularity is sufficiently small to assume that the load is arbitrarily divisible. There are no precedence constraints such that parts of the load can be processed independently of each other. Hence, DLT represents well-structured highly parallelizable data-intensive computations. DLT originated in the late 1980s, when computations on clusters of workstations [2], and on chains of intelligent sensors [26] were analyzed. Though the divisible load model is based on simple assumptions, this can be considered its strength because its input data can be easily obtained. The accuracy of DLT has been tested in many publications, e.g. [2, 34, 56]. The difference between the model and reality was in the range of 1% and better. Furthermore, basic DLT models are tractable where many other scheduling models for parallel applications are **NP**-hard. Thus, DLT is a good compromise between model accuracy and cost. Not surprisingly, DLT proliferated in many ways. Due to space limitations we direct interested readers to the surveys [22, 28, 74, 75] on DLT.

In the divisible load scheduling problems considered in the thesis three aspects must be particularly considered: 1) load scattering algorithm, 2) generally the representation of the energy cost of the computation and in particular,

3) non-linearity of time and energy dependencies on the size of processed load in systems with hierarchical memory. The load can be distributed to processors in *single* or in *multiple installments* scattering. In the former case each processor may received a chunk of load for processing at most once. In the latter case each processor may receive more than one load chunk for processing. In the single-installment communications load chunks tend to be large which delays start of the computations and results in larger memory footprint. Conversely, in the multi-installment communications messages are shorter, computations start earlier and memory footprint is smaller. However, scheduling multi-installment communications is harder. Divisible load processing problems of similar nature have been studied, e.g., in [16, 25, 32, 36, 55, 76, 79, 81]. Multi-installment scattering of the load to heterogeneous processors was studied in [79]. Two heuristics were proposed assuming that the sequence of communications with the machines is a known repetitive pattern. Here we allow for any sequence of communications, and what is more, the sequence need not be repetitive.

Almost all contemporary computer systems use hierarchical memory systems. Memory hierarchy may include the following levels:

1. CPU registers,
2. CPU cache levels L1, L2, L3 and even L4,
3. Random Access Memory (RAM) a.k.a. core memory (often acting as a cache for lower memory levels),
4. Solid State Drives (also acting as a cache of HDD [48]),
5. Hard Disk Drives (also acting as a cache for remote storage),
6. network storage (e.g. NAS using AoE, FCoE, NFS, SMB, and similar protocols),
7. tape, optical devices and other forms of long-term storage.

When going down the hierarchy, from CPU registers to the external storage, several things change. Capacity increases while time performance decreases. Usually time performance deteriorates both in terms of latency and throughput. The size per storage device increases and price per storage unit (e.g. GB) decreases while moving down the hierarchy. The last two issues play role in designing data centers rather than in scheduling parallel applications. Things get more complicated if one includes in this scheme memory of modern graphics cards [63]. Introducing new forms of fast storage, such as SSD, between RAM and HDD shifts the balance in time and energy costs between the I/O software stack and hardware. Since the intermediate SSD storage is faster, the I/O operations are called more often thus exposing computational costs of I/O software stack [87]. Such complex interactions lead to counterintuitive conclusions (cf. *"Software Considered Harmful"* in [87]). It demonstrates that the more a study of the impact of changing system parameters on time and energy performance is needed. Memory architecture is a broad research and engineering field certainly exceeding scope of this thesis. Interested readers might find further details, e.g., in [48, 73, 86]. An important consequence of memory hierarchy existence is nonlinear dependence of the computation time and used energy on the size of processed data. In [36] hierarchical memory systems have been analyzed with respect to time performance. Non-linear complexity divisible load scheduling was studied in [16]. It was demonstrated that the classic DLT approach based on load equipartition is not well suited for non-linear complexity algorithms unless clever data partitioning methods are applied. How this can be achieved was demonstrated for vector outer-product and for matrix multiplication. Our case is slightly different because the nonlinear execution time is a result of memory hierarchy and not the data processing algorithm itself. Consequently, simple load partitioning algorithms are applicable in our case.

In the cases when the memory system can be perceived as non-hierarchical, we will be calling it *flat memory* system.

Energy may be considered a special type of cost. Scheduling divisible computations for minimum cost has been analyzed in [25, 29, 30, 64, 81]. Scheduling with monetary cost has been considered in [25, 81]. In [76] a polynomial-time algorithm has been proposed to build time-cost trade-off when communication, computation times, and costs are proportional to the size of the load and all communication links have the same speed. [55] is the first work studying a flat (non-hierarchical) memory model. A heuristic has been proposed for the case with communication times proportional to the size of the load. Let us note that memory use model in [55] is different than it this thesis. If total load size exceeds size of memory then a feasible solution does not exist in [55]. However, in this thesis memory limits are soft: a feasible solution always exists, though possibly with bad performance. Energy in processing divisible loads on homogeneous flat memory systems were subject of [32]. Interrelations between system parameters were represented by maps of equal energy consumption similar to weather maps. In [30, 81] only costs of computation were studied. In [29, 64], the cost of communication and energy consumption have been also included. Methods of designing effective communication have been proposed in [64]. In [30] scheduling divisible loads has been formulated as a linear program minimizing schedule length with cost, memory, and processor availability constraints. Thus, it has been demonstrated in [30, 81] that scheduling divisible loads for minimum schedule length and cost is effectively a bicriterial problem.

The timing model of parallel computation here is different than in the earlier publications assuming general type of computation cost. The costs of computation initiation (startup costs) will be taken into account. We assume that computing system can be in two states: active state or idle state with reduced power usage. The idle state represents various techniques, such as DVFS, ACPI, used to achieve energy-proportional computing. Moreover, in this work we set further goals than in [29, 30, 64, 81]. The methods of load partitioning and computation/communication cost calculation will be components of a generic performance analysis method.

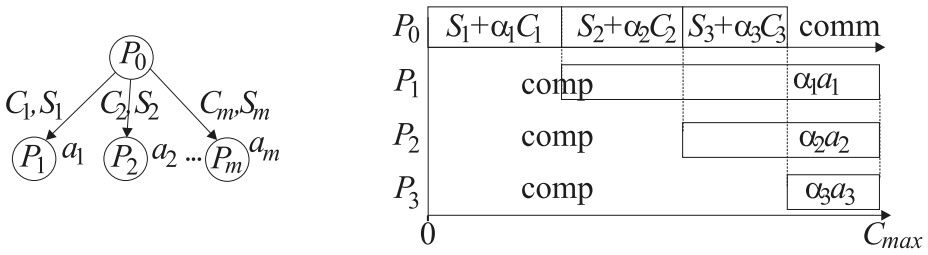


Figure 2.3: Start interconnection and a schedule without returning results.

2.2.1 Reference Star Model

In this section a classic single-installment divisible load scheduling in a star topology (single level network) method is introduced for further reference. We assume that initially all load V is stored by the originator P_0 (root, master). It is then divided into parts $\alpha_1, ..., \alpha_m$ and sent to processors $P_1, ..., P_m$ for processing. The originator is only dividing and sending the load. There are no computations on P_0. Computations on each processor P_i are started after receiving whole load chunk α_i for $i = 1, \ldots, m$. $P_1, ..., P_m$ are not communicating between each other. Star network can model many parallel systems where computing master-worker paradigm is used, such as: CPUs in SMP system sharing a bus, network workstations connected to the same Ethernet segment, computing clusters connected to a master controller via Internet. Parameters a_i, C_i, S_i depend on both, the computing environment and the application.

Starting computation on a remote computer involves waking it up. Hence, startup time S_i elapses before P_i ($i = 1, \ldots, m$) can start receiving the load. Startup time S_i is an important element in modeling DLT applications. Without the startup, arbitrary number of processors may be activated which is unrealistic [22, 28]. Load of size α_i is transmitted in time $S_i + \alpha_i C$. On the receipt of the load P_i immediately starts computations which last $\alpha_i a_i$ units of time. When the communication with P_i is finished, the originator activates processor P_{i+1} and sends to it load α_{i+1}. The procedure is repeated until starting computations on all m processors. Let us assume that the time of returning results is negligibly

short. The requirement of negligibly short result return time can be relaxed, and the process of result collection can be represented in DLT. It can be shown [22, 26, 74] that for the optimality of the schedule length all the processors must finish computations simultaneously. This is often called an *optimality criterion* in DLT. The problem is to find load partitions $\alpha_1, ..., \alpha_m$ such that the whole schedule is the shortest possible. We can solve this using a system of linear equations:

$$\alpha_i a_i = S_{i+1} + \alpha_{i+1}(a_{i+1} + C_{i+1}) \text{ for } i = 1, ..., m-1 \tag{2.1}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{2.2}$$

The above linear equations system can be solved in $O(m)$ time because of its special structure. We can express α_i (for $i = m, \ldots, 1$) as a linear function $k_i\alpha_m + l_i$ of α_m, where $k_m = 1, l_m = 0, k_i = k_{i+1}\frac{a_{i+1}+C_{i+1}}{a_i}, l_i = \frac{S_{i+1}}{a_i} + l_{i+1}\frac{a_{i+1}+C_{i+1}}{a_i}$. Then we will have

$$\alpha_m = \frac{V - \sum_{i=1}^{m} l_i}{\sum_{i=1}^{m} k_i} \tag{2.3}$$

Load distribution is simpler to calculate if $\forall i, S_i = 0$, so even closed-form solutions exist. Note that $k_i = \prod_{j=i}^{m-1} \frac{a_{j+1}+C_{j+1}}{a_j}$, for $i < m$, and $l_i = 0$. Then, the load sizes can be calculated from equation

$$\alpha_i = V \frac{\prod_{j=1}^{m-1} \frac{a_{j+1}+C_{j+1}}{a_j}}{\sum_{h=1}^{m} \prod_{j=h}^{m-1} \frac{a_{j+1}+C_{j+1}}{a_j}} \tag{2.4}$$

For homogeneous processors $\forall i, a_i = a, C_i = C$ the above equations reduces to $\alpha_i = V\frac{\kappa^{m-1}(\kappa-1)}{\kappa^m-1}$, where $\kappa = \frac{a+C}{a}$. Closed-form solutions can be derived when system is homogeneous or when startup times are negligible.

In the solution (2.3) of equation system (2.1)-(2.2), α_m could be negative. If so, the system is infeasible and load size V is too small to activate all processors. In the heterogeneous systems two questions arise: (1) which processors subset should be used, (2) what is the optimum sequence for sending load to processors. These questions are answered in the thesis albeit in a more general setting.

It has been shown in [98] that the problem of selecting optimum set of processors is **NP**-hard even if all communication rates C_i are equal zero.

2.2.2 Star with Flat Memory

In this section a simple model of single-installment divisible load processing in star network with limited flat memory is introduced. In single installment model a processor receives only one load chunk and the memory must be able to fit the incoming amount of load. This may cause load size limitations. In the case of flat memory model access time to all memory cells is constant. In such a way we can approximate a more complex hierarchical memory system by restricting it to just one memory level (which may be realistic representation of the architecture of some mobile or embedded devices). Computing speed is independent here of the size of used memory block. Yet, memory is a limited resource, and only some amount can be accessed in constant speed. Let B_i denote the size of memory available for processor P_i. A linear programming approach finding load partitioning $\alpha_1, \ldots, \alpha_m$ has been proposed in [35]:

$$\text{minimize } C_{max} \tag{2.5}$$

subject to:

$$\alpha_i a_i + \sum_{j=1}^{i} (S_j + \alpha_j C_j) \leq C_{max} \text{ for } i = 1, .., m \tag{2.6}$$

$$\alpha_i \leq B_i \text{ for } i = 1, .., m \tag{2.7}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{2.8}$$

$$\alpha_i, C_{max} \geq 0 \tag{2.9}$$

In the above formulation schedule length is minimized by (2.5). A completion time of computation on processor P_i cannot exceed schedule length by (2.6). According to (2.7), load assignments do not exceed sizes of memory buffers. Constraint (2.8) guarantee that whole load is processed. Let us observe that the above formulation becomes infeasible if load size is larger than the total memory size, i.e., when $V > \sum_{i=1}^{m} B_i$. Note that the linear program assumes that all processors take part in the computations and the sequence of starting computations is given. However in general, when start-up times S_i are nonzero and size V of the load is too small some processors may be dropped from the computation. The problem of selecting optimal sequence, subset of processors and load part sizes will be subject of this thesis in the energy optimization context.

The problem of scheduling divisible loads with nonzero communication start-up times and limited memory buffers has been shown to be **NP**-hard in [37] and s**NP**h in [17]. Various methods of solving the above problem have been compared experimentally in [21, 37].

2.3 Isolines and isoline maps

In this section we present the concept and origins of isoline maps, which will be used as visual aids in showing time and energy performance phenomena and relationships.

The concept of graphical representation of points of equal value of certain parameter as lines in two-dimensional pictures is widely used in science and technology. Such lines are often referred to as isolines or contour lines. Examples of two-dimensional depictions of complex physical object include [93] elevation contour maps in cartography, isobar, isotherm, isohyet maps in meteorology, enthalpy-entropy chart in thermodynamics [94]. The reason for such a wide

use of isoline maps is that such visualizations proved very effective in building understanding of sensitivities and relationships of complex phenomena in other areas of science and technology.

Performance of parallel computations is measured by two classic metrics: speedup $\mathcal{S}$ and efficiency $\mathcal{E}$:

$$\mathcal{S}(m) = \frac{T(1)}{T(m)} \qquad \mathcal{E}(m) = \frac{\mathcal{S}}{m} = \frac{T(1)}{mT(m)}, \tag{2.10}$$

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 the number of processors 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. Usually 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 [45]. 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. [3], 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 [33]. Such a visualization will be called an *iso-*

efficiency map of a parallel computation. Isoefficiency maps will be used in this thesis. Furthermore, this concept is extended to the *isoenergy maps*, i.e., maps of points of equal energy consumption in the space system parameters.

3 Computation Time and Energy Consumption Models

In this chapter we report on the results of experiments conducted to validate computation time and consumed energy models used in the following sections.

3.1 Testbed

In order to establish compute time energy models measurements were made on different computers with exemplary applications: quicksort, searching for a string in text, md5 hash calculation (rainbow tables), edge detection in bitmap pictures and matrix transposition. These applications were implemented in gcc and run under FreeBSD 8.1 and Ubuntu 14.04 LTS. The wattmeter (Lutron DW-6090) with power resolution 1W, and time resolution 1s was used. The measured computer was connected via the wattmeter to record energy consumption. Some of measured computers had cooling fan speeds dependent on CPU and system temperature. Changing fan speed caused a few Watts difference in measurement. In order to make the results independent of ambient temperature, or thermal results of the earlier experiments we decided to power the fans from outside power source (Fig. 3.1 and Fig. 3.2). Lutron wattmeter is connected to another computer/laptop which is logging data. Schematics of the testbed are shown on Fig. 3.3.

Figure 3.1: Measuring station.

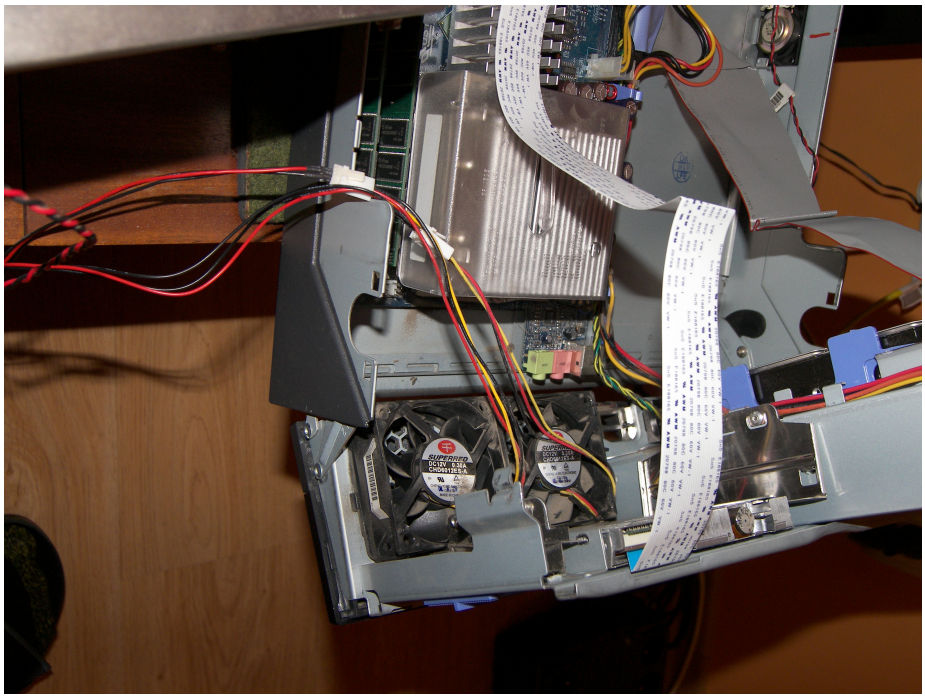


Figure 3.2: Measuring station - cooling fans powered from external power supply.

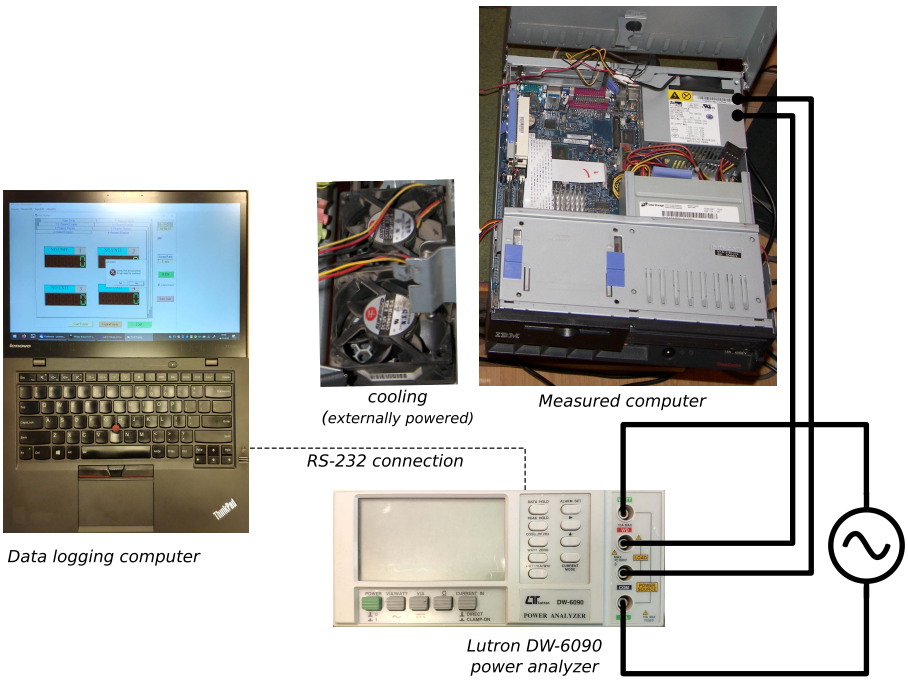


Figure 3.3: Scheme of measurement testbed.

3.2 Flat Memory Model

Practical viability of the DLT linear timing model for distributed computation has been demonstrated, e.g., in [2, 34, 56]. In the flat memory model whole memory is considered a uniform resource with the same performance independently of the size of used data structures. A key assumption of energy use model in flat memory systems is that for the given application and platform power consumption is constant and independent of the amount of processed load α_i. In other words, the application on the given platform achieves a point of stable power usage. This assumption was verified experimentally. Power usage has been measured on three different computers, with five exemplary applications: quicksort, searching for a string in a text, md5 hash calculation (rainbow tables), edge extraction in bitmap pictures and matrix transposition. The applications were implemented in gcc and run under FreeBSD 8.1. All the applications run

Table 3.1: Power versus problem size.

Size α_i	50MB		100MB		200MB		400MB		800MB	
App.	μ	c_v	μ	c_v	μ	c_v	μ	c_v	μ	c_v
Intel Pentium IV 2.8GHz, 1GB RAM DDR 400MHz CL2.5										
quicksort	127.1	1.4	127.6	1.1	127.8	1.0	128.1	1.0	128.7	1.7
string search	133.9	0.8	134.2	1.0	134.1	1.2	133.8	0.8	134.7	1.3
md5	127.6	0.7	128.3	1.3	128.5	1.3	128.3	1.3	128.3	1.2
edge detection	127.4	1.2	127.0	0.6	127.6	1.1	127.2	0.9	128.2	1.3
matrix transpose	130.1	1.0	130.9	1.6	129.6	0.9	130.2	1.3	130.2	1.6
idle	$\mu = 72.7$, $c_v = 2.2$, $k \approx 1.8$									
hibernation	$\mu = 10.0$, $c_v = 2.2$, $k \approx 13.0$									
AMD Athlon 64 X2 4800 2.5GHz, 4GB RAM DDR2 667MHz CL3										
quicksort	115.8	0.9	116.6	1.0	117.4	1.4	117.5	2.0	117.9	2.5
string search	126.7	1.1	125.9	0.5	125.4	0.7	126.0	0.4	125.7	0.7
md5	111.7	0.4	111.4	0.6	111.7	0.5	111.8	0.9	111.6	0.5
edge detection	128.4	0.4	128.4	0.4	128.3	0.4	128.5	0.4	127.8	1.1
matrix transpose	128.6	0.4	128.5	0.5	128.9	0.5	128.7	0.4	129.0	0.5
idle	$\mu = 76.8$, $c_v = 1.6$, $k \approx 1.6$									
hibernation	$\mu = 6.3$, $c_v = 8.7$, $k \approx 19.4$									
AMD Phenom II X4 945 3.00GHz, 8GB RAM DDR2 800MHz CL5										
quicksort	126.8	0.6	127.4	1.0	127.1	0.9	126.8	0.6	127.4	1.2
string search	125.7	0.7	126.3	1.0	126.0	0.5	126.1	0.7	125.6	0.5
md5	127.4	0.4	126.2	0.4	126.2	0.6	126.5	0.8	126.2	0.7
edge detection	131.9	0.5	131.2	0.7	131.2	0.7	129.8	0.8	130.7	0.6
matrix transpose	128.9	0.7	129.6	0.9	128.8	0.6	129.0	0.8	128.9	0.7
idle	$\mu = 73.0$, $c_v = 2.7$, $k \approx 1.8$									
hibernation	$\mu = 6.3$, $c_v = 8.4$, $k \approx 20.2$									

Table 3.2: Power versus problem size and computational intensity I in POLY benchmark. Intel Pentium IV 2.8GHz, 1GB RAM DDR 400MHz CL2.5.

Size α_i	50MB		100MB		200MB		400MB		800MB	
I	μ	c_v	μ	c_v	μ	c_v	μ	c_v	μ	c_v
0	130.8	0.6	131.0	0.3	131.2	0.4	131.4	0.4	132.0	1.4
1	133.4	0.4	133.8	0.9	132.9	0.4	133.2	0.4	133.7	0.8
2	106.8	1.5	106.2	0.4	107.1	1.6	107.1	1.7	106.9	1.4
4	130.4	0.4	130.9	1.0	126.4	0.4	126.1	0.3	126.7	0.9
8	122.7	0.4	122.7	0.5	125.9	1.3	125.6	0.5	126.2	1.1
16	115.6	0.4	115.6	0.5	116.3	0.4	116.5	1.0	116.4	0.4
32	115.6	1.1	116.5	0.4	115.1	0.5	115.5	0.9	115.0	0.5

for at least 1 min. The experiments were repeated 3 times. The results are collected in Tab. 3.1. The power ratings are in Watts. The applications are reported in lines, and different load sizes in columns. Each entry consists of two numbers: average power usage μ in Watts and its coefficient of variation c_v in percents (%). The 'idle' entry gives power usage of the computer switched on, with operating system loaded, and waiting for the code to be executed. The value of k given in this line is the ratio of average power consumption of all active state measurements, and the average idle state power usage. Analogous values are given for the hibernation state. Parameter k will be further used in Chapter 4. In the active state the variation in power usage is of the order of 1%. Thus, it is low. No spikes in power usage were observed. No apparent correlation between problem size α_i and power usage could be found. Only for quicksort there is hardly any correlation observable, but the changes are on the order of noise (represented by the coefficient of variation), and were observed on two of the three computers. The applications involve substantial CPU to core memory communication. Quicksort, string search, matrix transpose are strictly memory-bounded. In order to further verify the impact of CPU-memory communication on power usage we applied POLY benchmark [47] measuring performance vs different computational intensities I. Computational intensity I is the number of floating point operations per memory reference. POLY consists in evaluating polynomials by Horner's rule. For example, a third-degree polynomial `Y[j]=S0+X[j]*(S1+X[j]*(S2+X[j]*S3))` has $I = 3$ because two memory references `Y[j], X[j]` are made per 6 floating-point calculations. Computational intensity I can be adjusted by changing the degree of a polynomial. In Tab. 3.2 we report power usage and its variability for different computational intensities and problem sizes. It can be seen that frequent memory references (smaller I) incur higher power usage. For changing problem sizes α_i power usage remains nearly the same. Similar results were obtained for Strassen matrix multiplication, merge-sort, radix-sort, even if virtual memory was partially used. The lack of the impact of hierarchical memory usage is a result of strong locality of

memory references in these algorithms. Though memory interaction plays an important role in determining power consumption, for computations with fixed resource requirement profile, determined by computational intensity I, power usage is constant. We conclude that there are applications for which power consumption is independent of the size of solved problem α_i. Furthermore, dispersion of used electric power is on the order of 1% and often smaller, which is acceptable for modeling purposes. Thus, linear model of energy vs size of processed data can be considered as confirmed for flat memory systems.

3.3 Hierarchical Memory Model

The time and the energy required for the computations on a load chunk depend on the chunk size α. An important determinant is how big α is compared to the size of the main memory. In order to verify the relationship between computing time, consumed energy, and the size of the load chunk a number of computational tests have been conducted. Example results are collected in Fig. 3.4 and Tab. 3.3. Three example platforms and three applications (image edge detection, quicksort, search for a string in a text block) are reported upon. In Fig. 3.4 dependence of computing time and energy on load size α is shown. The dashed lines represent best linear regression fit into the measured data. The values in Tab. 3.3 have been obtained using linear regression fit to the measurements shown in Fig. 3.4. Note that the vertical axes in Fig. 3.4 are logarithmic, and hence, in this coordinate system the linear functions are not straight lines. It can be verified that both runtime and energy consumption increase significantly faster with problem size when out-of-core memory is used. Interestingly, usually *power* consumption of the out-of-core computations is lower than on-core. However, the speed is by far lower, and hence, the overall *energy* consumption increases much faster with work size α. The point of switching from one dependence to the other is smaller than the hardware RAM size because some memory is reserved by the operating system and runtime environments.

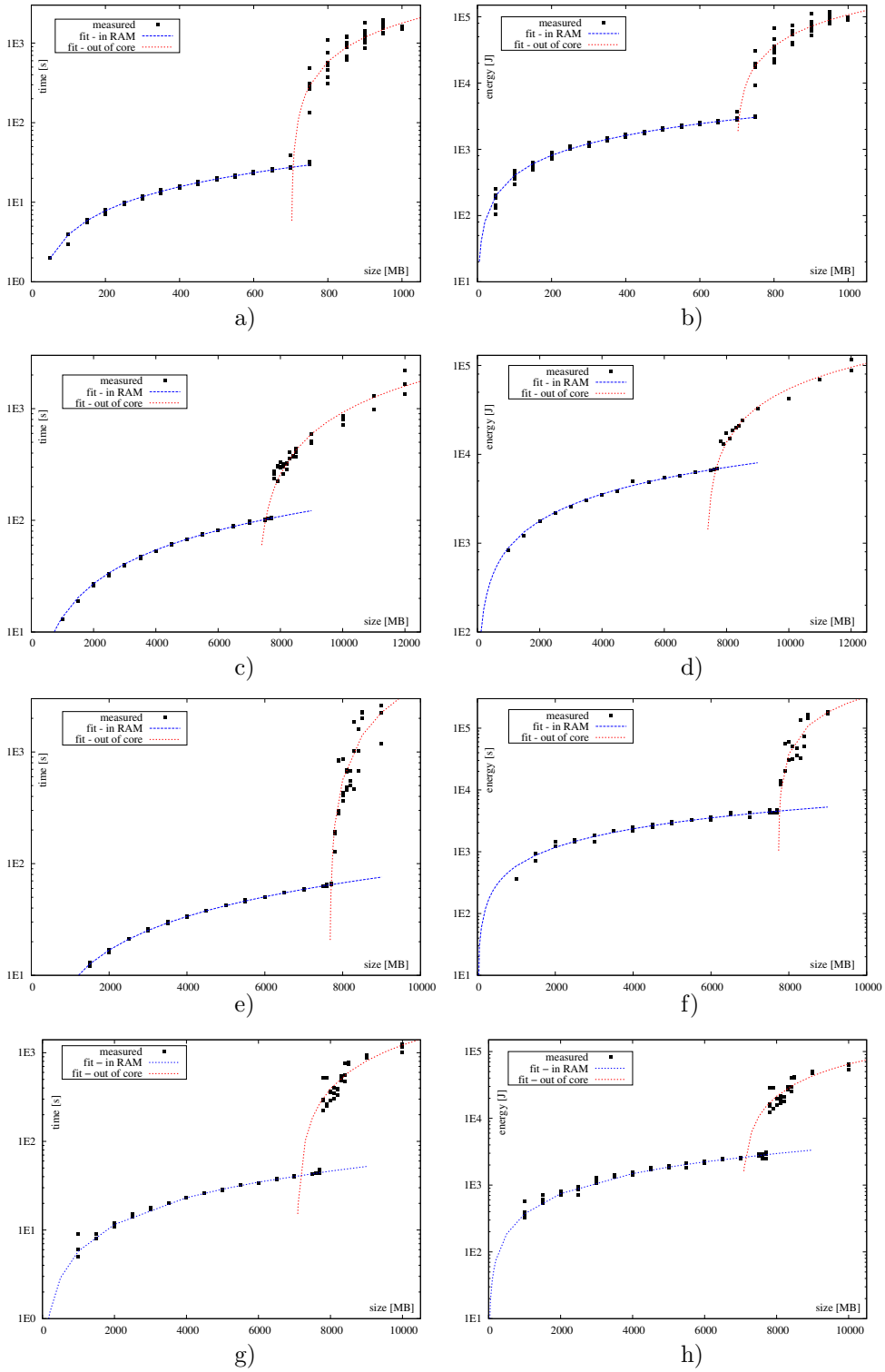


Figure 3.4: Time and energy dependence on load size. Edge detection: a) time, b) energy; quicksort : c) time, d) energy; string search, system 3 in Tab 3.3: e) time, f) energy; string search, system 4 in Tab 3.3: g) time, h) energy, vs problem sizes. Logarithmic vertical axes. Continuous lines are fit using linear regression.

It can be observed that the dependence of computing time and consumed energy on chunk size can be represented by piecewise-linear functions. For example, the time of processing load of size α on machine M_i is

$$\tau_i(\alpha) = \max\{a_{1i}\alpha, a_{2i}\alpha + b_{2i}\}. \tag{3.1}$$

Component $a_{1i}\alpha$ corresponds with computations in core with rate a_{1i} (reciprocal of speed). The second component represents out-of-core computations. Functions τ_i have two properties: $\tau_i(0) = 0$ and $\tau_i(\rho_i) = a_{1i}\rho_i = a_{2i}\rho_i + b_{2i}$, for $i = 1, \ldots, m$, where ρ_i is the size of the main memory on machine M_i available to the application (not necessarily the whole hardware RAM). Beyond ρ_i the machine starts using out-of-core memory. The energy consumed in the computations is determined by an analogous function:

$$\varepsilon_i(\alpha) = \max\{k_{1i}\alpha, k_{2i}\alpha + l_{2i}\} \tag{3.2}$$

satisfying conditions $\varepsilon_i(0) = 0, \varepsilon_i(\rho_i) = k_{1i}\rho_i = k_{2i}\rho_i + l_{2i}$. For memory size ρ_i, both τ_i and ε_i satisfy:

$$\rho_i = b_{2i}/(a_{1i} - a_{2i}) = l_{2i}/(k_{1i} - k_{2i}). \tag{3.3}$$

Let us observe that since out-of-core processing time and energy increase much faster than on-core, we have: $a_{1i} < a_{2i}, b_{2i} < 0$ $k_{1i} < k_{2i}, l_{2i} < 0$, for all machines M_i. Values of these coefficients can be obtained by use of linear regression on intervals of load sizes α as shown in Tab. 3.3. Only b_{2i}s are given in Tab. 3.3, because l_{2i}s can be calculated via the available RAM formula (3.3) presented above: $l_{2i} = b_{2i}(k_{1i} - k_{2i})/(a_{1i} - a_{2i})$. Note that coefficients $a_{1i}, a_{2i}, b_{2i}, k_{1i}, k_{2i}, l_{2i}$ depend both on the machine and the application.

Let us compare our scheduling model with the existing approaches. There are papers, e.g. [55], assuming hard memory limits. It means that instances with $V > \sum_{i=1}^{m} \rho_i$ are infeasible. A border between on-core and out-of-core

Table 3.3: Example time and energy function parameters.

Case: machine, application	a_1 [s/MB]	a_2 [s]	b_2 [s/MB]	k_1 [J/MB]	k_2 [J/MB]
1. Pentium IV@2.8GHz, 1GB RAM@266MHz, HDD Caviar WD400,FreeBSD 9.0, image edge detection	0.0392	5.9943	-4208.2	4.0509	355.2
2. i5@3.2GHz, 8GB RAM@1.6GHz, Ubuntu 14.04LTS, Seagate Sti1000dm003 quicksort	0.0136	0.3331	-2404.8	0.8925	20.46
3. i5@3.2GHz, 8GB RAM@1.6GHz, Ubuntu 14.04LTS, Seagate Sti1000dm003 string search	0.0084	1.9345	-14949	0.5881	143.1
4. AMD A8-7670K@3.6GHz, 8GB RAM@1.6GHz, Ubuntu 14.04LTS Seagate Sti1000dm003, string search	0.0058	0.4151	-2928.1	0.3717	21.77

Table 3.4: Difference between hierarchical memory and proportional cost models.

V	[MB]	10000	12000	14000	16000	18000
$\alpha_1 = \frac{V(a_{12}+C)}{a_{11}+a_{12}+C}$	[MB]	5714	6857	8000	9143	10286
$\alpha_2 = \frac{Va_{11}}{a_{11}+a_{12}+C}$	[MB]	4286	5143	6000	6857	7714
$E_{prop} = (\alpha_1 k_{11} + \alpha_2 k_{12})$	[J]	4000	4800	5600	6400	7200
$E^R = \varepsilon_1(\alpha_1) + \varepsilon_2(\alpha_2)$	[J]	4000	4800	16400	41886	72000

$m = 2$, $a_{11} = a_{12} = 0.006$s/MB, $a_{21} = a_{22} = 0.4$s/MB, $b_{21} = b_{22} = -2955$s, $k_{11} = k_{12} = 0.4$J/MB, $k_{21} = k_{22} = 22$J/MB, $l_{21} = l_{22} = -162000$J/MB, $\rho = 7500$MB, $C = 0.002$s/MB, $S = O = 0, P^I = P^S = 0$.

memory cannot be incontrovertibly inserted in such a model. In our case, by equations (3.1), (3.2), a feasible solution always exists, albeit possibly with bad performance. It is still possible to apply the very basic DLT approach [22] assuming that computing time, and energy as a kind of cost, are proportional to the assigned load size. Unfortunately, in this model energy can be very far from reality in a hierarchical memory system. Tab. 3.4 gives an example of the disparity of these two approaches. In Tab. 3.4 E_{prop} is the energy consumed in the computations according to the proportional cost model, E^R is the same type of energy calculated according to equation (3.2). System parameters and the formulae used to calculate E_{prop}, E^R are given in Tab. 3.4. It can be concluded that the existing approaches cannot be easily adjusted to our situation, or are significantly inconsistent with reality.

4 Isoenergy Maps with Unlimited Memory

In this chapter performance of data-parallel applications is analyzed by use of equal energy maps. As already mentioned, the reason for recursing to relations of equal effectiveness, is that they facilitate building understanding of complex relationships. For example, in [57] the influence of supply voltage (V_{dd}) and threshold voltage (V_{th}) on performance and power consumption of CMOS VLSI chips is studied. In the realm of parallel processing, it has been observed that with growing number of used processors the size of the problem must grow to keep computation efficiency constant [44, 45]. It can be expected that even more complicated relationships exist in the DLT models. Let us note that in this chapter we assume flat memory model of non-restricting size.

4.1 Isoenergy Maps for Amdahl's and Gustafson's Laws

We will introduce here two models of energy consumption related to SMP systems such as multicore CPUs. The models are derived from the laws of parallel processing performance: Amdahl's law [6], and Gustafson's law [46]. Both laws divide a parallel application into two parts: a sequential part, and a perfectly parallelizable parallel part. Let 1 be the total size of the computation, and

$f \leq 1$ the size of the parallel part. We assume a system of m processors (cores) which can be either idle or active. When idle, the cores use k times less electric power than if running in the active state.

According to the Amdlahl's law, the sequential part is executed in time $1-f$, and the parallel part is executed in time f/m on m processors. It was proposed in [95] that during the sequential part only one core is used, while the remaining $m-1$ cores are idle using k times less power. In the parallel part all m cores are active. Hence, the energy used by a parallel application executed on m cores is [95]:

$$E = (1-f)\left(1+\frac{m-1}{k}\right)+\frac{f}{m}m = 1+(m-1)(1-f)/k. \tag{4.1}$$

In the Gustafson's law the parallel part takes time f on m cores, and the sequential part time $1-f$ on one core. Analogously to (4.1) during the sequential part $m-1$ cores are idle and each core consumes k times less power than if active. Hence, the energy used up is:

$$E = (1-f)\left(1+\frac{m-1}{k}\right)+mf. \tag{4.2}$$

Note that energy in (4.1), (4.2) is expressed in relative units, where 1 is the energy consumed by one core executing the whole application.

Both (4.1), and (4.2) allow to represent one of the parameters, as a function of energy E and the remaining parameters. For example, for Amdahl's law from (4.1) k is:

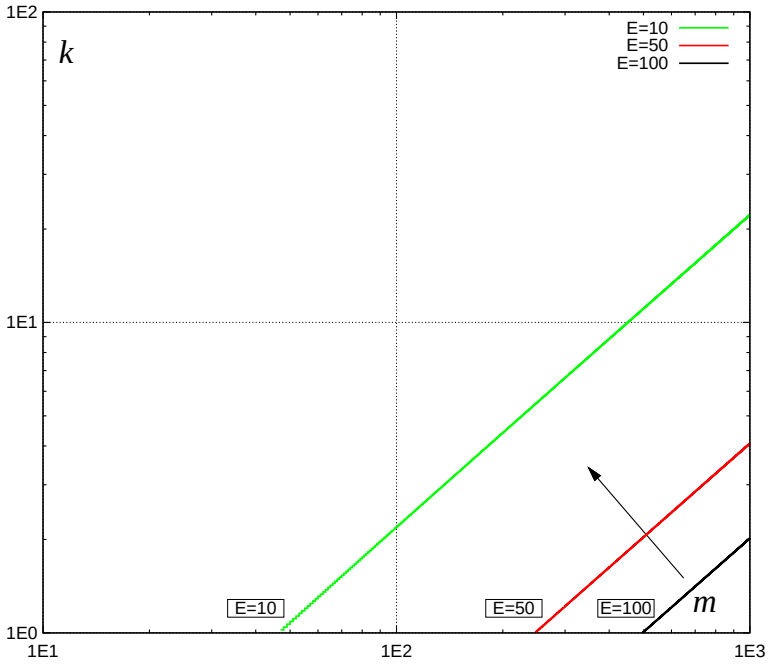
$$k = \frac{(1-f)(m-1)}{E-1} \tag{4.3}$$

and for Gustafson's law, we obtain from (4.2):

$$k = \frac{m-1}{(E-mf)/(1-f)-1}. \tag{4.4}$$

Isoenergy maps (m,k) for $f = 0.8$ are shown in Fig. 4.1a for Amdahl's law (eq. (4.3)), and in Fig. 4.1b for Gustafson's law (eq. (4.4)). Increasing processor number m is the main way of reducing the computation time in parallel

a)



b)

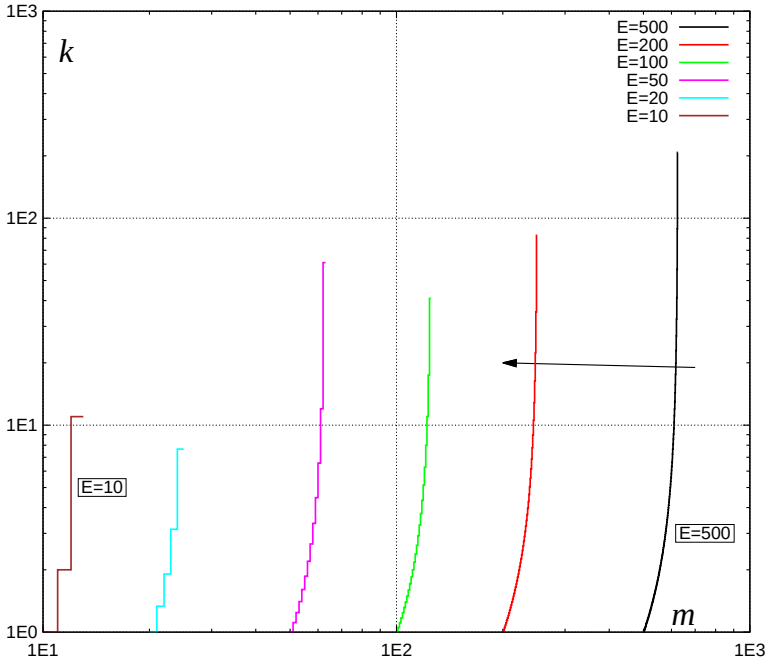
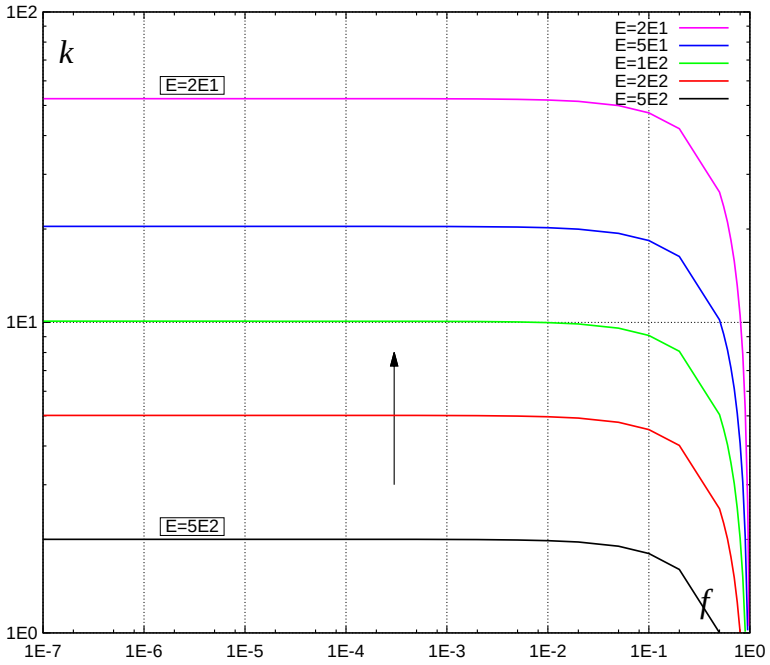


Figure 4.1: Isoenergy map for processor number m, and idle power reduction factor k. a) Amdahl's law, b) Gustafson's law. Arrows show direction of decreasing energy E.

processing. Hence, map (m, k) demonstrates if energy consumption incurred by increasing parallelism can be compensated by effective application of low-energy modes embodied by k. In Fig. 4.1a (Amdahl's law) coefficient k cutting power usage grows nearly linearly with m. Moreover, k must increase with decreasing energy E. In Fig. 4.1b (Gustafson's law) k grows rapidly with m, and not for all m a feasible k exists. Note that m is a discrete parameter. Therefore, the lines have step-wise form, and the isolines of k finish in certain point rather than approaching a vertical asymptote. Since the isolines are roughly parallel to k axis, the dependence of energy use on k is weak, and strong on m. The apparent difference of the isoline shapes can be explained as follows. In Amdahl's law (4.1) the energy cost of parallel processing is constant because $\frac{f}{m}m$ reduces to f. Energy savings can be done only in the sequential part of the application. Since the sequential part is not dominating in well designed parallel applications, the energy saving coefficient k has any influence only if $(m-1)(1-f)$ is very big. On the contrary, in equation (4.2) parallel processing costs mf in energy. Since f is big in a well designed parallel application, it is hard to compensate energy use by aggressively clamping power down in the sequential part. Application of the isoenergy maps in Fig. 4.1 in a practical scenario could look as follows: A parallel application programmer, or a user, wants to use more processors m, but keep energy constant. Thus, it is necessary to reduce energy use in the idle state, i.e. increase parameter k. It can be achieved by modifying CPU and/or memory, designing a subsystem in the operating system such that the application can request CPU and memory suspension, incorporating in the operating system algorithms discovering idle cores, better determining sequential parts in the application at the development stage. The two pictures in Fig. 4.1 differ in assessing scalability of this approach. In Fig. 4.1a it is linearly scalable. In Fig. 4.1b increasing k gives only diminishing returns.

Fig. 4.2 depicts isoenergy map of k, and the size of the parallel part f, at $m = 1000$. Let us note that for $m = 1000$ energy levels $E \geq 1000$ do not represent any savings because the energy used by $m = 1000$ processors in the schedule

a)



b)

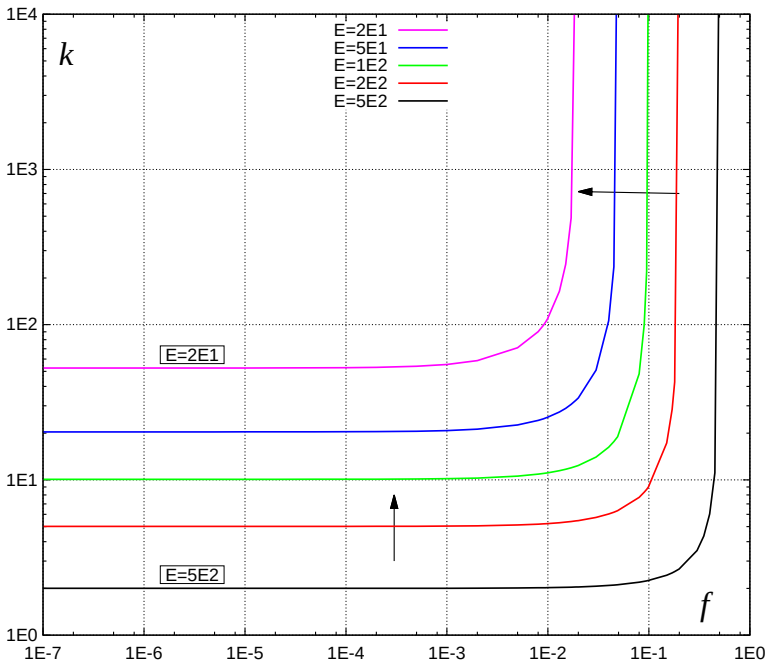


Figure 4.2: Isoenergy map for parallel fraction f, and idle power reduction factor k, for $m = 1000$. a) Amdahl's law, b) Gustafson's law. Arrows show direction of decreasing energy E.

of the length equal to the sequential one is just $E = 1000$. Hence, only $E < 1000$ are shown in Fig. 4.2. The map in Fig. 4.2 shows how low-energy modes (represented by k) interact with the potential application parallelism (represented by f). The isolines are quite similar when f is very small, which is rare in well designed parallel applications. When f is close to 1, the isolines become radically different. In Fig. 4.2a (Amdahl's law) the power reduction coefficient k decreases which means that it is becoming less and less needed in energy saving because the sequential part in the parallel application is disappearing and the parallel part consumes constant energy f. On contrary, in Fig. 4.2b (Gustafson's law) k must rapidly grow to compensate increasing contribution of the energy consumed by the cores working in parallel. It follows, that compensating energy consumption incurred by parallel processing of computationally intensive applications (represented by high f) using processor suspension modes is ultimately futile. It is an indication for a decision-maker that for such applications energy optimization must be achieved in other ways. Fig. 4.1 and Fig. 4.2 show that Amdahl's and Gustafson's laws represent essentially different perceptions of parallel application performance. In the Amdahl's law parallel processing in the energetic sense is essentially for free. In the Gustafson's law the size of the problem grows with the number of cores m, and hence, parallelism incurs costs. It can be concluded that from the energy point of view Amdahl's law misses an important component of the cost of parallel computation.

4.2 Isoenergy Maps for Divisible Computations

The models introduced in the previous section lack such important details as costs of communication and starting additional processors. In this section an energy use model addressing such deficiencies is introduced. It represents processing divisible loads. Optimum schedules for divisible computations provide timings necessary to calculate energy consumption. Though the schedules are

constructed in the standard way for DLT [22, 26, 28, 29, 74] we report necessary details to make the presentation self-contained. Given the timings of the schedule we can multiply it by electric power to derive formulae for energy consumption. In this step we partially use results from [29]. Empirical evidence supporting our electric power use model was provided in Chapter 3.

4.2.1 The Energy Usage Model

We assume that at the beginning of the computation load of size V resides at originator (initiator, file server) P_0. The originator is in a star interconnection with homogeneous worker computers (processors) $P_1, \ldots, P_m$. The star interconnection may represent multiple CPUs sharing a bus, a cluster of workstations, or a set of machines in a global grid. The sole role of the originator is to control the computation, and distribute the load. Communications are performed between the originator and the worker processors only (cf. Fig. 4.3). Volume V of load is partitioned into chunks of size $\alpha_1, \ldots, \alpha_m$, and sent to $P_1, \ldots, P_m$, respectively. Starting computation on a remote computer involves waking it up, loading virtual machines, application code and its libraries. Hence, startup time S elapses before P_i $(i = 1, \ldots, m)$ can start receiving the load. Startup time S is an important element in modeling DLT applications. Without the startup, arbitrary number of processors may be activated which is unrealistic [22, 28]. Load of size α_i is transmitted in time $\alpha_i C$. On the receipt of the load, P_i immediately starts computations which last $\alpha_i a$ units of time. When the communication with P_i is finished, the originator activates processor P_{i+1} and sends to it load α_{i+1}. The procedure is repeated until starting computations on all m processors. Now the goal is to partition load V into chunks $\alpha_1, \ldots, \alpha_m$ such that schedule is as short as possible. Let us assume that the time of returning results is negligible. It can be shown [22, 26, 74] that for the optimality of the schedule length all the processors must finish computations simultaneously. Though we assumed that the result collection time is negligible and that communication with the worker processors is executed only once, the result collection

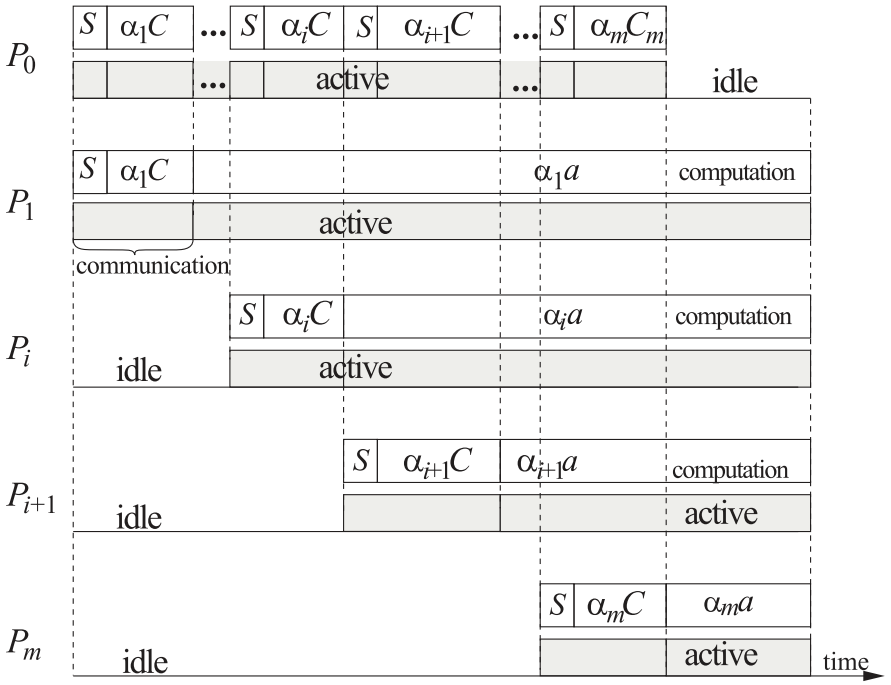


Figure 4.3: Communication, computation and power use schedule.

time, parallel communication, concurrent communication and computation can be comprised in DLT models when necessary [22, 26, 28, 29, 74]. Most of these options for processing the load will be subject of Chapter 5. We assume that load chunks α_i fit in the core memory such that parameters a, C are not affected by the load sizes. Since all processors finish computations simultaneously (see Fig. 4.3) processor P_i $(i = 1, \ldots, m-1)$ computes as long as it takes to activate P_{i+1}, send and process its load α_{i+1}. Moreover, all load must be processed. Hence, we have a system of linear equations:

$$a\alpha_i = S + (C + a)\alpha_{i+1} \qquad \text{for} \quad i = 1, \ldots, m-1 \tag{4.5}$$

$$\sum_{i=1}^{m} \alpha_i = V. \tag{4.6}$$

Denoting $\kappa = 1 + C/a$ it can be derived from (4.5):

$$\alpha_i = \frac{S(1-\kappa^{m-i})}{a(1-\kappa)} + \kappa^{m-i}\alpha_m \quad \text{for } i = 1, \ldots m, \tag{4.7}$$

where α_m from (4.6), and (4.7) is

$$\alpha_m = \frac{V(1-\kappa)}{1-\kappa^m} - \frac{S(m(1-\kappa)-1+\kappa^m)}{a(1-\kappa^m)(1-\kappa)}. \tag{4.8}$$

Note that only $\alpha_m \geq 0$ has practical sense, yet negative values can be obtained from equation (4.8). Such a situation may arise if the volume of load V is too small to employ all m processors. In the further discussion the combinations of parameters a, C, m, S, V such that $\alpha_m < 0$ will be called *infeasible*. From α_1 schedule length T can be calculated:

$$T = S + (C+a)\alpha_1 = Sz_1 + Vz_2 \tag{4.9}$$

where from (4.7), (4.8)

$$z_1 = 1 + \frac{\kappa}{1-\kappa}\left[1 - \kappa^{m-1} - \frac{\kappa^{m-1}(m(1-\kappa)-1+\kappa^m)}{1-\kappa^m}\right] \tag{4.10}$$

$$z_2 = (C+a)\frac{(1-\kappa)\kappa^{m-1}}{1-\kappa^m}. \tag{4.11}$$

Practical viability of the above model of distributed computation has been confirmed, e.g., in publications [2, 34, 56]. Parameters a, C, S are platform- and application-dependent and can be measured as shown in Chapter 3.

The energy consumption can be split into three components: E^I – the energy consumed in the idle state, E^{RN} – energy beyond the idle state consumed in communication, and E^{RC} – the energy beyond the idle state consumed in computation. Hence, the total energy consumed by the system is

$$E = E^I + E^{RN} + E^{RC}. \tag{4.12}$$

Let P^C denote the power consumed by the active processors, and P^N the power consumed in the network equipment when communicating. We assume, that the energy costs brought by components of a computer, such as CPU, RAM, HDD, NIC, power supply unit, fans, cooling equipment are specific for the platform and the application, and are all comprised in P^C. This value is constant in the performed computation, as shown in Section 3.2. A more detailed model of energy consumption in computers will be used in Chapter 6. We assume that in the idle state both the network, and the computers use k times less energy, than in the active state. Parameter k represents in a synthetic way the degree of proportionality in energy use. For example, recently introduced FVER index of datacenter energy performance [67] is equal to $1 + 1/(k-1) = k/(k-1)$. If the system were idle all the schedule length T, the energy consumed would be:

$$E^I = T((m+1)P^C + P^N)/k. \tag{4.13}$$

The network is active when performing communications, after finishing load distribution it switches back to the idle state. Hence, energy E^{RN} consumed in the network beyond the idle state energy is:

$$E^{RN} = P^N \frac{k-1}{k}(mS + CV). \tag{4.14}$$

Processor P_i, consumes energy $P^C \frac{k-1}{k}(S + \alpha_i(C+a))$ above the idle state. The energy consumed by all processors beyond the idle state by (4.6) is:

$$\begin{aligned} E^{RC} &= P^C \frac{k-1}{k}\left(mS + CV + \sum_{i=1}^{m}(S + (C+a)\alpha_i)\right) \\ &= P^C \frac{k-1}{k}(2mS + 2CV + aV). \end{aligned} \tag{4.15}$$

Thus, from (4.12)-(4.15) we obtain:

$$E = T((m+1)P^C + P^N)/k + P^N\frac{k-1}{k}(mS + CV) + P^C\frac{k-1}{k}(2mS + 2CV + aV). \tag{4.16}$$

The energy use model presented above is fairly generic and can be extended to accommodate more details. This will be subject, at least to some extent, of the following chapters.

4.2.2 Isoenergy Lines Calculations

Now we proceed to the method of plotting the isoenergy lines. In some cases it is possible to derive analytically value of one parameter as a *function* of all other parameters and the energy. For example, it is possible to derive k from (4.16) to obtain equation (4.17). Then all two-dimensional isoenergy maps involving k can be obtained by sweeping a range of some parameter X and calculating k for the given X, at certain energy level E and other parameters fixed. Below we list parameters obtained analytically as functions.

$$k = \frac{T[(m+1)P^C + P^N] - (P^N + 2P^C)(mS + CV) - P^C aV}{E - (P^N + 2P^C)(mS + CV) - P^C aV} \tag{4.17}$$

$$P^C = \frac{Ek - P^N[(k-1)(mS + CV) + T]}{(m+1)T + (k-1)(2mS + (2C + a)V)} \tag{4.18}$$

$$P^N = \frac{Ek - P^C[(m+1)T + (k-1)(2mS + (2C + a)V)]}{T + (k-1)(mS + CV)} \tag{4.19}$$

$$V = \frac{Ek - Sz_1[(m+1)P^C + P^N] - (k-1)mS(P^N + 2P^C)}{z_2[(m+1)P^C + P^N] + (k-1)(P^N C + (2C + a)P^C)} \tag{4.20}$$

$$S = \frac{Ek - Vz_2[(m+1)P^C + P^N] - (k-1)V[P^N C + P^C(2C + a)]}{z_1[(m+1)P^C + P^N] + m(k-1)[P^N + 2P^C]} \tag{4.21}$$

where T, z_1, z_2 were given in equations (4.9), (4.10), (4.11).

Unfortunately, for parameters a, C, m no explicit function representation is known. Furthermore m is discrete. In these cases the isoenergy line can be found numerically for a given energy level E. Let (X, Y) denote an isoenergy map for parameters X and Y. For example, in the isoenergy map (a, C) let parameter a be an independent variable. We set certain value of a, while the remaining system parameters are fixed. Using equation (4.16) the value of C at which energy is E can be found numerically, e.g., by binary search over C. This procedure is repeated for a range of a values. The isoenergy maps involving any pair of parameters a, C, m were constructed numerically.

4.3 Isoenergy Maps Examples

In this section we give examples of typical isoenergy map forms for divisible computations. Let us first make some general observations. Note that some amount of energy inevitably must be used. For example, at least energy $P^C aV$ must be consumed in the computations. Furthermore, some parameter combinations may be infeasible, as observed in Section 4.2. Consequently, some isoenergy lines (e.g. for $E < P^C Va$) or some parts of the line may be inaccessible.

Since our model has 8 parameters, there are $\binom{8}{2} = 28$ two-dimensional isoenergy maps. Due to limited space only a subset will be presented. A bigger collection of isoenergy maps can be found at [31]. Rather than studying obvious candidates for energy saving like a, V, P^C, we analyze the less obvious ones and the relation between the cost of communication and computation. Unless stated otherwise, the isoenergy maps were constructed for reference parameters: $m = 1000, a =$ 1E-3, $C =$1E-8, $S = 100, V =$1E11, $k = 3, P^C = 200, P^N = 50$. The above values can be interpreted as follows. There are $m = 1000$ processors. A load unit is processed in 1 ms ($a =$ 1E-3), and communicated in 10ns ($C =$1E-8). If the load units were 10 bytes, then processing speed of a processor would be 10kB/s, communication speed 1GB/s, and the size of load would be 1TB. A processor uses $P^C = 200$W, and the network P^N=50W. In the idle state power

usage is $k = 3$ times smaller. In such a configuration the load is processed in $T \approx$ 1.51E5s using $E \approx$ 2.34E10J (6500kWh). In the isoenergy maps we explore wide ranges of the parameters. For example, values up to 1E4W for P^C, or 1E9W for P^N are shown in Fig. 4.5. It may be disputable if such values make sense, especially if we assume a single chip point of view. The rationale for such wide range of values are the following: A processor in our model might range from a single shader in a graphics card to an entire datacenter, hence wide range of P^C. If we intend to analyze a datacenter with many processors then P^N may be indeed big [4, 38]. Our principal objective is to uncover *shapes* of the isoenergy lines to observe mutual interactions between the platform and application parameters. If we discover some phenomenon in the extreme range of parameters, we can safely exclude such phenomena in practice. Hence, the investigated parameter ranges should not preclude exposing the shapes of the isoenergy lines.

4.3.1 Problem Size vs Communication Rate

The map (V, C) for the reference parameters is shown in Fig. 4.4. The configurations in the upper-left corner, where communication is slow and problem size small, are infeasible as explained in Section 4.2. With the progress in high performance computing sizes V of the solved problems inevitably grow. Fig. 4.4 determines whether the energy consumption induced by growing problem size can be compensated by faster communication. Map (V, C) shows that energy consumption grows mainly with V. Only for C greater than certain threshold does the energy use depends also on C. The threshold value of C is determined by the relation between the energy consumed in the computation VaP^C and the energy used up by processors waiting in the idle state to start computations, which is roughly $VC(P^C m + P^N)/k$. Hence, for $P^C m \gg P^N$ and C greater than approx. ak/m =3E-6 can the two parameters compensate each other. It means that communication must be faster to keep with growing V and maintain constant energy use. Alternatively, it can be said that for $C > ak/m$ communi-

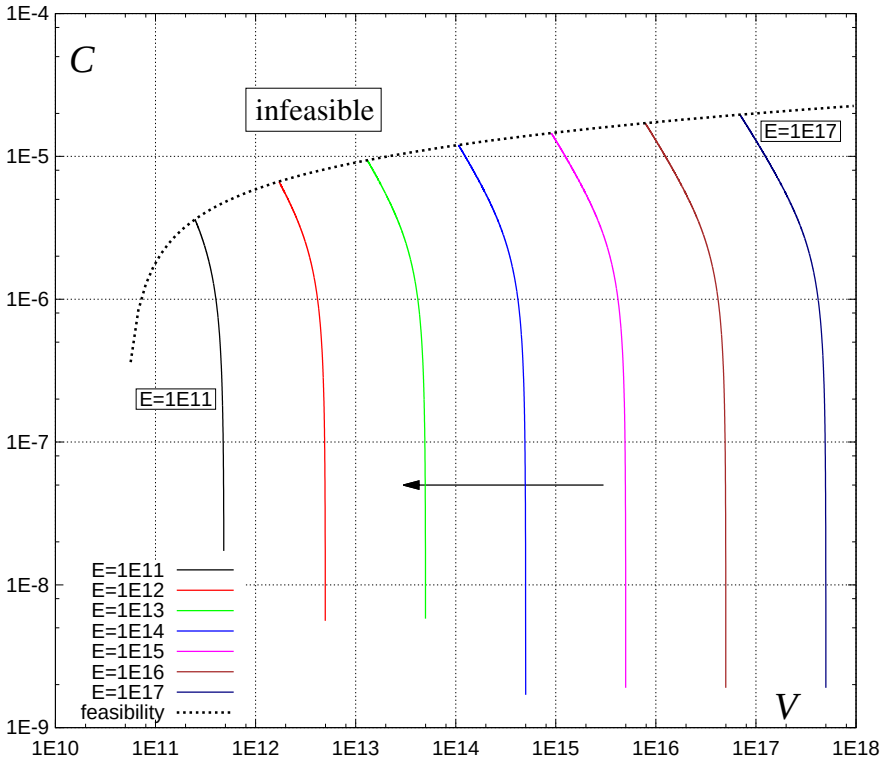


Figure 4.4: Isoenergy map for load size V, and communication rate C. The arrow shows direction of decreasing energy E.

cation is too slow and incurs unnecessary energy cost by holding the processors idle. Application of isoenergy map (V, C) in a real-world scenario could look like this: It is planned to process increasing problem sizes V using the existing application and datacenter. What should be done to prepare for such change and curb energy use? The energy consumption will inevitably increase with V. The main component is computation, product VaP^C must be minimized. This can be achieved by better algorithms and programming (a), better hardware (P^C). Further changes depend on the interaction between a, C, m, k, P^C, P^N. If $P^C m \gg P^N$ and $C < ak/m$ no changes in networking subsystem are needed. For the example application parameters introduced at the beginning of this section $P^C m \gg P^N$ condition holds, C=1E-8< ak/m = 3E-6 and changes in the communication subsystem are ineffective. Assume however, that by more advanced algorithm and better programming speed of the application increases ten-fold, i.e. a is reduced to a=1E-4. Then, at roughly m = 3E4 processors it will be necessary to increase communication speed $(1/C)$, roughly proportionally to the increase of V, to curb energy costs.

The shape of the isolines on map (V, P^N) is quite similar, and for very big values of P^N, the network power influences energy usage on the scale comparable with V. The isolines on map (V, S) have similar shape as in Fig. 4.4, but the lines are parallel to S axis in nearly whole feasible range of parameters. Hence, reducing startup time S is in general insufficient to compensate energy consumption incurred by growing problem size V.

4.3.2 Processor Power vs Network Power

The isoenergy map (P^C, P^N) is shown in Fig. 4.5. Processor power P^C and the network power P^N are two main parameters determining overall energy consumption. This map shows whether growing network power P^N can be compensated by reducing processor power P^C, or vice versa. The isoenergy lines have knee-like shape with energy decreasing toward lower-left corner, i.e. when P^N and P^C are decreasing. In the upper part of the map the isoenergy line is

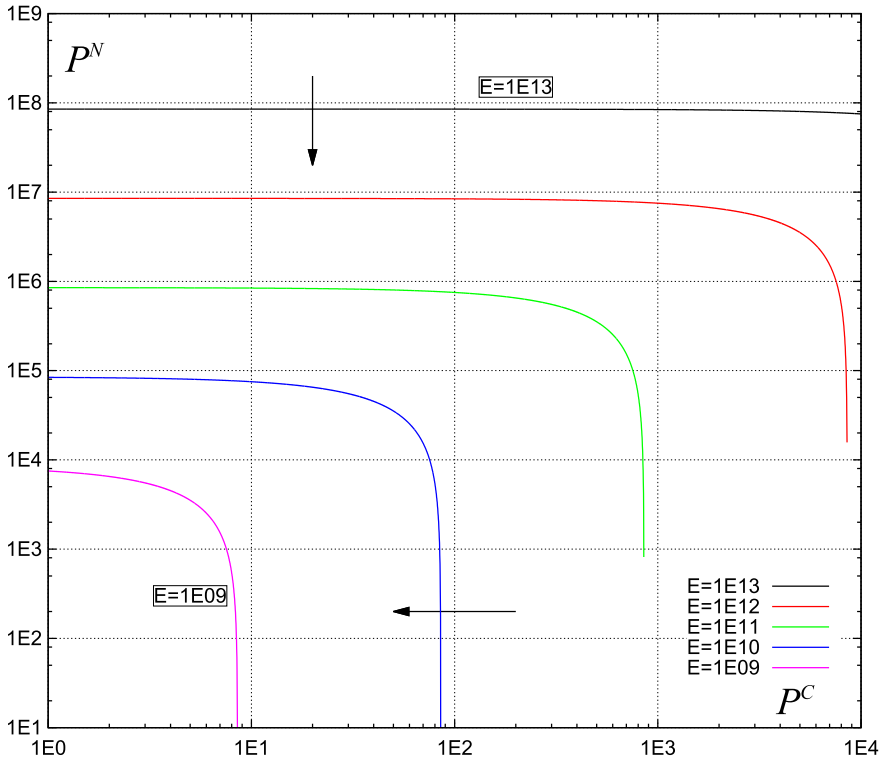


Figure 4.5: Isoenergy map for network power P^N, and processor power P^C. Arrows show direction of decreasing energy E.

parallel to P^C axis, and P^N is determining energy consumption. The energy use is dominated here by component TP^N/k from E^I (see equation (4.13)). On the lower part (parallel to P^N axis) energy consumption is determined by the energy aVP^C needed to compute the whole load. The two parameters can compensate each other only in a narrow range of values. The shapes of the isolines on (P^C, P^N) map remain the same for wide range of system parameters [31]. In the areas where either P^C, or P^N play role, their reductions correspond with roughly proportional reductions in energy consumption. A real world use of map (P^C, P^N) could be as follows. A system designer reduces computer power P^C, by using better hardware: CPUs, memory, power supplies and cooling, but ignores energy in the communication subsystem. Isoenergy map (P^C, P^N) demonstrates that it will ultimately lead to exposing energy used in networking, and then P^N will have to be reduced, too. Though P^C, P^N hardly ever can compensate for one another, they have to be minimized in unison because minimizing one exposes the second power type.

4.3.3 Processor Number vs Network Power

Isoenergy map (m, P^N) is shown in Fig. 4.6. Users increase processor numbers m to take advantage of concurrency, and reduce the computation time. Thus, map (m, P^N) shows whether energy costs resulting from greater processor number m can be compensated by reducing network power P^N. The isolines have step-wise form because processor numbers m are discrete. For the assumed parameters, configurations with more than $m = 1411$ are infeasible. It can be seen that for big load sizes and sufficient energy budget (Fig. 4.6 upper part), P^N on the isoenergy line increases roughly linearly with m. This means that the energy use decreases with decreasing P^N and increasing m. The former could be intuitively expected, the latter can be explained as follows. Load size V is sufficient to effectively exploit many processors, and computations scale well with m. With each new processor schedule gets shorter which spares energy because load chunks are smaller, computations start earlier, and we pay less for keep-

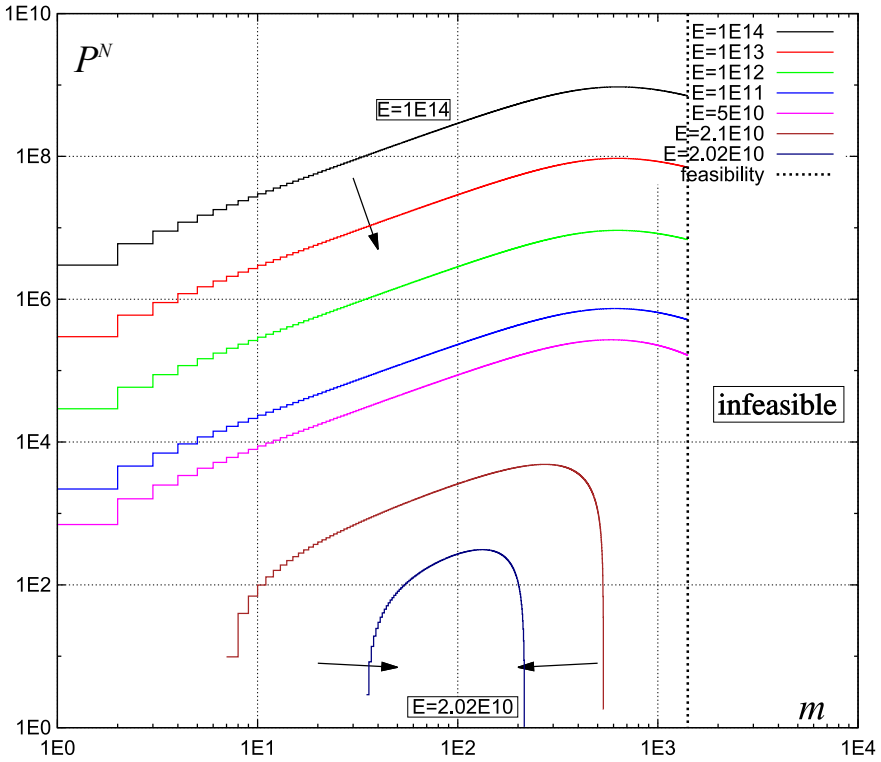


Figure 4.6: Isoenergy map for processor number m, and network power P^N. Arrows show direction of decreasing energy E.

ing idle (waiting) processors. Hence, certain inefficiencies in the communication can be compensated by concurrency of computations. This applies to quite big values of P^N and E, or small m. However, even the topmost lines in Fig. 4.6 bent down at the right ends, showing that this way of energy savings is limited. On the other hand, when the energy budget is tighter (Fig. 4.6 the lower part) the situation changes radically. By a tight energy budget we mean here that the energy level is at the minimum attainable by changing the two parameters m, P^N only. Both ends of the isoline are almost parallel to axis P^N, and energy use has a minimum for some small m. This shape emerges in the following way. On the left end certain isolines are inaccessible because other components incur high energy consumption which cannot be compensated by P^N alone. On the right end increasing processor number brings higher activation costs which hit against tight energy budget, and again P^N alone becomes insufficient to keep energy use constant. On the one hand, it is hard to expect that users will resign from increasing processor numbers m since it is an essential idea of parallel processing. On the other hand, there is a limit to energy savings by reducing P^N when m is growing. Consequently, reducing the overall energy consumption by limiting P^N does not scale well with m.

4.3.4 Processor Number vs Startup Time

Isoenergy map for processor number m and startup time S is shown in Fig. 4.7. The upper-right corner of the map contains infeasible configurations. Map (m, S) shows how startup times S should change with growing processor number m to avoid using additional energy, and to what extent such compensation is possible. Observe that for a fixed S the energy E has minimum at certain m. The shape of the isoenergy lines can be explained as follows. When the number of processors is small, the contribution of the startup to the overall energy consumption is also small, and S alone cannot compensate for energy usage incurred by other parameters. Hence, some isolines cannot be seen for arbitrarily small m. When the number of processors increases two factors come

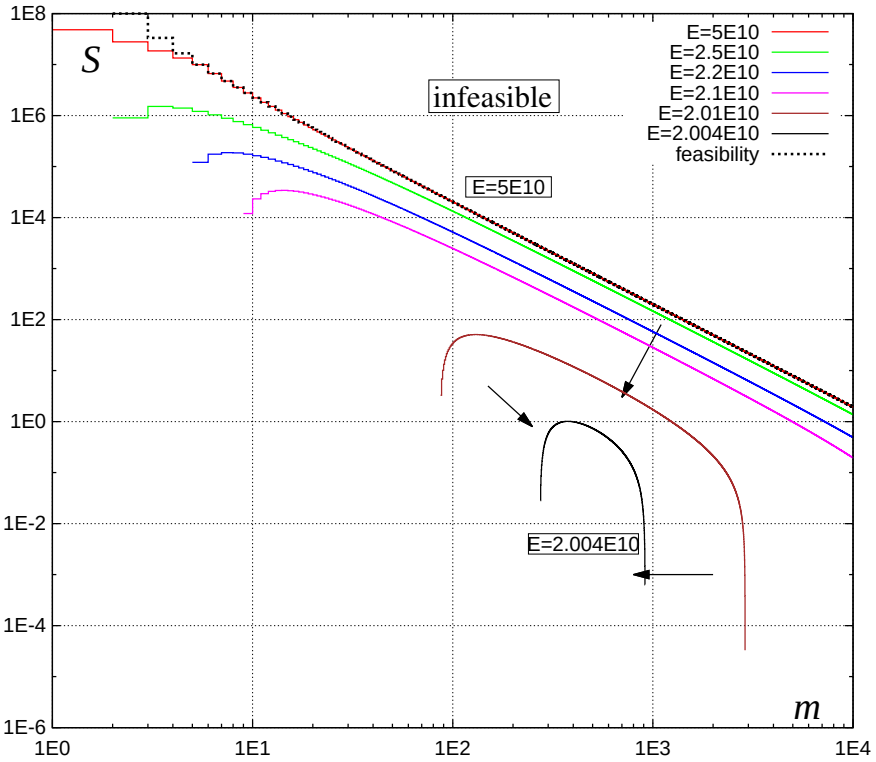


Figure 4.7: Isoenergy map for processor number m, and startup time S. Arrows show direction of decreasing energy E.

into play: Schedule length decreases, and the cost of processor activations increases. Initially, adding a few processors reduces schedule length. This spares some energy because the system is used for a shorter time. These savings can be "wasted" in longer startups. Therefore, the isoenergy lines are bent down for small m. If the processor number increases even further the reduction of the schedule length is insufficient to compensate the energy needed for the startups, and S must decrease to keep energy constant. The form of the right end of the isoline depends on the energy budget. Again, a tight budget means here that the energy level approaches minimum achievable by changing the two parameters m, S. If energy budget is tight then the relative contribution of startups quickly increases with m. Consequently, startup times must radically decrease which ultimately becomes unattainable. Thus, in the areas where the isolines are parallel to axis S, no further reduction of S will provide any energy savings. This signifies lack of scalability of energy optimizations based on S only. Note that energy range is narrower in Fig. 4.7 than in the earlier figures. It is approximately 60% of the highest isoline. Moreover, values of S in Fig. 4.7 are in range from moderate to exceptionally big. This means that energy savings here are shallower than by changing values P^C, P^N and product aV discussed earlier.

4.3.5 Processor Number vs Power Reduction Factor

In Fig. 4.8 isoenergy map (m, k) is shown. The shapes of the isolines can be explained as follows. For small m schedule length decreases with each new processor. This reduces the idle part of energy (E^I in equation (4.13)), and a bit more energy can be "wasted" by decreasing k. When the effect of shortening schedule length T with growing m slows down, and still energy use increases with each added processor, then constant energy use can be maintained by aggressively cutting of power from the idle equipment, and k ascends with m. Consequently, for fixed k energy use has a minimum at certain processor number m. The increase of k is particularly sharp for tight energy budgets. In effect,

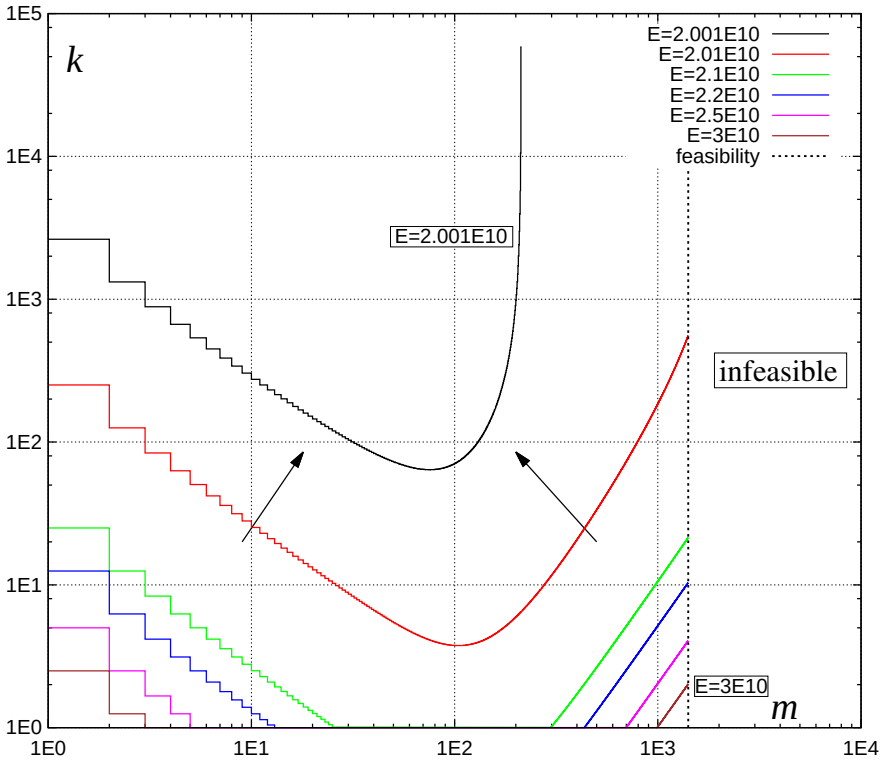


Figure 4.8: Isoenergy map for processor number m, and idle power reduction factor k. Arrows show direction of decreasing energy E.

using low power modes is inefficient when m is big. A similar conclusion was drawn for (m, k) and Gustafson's speedup law (cf. Fig. 4.1b). The lowest isoenergy line energy level is more than 67% of the highest one. Thus, savings by cutting off power from the idle equipment may have smaller effect than reducing P^C, P^N, aV.

4.3.6 Processor Number vs Communication Rate

Isoenergy map for processor number m and communication rate C is shown in Fig. 4.9. Map (m, C) shows how communication rates C should change with growing processor number m to avoid using additional energy, and to what extent such compensation is possible. Note that the isolines have step-wise form because processor numbers m are discrete. The top-right corner of the chart contains infeasible configurations. As observed in Section 4.2.1 it is not possible to run computations on large number of processors m when communication speed is very low (i.e C is large) because the whole load can be processed on a subset of processors. Closer examination of formula (4.8) reveals that in the numerator of the second part of the sum term $1-\kappa^m$ becomes dominant and with m tending to infinity the second part of the sum tends to S/C. Consequently, component $V(1-\kappa)/(1-\kappa^m) \approx V(C/a)^{1-m}$ is the biggest in (4.8). Thus, with growing m, communication rate C must quickly decrease to guarantee feasibility of the load partitioning. It can be observed that in general in the isolines C decreases with m which is intuitively expected because adding each new machine increases activation energy costs which must be compensated by shorter communication. Only in a narrow interval of small processor number and low energy budgets are the isolines are lightly concave. It is because some initial increase in the processor number reduces schedule length and hence also idle waiting which allows for slightly slower communication (C can grow a bit). Note that communication rate values are extremely large in Fig. 4.9 for small m. This means that communication rate C – processor number m interaction plays important role only if processor numbers are large.

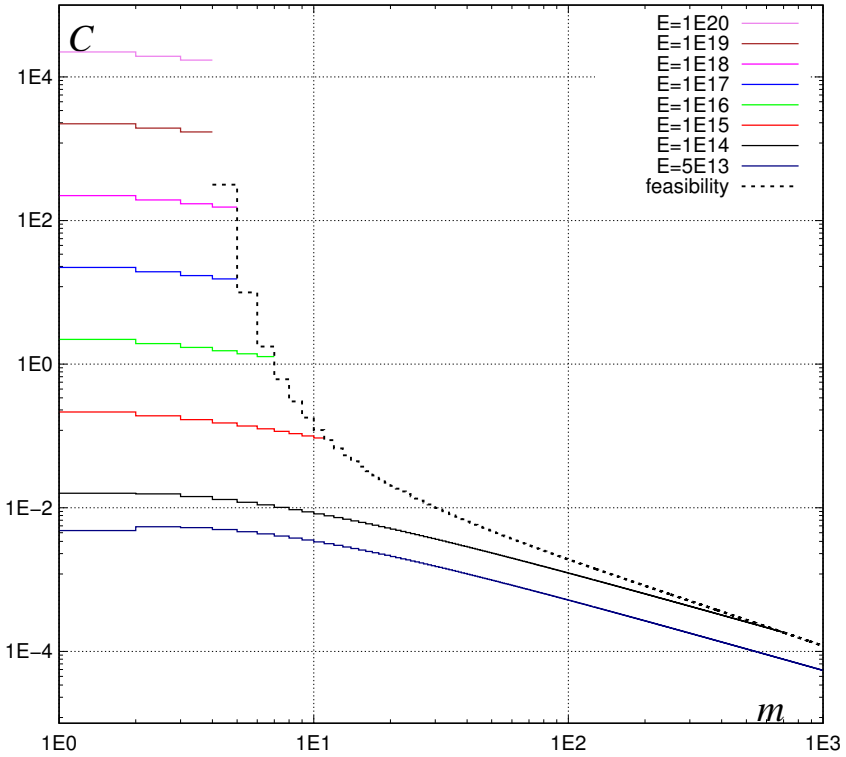


Figure 4.9: Isoenergy map for processor number m, and communication rate C for a = 1E-2.

4.3.7 Processor Number vs Processor Power

Isoenergy map (m, P^C) is shown in Fig. 4.10. It can be observed that for large energy budgets, P^C can be large and since the energy, firstly, depends on the computation time Va, and secondly, it dominates the whole energy consumption, the isolines are nearly parallel to m axis. This means that there is no compensation between m and P^C in this area. The situation is different for tight energy budgets. On the one hand, some energy will be consumed anyway, e.g. $P^N CV$ in distributing the load. This cannot be compensated by reducing P^C or m. Hence, the isolines for small energy budgets turn down (toward small P^C) and end abruptly. On the other hand, increasing m requires energy in activating new machines. This cost can be compensated by reducing P^C. Consequently, also for large m the isolines turn down. It means that with respect to energy costs and for tight energy budgets, there is an optimum number of processors which balances energy costs and gains of parallelism.

4.3.8 Processor Power vs Load Size

In Fig. 4.11 isoenergy map for processor power P^C and load size V is shown. Infeasible solutions are located at the bottom of the isoenergy map where problem sizes are too small to run computations on all processors. It can be seen that with growing processor power load sizes have to decrease to keep energy at constant level. This is an expected behavior because energy VaP^C consumed in the computations has to be kept constant. The isolines are very regularly spaced in the map and almost parallel to each other. The (P^C, V) isoenergy map confirms a quite natural conclusion that bigger problems need more efficient processors to keep the same energy levels.

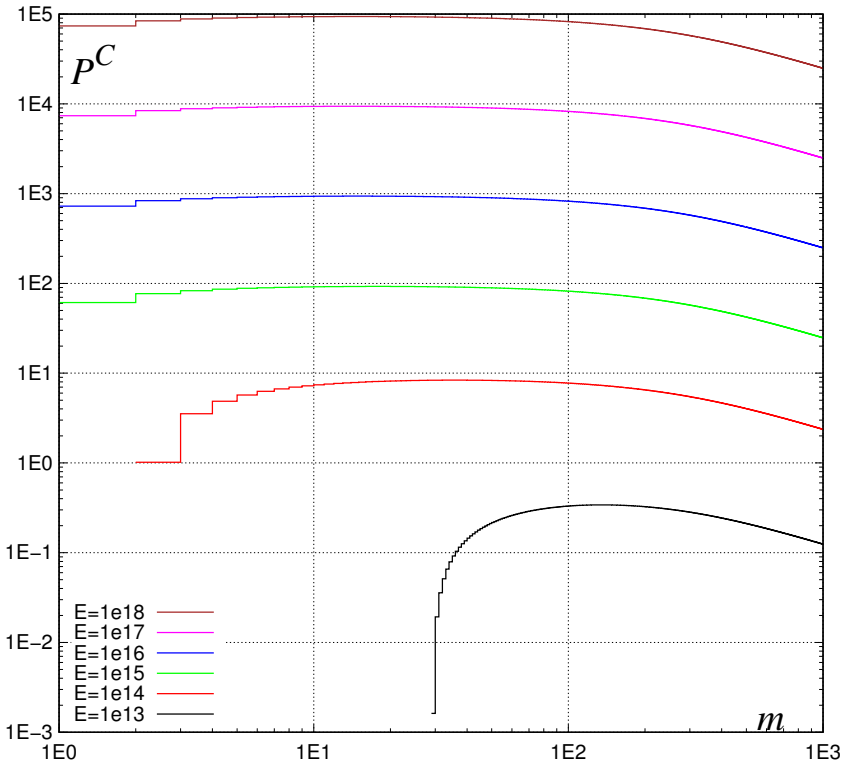


Figure 4.10: Isoenergy map for processor number m, and processor power P^C for $C = 1\text{E-}2$.

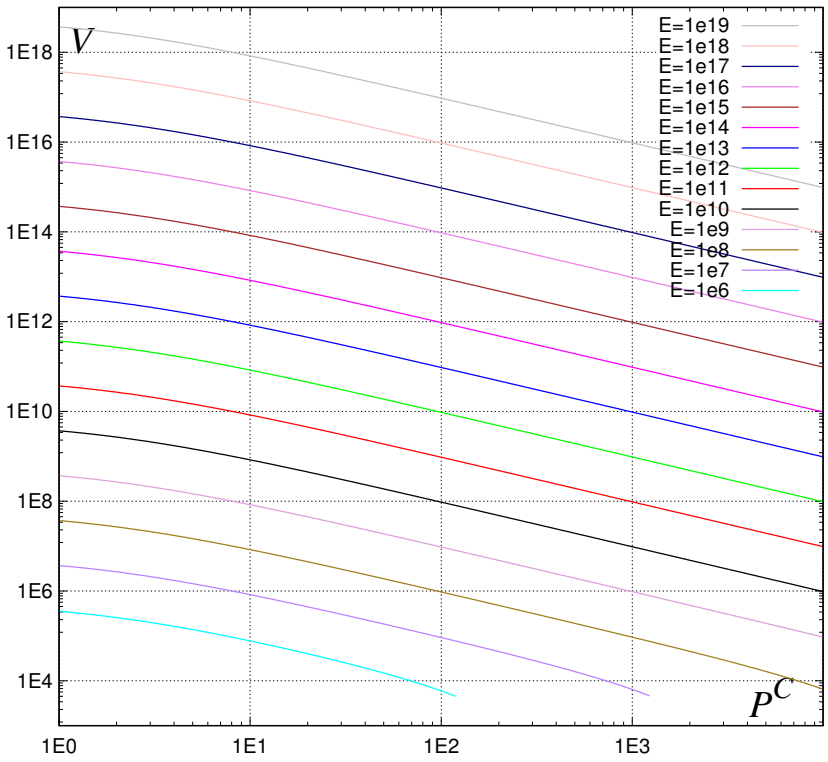


Figure 4.11: Isoenergy map for processor power P^C, and load size V for $m = 10$.

4.3.9 Network Power vs Startup Time

Isoenergy map for network power P^N and startup time S is shown in Fig. 4.12. The top of the map contains infeasible configurations where startup times are too large to activate all processors. For $E = 201E13$ the map has knee-like shape. For bigger energy levels we can see only part of the knee shape. This shape of isoenergy line means that with respect to keeping energy usage stable in very wide ranges of its values S remains unrelated to growing network power P^N. At some extreme values of P^N, startup S can compensate growing network power because power P^N is used in the network when a machine starts. However, the range of P^N and S must be large for this compensation effect to emerge. Let us also note that most of the (P^N, S) is covered by a plateau with little energy changes. This means that in practical ranges of system parameters S and P^N are secondary factors determining energy consumption.

4.3.10 Network Power vs Computation Rate

Isoenergy map for network power P^N and computation rate a is shown in Fig. 4.13. The bottom of the map contains infeasible configurations where low computation rate a (computation is too fast) results in computation time too short in relation to the communication time to allow starting all the processors. In a wide area of a and P^N pairs isoenergy lines are parallel to the P^N axis. It means that energy consumption VaP^C in computation is a key component of the total energy consumption and the changes in a cannot be compensated by changes in networking power P^N. Conversely, for tighter energy budget and fast computations, like $E = 1E14$ and $a \approx 1E\text{-}4$, the isoline has knee-like shape. This means that for very large network power P^N and fast computation (low a) the energy consumed in the network becomes dominant. Then, decreasing computation rate a (faster computations) can compensate growing network

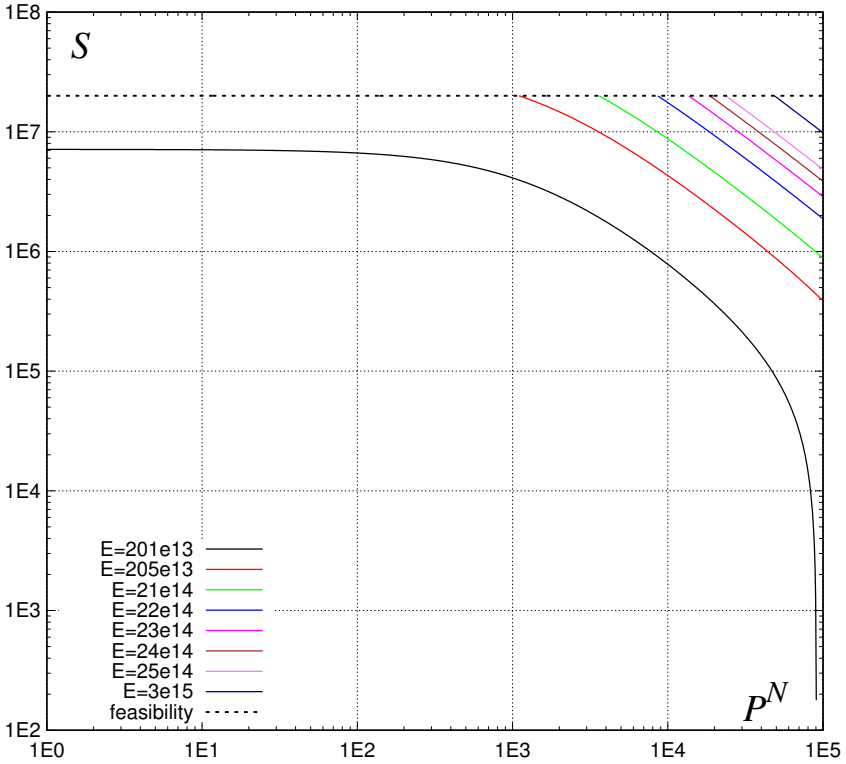


Figure 4.12: n Isoenergy map for network power P^N, and startup time S for $k = 100$.

power P^N. It can be observed that such an effect is limited by feasibility of system configurations because small a required for the compensation to emerge also restricts the feasibility area where the effect appears.

4.3.11 Communication Rate vs Startup Time

A (C, S) isoenergy map is shown in Fig. 4.14. The top-right area of the map is infeasible. It means that both too large startup times and too slow communications (C is large) prevent activating the required number of processors. We can see knee-like shape of the isolines in Fig. 4.14. Again, such knee-like shapes signify that two effects can dominate the energy consumption and they compensate each other only in a narrow area of system configurations. For very large S and small C the machine activation cost $mS(P^C + P^N)$ dominates energy VCP^N consumed in communications. Consequently, energy is determined by S and independent of C (top of the picture). Conversely, for large C and small S (the right side of the picture) energy consumption in communications is larger than the machine startup energy, and total energy consumed does not depend on S. The two parameters can compensate one another only is a relatively narrow area when $mS(P^C + P^N) \approx VCP^N$. Note similarity of the isoenergy map to the map (P^N, P^C) in Fig. 4.5.

4.4 Conclusions

In this section we proposed a new concept of isoenergy map. Such maps show relationships between parameters determining energy use in processing divisible loads. They indicate how changes in one parameter of the system must be matched by changes in other parameters, to achieve the energy savings. The maps can be also used to identify conditions when some parameters have no influence on the overall energy use.

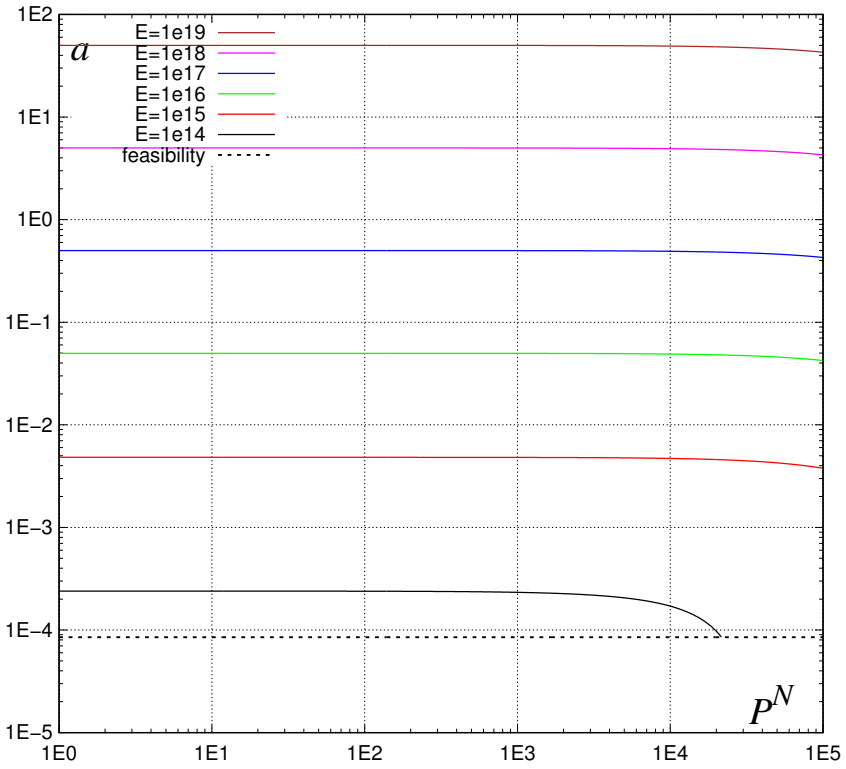


Figure 4.13: Isoenergy map for network power P^N, and communication rate a.

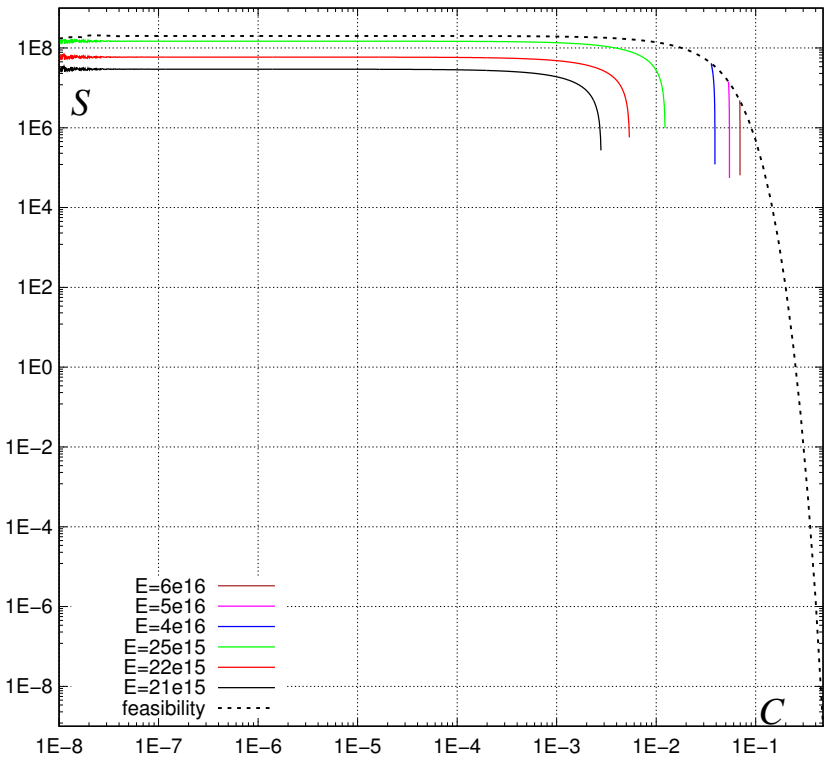


Figure 4.14: Isoenergy map for communication rate C, and startup time S at $a = 10$.

The idea of isoenergy maps has been applied to analyze performance for three energy use models. It has been observed that Amdahl's and Gustafson's perceptions of parallel application are essentially different, and consequently give opposing indications. For divisible loads we expected that increasing processor number m may bring only costs. But it appears that increasing m reduces schedule length and subsequently minimizes the consumed energy. Our analysis confirms that that intuitively obvious parameters of processor power P^C, network infrastructure power P^N or the time of computation aV are the most influential and hence most scalable determinants in energy saving. Parameters like idle state power consumption factor k and startup time S allow for shallower energy reductions. There are pairs of parameters, e.g. the number of processors and startup time (m, S) or network power consumption (m, P^N), which can compensate each other to keep energy use constant. On the other hand, in certain conditions some pairs, e.g. processor power and network power (P^C, P^N) or communication rate and startup time (C, S), are mutually independent and changing one parameter may be counterproductive because the other one should be optimized. Our study revealed more intricate relationships which manifest in local minima on the isoenergy maps. The minima imply that inefficiencies in one parameter (e.g. P^N, P^C, S, k) can be, but only to a limited extent, compensated by some other parameter (e.g. increasing processor number m). This means that scalability of such optimizations is again restricted.

Isoenergy maps provide a generic method of analyzing energy performance trade-offs. Any scheduling model, or energy use model may take advantage of isoenergy maps as a visualization front-end. In the complex relationships ruled by many factors isoenergy maps give holistic view and sense of direction for optimization efforts.

The models presented here fit well structured, intensive computations. Therefore, future work should be targeted at other types of applications, e.g., with multiple phases of computation and communication like NAS Grid Benchmarks [90] or Map-Reduce applications [19, 58]. Transaction-based loads typical of

OLTP or web-servers expose far greater variety of resource use profiles. Moreover, perception of the computing platform is very basic and further details should be incorporated. For example, systems with limited and hierarchical memory imposing limits on the workload sizes will be subject of the following chapters.

5 Homogeneous Systems with Hierarchical Memory

In this chapter a homogeneous computer system with hierarchical memory is considered. Two types of load distribution are introduced. Firstly, single installment load distribution and MIP methods used to solve the problem. Secondly, methods using multi installment distribution are presented. Simple heuristics and optimal MIP methods are used for solving the problem.

5.1 Single-installment Processing

In this section single installment load distribution and MIP methods are used to portion of the load sent to the machines.

5.1.1 Mathematical Model and Solution Procedure

In this section we formulate a problem of time- and energy-efficient scheduling of divisible loads in systems with hierarchical memory. We assume that there is a load of size V to be processed on m homogeneous machines. We will be looking for the lowest possible energy E subject to a limited processing time T.

The time schedule of communications and computations is shown in Fig. 5.1a. The process starts with the load held by the originator (initiator, resource allocator, etc.), computer further denoted as M_0. The originator is connected with

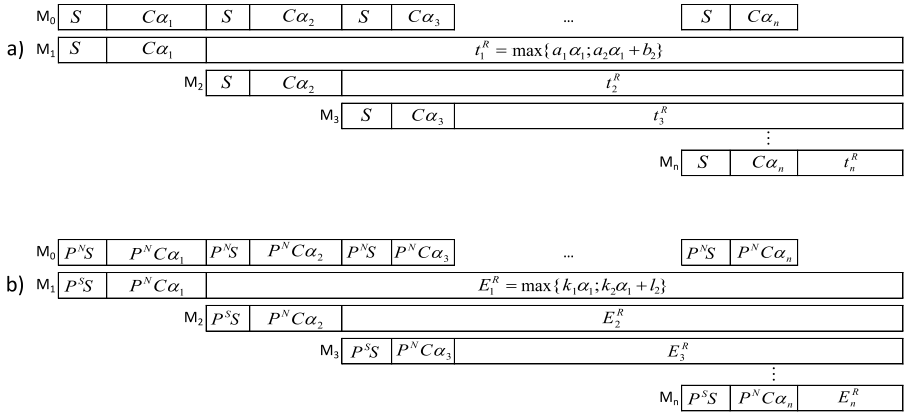


Figure 5.1: Schedule of the DLT computation a) timing b) energy usage.

all worker machines (computers, processors) $M_1, \ldots, M_m$, by means of some network with communication rate C. The originator is dividing and distributing the load. Communications are performed only between M_0 and the workers, one at a time. The load of volume V is sent in chunks of size $\alpha_1, \ldots, \alpha_m$ to machines $M_1, \ldots, M_m$, respectively. Machines take nonzero startup time S before they become capable of performing communication. This might represent a simple waking up time, as well as more complicated processes like loading appropriate VMs or platforms in dynamically scalable clouds. When the transfer of a chunk of load to machine M_i is finished, M_i starts processing it, while the originator activates machine M_{i+1} in order to send to it load α_{i+1}. The procedure is repeated until starting computations on all m processors.

It is often assumed in DLT that the time of returning results is negligible. Discussion on extensions of the DLT networking model including results collection time, parallel communications or communications concurrent with computations can be found in [22, 26, 28, 74]. For intelligibility of the further analysis such alternative communication and computation strategies are not considered in this section.

It is assumed that a machine can be in one of four states: idle (I), starting up (S), networking (N) or running, i.e. performing computations (R). With these states power consumption rates P^I, P^S, P^N, P^R and durations t_i^I, S, t_i^N, t_i^R

are connected, respectively. As these states might be of complicated nature, we assume that the powers are averages representing the on-going processes. The product of power consumption and time gives overall energy usage. Startup time S is the time a machine needs to wake up from idle and become operational, i.e. start networking or computations. The value of S is equal for all machines. The idle state can represent various situations in reality. Firstly, the machine can be turned off or hibernated to HDD and waiting for a signal to boot up. The corresponding P^I value will be the lowest, possibly a few Watts, but the startup time will be the longest one, even up to dozens of seconds depending on the software to be loaded. Secondly, the machine can be suspended to RAM, then the startup time will be at the level of seconds, but the power consumption will be around several dozens of Watts. Finally, the machine can be on and waiting to start executing a new task within a second, but the idle power rate will be up to 100W.

The energy consumed by machine M_i can be calculated as:

$$E_i = E_i^S + E_i^I + E_i^N + E_i^R$$

The running energy E_i^R depends on the size of assigned load α_i as determined by equation (3.2). As the communication rate is C, the total communication time is calculated as $t_i^N = C\alpha_i$ and energy as $E_i^N = P^N C\alpha_i$ for machine M_i. If some machines are not used in the schedule, their loads are $\alpha_i = 0$ so in effect $E_i^N = 0$ and $E_i^R = 0$. However, we need a binary decision variable x_i indicating whether machine i is used in the schedule. Thus, the energy consumed in the startup is $E_i^S = x_i S P^S$. The idle time can be calculated from the length of the schedule T and the remaining three times $t_i^I = T - Sx_i - C\alpha_i - t_i^R$. Hence, we get:

$$E_i = x_i S P^S + t_i^I P^I + C\alpha_i P^N + \max\{k_1\alpha_i, k_2\alpha_i + l_2\} \tag{5.1}$$

Energy E_0 consumed by the originator M_0 is calculated differently. It uses power P^N when other machines are starting up or during communication and the originator goes idle when all the load is distributed. Originator does not go idle when some other machine is waking up, because this would require some wake up time from him too. Including suspension of the originator into the model would complicate the model beyond reasonable need. Thus, we get:

$$E_0 = P^N(\sum_{i=1}^{m} x_i S + \sum_{i=1}^{m} t_i^N) + P^I(T - \sum_{i=1}^{m} t_i^N - \sum_{i=1}^{m} x_i S) \tag{5.2}$$

Considering that

$$\sum_{i=1}^{m} t_i^N = C \sum_{i=1}^{m} \alpha_i = CV$$

equation (5.2) can be transformed to a more convenient form:

$$E_0 = P^N(\sum_{i=1}^{m} x_i S + CV) - P^I(\sum_{i=1}^{m} x_i S + CV) + P^I T =$$

$$= P^I T + (\sum_{i=1}^{m} x_i S + CV)(P^N - P^I) \tag{5.3}$$

Total consumed energy is:

$$E = \sum_{i=1}^{m} E_i + E_0 \tag{5.4}$$

The problem of time- and energy-efficient scheduling can be formulated as an integer linear program for minimizing schedule length:

$$\min T, \text{subject to } E \leq E', \tag{5.5}$$

where E' is energy limit, or as an integer linear program for the minimization of energy consumed:

$$\min E, \text{subject to } T \leq T', \tag{5.6}$$

where T' is some limit on makespan. In both cases, it is further required that:

$$\sum_{i=1}^{m} \alpha_i = V \tag{5.7}$$

$$\sum_{i=1}^{j} x_i S + C \sum_{i=i}^{j} \alpha_i + t_j^R \leq T \qquad \forall j = 1, \ldots, m \tag{5.8}$$

$$t_i^I + S x_i + C \alpha_i + t_i^R = T \qquad \forall i = 1, \ldots, m \tag{5.9}$$

$$\alpha_i \leq V x_i \qquad \forall i = 1, \ldots, m \tag{5.10}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, \ldots, m \tag{5.11}$$

In the above formulations, the sum of all load chunk sizes must be equal to the whole load (5.7). Inequality (5.8) ensures the proper timing: the startups and communications of machines $M_1, \ldots, M_{j-1}$ have to elapse before machine M_j starts up. The startups, communication and computation of machine M_j must end before the end of schedule T. Note that constraint (5.8) is an implicit equivalent of optimality criterion used in DLT. Equation (5.9) allows to calculate the value of idle time t_i^I. Inequality (5.10) sets x_i to 1 if machine M_i is used in the schedule. When minimizing schedule length with the objective function (5.5) and constraints (5.7)-(5.11) energy E necessary for that schedule is calculated from (5.4). When minimizing the energy usage with the objective function (5.6), and constraints (5.7)-(5.11) schedule length T has to be given, as we do not perform direct bicriterial optimization with the above formulation.

5.1.2 Time-Energy Trade-off in Close-up

In our model we have a set of 12 parameters: $V, C, S, P^S, P^I, P^N, k_1, k_2, a_1, a_2$, size of the RAM indirectly represented by l_2 and b_2, and machine number m. Testing all relationships between all possible values of these parameters is not doable in the limited space of this thesis. Thus, we decided to stick to the analysis of the relationships between schedule length and energy consumption. In the following charts all parameters have fixed values except for m and one

Table 5.1: Index of the analyzed parameter ranges.

Parameter	Unit	Default value	Range Min	Range Max	Studied in Fig.
V	[MB]	10000	200	100000	5.2 - 5.6
C	[s/MB]	0.006	0.00001	0.1	5.7
S	[s]	70	0.1	100	5.8
P^S	[W]	101	101	112	-
P^I	[W]	6	6	79	-
P^N	[W]	91	91	116	-
k_1	[J/MB]	13.00	9.03	18.72	5.9
k	[J/MB]	294.43	150	500	5.10
a_1	[s/MB]	0.08	0.025	0.4	5.9
a	[s/MB]	2.37	0.53	2.37	-
ρ	[MB]	996	100	100000	5.11

parameter the impact of which will be analyzed. With that setting we generate a series of (T, E) values using increasing number of available machines. A guide to the analyzed parameter ranges is given in Tab. 5.1.

Now let us discuss shortly the default values we used in our analysis. The units used in Tab. 5.1 are seconds, MegaBytes, Watts. The size of the load $V = 10000$ is 10GB. The value of $C = 0.006$ means that 1MB of data will be transfered in 0.006s and represents network with bandwidth ca. 1300Mbit/s. Power consumption values $P^I = 6, P^S = 101, P^N = 91$ are chosen from the range of values observed on real machines in experiments described in Section 3.3. The power rate at the startup $P^I = 6$ and the startup time $S = 70$s represent computers waking up from hibernation to HDD, and then loading system or other necessary software. This is again a real measured value and time of 70s is acceptable in schedules of lengths usually between 1000s and 10000s. Unless stated otherwise the above values were used for all charts. Parameters describing processing rate and energy cost of computations $a_1 = 0.08, a_2 = 2.37, k_1 = 13.00, k_2 = 294.43$ were chosen from the range of measured values presented in Tab. 3.3. The machine represented in this analysis had 996MB of RAM available for the data. This may represent a 1GB machine with lightweight

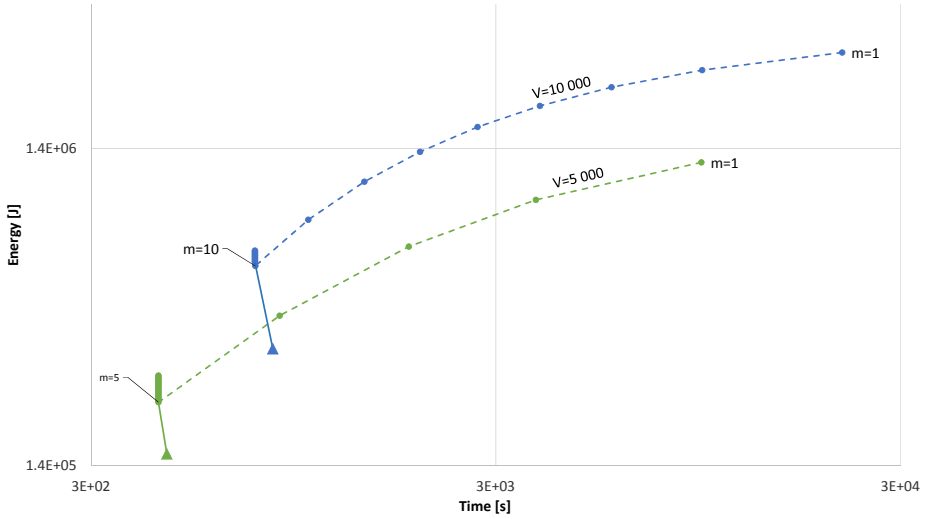


Figure 5.2: Minimum time and minimum energy points for different load sizes (log scale).

operating system and software, or for example 1.5GB VM or even 2GB machine with much heavier environment. Values of $l_2 = -280303.72$ and $b_2 = -2274.89$ where calculated to represent RAM of 996MB.

To obtain data for the charts presented in this study, the ILP model proposed in the previous section was programmed in CPLEX 12.6 software. Time minimization (5.5) was performed first with changing the number of machines available for the computations: every point in a chart is a result of solving one minimization problem. Usually values of m up to 20 or 30 were tested, because larger values increase solution time beyond a few minutes. Since changes of m are of discrete nature, the corresponding points in the charts are connected with dashed lines to guide the eye and help analyzing the results. When an energy-makespan relationship for fixed machine number is represented by more than just points, we proceed differently. For the selected numbers of machines we examined minimum energy derived from (5.6) with schedule length increasing with the resolution of 1s. As this change is continuous in nature (any arbitrary time value can be used) we marked these schedules with solid lines.

In Fig. 5.2 energy and makespan for two different problem sizes V are depicted. This figure will suit us to discuss some phenomena, but also to explain how to read the following charts, as they contain many different curve shapes. The diagonal dashed lines for $V = 5000$ and $V = 10000$ mean that allowing more machines for computations was both decreasing schedule length and energy used. However, the curves have pipe-like shape, i.e. the set of points on the left end of the curve forms a vertical line. This means that allowing more computers than in the point at the bottom of the pipe did not shorten the schedule while it increased energy usage. The reason is that it is impossible to use more machines. Machines start in sequence; machine M_j can start computations at time $\sum_{i=1}^{j} x_i S + C \sum_{i=i}^{j} \alpha_i$ and at some number of machines j this time exceeds T (here the optimum schedule length). Thus, the machines that were available, but not used, were only wasting power P^I in the entire time T. Note that the maximum number of usable computers changes with V and should be set prior to computations to avoid wasting energy E^I. Due to single installment load distribution the loads α_i are uneven. Consequently, for both data series (the dashed lines of the shortest schedules) virtually all machines performed out-of-core computations. For example, for $V = 10000$ only the last two machines are computing on-core. If we increase schedule length, more machines can get a load equal to 996MB of RAM and compute on-core which is energetically cheaper. For $V = 10000$ it is possible to give 996MB to at most nine machines and 1036MB to the remaining one, in this way obtaining the schedule with the lowest energy marked on the chart by a triangle. Between the point of the shortest time (bottom of the pipe) and the point of the lowest energy (triangle) we have a line of the minimal energy for a given schedule length. This line can be understood as a trade-off line where we can reduce computation time at the cost of increased energy consumption. And vice versa, we can reduce energy intake, but at the cost of longer processing.

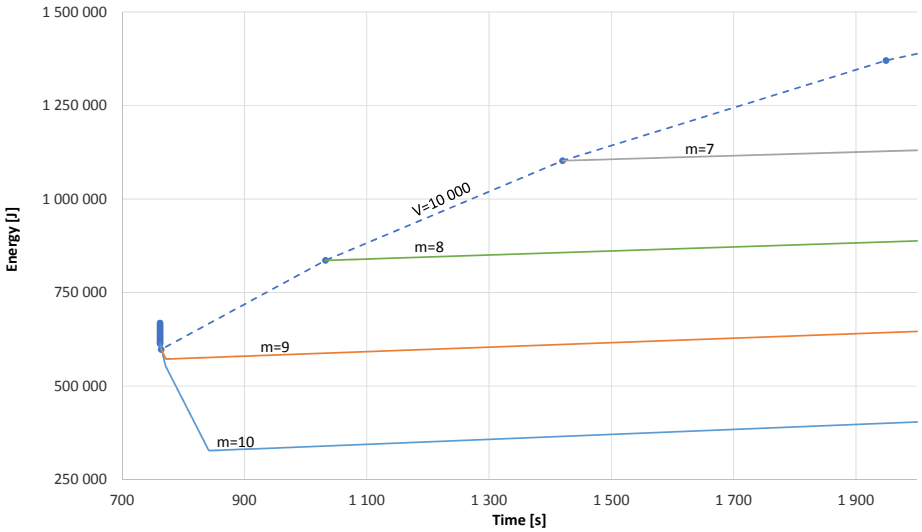


Figure 5.3: Minimum energy line close-up for 7 to 10 machines.

Now let us zoom on the area of points representing the shortest schedules with 7 to 10 machines and follow the analysis in Fig. 5.3. We extended the line representing the minimum energy schedules from the point of the lowest energy to the right. It is visible that the energy grows with time; this is due to the idle time of machines in the schedule. Actually with every second $(m+1) * P^I$ of energy is added. This line also separates the region of infeasible solutions which is located below the line. Similar lines have been observed for smaller machine numbers. For $m = 9$ energy savings which can be achieved by the time-energy trade-off are more limited. Due to single-installment load distribution it is not possible to build a schedule with more than only two machines operating on-core. Similarly, for $m = 8$ and $m = 7$ no energy savings below the level achieved at the shortest schedule are possible by lengthening the schedules. Here the longer schedules with some machines operating on-core still use more energy (than the minimum) because of overloading the machines computing out-of-core.

Fig. 5.4 shows these phenomena for schedules on 9 to 15 machines with even higher resolution. It is visible now that the time-energy trade-off line for $m = 10$ has two knees. Let us trace the line rightward from the point of the shortest makespan. We start in the steepest descent area, where only the last

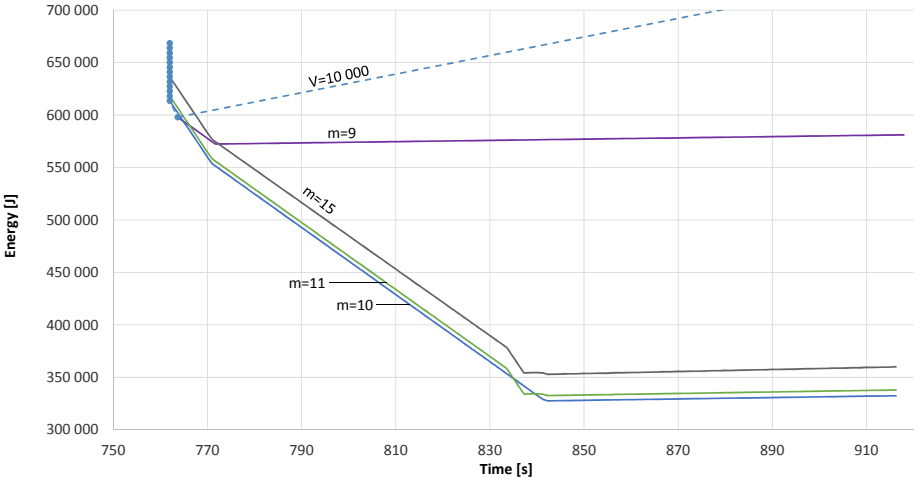


Figure 5.4: Time-energy trade-off close-up for 9 to 15 machines.

two machines are receiving loads α_i smaller than their RAM size. This limited use of the on-core processing is again due to single-installment load distribution. Lengthening the schedule allows to shift the load processed out-of-core in the eight first machines to the last two machines. When the schedule is long enough, the penultimate machine receives load of the RAM size and only the last machine can receive a load smaller than the RAM size, then the minimum energy line reaches its first knee around $T = 770$. The process of shifting the load to the last machine, still working on-core, continues with increasing schedule length until arriving at the second knee (being the point of the lowest energy of $T \approx 840$). Here the last machine receives a load equal to its RAM size. The line is steeper in the area where the RAM of the two last machines is being progressively filled with load, than in the area where this happens only to last machine. Another interesting effect can be observed with the minimum energy line for $m = 11$. The machines can finally conduct all the computations on-core, but the schedule must be long enough to start $m = 11$ processors. When such length is reached, the line gets another bend ($T \approx 835$), and in a small area computations on 11 machines are more energy-efficient than on 10. For machine numbers $m > 11$, we portrayed here only a line for $m = 15$ because all these lines have the same

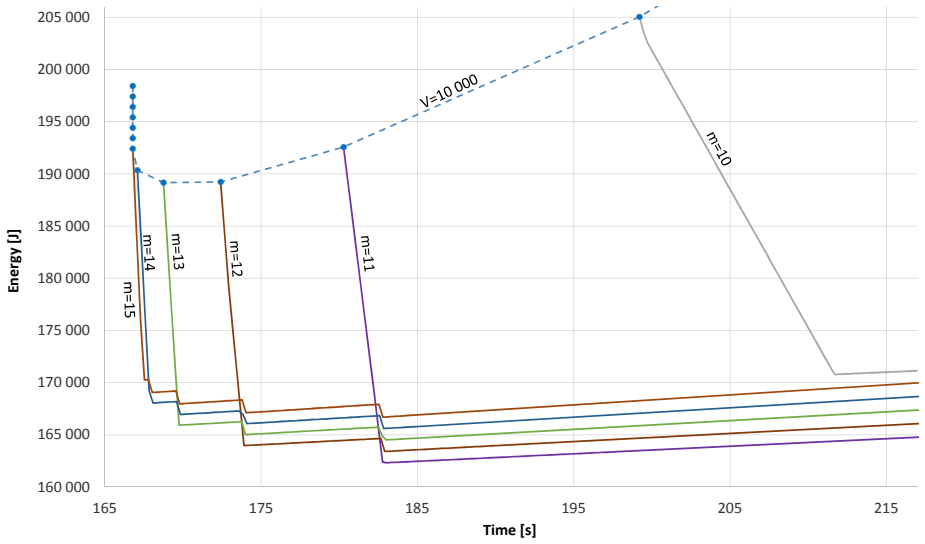


Figure 5.5: Time-energy trade-off for systems with shorter startup $S = 7$.

shape as it is not effective to include more than 11 machines in the schedule. The lines differ only by an offset of the startup and idle energy used by the excessive idle machines.

Fig. 5.5 presents a system where machines start quicker, i.e. in $S = 7$. The rest of the parameters remain unchanged. The figure shows minimum energy lines for $m = 10$ to $m = 15$ machines. The (dashed) line of the shortest schedules is slightly smoother than before which will be discussed in the next section (see Fig. 5.8). As the startup is shorter, it is possible to use more machines in the schedules and thus reduce the makespan. As previously, the (solid) lines of the time-energy trade-off start at the points of the shortest schedule. The shapes of the minimum energy lines for $m = 10$ and $m = 11$ are similar as in the previous examples; however for $m > 11$ new phenomena emerge. Previously, straight lines run from the knee of the lowest energy to the rightmost part of the chart. Now they have segments where the minimum energy is stair-casing down. Note that the steps of the "stairs" and points of minimum energy always appear at the same schedule lengths. Each such step represents excluding one machine from the schedule. The size of the energy drop is $S * (P^N + P^S) - 2SP^I$ which is the energy consumed by a starting machine and the originator waiting for

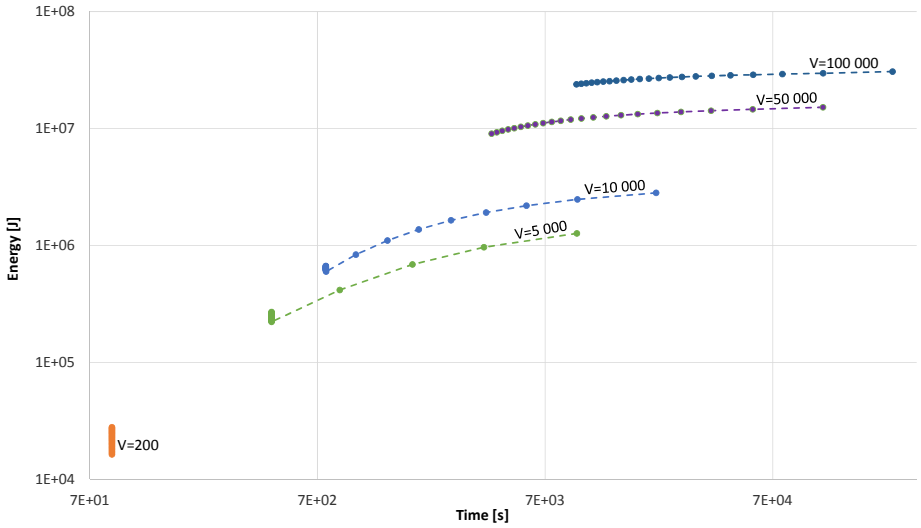


Figure 5.6: Energy vs time for different load sizes V.

the machine to start minus the energy used by the two machines remaining idle. The number of steps depends on the number of machines which can be switched off to reach the lowest energy schedule. Here, the lowest energy is achieved at $m = 11$, with all machines operating on-core. Thus, e.g. the schedule with $m = 12$ can switch off one machine, giving it one drop, and so on.

5.1.3 Impact of Other Parameters on Time-Energy Trade-off

In this section we will examine the relationships between time and energy consumption in a broader scene. Therefore, we will analyze mainly the shortest schedules, skipping in some charts the whiskers of the time-energy trade-off for clarity. The time-energy trade-off will be still present in the discussion. However, we will focus on the impact of the system parameter changes on the performance of the computations.

In Fig. 5.6 we analyze the impact of the problem size V. For $V = 200$ we observe a straight vertical line, perpendicular to the time axis. This means that the time of the computations could not be improved by applying more

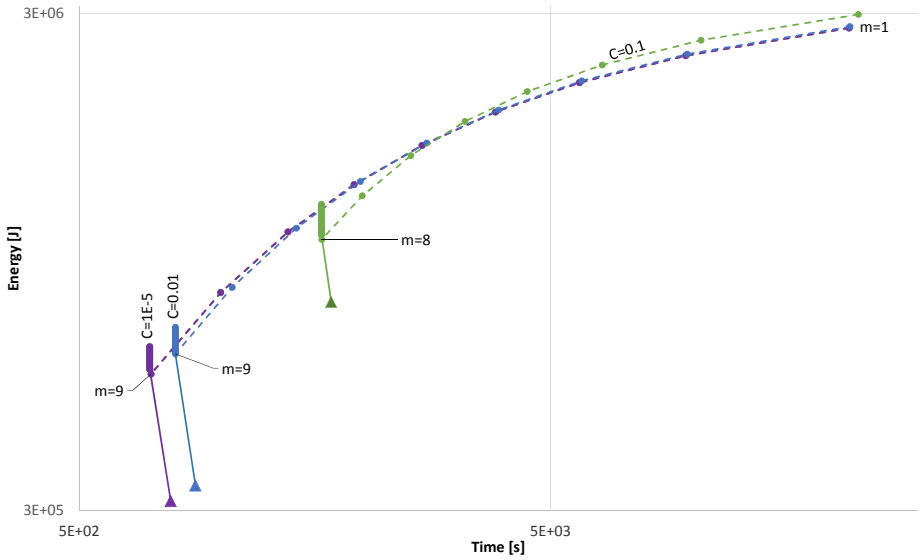


Figure 5.7: Energy vs time for different communication rates C.

machines while the energy cost was still growing. The reason is that there was not enough load to exploit more than one computer and all machines beyond M_1 were merely wasting energy in the idle state for the entire time T. The lines for $V = 5000$ and $V = 10000$ were discussed previously (section 5.1.2). For $V = 50000$ and especially for $V = 100000$ the curves become nearly horizontal, i.e. almost perpendicular to energy axis. In this area it is possible to significantly shorten computations by adding more machines, but savings in the energy will be very limited because for $V > 1000$ virtually all machines perform out-of-core computations, which is a result of single-installment load distribution.

In Fig. 5.7 we can observe how different values of communication rate C affect the computations. For the slowest communication at $C = 0.1$, the number of machines that can receive load is the smallest ($m = 8$) and the schedules are longer and more energy-consuming than for faster communications. We get better results with $C = 0.01$ and C=1E-5. However, the difference between them seems small considering the range of change in the network speed. The curves for communication rate C <1E-5 are not visible in the chart because they overlap the curve of C =1E-5. This shows that there are limitations of speeding

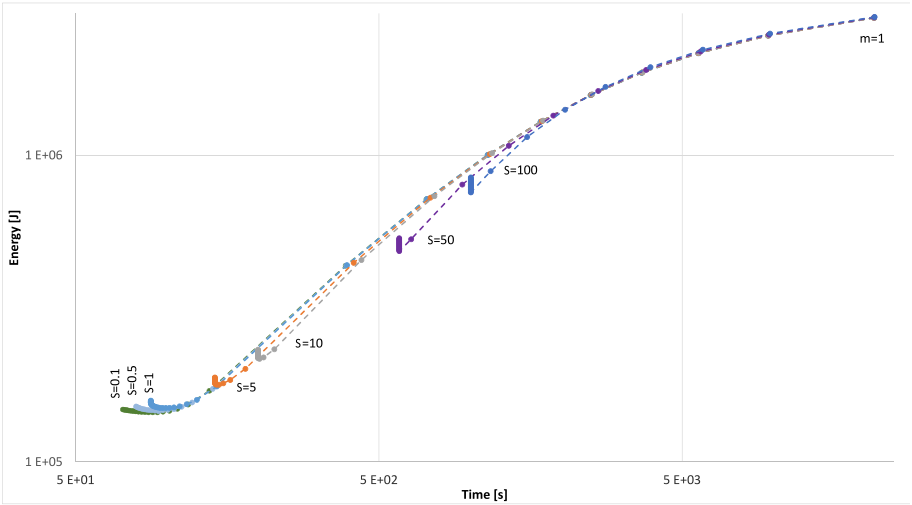


Figure 5.8: Energy vs time for different startup times S.

the computations up and reducing the energy usage by means of improving only the networking capabilities. This observation is in line with the isoenergy map (C, m) in Fig. 4.9 showing that parameter C alone has limited capability of reducing energy usage.

Startup time S is a parameter limiting the number of machines that can be included in the schedule. In the previous charts the significant value of S was one of the main bottlenecks preventing the increase of machine number. In Fig. 5.8 we study the effect of changing S on the performance of the computations. For $S = 100$ it is possible to use only 8 machines and we can see a very clear pipe-like shape in Fig. 5.8. For $S \geq 50$ the shortest schedule is also the one with the minimum energy. However, for $S \leq 10$ the minimum energy is reached before arriving at the shortest schedule and hitting the limit of the number of machines that can be used in the schedule. Although it could be expected that the minimum energy is achieved when all, or as many as possible, machines receive a load small enough to process it in RAM, it is not the case. For $S = 10$ half of the machines still perform out-of-core computations in the minimum energy schedule. In the case of $S = 1$ the number of machines at minimum energy schedule grows to 28, and the pipe shape is more outstretched. With

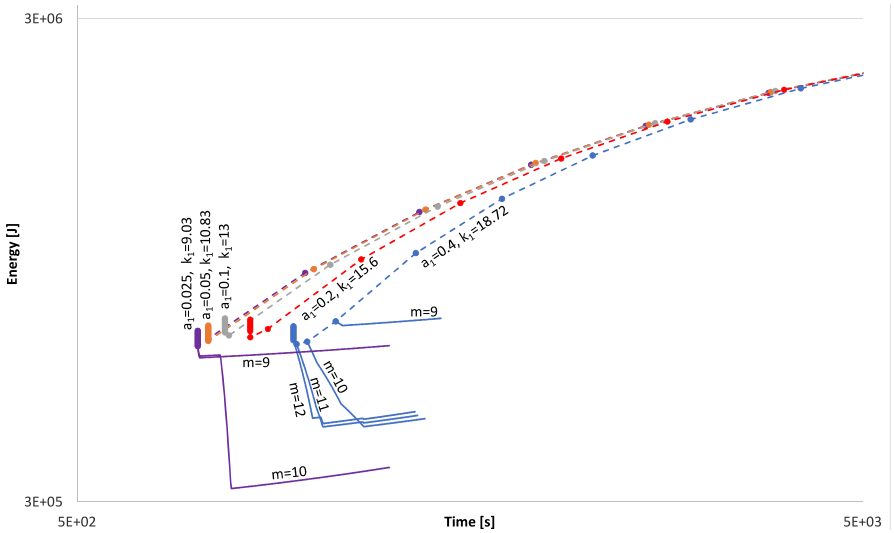


Figure 5.9: Energy vs time for different on-core computation rates a_1.

decreasing S the share of machines computing out-of-core shrinks, but actually never disappears. For really small startups, such as $S = 0.5$ and $S = 0.1$, the impact of the startup time on the number of usable machines is milder as we see no sharp upright pipe shape. Since the curves noticeably bend upward at their leftmost ends it means that the shorter computation time is gained at the cost of increased energy consumption. Yet, the shape of this line depends also on the load size V (see Fig. 5.6).

The impact of computation rate a_1 is shown in Fig. 5.9. Let us observe that computation rate a_1 and energy cost k_1 are mutually related. Decreasing a_1 means shorter computation and consequently smaller energy consumed per load unit. Yet, as observed in Section 3.3 this relationship is not linear because computer systems are not energy-proportional. For example, power consumption of the computing equipment does not halve with dividing CPU speed by two. Therefore, with halving a_1 (i.e. doubling the speed) we divided k_1 by 1.2, starting with $a_1 = 0.1, k_1 = 13$ as reference values. Fig. 5.9 zooms in on the area where the most interesting observations can be made. The processing rate here affects the number of machines that can be used in a schedule. For the

fastest processing at $a_1 = 0.025$ and $a_1 = 0.05$ it is $m = 9$, and m increases for the cases of slower processing, up to $m = 12$ at $a_1 = 0.1$. For slower processing there are visible schedules with lower energy which may seem to be a paradox. The explanation is as follows. For $m = 9$ (marked on chart) all processing is still done out-of-core, and with slower computations more machines can be included into the schedule and the load is divided into smaller chunks. Thus, on some machines the load is fitting into memory, in effect allowing more energy-efficient on-core computations. Now let us discuss whiskers marking the lines of minimum energy. For clarity of the chart, whiskers are shown only for the border values $a_1 = 0.4$ and $a_1 = 0.025$, and are drawn only partially in the area of the linear growth. Note that the shapes of the whiskers for $a_1 = 0.4$ resemble the ones discussed earlier. The minimum energy curve for $m = 9$ has the same shape as in Fig. 5.4 and for the cases of $m = 11, m = 12$ there are staircase patterns as in Fig. 5.5. The curve for $m = 10$ has one more bending point. The curve is changing its slope three times, because in the shortest schedule there were three machines with load α_i smaller than the RAM size. This allowed to off-load the out-of-core computations and fill the three machines up to the RAM size in three stages of increasing the makespan. A different shape of minimum energy line can be observed for $a_1 = 0.025$ and $m = 10$. Within the minimum length schedule it is not possible to include the tenth machine into computations. Still, for its minimum energy line two steps can be seen. Firstly it obtains energy minimum using only 9 machines. The line is slightly higher than the one for $m = 9$ because of the idle energy of the tenth machine. Then after adding some more time to the makespan the tenth machine can be included in the computation, reaching the energy-optimal energy point for the computation rate $a_1 = 0.025$.

In Fig. 5.10 the impact of changing the energy cost of out-of-core computations k_2 on total energy consumption is depicted. Out-of-core computations are the costliest part of the analyzed the schedules. The startup time was shortened to $S = 7$ to allow better insight into the discussed phenomena. At a given value

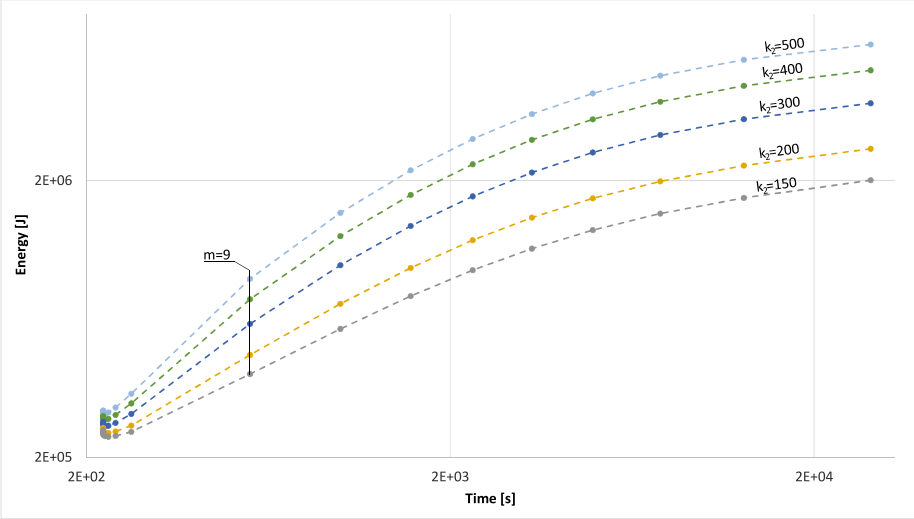


Figure 5.10: Energy vs time for different energy factors k_2 in out-of-core computations.

of m the points on all the curves align to vertical lines (see the line marked for $m = 9$). There are the same minimum schedule lengths for the same number of machines, as k_2 has no impact on the schedule length. If we treat the $k_2 = 300$ curve as a reference, then we have systems with costlier out-of-core computations $k_2 = 400$ and $k_2 = 500$, as well as less less costly systems at $k_2 = 200$ and $k_2 = 150$. Yet, all of the curves converge to a much smaller difference in energy consumption with an increasing number of machines. Part of this happens before loads α_i start to fit into RAM; even on $m = 9$ all machines still work out-of-core. The savings in energy usage are a sheer result of the parallelism shortening the schedule. For bigger values of m the convergence is even more apparent because more machines operate on-core. This shows that, the reductions of the energy intake of the equipment have limits. Conversely, energy costs of worse hardware can be often compensated with better parallelism.

Fig. 5.11 represents an example of a slightly different configuration. Parameters $a_1 = 0.066, a_2 = 0.53$ represent a faster machine with an SSD drive. This results in higher energy demand $k_1 = 9.03, k_2 = 82.66$, $P^S = 112$, $P^N = 116$. Machines are waiting almost ready to start processing with $S = 5$ and $P^I = 79$.

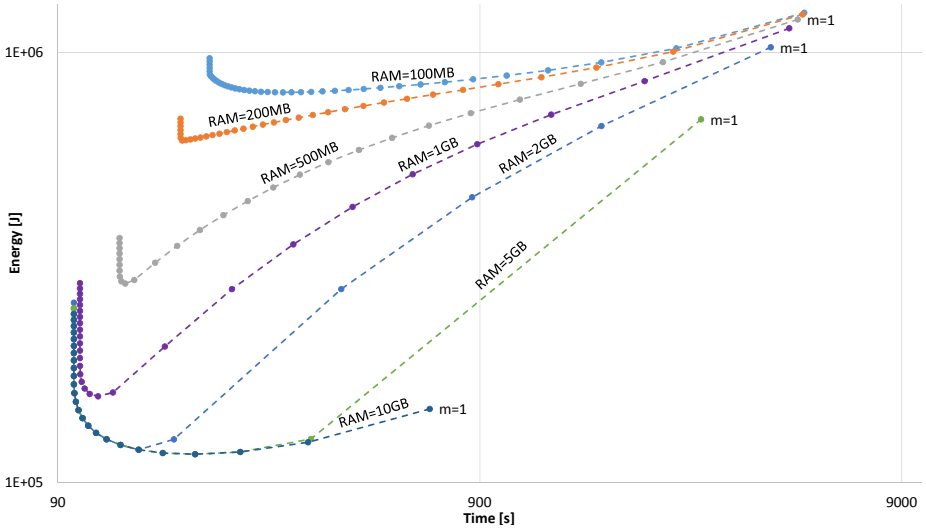


Figure 5.11: Energy vs time for different RAM sizes ρ.

Also communication is faster $C = 0.002$, which represents effective connection speed of ca 4000 Mbit/s. The parameter changed in Fig. 5.11 is RAM size ρ on the machines. Parameters b_2 and l_2 were changing accordingly. For ρ =10GB all the data fit into RAM, and the curve here has quite wide area of time-energy trade-off. For ρ =5GB and ρ =2GB with increasing number of computers the chunks of load start fitting into memory, and the curves are overlaying with the one for ρ =10GB. For the remaining four sizes of RAM, we observe pipe shapes similar to the previous charts. Still, there are differences: the number of machines which can be included into the schedule before the curve upright turn greatly increases because the system is generally faster. This also broadens the area near the optimum energy, from a few machines up to 20 machines even for RAM=100MB.

5.2 Multi-installment processing

In this section divisible load processing methods using multi installment distribution are presented. Simple heuristics algorithms and optimal MIP methods will be used for solving the scheduling problem.

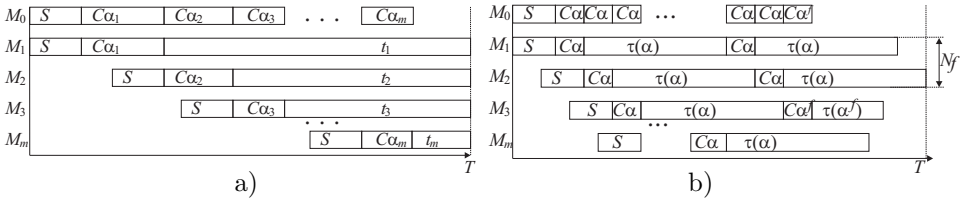


Figure 5.12: a) Single-installment schedule. b) Multi-installment schedule.

5.2.1 Simple Multi-installment Scheduling Methods

Let us remind that processing time in load size α is $\tau(\alpha) = max a_1\alpha, a_2 + b_2$ and energy is $\epsilon(\alpha) = max k_1\alpha, k_2 + l_2$ (see equations (3.1), (3.2)). Let us assume that 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 $(m-1)C\alpha \leq \tau(\alpha)$ which can be reformulated as $m \leq a_1/C + 1$ for $\alpha \leq \rho$ and $m \leq a_2/C + 1 + b_2/(C\alpha)$ 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 $m > 1$.

The number of iterations in which all m machines obtain load α is $N_o = \lfloor \frac{V}{\alpha m} \rfloor$. The number of chunks of size α in the last 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. 5.12b):

$$T = S + N_o(C\alpha + \tau(\alpha)) + \begin{cases} N_f C\alpha + \max\{\alpha^f C + \tau(\alpha^f), \tau(\alpha)\} & N_f > 0 \\ \max\{(m-1)C\alpha, \alpha^f C + \tau(\alpha^f)\} & N_f = 0 \end{cases} \tag{5.12}$$

Deriving energy consumption requires calculating idle times on $M_1, \ldots, M_m$, as well as computing and communication durations. At the start of the schedule M_i is idle until time $C(i-1)\alpha$. Thus, total energy used before machines activation is $E_A^I = P^I \sum_{i=1}^{m}(i-1)C\alpha = P^I(m-1)m/2C\alpha$. 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. 5.12b). The idle time at the end of the schedule 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

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

$$(m - N_f)\tau(\alpha) + \frac{1}{2}(m-1)(2N_f - m)C\alpha - C\alpha^f - \tau(\alpha^f). \tag{5.13}$$

Suppose that $\alpha^f C + \tau(\alpha^f) \geq \tau(\alpha)$, which means that M_{N_f+1} has no idle time at the end of the schedule. 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

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

$$(m-1)(\tau(\alpha^f) + C\alpha^f) - N_f\tau(\alpha) + \frac{1}{2}(m-1)(2N_f - m)C\alpha. \tag{5.14}$$

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

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

$$E = E_A^I + E^S + E^R + E_B^I + E_0. \tag{5.15}$$

Now we will propose methods of choosing load chunk α.

Simple static chunk (SSC) algorithm assumes that load chunk sizes are equal to the size of 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 some processors may remain idle.

Static chunk with underload (SCU) algorithm assumes $\alpha_{SCU} = V/(q_1 m)$. Thus, algorithm SSU 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 $\alpha_{SCO} = V/(m \max\{1, q_2\})$. In this formula value 1 means that at least one load distribution iteration will be done.

Guided Self-Scheduling Adaptation (GSS) algorithm uses the idea of the classic loop scheduling algorithm [28]. Let V' be the size of load remaining on M_0 to be distributed. Chunk sizes are calculated as $\alpha_{GSS} = \min\{V', \max\{1, \min\{V'/m, \rho\}\}\}$. Thus, otherwise than in the three previous algorithms, load chunk sizes decrease in the course of the schedule. Assuming that $V > \rho$, the algorithm starts with load chunk sizes of RAM size. When $V' < \rho$, GSS gradually decreases chunk sizes and thus also 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

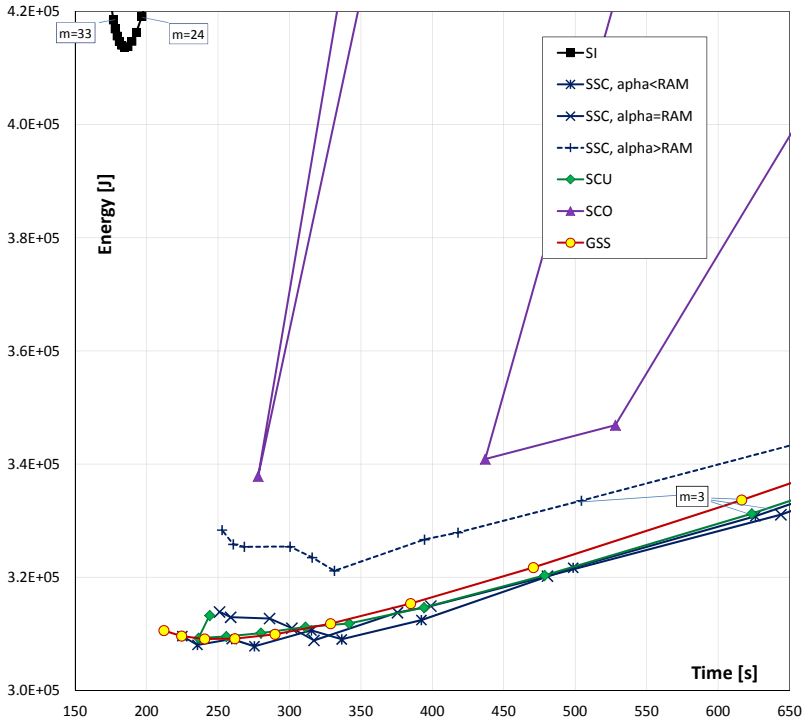


Figure 5.13: Time-energy diagram for the default system.

structures representing processed load or some size which sufficiently amortizes the fixed overheads in processing one load chunk. In the further considerations we assume that it is 1MB. 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 $V/\rho > m$ GSS immediately 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, SSU, SCO without entailing idle time on $M_1, \ldots, M_m$

5.2.2 Performance Comparison

In this section we compare performance of the above introduced scheduling algorithms. Unless stated to be otherwise, the system and application parameters were the following: $V = 10$GB, $a_1 = 0.082$s/MB, $a_2 = 2.366$s/MB, $b_2 = -2274.9$s, k_1=13J/MB, $k_2 = 294$J/MB, $l_2 = -280$kJ, C =7.8ms/MB,

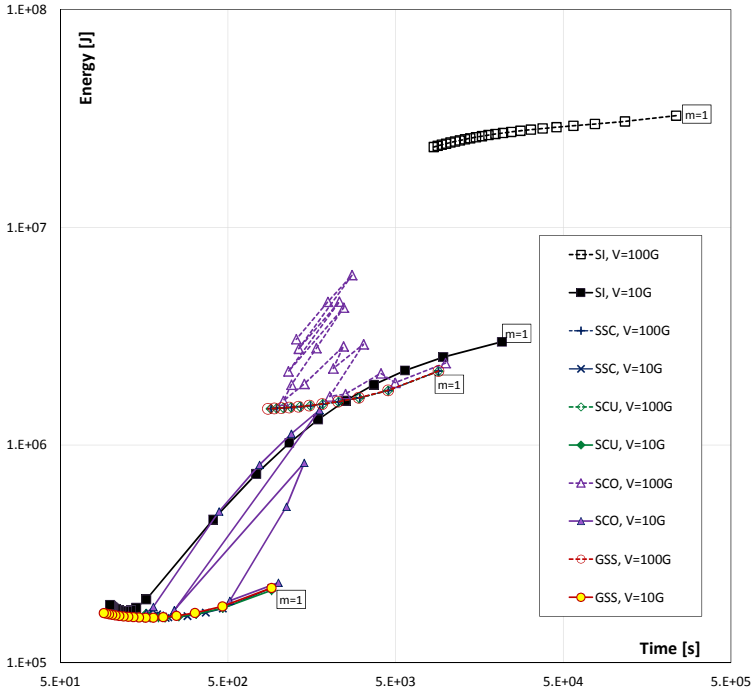
$S = 10$s, $P^I = 14$W, $P^N = 91$W, $P^S = 101$W, ρ=996MB. It can be verified that processing out-of-core is roughly 28 times slower per MB than processing on-core. The energy consumption per MB is roughly 23 times higher out-of-core. Communication rate C corresponds with communication speed of ≈1Gb/s. P^I, S, P^S represent a system which quite effectively switches from idle to running state, e.g. from hibernation to an SSD disk. The size of RAM accessible for storing data is $\rho = 996$MB. Values in similar range were found experimentally in real systems (Chapter 3). Since our problem is bicriterial, we will show performance as consumed energy E vs schedule length T for changing number of machines m.

We start with a time-energy chart in Fig. 5.13 for the above reference parameters to introduce the phenomena guiding performance. The dependencies are shown only partially for better visibility (but will be shown for a wider range of time and energy in the next figure). Since schedule length T (in general) decreases with growing number of used machines m, the smallest m is shown on the right-hand-side of the chart and the dependencies progress leftward with growing m. It can be observed that with growing m not only T decreases but so does used energy E. 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 seen also in the following figures and was already observed in the previous section (cf. the pipe shapes). 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, therefore even small excesses

of chunk sizes above ρ are escalated to big increases in 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: 680MB, 996MB, 998MB, where ρ=996MB. 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 a better load balancing.

In Fig. 5.14a time-energy chart is shown for $V = 10$G and $V = 100$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 single installment method (SI) 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 as discussed in section 5.1.2. Finally, at $V = 10$G and $m > 11$ its performance becomes comparable with multi-installment methods. In Fig. 5.14b time-energy chart is shown for $\rho = 100$MB and $\rho = 10$GB. For SI dependencies for $\rho = 1$GB, 10GB are shown because SI's results for $\rho = 100$MB are out of the range shown in Fig. 5.14b. 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 (bottom-left part of the chart). SI is capable of constructing such shorter schedules, but it activates new machines which brings energy costs bigger than in the other methods. SSC

a)



b)

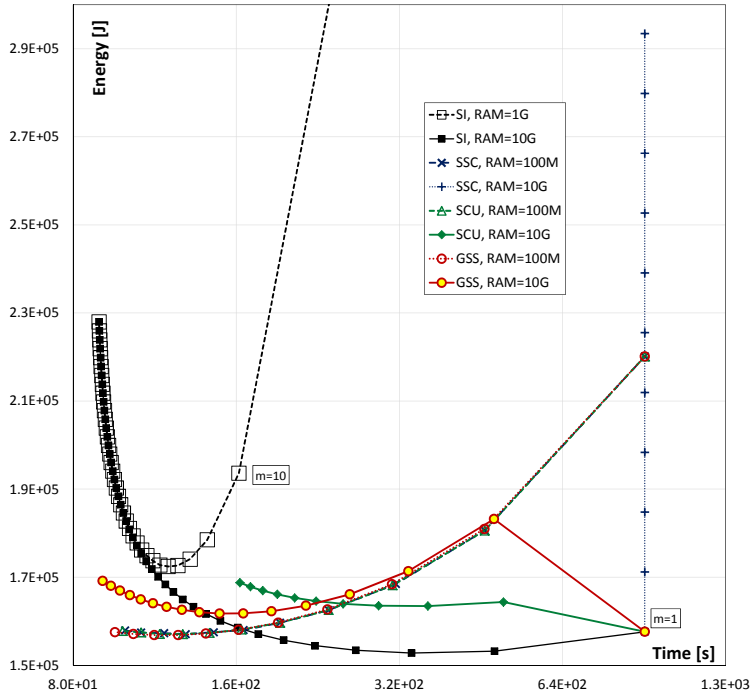
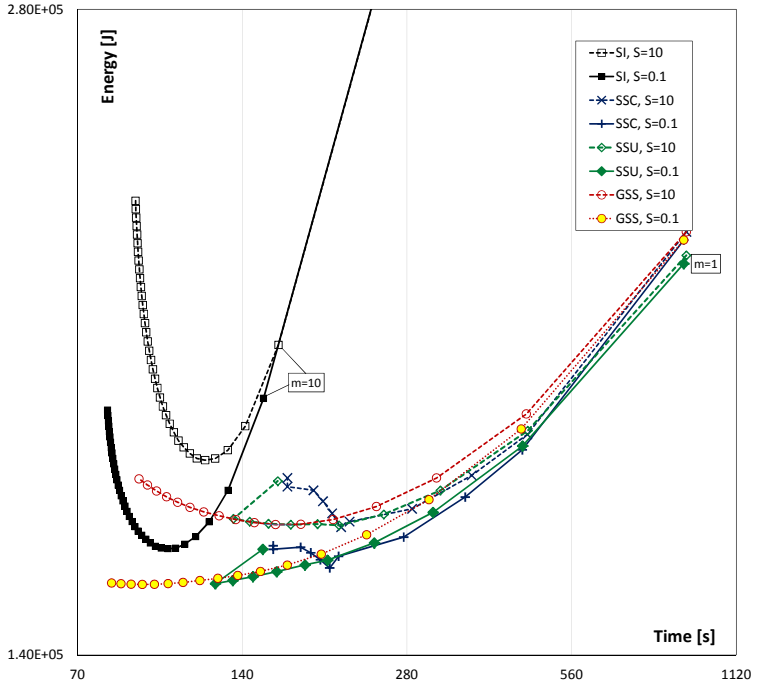


Figure 5.14: Time-energy dependence a) for $V = 10$G and $V = 100$G, b) for varying ρ.

method for $\rho = 10$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 $\rho = 100$MB is better than for $\rho = 10$GB because small load chunks reduce initial and final idle times. It can be also observed that GSS for $\rho = 10$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. Yet, GSS uses more energy in this case. Both GSS and SI approach the minimum schedule length determined by communication time: $S + CV$. However, GSS is more energy-efficient.

In Fig. 5.15a time-energy relation is shown for two values of the startup time $S = 0.1$s and $S = 10$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. 5.15b 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 heuristics is similar, GSS algorithm has an advantage of using more machines than SSC, SCU and consequently is able to build shorter schedules though at higher energy costs. The single installment 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. speeds increases).

a)



b)

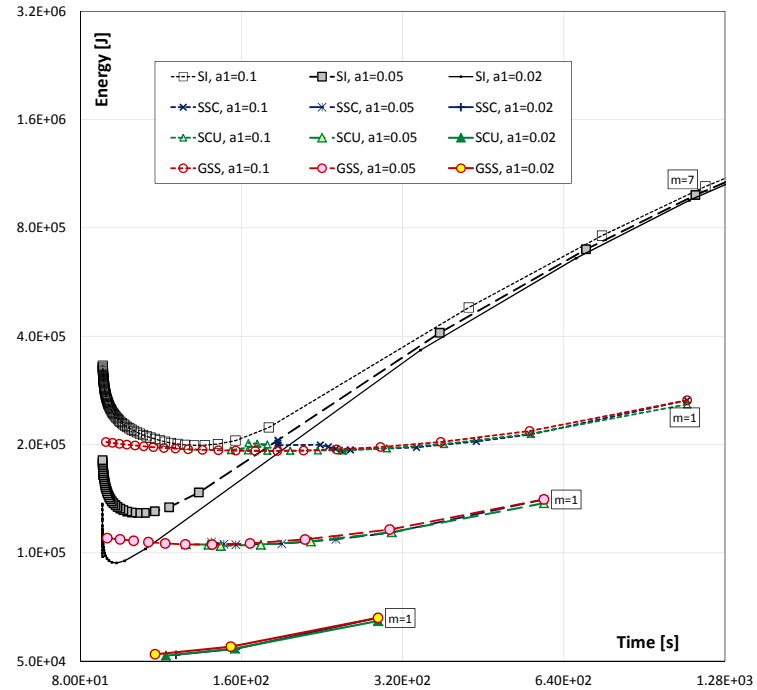


Figure 5.15: Time-energy dependence a) for $S = 10$s and $S = 0.1$s, b) for changing a_1.

5.2.3 Optimum Multi-installment Methods

In the following we propose an algorithm constructing the shortest multi-installment load distribution in a homogeneous system. Then, the performance of such load processing method will be studied by use of isoefficiency maps. Thus, by referring to isoefficiency maps we focus on time performance only. The time-energy performance in multi-installment processing will be examined in Chapter 6.

Mathematical Model of Parallel Application

Again, we will be assuming that execution of the data-parallel application is initiated by a root processor M_0, which 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 single level tree and M_0 communicates directly with worker processors. The machine starting process lasts for S time units. 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). Note that by introducing overhead O we apply more precise communication time model. 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.

Let us remind that dependence of the computing time on load of size α in a system with two memory levels is represented by function

$$\tau(\alpha) = \max\{a_1\alpha, a_2\alpha + b_2\} \tag{5.16}$$

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. 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 the shortest computation schedule can be formulated as a mixed integer linear program (MIP):

$$\text{minimize } T \tag{5.17}$$

subject to:

$$t_j + C\sum_{i=1}^{m}\alpha_{ij} + O\sum_{i=1}^{m}x_{ij} \le t_{j+1} \quad j = 1,\ldots,n \tag{5.18}$$

$$\begin{gathered} q_{ij} + C\alpha_{ij} + Ox_{ij} + \tau_{ij} \le T \\ j = 1,\ldots,n \quad i = 1,\ldots,m \end{gathered} \tag{5.19}$$

$$\begin{gathered} q_{ij} + C\alpha_{ij} + Ox_{ij} + \tau_{ij} \le q_{il} + (1 - x_{il})Z \\ i = 1,\ldots,m \quad j = 1,\ldots,n-1 \quad l = j+1,\ldots,n \end{gathered} \tag{5.20}$$

$$S\sum_{i=1}^{m}x_{ij} \le t_j \qquad j = 1,\ldots,n \tag{5.21}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\alpha_{ij} \ge V \tag{5.22}$$

$$\alpha_{ij} \le Vx_{ij} \qquad i = 1,\ldots,m \quad j = 1,\ldots,n \tag{5.23}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{5.24}$$

$$\begin{gathered} Zx_{ij} \geq q_{ij} \geq 0 \\ t_j \geq q_{ij} \geq t_j - Z(1 - x_{ij}) \\ i = 1, \ldots, m \quad j = 1, \ldots, n \end{gathered} \tag{5.25}$$

$$\begin{gathered} a_1\alpha_{ij} + Zu_{ij} \geq \tau_{ij} \geq a_1\alpha_{ij} \\ a_2\alpha_{ij} + b_2 + Z(1 - u_{ij}) \geq \tau_{ij} \geq a_2\alpha_{ij} + b_2 \\ i = 1, \ldots, m \quad j = 1, \ldots, n \end{gathered} \tag{5.26}$$

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 by the parallel application and computing platform, while Z is a large number. 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 (5.18) 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 (5.19) ensure that computations finish before the end of the schedule. Constraints (5.20) 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 (5.21) the processor which is receiving the jth load chunk is already started when sending the jth chunk begins. Inequality (5.22) guarantees that the whole load is processed. Constraint (5.23) ensures that a processor that is not receiving the jth load chunk receives load of zero size in the jth communication. By (5.24) only one machine can receive the jth load chunk. Inequalities (5.25) ensure that the auxiliary variable q_{ij} is equal to $t_j x_{ij}$. Using product $t_j x_{ij}$ directly is not possible in a linear program. It is possible to obtain the same value by linearizing constraints (5.25) and an additional variable q_{ij}. Inequalities (5.26) 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.

5.2.4 Isoefficiency Maps

Isoefficiency Map Construction

Schedule length T calculated by solving (5.17)-(5.26) 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 $\mathcal{E}(m, n, V) = T(1, 1, V)/(mT(1, 1, V))$. 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. A collection of isoefficiency lines drawn in some area of $m \times V$ space is an *isoefficiency map*.

Due to the complex nature of the formulation (5.17)-(5.26) 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 $V_{\max}(m, n)$. 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 on Intel i7@2GHz, the MIP optimality gap was set to 0.5%. Consequently, the obtained solutions mostly were not guaranteed optimum. Still, the solutions are always feasible and can be considered as good approximations of the optimum solutions of (5.17)-(5.26).

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 (cf. Section 3.3), we will concentrate on the frequent phenomena rather than on particular performance numbers.

In Fig. 5.16 isoefficiency map for the load sizes smaller than $V_{\max}(m,n)$ is shown, and in Fig. 5.17 for the loads above $V_{\max}(m,n)$. For better clarity, maximum values of efficiency $\mathcal{E}_{\max}(m,n,V)$, and the corresponding load sizes $V_{\max}(m,n)$ are shown in Tab. 5.2. The line of maximum efficiency $\mathcal{E}_{\max}(m,n)$ is denoted as MAX in Figs 5.16, 5.17 and the isolines are labeled with their ef-

Table 5.2: Maximum efficiency and corresponding load sizes vs m at $n = 20$ installments.

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

ficiency levels. The efficiency for $m = 1$ is always 1, so 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 on-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 Tab. 5.2 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 on-core over out-of-core processing, the costs of starting the machines, communicating and avoiding idle time. MIP (5.17)-(5.26) 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 the $\mathcal{E}_{\max}(m,n)$ is neither smooth nor does it show an obvious trend.

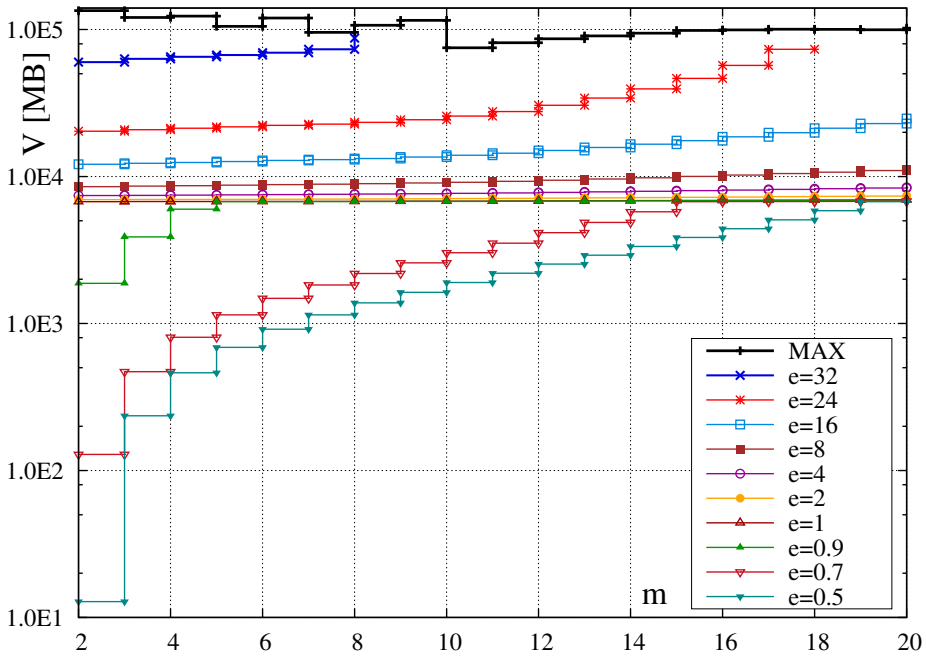


Figure 5.16: Isoefficiency map for the load sizes V below maximum efficiency. Logarithmic vertical axis.

Let us consider the part of the isoefficiency map for problem sizes smaller than $V_{\max}(m,n)$ as shown in Fig. 5.16. For such load sizes machines in set $M_1,\ldots,M_m$ compute on-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 (5.17)-(5.26) 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:

$$\frac{S+O+CV+a'V}{mS+nmO+mCV+a_1V} \leq \mathcal{E}(m,n,V) \leq \frac{S+O+CV+a'V}{mS+mO+a_1V}. \qquad (5.27)$$

In the numerator of (5.27) $a_1 \leq a' \leq a_2$ is an equivalent rate of processing on one machine. Product $mT(m,n,V)$ in denominator of (2.10) can be interpreted 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 (5.27) 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 $\mathcal{E}(m, n, V)$ in (5.27) 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. 5.16 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. 5.16 around $V = \rho = 6739$MB where a bunch of isolines coalesce. This is a result of using out-of-core memory while calculating $T(1, 1, V)$ used in the efficiency formula. At $V \approx \rho$ the single reference machine starts to use out-of-core memory which extremely expands $T(1, 1, V)$ and $I(e, m, n)$ has to increase only marginally to attain the required efficiency level. Consider, e.g., the upper bound of (5.27). 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, V grows very slowly in the isolines near $V \approx \rho$.

In the part of the isoefficiency map above $V_{\max}(m, n)$ (see Fig. 5.17) 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 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 m 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 is as small on each machine as possible.

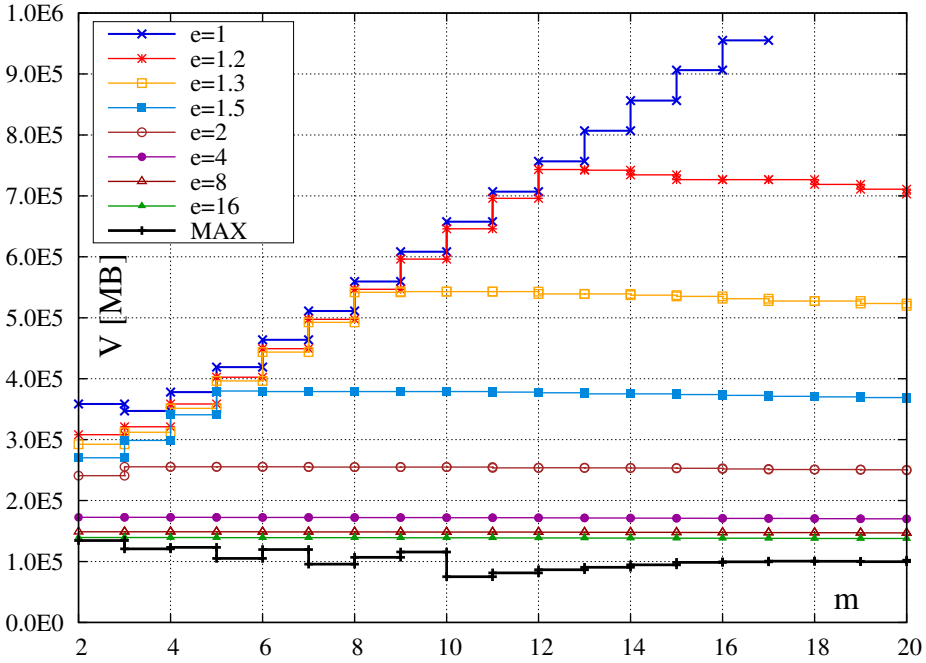


Figure 5.17: Isoefficiency map for the load sizes V above maximum efficiency.

This load partitioning pattern results in the following efficiency formula

$$\mathcal{E}(m,n,V) \approx \frac{S+O+V(C+a_2)+b_2}{mS+nO+CV+(n-m)\rho a_1+m[(V-(n-m)\rho)a_2/m+b_2]}. \quad (5.28)$$

In the denominator of (5.28) 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. Furthermore, $(n-m)\rho a_1$ is the area of computing in core, and $m[(V-(n-m)\rho)a_2/m+b_2]$ out-of-core. From (5.28) estimation of the isoefficiency function can be derived:

$$I(e,m,n) \approx \frac{b_2(en-1)+S(em-1)+O(en-1)}{(C+a_2)(1-e)}. \quad (5.29)$$

In the derivation of (5.29) property $\rho = b_2/(a_1-a_2)$ of (5.16) has been used. Note that $b_2 < 0, e > 1, n > m, |b_2| \gg S \gg O$, and $I(e,m,n) > 0$. Moreover, load size necessary for certain efficiency e is almost independent of the number

of machines m in (5.29). Thus, (5.29) represents well the bottom-right part of Fig. 5.17 where isoefficiency lines are nearly parallel to the horizontal axis. The top-left part of Fig. 5.17 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 $(Va_2 + b_2)/(m[Va_2/m + b_2]) \approx 1$ 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 (5.28). Yet, with decreasing m the amount of the overheads decreases and it is becoming impossible to build a schedule with some low efficiency level unless some idle time is added. Since introducing artificial idle time is counterproductive, we do not show isoefficiency lines for $e < 1$ in Fig. 5.17. The isoefficiency lines coalesced along top-right to bottom-left diagonal all represent schedules approaching the situation when (unneeded) idle times are kept in the schedule for efficiency e close to 1.

5.3 Conclusions

In this section the problem of scheduling divisible loads for the criteria of energy and makespan in homogeneous systems with memory hierarchy has been considered. The performance evaluation has shown that there is a trade-off between the two criteria. The time- and energy-performance is ruled by: i) sizes of load chunks which determine on-/out-of-core processing, ii) number of effectively usable processors which imposes lower bound on schedule length, iii) amount of idle time which affect wasted energy. The trade-off as well as the overall performance is ruled by a complex interplay between the speed and power of computing on-core vs out-of-core, costs of activating new machines, communication delays, and the size of the solved problem. It can be observed that in the wide ranges of system parameters parallel processing has a synergistic effect on energy and makespan: it is possible to economize on both criteria by adding

new machines. However, this phenomenon is limited to big size computations and short startup times. Bigger startup times quickly cut off chances for time and energy savings. Thus, parameters affecting the parallelism will also influence optimality of the schedules length. Moreover, it could be observed that the energy savings obtained by the change of one parameter, or one part of the system, are usually limited. Progress in all areas is needed for a steady reduction of power consumption required to fuel high performance computations and big data centers.

In Section 5.2.4 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 on-core which is faster than if the same load was put on one machine, necessarily out-of-core. For problem sizes smaller than the maximum efficiency size $V_{\max}(m, n)$, the efficiency decreases with increasing machine number. For problem sizes larger than the maximum efficiency size, the efficiency is almost independent of machine number. The idea of isoefficiency maps for systems with hierarchical memory can be extended to other pairs of system parameters than m and V as a future research subject.

Although almost all contemporary computer systems have hierarchical memory, representing this hierarchy seems a novel idea in scheduling and performance models of parallel computations. Thus, the scheduling model proposed here is a valuable instrument in analytical performance modeling of distributed systems. Since homogeneous systems were examined in this section, heterogeneous systems are the next and tempting research subject. These will be examined in Chapter 6.

6 Heterogeneous Systems with Hierarchical Memory

In this chapter load distribution strategies in heterogeneous systems with hierarchical memory are analyzed. The impact of system heterogenity is the key interest of this chapter. Firstly, simple heuristic algorithms using processor sorting rules are introduced. Secondly, Mixed Integer Linear Programming solutions using multi-installment distribution are presented. Then the effect of heterogenity on system performance is studied. Algorithms performance is compared in Section 6.4.

6.1 Mathematical Model

We assume that computations are performed in a single-level tree system with root M_0 and worker machines $M_1, \ldots, M_m$ in the leaves. The machines M_i can be in one of four states:

1. *idle* - consuming power P_i^I,
2. *starting* - which takes time S_i and power P_i^S,
3. *networking* - busy-waiting or receiving the load, using power P_i^N,
4. *computing* - when the received load is processed.

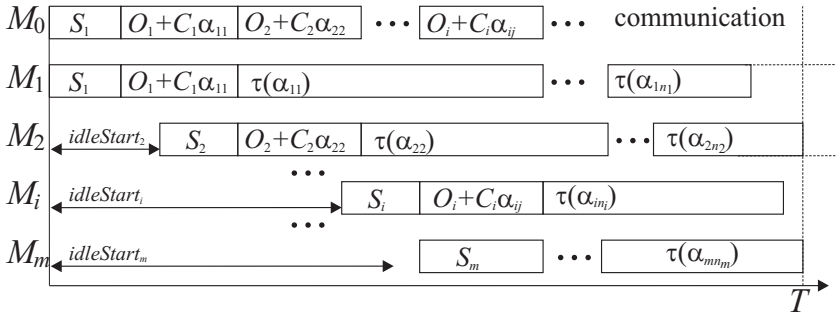


Figure 6.1: Multi-installment schedule. n_i denotes the last chunk sent to machine M_i.

Communications and computations proceed according to the following scheme (see Fig. 6.1). Initially volume V of load is held by the originator M_0 which is in the networking state, while machines $M_1, \ldots, M_m$ are idle. M_0 is not processing the load. M_0 is scheduling of the computations by sending load chunks to the chosen processors. If M_0 is capable of processing some load in parallel with communications, then this capability may be represented as an additional worker processor. M_0 activates the machines which takes energy

$$E_i^S = S_i P_i^S \tag{6.1}$$

on machine M_i. Note that not all machines have to be used. For example, some computer which is too slow, or is consuming too much power, may be kept idle. Let us observe that idle state energy should not be ignored because, unless completely disconnected from electric network, idle machines still contribute to the costs that the system owner must bear. The originator activates machines just-in-time which means that completion of the starting operation coincides with the beginning of receiving the load to process. The duration and energy cost of sending a wake up signal is negligible and starting some machine M_j can be performed in parallel with some other machine M_i communicating with M_0. Transferring α units of load to M_i takes time

$$t_i^{comm}(\alpha) = O_i + \alpha C_i, \tag{6.2}$$

where O_i (e.g. in seconds) is a fixed overhead also called communication startup time, C_i is communication rate (in seconds per byte). In the whole communication time machine M_i draws power P_i^N. Then, the energy cost of communication is

$$E_i^N = (O_i + \alpha C_i)\, P_i^N. \tag{6.3}$$

The computation starts after the entire load chunk is received. The load is distributed in a multi-installment manner and a processor may receive more than one load chunk. M_0 sends the load to the worker processors one at the time, i.e. the load is distributed sequentially. Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be the sequence of the communications, where σ_i is the index of the machine receiving the load in the ith communication from M_0 and n is the number of communications. As discussed in Section 3.3, time $\tau_i(\alpha)$ and energy $\varepsilon_i(\alpha)$ required for the computations on load chunk of size α on machine M_i are given by equations (3.1) and (3.2), respectively. We assume that the size of produced results is small and the time of returning the results to M_0 is very short compared to the whole schedule length. Hence, the result return operation need not be explicitly scheduled (result return can be easily tackled in DLT, see [22, 28, 74]). After processing the received load chunk a machine is busy-waiting to receive another load chunk. The busy-waiting is energetically equivalent with networking and M_i consumes power P_i^N in this state.

The problem considered here consists in constructing a schedule of minimum length T and energy E. Let $\mathcal{M} = \{M_1, \ldots, M_m\}$ be the set of worker processors. A scheduling algorithm for our problem must determine:

- subset $\mathcal{M}' \subseteq \mathcal{M}$ of machines participating in the computation,
- sequence σ of load distribution communications between M_0 and worker machines in $\mathcal{M}'$,
- the load chunk sizes.

This problem is effectively bicriterial. Multiple criteria problems can be handled in various ways [89]. In order to deliver the relationship between E and T, we will minimize one criterion – energy E, for a constrained value of the other criterion – schedule length T.

6.2 Solution Methods

In this section we introduce two types of strategies for load distribution. Firstly, we introduce two groups of fast heuristics which distribute load iteratively, choose the sequence of communications and regulate load chunk sizes according to some simple rules. These heuristics extend algorithms from Section 5.2.1 to the heterogeneous case. Secondly, a method of constructing an optimum multi-installment schedule is presented which sequences communications and sizes load chunks using mixed integer linear programming (MIP).

6.2.1 Fast Heuristics

Our heuristic algorithms are defined by the processor sorting rule (PSR) and the load chunk sizing algorithm. Beyond these two components, the mode of operation is similar as described in Subsection 5.2.1.

Initially all processors are idle. The originator starts the first idle machine on the PSR list and sends it a load chunk. The processors are activated until exhausting idle machines or until receiving a request for new load from some ready processor. Then, the originator sends load chunks to the ready processors first. A processor is ready at some moment t if it has already been started, received and processed its load chunk by t. Let $\mathcal{M}_R(t)$ be the set of processors ready at t. The originator ranks online processors in $\mathcal{M}_R(t)$ according to PSR and sends a load chunk to the processor on the topmost position. Consequently, processor M_i preferred by PSR may receive a new load chunk earlier than some other processor M_j which joined $\mathcal{M}_R$ before M_i. Let us also note that further

idle processors will be started when no processors are ready (because they are computing and the originator is not communicating with them). This procedure is repeated until exhausting load V.

Processor Sorting Rules (PSR). The considered processor priority rules order the processors according to: **k1** – non-decreasing k_{1i}, **k2** – non-decreasing k_{2i}, **l2** – non-decreasing l_{2i}, **a1** – non-decreasing a_{1i}, **a2** – non-decreasing a_{2i}, **b2** – non-decreasing b_{2i}, **C** – non-decreasing C_i, **S** – non-decreasing S_i, **PI** – non-decreasing P_i^I, **PN** – non-decreasing P_i^N, **PS** – non-decreasing P_i^S, **O** – non-decreasing O_i, **RAM** – non-increasing ρ_i, **Rnd** – order the processors randomly. The Rnd rule is introduced as reference method, to verify the utility of the other rules.

From the four load chunk sizing methods introduced in Section 5.2.1 we will use here only GSS and SSC. The former had performance (see Section 5.2.2) similar or better than SCU. The SCO method exposed strong performance irregularities. Hence, we omit SCO and SCU here. Let us shortly remind the idea of SSC and GSS.

Simple Static Chunk (SSC). This algorithm assumes that load chunk sizes are equal to the size of available RAM of the machine, i.e. $\alpha_i = \rho_{\sigma_i}$. Thus, SSC avoids using out-of-core memory. A disadvantage of SSC algorithm is lack of balancing the load in the final stage of computation. It means that in the last iteration many processors may be idle while a few processors strive with unnecessarily big load chunks. However, predicting which processors will be used in the last iteration is hard because the algorithm is running online in a heterogeneous system.

Guided Self-Scheduling Adaptation (GSS). Let V' be the size of the load remaining to be distributed and M_{σ_i} be the processor about to receive the ith load chunk. The size of the chunk is calculated as

$$\alpha_i = \min\{V', \max\{1, \min\{V'/m, \rho_{\sigma_i}\}\}\}.$$

By referring to the remaining load V', load chunk sizes decrease in the course of the schedule. Assuming that the initial size of the load is greater than memory size ($V' = V > \rho_{\sigma_1}$), the algorithm starts with load chunk sizes of RAM size. When $V' < \rho_i$, GSS gradually decreases chunk sizes and in this way equalizes the spread of machine completion times. GSS does not use chunk sizes smaller than some fixed size, by convention denoted as 1 in the above equation. In the further considerations we assume that it is 1MB.

Computational complexity of the heuristics is $O(V/\min\{\rho_i\}\log m)$ and $O(V \log m)$ for SSC and GSS, respectively. The $\log m$ component is a result of applying, e.g., processor priority queue and enforcing some PSR. Terms $V/\min\{\rho_i\}, V$ are upper bounds on the number of load chunks. Though complexity of the algorithms depends on V, they are very fast in practice as will be shown in Section 6.4.

It is possible to apply *all* the above PSRs and choose the best result. Some studies demonstrate [61] that combining many simple methods is a lightweight method of improving solution quality. Such heuristics will be referred to as **super-SSC**, or **super-GSS**, in the following text.

Let us now proceed to the technical matters of time and energy calculation. Since values of σ, α_i, T cannot be determined in the analytical way for a heterogeneous system, they are found a posteriori for some schedule $\mathcal{S}$, e.g., obtained by simulation or from runtime logs. Given schedule $\mathcal{S}$, the consumed energy is:

$$E = E_0 + \sum_{i=1}^{m} (E_i^I + E_i^S + E_i^R + E_i^N), \tag{6.4}$$

where: E_0 is the energy consumed by the originator, E_i^I is the energy consumed by M_i in the initial idle state, E_i^S is the energy consumed by machine M_i while starting, E_i^R is the energy consumed by M_i while processing the assigned load, E_i^N is the energy consumed by M_i in networking and busy-waiting. Since the originator can only communicate or busy-wait, $E_0 = P_0^N T$. The M_i energy in the idle state is $E_i^I = P_i^I idleStart_i$, where $idleStart_i$ is the time before M_i

begins waking up (see Fig. 6.1). If some machine is not used in the computation, then $idleStart_i = T$. The energy consumed in starting this machine is $E_{Si} = S_i P_i^S y_i$, where $y_i = 1$ if M_i participates in the computation, $y_i = 0$ otherwise. The energy consumed in computations on M_i can be calculated as:

$$E_i^R = \sum_{j:\sigma_j=i} \max\{k_{1i}\alpha_j, k_{2i}\alpha_j + l_{2i}\}.$$

In the busy-waiting and communications M_i draws power P_i^N. Hence,

$$\begin{aligned} E_i^N = P_i^N (T - idleStart_i - S_i y_i - \\ \sum_{j:\sigma_j=i} \max\{a_{1i}\alpha_j, a_{2i}\alpha + b_{2i}\}). \end{aligned} \tag{6.5}$$

6.2.2 Mixed Integer Linear Program

In this section we formulate the problem as a mixed integer linear program (MIP). Both divisible load scheduling [98], and mixed integer linear programming in general, are **NP**-hard. This means that according to the current state of knowledge (unless **P**=**NP**), to solve these problems to optimality exponential runtime algorithms are required. In our case, the worst-case computational complexity grows exponentially in the number of processors m and the number of installments n. However, for reasonable problem sizes MIPs can be solved fairly well by modern solvers. Thus, utility of MIPs must be assessed on the practical basis rather than by the worst-case pessimistic estimation. This will be subject of Section 6.4. The notations used in the following linear program are collected in Tab. A.3.

Given schedule length limit T, the minimum-energy schedule can be calculated by solving the following mixed integer linear program:

$$\min E = E_0 + E^R + E^I + E^S + E^N \tag{6.6}$$

subject to:

$$E_0 = P_0^N T \tag{6.7}$$

$$E^R = \sum_{i=1}^{m}\sum_{j=1}^{n} E_{ij} \tag{6.8}$$

$$\begin{gathered} E_{ij} \geq k_{1i}\alpha_{ij} \quad E_{ij} \geq k_{2i}\alpha_{ij} + l_{2i}x_{ij} \\ i = 1,\ldots,m \quad j = 1,\ldots,n \end{gathered} \tag{6.9}$$

$$E^I = \sum_{i=1}^{m}(idleStart_i P_i^I + idle_i P_i^N) \tag{6.10}$$

$$E^S = \sum_{i=1}^{m} S_i P_i^S y_{in} \tag{6.11}$$

$$E^N = \sum_{i=1}^{m}\sum_{j=1}^{n}(O_i x_{ij} + \alpha_{ij} C_i) P_i^N \tag{6.12}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1,\ldots,n \tag{6.13}$$

$$\alpha_{ij} \leq V x_{ij} \qquad i = 1,\ldots,m \quad j = 1,\ldots,n \tag{6.14}$$

$$\sum_{i=1}^{m} S_i x_{ij} \leq t_j \qquad j = 1,\ldots,n \tag{6.15}$$

$$t_j + \sum_{i=1}^{m} \alpha_{ij} C_i + \sum_{i=1}^{m} O_i x_{ij} \leq t_{j+1} \quad j = 1,\ldots,n \tag{6.16}$$

$$\begin{gathered} q_{ij} + C_i\alpha_{ij} + O_i x_{ij} + \tau_{ij} \leq q_{il} + (1 - x_{il})Z \\ i = 1,\ldots,m \quad j = 1,\ldots,n-1 \quad l = j+1,\ldots,n \end{gathered} \tag{6.17}$$

$$\begin{gathered} q_{ij} \leq Z x_{ij} \quad q_{ij} \geq 0 \\ q_{ij} \leq t_j \quad q_{ij} \geq t_j - Z(1 - x_{ij}) \\ i = 1,\ldots,m \quad j = 1,\ldots,n \end{gathered} \tag{6.18}$$

$$\begin{aligned} a_{1i}\alpha_{ij} + Z'u_{ij} \geq \tau_{ij} \quad \tau_{ij} \geq a_{1i}\alpha_{ij} \\ a_{2i}\alpha_{ij} + b_{2i}x_{ij} + Z''(1-u_{ij}) \geq \tau_{ij} \\ \tau_{ij} \geq a_{2i}\alpha_{ij} + b_{2i}x_{ij} \\ i = 1,\ldots,m \quad j = 1,\ldots,n \end{aligned} \tag{6.19}$$

$$\begin{aligned} q_{ij} + C_i\alpha_{ij} + O_i x_{ij} + \tau_{ij} \leq T \\ j = 1,\ldots,n \quad i = 1,\ldots,m \end{aligned} \tag{6.20}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_{ij} \geq V \tag{6.21}$$

$$\begin{aligned} idleStart_i \geq t_j - (1-(y_{ij} - y_{i,j-1}))Z - S_i y_{ij} \\ idleStart_i \leq t_j + (1-(y_{ij} - y_{i,j-1}))Z - S_i y_{ij} \\ j = 2,\ldots,n \quad i = 1,\ldots,m \end{aligned} \tag{6.22}$$

$$\begin{aligned} idleStart_i \geq t_1 - (1-y_{i1})Z - S_i y_{i1} \quad i = 1,\ldots,m \\ idleStart_i \leq t_1 + (1-y_{i1})Z - S_i y_{i1} \quad i = 1,\ldots,m \\ idleStart_i \geq T - y_{in}Z \quad i = 1,\ldots,m \\ idleStart_i \leq T + y_{in}Z \quad i = 1,\ldots,m \end{aligned} \tag{6.23}$$

$$x_{ij} \leq y_{ij} \qquad i = 1,\ldots,m \quad j = 1,\ldots,n \tag{6.24}$$

$$y_{ij} \leq y_{i,j+1} \qquad i = 1,\ldots,m \quad j = 1,\ldots,n-1 \tag{6.25}$$

$$y_{ij} \leq y_{i,j-1} + x_{ij} \qquad i = 1,\ldots,m \quad j = 2,\ldots,n \tag{6.26}$$

$$y_{i1} = x_{i1} \qquad i = 1,\ldots,m \tag{6.27}$$

$$\begin{aligned} idle_i = T - (idleStart_i + \sum_{j=1}^{n} C_i\alpha_{ij} + S_i y_{in} + \\ + \sum_{j=1}^{n} O_i x_{ij} + \sum_{j=1}^{n} \tau_{ij}) i = 1,\ldots,m \end{aligned} \tag{6.28}$$

Let us note that by use of binary decision variables x_{ij}, y_{ij} any subset of the m machines and any communication sequence of length n can be achieved. In the above MIP total energy usage is minimized by equation (6.6). Equations (6.7)-(6.12) define components of the energy consumption. In particular, by (6.8) energy spent in the computations is a sum of the energy parts E_{ij} spent on computing on load chunks j on machines M_i. The dependence of energy parts E_{ij} on load chunk sizes α_{ij} is defined by inequalities (6.9). Since E^R is minimized, (6.9) guarantees that $E_{ij} = \max\{k_{1i}\alpha_{ij}, k_{2i}\alpha_{ij} + l_{2i}\}$. Since $l_{2i} < 0$, term $l_{2i}x_{ij}$ in (6.9) makes the constraint more restrictive when $x_{ij} = 0, \alpha_{ij} = 0$ and helps easier solving the MIP. In (6.10) energy E^I is the sum of the energy $idleStart_iP_i^I$ used while waiting before starting the processors and energy $idle_iP_i^N$ consumed later in the busy-waiting. In (6.11), $y_{in} = 1$ only if processor M_i is used in some installment. Consequently, energy cost $S_iP_i^S$ of starting M_i is paid once, and only if M_i is indeed activated. Energy cost of networking is calculated in (6.12). By equation (6.13) only one machine may receive some load from the originator in installment j. Inequality (6.14) guarantees that a machine is not receiving any load if it is not chosen to take part in the jth communication. If a machine is chosen to take part in the computation, then by inequality (6.15) there is always enough time for starting the machine. The jth communication fits in time interval $[t_j, t_{j+1}]$ by (6.16). Let us remind that $q_{ij} = t_jx_{ij}$. Hence, constraints (6.17) ensure that if some machine M_i receives the jth and lth load chunks, then there is enough time between t_j and t_l to transfer the load to M_i and process it. Constraints in (6.18) serve the purpose of linearizing product t_jx_{ij}, that is, they guarantee that $q_{ij} = t_jx_{ij}$. Such a product cannot be directly used in a MIP because it would change the formulation into a quadratic programming problem. However, it is possible to substitute t_jx_{ij} with additional variable q_{ij} and constraints (6.18). Constraints (6.19) guarantee that processing load chunk j on M_i lasts for time $\tau_{ij} = \max\{a_{1i}\alpha_{ij}, a_{2i}\alpha_{ij} + b_{2i}\}$. Let us observe that for $u_{ij} = 0$ (6.19) guarantee that $a_{2i}\alpha_{ij} + b_{2i} \leq \tau_{ij} = a_{1i}\alpha_{ij}$ which implies $\alpha_{ij} \leq \rho_i$, and vice versa, for $u_{ij} = 1$, $a_{1i}\alpha_{ij} \leq \tau_{ij} = a_{2i}\alpha_{ij} + b_{2i}$

and $\alpha_{ij} \geq \rho_i$. The two big constants Z', Z'' in (6.19) have been chosen to avoid using arbitrarily large numbers, which are hard to operate upon for MIP solvers, and still make the formulation as tight as possible. Since for any feasible solution $\tau_{ij} \leq a_{2i}V$, the first big constant has been set to $Z' = (a_{2i} - a_{1i})V$ which guarantees that the first inequality in (6.19) is not binding if $u_{ij} = 1$. The Z'' constant, has been set to $Z'' = -b_{2i}$ which guarantees that $a_{2i}\alpha_{ij} + b_{2i}x_{ij} + Z''(1 - u_{ij})$ for $u_{ij} = 0$ is not binding because $a_{2i}\alpha_{ij} + b_{2i}x_{ij} - b_{2i} \geq a_{1i}\alpha_{ij} = \tau_{ij}$ because $0 > b_{2i}, a_{2i} > a_{1i}$. Inequalities (6.20) guarantee that computations on each machine finish before the end of the schedule. By (6.21) all work is executed. Inequalities (6.22), (6.23) serve the purpose of calculating time $idleStart_i$ which is the idle time before activating machine M_i (Fig. 6.1). Note that $(y_{ij} - y_{i,j-1}) = 1$ and the inequalities in (6.22) are binding only if $y_{ij} \neq y_{i,j-1}$ which happens if the jth communication is the first message sent to M_i. Thus, $idleStart_i = t_j - S_i$ where j is the first load chunk sent to M_i. Inequalities (6.23) are a boundary case of (6.22). The constraints in (6.24)-(6.27) define the trigger variables y_{ij}, used in (6.11), (6.22), (6.23), (6.28) such that y_{ij}s are equal to 0 before the first message sent to M_i and equal to 1 from the first message sent to M_i on. Equations (6.28) allow to calculate the length of busy-waiting intervals on machines M_i (used in (6.10)).

For practical matters, let us note that the big constant Z, used in constraints (6.17), (6.18), (6.22), (6.23) can be substituted with T if (6.6)-(6.28) is solved for some known value of T. MIP (6.6)-(6.28) can be reformulated to calculate minimum schedule length T, or to minimize T subject to a limit on the usage of energy E. In the former case (T is minimized), constraints (6.7)-(6.12) are removed, while the big constant Z can be set to some upper bound on the schedule length. For example, $Z = 2(\max_{i=1}^{m}\{S_i\} + V\max_{i=1}^{m}\{C_i\} + n\max_{i=1}^{m}\{O_i\} + \max_{i=1}^{m}\{\max\{a_{1i}V, a_{2i}V + b_{2i}\}\})$. In the latter case (minimization of T subject to E), T is again minimized while equation (6.6) becomes a constraint.

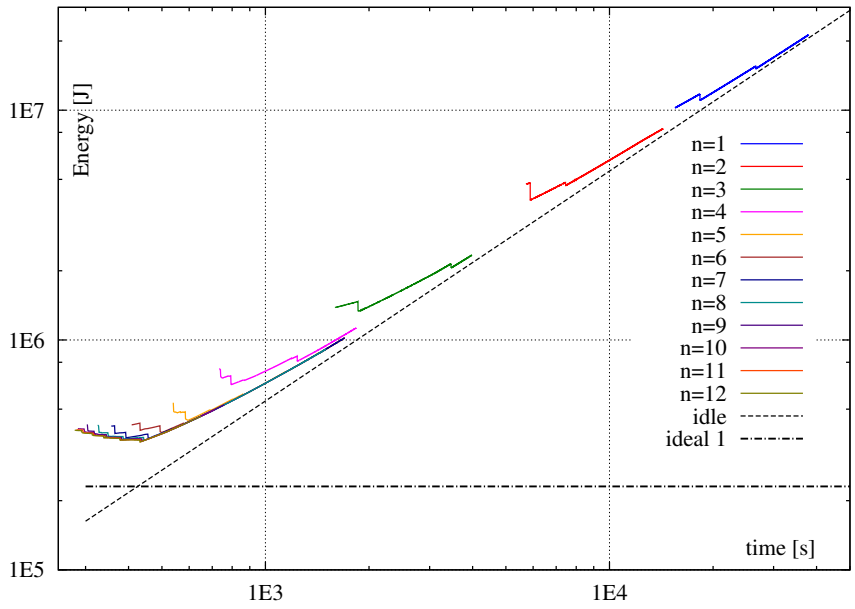


Figure 6.2: Energy vs time for changing number of installments n. Logarithmic axes.

6.3 System Performance Modeling

Parameters of heterogeneous systems on various divisible applications may differ significantly (cf. Tab. 3.3). Therefore, rather than discussing particular numbers we will analyze tendencies which appear in many test instances. The behavior of optimum problem solutions will be demonstrated on the example data shown in Tab. 6.1. The values in Tab. 6.1 have been generated pseudo-randomly. Generating test instances is described in more detail in the next section. Unless said to be otherwise $V = 24\text{GB}$. The time-energy trade-offs presented in this section were obtained by first finding the minimum schedule length T^* for the given m, n limits as described at the end of Section 6.2.2, and then MIP (6.6)-(6.28) has been applied to calculate minimum energy consumption for test values of T starting with T^* and increasing with a step of 1s. Gurobi version 7.5.2 with at most 0.2% optimality gap has been applied as the MIP solver.

Table 6.1: Test system parameters.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
a_{1i} [s/MB]	0.111677	0.122611	0.141048	0.070698	0.056562	0.114364	0.102537	0.085978	0.102269	0.087398
a_{2i} [s/MB]	3.078402	0.876470	4.887229	0.905717	2.125928	1.586452	2.126140	2.235132	1.151396	1.396836
b_{2i} [s]	-5227.37	-5642.63	-11272.18	-3623.98	-10913.84	-11577.97	-12467.41	-10427.70	-7262.05	-7199.29
k_{1i} [J/MB]	15.729	16.569	17.631	10.552	9.427	14.662	15.079	13.027	16.495	13.656
k_{2i} [J/MB]	377.0241	106.7044	473.5687	117.5493	281.2074	149.5524	309.5719	310.6940	208.6618	194.8711
l_{2i} [J]	-636602	-674663	-1082852	-464368	-1433370	-1060913	-1814371	-1444280	-1330178	-9963201
C_i [s/MB]	0.007	0.002	0.004	0.009	0.003	0.005	0.003	0.002	0.004	0.009
S_i [s]	62	80	3.4	1.1	2.3	1.2	65	5.1	75	82
P_i^I [W]	4	6	88	71	95	70	2	93	5	4
P_i^N [W]	99	90	96	93	113	113	108	120	111	95
P_i^S [W]	98	108	115	96	99	114	120	91	93	108
O_i [s]	0.62	0.8	0.034	0.011	0.023	0.012	0.65	0.051	0.75	0.82
ρ_i [MB]	1762	7485	2375	4340	5274	7865	6161	4852	6922	5498

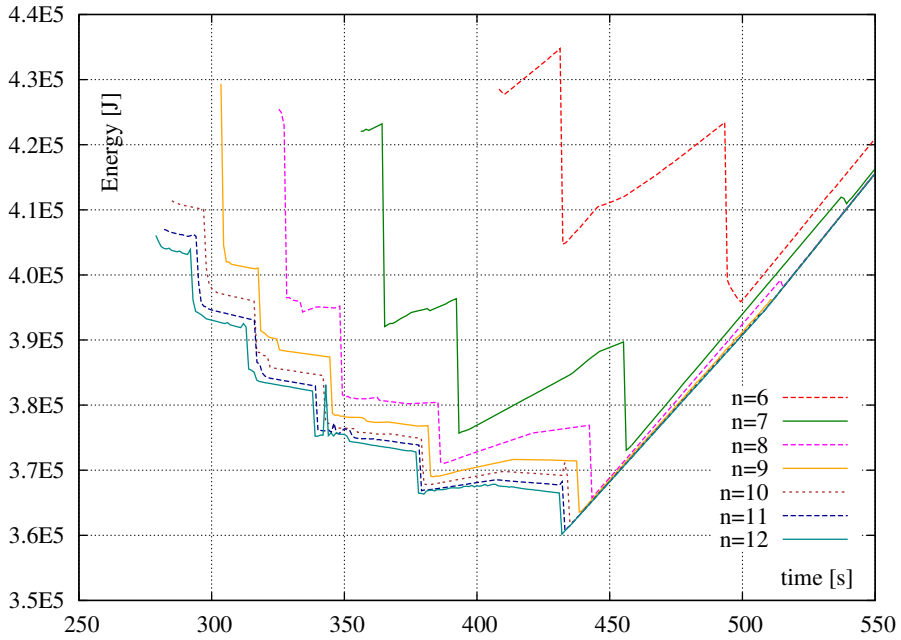


Figure 6.3: Closeup on energy vs time for changing number of installments n.

In Fig. 6.2 and Fig. 6.3 dependence of total energy usage on schedule length T for different numbers of load chunks n is shown. In Fig. 6.2 two lower bounds are also shown: a diagonal line representing energy usage if all processors remained idle (denoted "idle"), and a horizontal line of energy usage if all work were done in RAM on an ideal processor with all the best parameters of the given processors and no other processors were present (denoted "ideal 1"). The following phenomena can be observed in Fig. 6.2 and Fig. 6.3: Time–energy functions are not convex and may have many local optima, "cliffs" in energy usage separate intervals of monotonous time-energy dependencies. The existence of many local optima and non-monotonic nature of the relationships make optimization efforts hard. The "cliffs" appear when schedule length T is sufficiently large to switch off some machine (this will be discussed in the following text, see Fig. 6.5). The intervals of decreasing E with increasing T, e.g. at $T = 300$ for $n = 10, 11, 12$ in Fig. 6.3, represent the opportunity of shifting the load from faster but more energy-intensive machines to the slower but more energy-economic ones. The

fact that energy consumption increases with schedule length is a result of busy-waiting and idle processors. We will return to the impact of idle processors power in the following (Fig. 6.8). It can be observed in Fig. 6.2 and Fig. 6.3 that with increasing number of load chunks n performance of multi-installment processing improves, however, the returns are diminishing. This is intuitively expected because more installments mean smaller load chunks, shorter communications, earlier starting of the computations, and using less the out-of-core memory. The minimum number of chunks which allow processing the whole load in RAM is $n = 4$ here, but the processors with the biggest main memory are not necessarily the fastest or best in the energy efficiency sense. Thus, if schedule is long enough, it may be still profitable to use out-of-core memory even if RAM size is sufficient (this will be discussed in the following, cf. Fig. 6.6). It can be verified in Fig. 6.3 that using more than $n = 8$ installments does not give substantial performance gains. Thus, a small multiple of the minimum number of chunks which allow fitting the load in core memory should be sufficient in typical cases.

In Fig. 6.4 distribution of the load between the processors is shown with changing schedule length T for $n = 10$ installments. The total load size obtained in many messages is shown. In Fig. 6.4 dependence of energy E on schedule length T is also shown to allow coordinating the "cliffs" in energy usage with the changes of load distribution. Fig. 6.4 serves the purpose of illustrating how complex processor load distribution interplay in heterogeneous systems can be. It can be verified in Fig. 6.4 that steep reductions in energy usage coincide with removing processors from computation (load assignments become zero). The case of machines M_1, M_7, M_9, M_{10} in Fig. 6.4 (shown as continuous lines) is illustrative. At $T = 318$ M_1 is switched-off and its load is taken over by M_9 which is needed as a faster machine. But then at $T = 323$ the schedule is long enough to give up M_9 and use a slower but more energy-efficient M_1. As longer schedules are allowed (T grows) machine M_1 gives its load to the other, more energy-efficient machines, and is switched off at $T = 343$. This situation

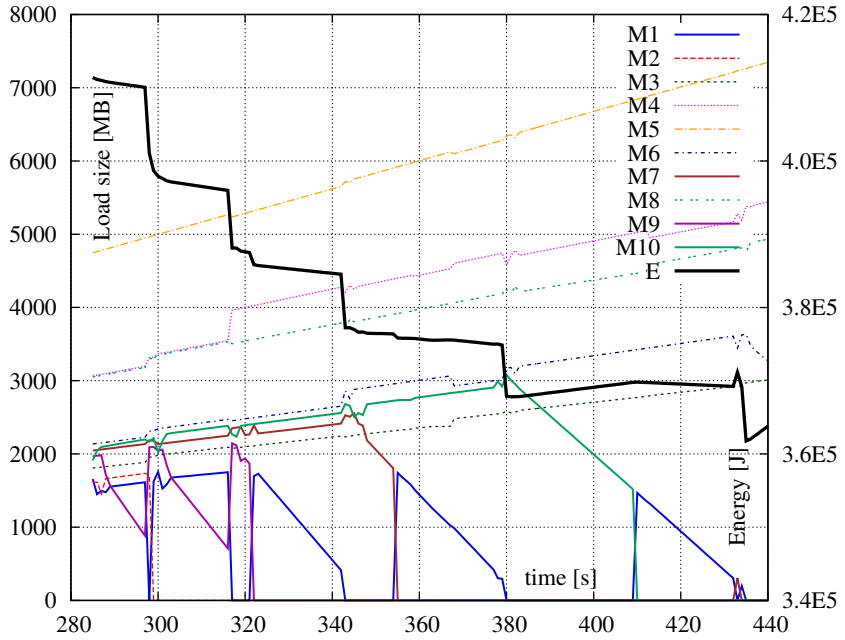


Figure 6.4: Distribution of the load between machines and energy consumption vs schedule length T. The left axis is for load sizes, the right axis is for energy E. Number of installments: $n = 10$.

Table 6.2: Processors omitted in the best schedules vs schedule length T, for $m = 10$ machines.

end of T range	292	313	338	346	377	379
omitted	none	2	2,9	1,2,9	2,9,10	1,2,7,9
end of T range	431	468	540	560	665	945
omitted	2,7,9,10	1,2,7,9,10	2,7-10	1,2,7-10	2,6-10	1,2,6-10

is repeated twice more: M_1 is switched on again at $T = 355$ to take the load of M_7, switched off at $T = 381$, and switched on once more at $T = 410$ to substitute M_{10}, and off at $T = 433$. Thus, machine M_1 which is comparatively slow but more energy-efficient than the other machines acts as a substitute for the faster M_7, M_9, M_{10}.

In Fig. 6.5 changes of energy usage with schedule length are shown for increasing sets of available machines. This means that from the machines in Tab. 6.1, subset $\{M_1, \ldots, M_m\}$ were available for the computation, where $m = 6, \ldots, 10$. Labels added at the relationships are indices of the used proces-

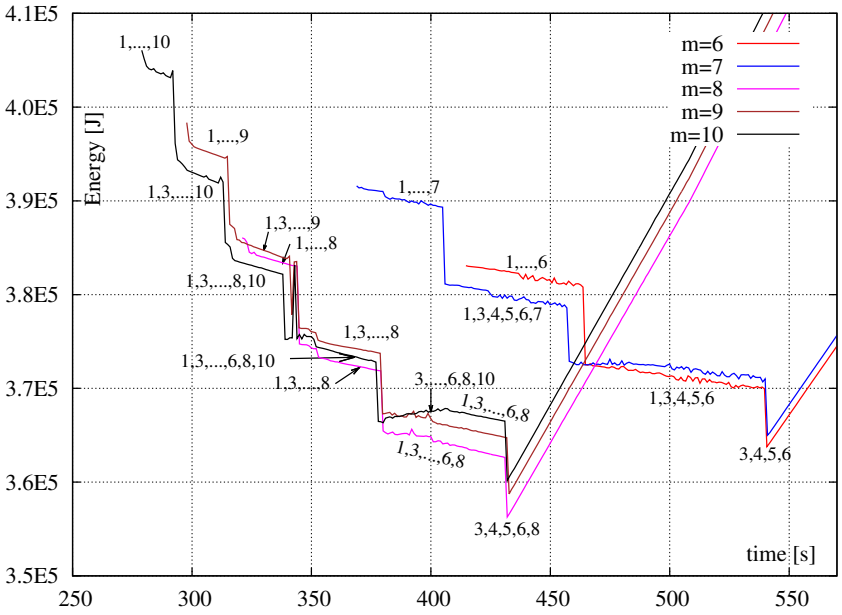


Figure 6.5: Energy vs time for changing number of machines m at the number of installments $n = 12$.

sors. For better clarity indices of machines absent in the optimum working set vs changing T are listed in Tab. 6.2 for $\{M_1, \ldots, M_{10}\}$. It can be observed that adding machine M_8 to the working set was profitable and the minimum energy use point moved to shorter schedules (see Fig. 6.5, $T \approx 430$). When schedules get shorter (see Tab. 6.2), additional machines are switched on and energy usage instantly increases as a direct result of the starting energy cost. Observe pivotal role of M_1 which joins and leaves the best processors (cf. Tab. 6.2). It is worth observing that in this heterogeneous system the trajectory of increasing energy E with shortening schedule length T can be different depending on the actual set of available machines. For example, for the set of $m = 10$ machines the energy used can be bigger than (because of the cost of holding more machines), smaller than (because the schedule is shorter), or in-between (a subset of the extra machines work) the energies consumed by smaller sets of $m = 8, 9$ machines (see Fig. 6.5 in T range [300,400]). Thus, a bigger set of machines gives

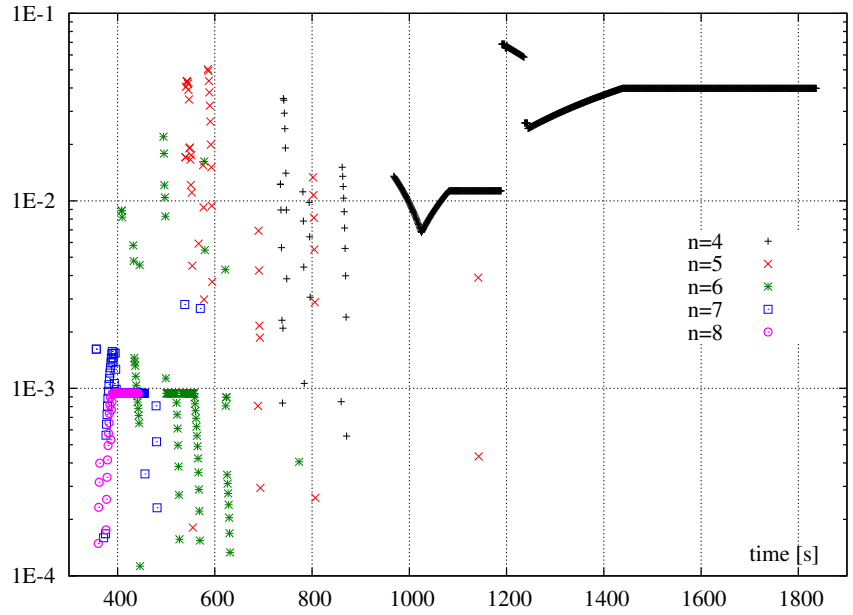


Figure 6.6: Out-of-core memory usage vs time for changing number of installments n.

an advantage of choosing more energy-efficient subset of processors, but also a disadvantage of holding machines which are available but not engaged in the computation.

Let us consider now the utility of using out-of-core memory. Even if the number of load chunks n is sufficient to process the whole load in the core (here $n \geq 4$), it may be profitable to use some out-of-core memory. This may allow for fewer communications or for starting fewer processors. In Fig. 6.6 usage of the out-of-core memory is shown for various schedule lengths T and installment numbers n. The fraction $\max_{i,j}\{\alpha_{ij}/\rho_i - 1\}$ is shown along the vertical axis. In other words, the relative excess of load chunk sizes α_{ij} over RAM size ρ_i is shown. For $n = 4$ installments and $T > 1240$ it is possible to process three load chunks in main memory of one processor while the remaining load is processed out of core in another processor, which allows to avoid starting other machines. Thus, in this case out-of-core memory usage persists though it seemed unnecessary. However, it can be seen in Fig. 6.6 that indeed with

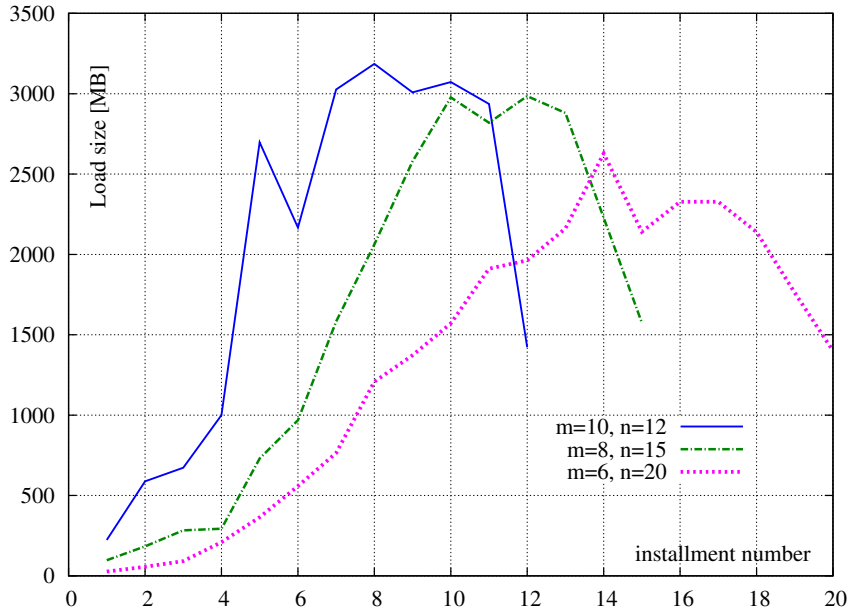


Figure 6.7: Average chunks sizes vs installment number for selected numbers of installments n and machines m.

increasing number of installments n the use of the out-of-core memory ceases. The above observations can be rephrased as confirming presence of a trade-off between the cost of starting more processors or using the less efficient out-of-core memory.

In Fig. 6.7 average sizes of installments are shown for different combinations of processor set size m and the number of installments n. The averages have been taken over schedule lengths T from the minimum schedule length T^* to the minimum-energy E^* schedule length (e.g. for $m = 10, n = 12$ for all the points $T = 278, \ldots, 432$ with a step of 1s, see Fig. 6.5). Each point in Fig. 6.7 represent an average from at least 110 samples. A common pattern in the shape of the relationships can be observed in Fig. 6.7: Initial installments are small, and they grow slowly until reaching a maximum in the second part of communication sequence. The sizes of the final installments quickly fall. The

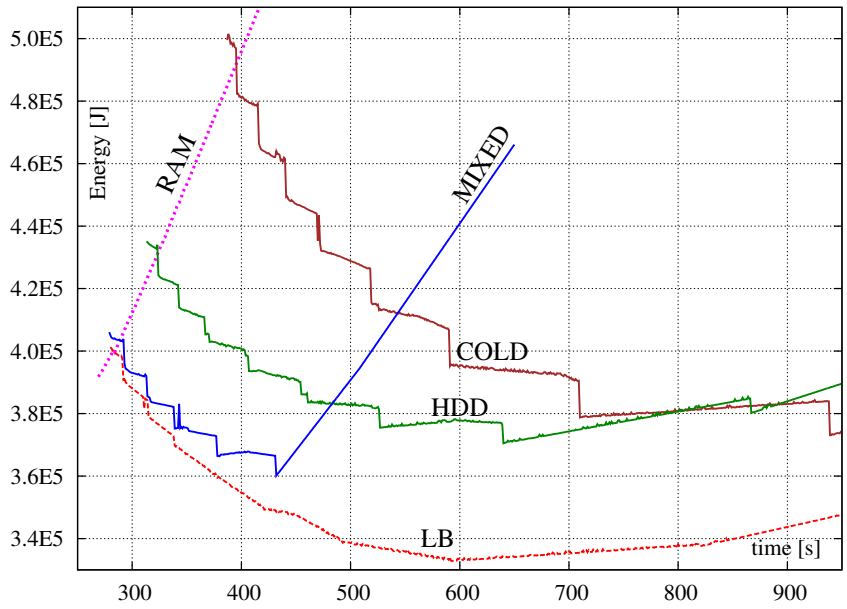


Figure 6.8: Energy vs time for various idle state power P_I configurations at $m = 10$ machines, and $n = 12$ installments.

small initial chunks allow to start the computations quickly by avoiding lengthy data transfers. The trailing messages are shorter to allow balancing the load on the machines which receive some work as the last one(s).

Let us return to the impact of idle processors power P_i^I on the time–energy trade-off. Even idle processors consume energy which is signified by the lower bound "idle" in Fig. 6.2. In Fig. 6.8 the time–energy trade-off is shown for the previous instance with various P^I, S_i settings. The original instance is shown as "mixed". The same instance with machines starting from hibernation to disk are denoted "HDD" ($S_i \in [40, 90]$s, $P_i^I \in [1, 7]$W, see the next section for details of test instance generation). The setting corresponding to starting from a suspension to RAM ($S_i \in [0.1, 6]$s, $P_i^I \in [70, 100]$W) are denoted "RAM". The case for cold-starting (as if the machines were disconnected from electric power, $P_i^I = 0, S_i = 120$s) is shown as "cold". Finally, the same instance with all processors idle power changed to $P_i^I = 0$, but retaining their original startup times is an optimistic lower bound (LB) on possible idle power vs startup time

cases. Hence, the LB line shows potential for possible improvements by modifications of energy-saving modes. It can be observed that the lower the idle power P_i^I is, the wider the range (both in time and in energy) between the shortest and the lowest-energy schedules. The more realistic cases, where P_i^I is bigger, have narrower options for optimizing the energy usage. Suspending machines to RAM allows for only marginally shorter schedules, but the energy cost only increases with T as P_i^Is are not significantly smaller than the power consumed in communication or computation. The cold- and HDD- start do not dominate the original case (mixed) because of long, and consequently, energy-costly startups. Therefore, short schedules, even though compute- and power-intensive, can be more effective than long schedules with many machines mostly in low energy mode. Furthermore, using a mix of machines – some in shallower and some in deeper suspension (mixed case) – is advantageous because it allows for a quick start of computations using the machines in shallower suspension, while simultaneously activating the machines in a deeper sleep mode. This conclusion is supported by a statistically significant correlation between the position in communication sequence and parameters S_i, P_i^I (e.g. machines with smaller S_i receive load earlier), and lack of strong correlation with other parameters. In general, keeping machines in idle state should be avoided (which is the strategy of many cloud infrastructure providers).

6.4 Algorithm Performance Comparison

In this section we compare performance of our algorithms. Quality of the schedules and the time taken to find them will be evaluated. Quality of the schedules is measured by schedule length T and energy usage E. In general, there is no unique way to compare performance of two algorithms constructing solutions as a trade-off between two criteria [100]. Moreover, it is not possible to reduce such trade-offs to one dimension (i.e. a single numerical score) without loosing some information [100]. In the case of MIPs, it would require considering

three-dimensional trade-off: between the solving algorithm runtime, the obtained schedule length T and the used energy E. Though conceptually possible, it would be unwieldy. The fast heuristics introduced in Section 6.2.1, however, construct only a single solution for a given instance, not a trade-off between T and E. This restricts options for algorithm performance comparison. Hence, in the further discussion we will reduce algorithm performance examination to just two reference points of practical importance: the shortest schedules (minimum schedule length T^*) and the lowest energy schedules (minimum energy E^*). Solution quality will be measured either as a distance from the shortest known schedule length T^*, or as the distance from the lowest known energy usage E^*. The algorithm runtime to construct the schedules, how this runtime is traded for solution quality, and sources of solution inefficiency will be discussed in the following.

To this end, a set of instances have been generated pseudorandomly. The numbers of machines were $m \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$. For each number m a set of 30 test instances has been generated, with $\rho_i \propto U[256, 8048]$MB, $k_{i1} \propto U[9, 18]$J/MB, $k_{i2} \propto U[45, 540]$J/MB, $a_{i1} \propto U[0.05, 0.15]$s/MB, $a_{21} \propto U[0.3, 4.4]$s/MB, $C_i \propto U[0.001, 0.01]$s/MB, $O_i \propto U[1\text{E-}6, 0.02]$s, $P_i^S, P_i^N \propto U[90, 120]$W, where $\propto U[a, b]$ means that the value was drawn from a uniform distribution in range $[a, b]$. Values b_{2i}, l_{2i} were calculated so that the two linear components of execution time and energy consumption, respectively, intersect at ρ_i (cf. Section 3.3). With probability 0.5 a machine was chosen to have a short startup (e.g. because it is suspended to RAM), otherwise it had a long startup (as if starting from hibernation to HDD). In the former case, $P_i^I \propto U[70, 100]$W and $S_i \propto U[0.1, 6]$s. In the latter case, $P_i^I \propto U[1, 7]$W and $S_i \propto U[40, 90]$s. Hence, idle power P_i^I and wakeup time S_i have been correlated in this way that machines with low P_i^I need more time to wake up, and vice versa, short wakeup time S_i is possible in shallower suspension mode using higher power. Unless stated to be otherwise $n = 12$ has been assumed for the MIP model (6.6)-(6.28). Gurobi version 7.5.2 has been used as the MIP-solver using 6 par-

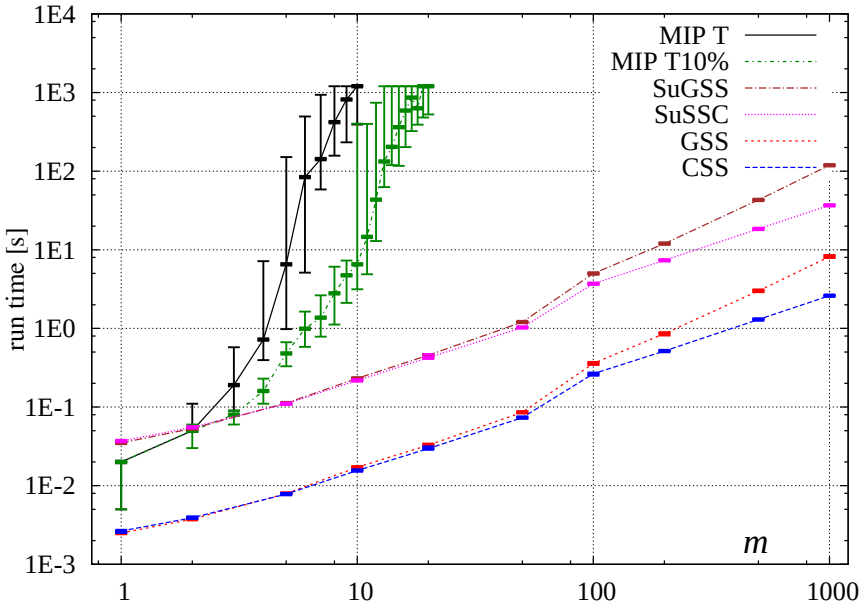


Figure 6.9: Algorithm runtimes vs number of machines m. Logarithmic axes.

allel threads and runtime limit of 1200s. Two versions of the model have been solved: with MIP optimality gap set to 0.2% and with the gap set to 10%. The experiments have been conducted on a PC computer with Intel i7@2.8GHz and Windows 7.

Algorithm runtime quartiles Q1, Q2 (median), Q3 vs changing number of machines m are shown in Fig. 6.9. For the algorithms based on MIP (6.6)-(6.28) runtimes for finding minimum schedule lengths T^* with 0.2% MIP gap (denoted as "MIP T") and 10% MIP gap (MIP T10%) are shown. The runtimes for finding the lowest energy schedules are similar and have been omitted to avoid cluttering the picture. As it can be seen the runtime cost of solving the MIP model quickly increases and the median runtime reaches the 1200s limit at $m = 9, n = 12$ for MIP gap 0.2%. Relaxing the optimality requirements (MIP T10%) helps, but still median runtime reached the time limit around $m = 20$ processors. Were it not for the limit of 1200s, it should be expected the MIP solver runtime would continue exponential growth. Thus, the MIP model can be used for moderate size instances. Conversely, the SSC and GSS

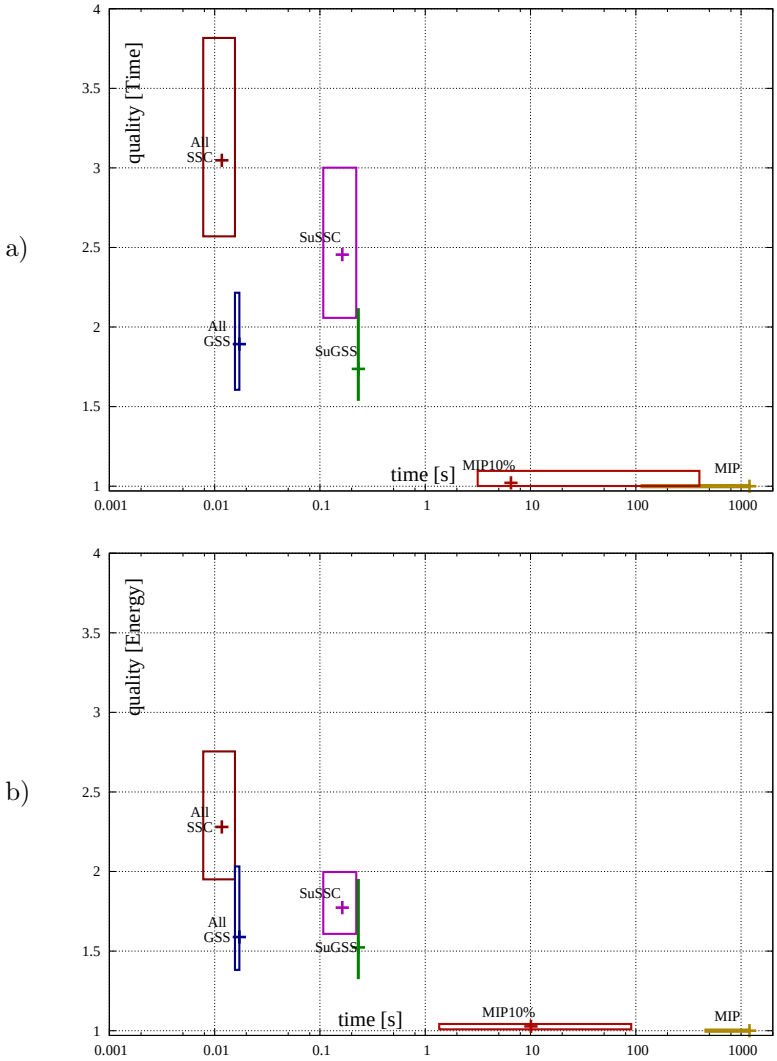


Figure 6.10: Solution quality vs algorithm runtime for a) schedule length criterion, b) energy criterion. Logarithmic runtime axes. $m = 10$ machines.

heuristics are by far faster and distribute the load to $m = 1000$ machines in less than 10s. The super-SSC and super-GSS (denoted SuSSC, SuGSS, respectively) are by an order of magnitude slower than SSC and GSS because they execute the heuristics with all processor sorting rules (PSRs). The dispersion of the heuristics' runtime is also much smaller and Q1, Q2, Q3 overlap in the picture. Now we should examine how these runtime costs are exchanged for the solution quality.

The trade-off between solution quality and algorithm runtime cost is shown in Fig. 6.10. Computational complexity of solving the problem depends on m, n, and in the case the heuristic algorithms also on V. In order to avoid concealing the algorithm runtime vs solution quality relationship by this dependence of computational complexity on m, n, V, these parameters have been set to $m = 10, n = 12, V = 24000$MB. In Fig. 6.10 boxes represent algorithm runtimes (horizontally) and relative distance from the best obtained solution (vertically) on a population of 30 test instances. The boxes span from quartile Q1 to Q3 in time (horizontally). Quality span is represented analogously along the vertical dimension. Medians (Q2) of runtime and quality are also marked. There are 14 PSRs of the fast heuristics SSC and GSS. Statistical analysis (ANOVA) revealed that neither in the runtime nor in the solution quality has any sorting rule a statistically sound advantage for the considered m, n, V. Thus, to avoid cluttering the picture the results of all the sorting rules are put together in boxes distinguishing only SSC and GSS methods (denoted "All SSC", "All GSS", respectively). The results for the super-SSC and super-GSS methods, which choose the best result among all processor sorting rules, are shown as SuSSC and SuGSS, respectively. It can be seen in Fig. 6.10 that the solutions constructed by solving the MIP model (6.6)-(6.28) are always the best with respect to the solution quality. But this guarantee of quality comes at the cost of the runtime, the highest among all the studied methods. Relaxing the MIP gap to 10% (MIP 10%), helps with respect to the runtime with only minor loss in solution quality. However, as shown in Fig. 6.9, this approach has limited scalability because at $m = 20$ also the relaxed MIP model exceeds the 1200s time limit. The heuristic solutions are on average 1.7 – 3 times worse in schedule length T (Fig. 6.10a) and 1.5 – 2.3 times worse on energy E criterion (Fig. 6.10b). Conversely, heuristic methods are by 4 – 5 orders of magnitude faster than solving the MIP model. It can be observed that GSS algorithm is better than SSC, but it is slightly slower (approx. 50% longer runtimes). The super-SSC improves solution quality compared to the original SSC methods, but it is still

Table 6.3: Idle time fraction in schedule length.

Quar-	All machines			Only active machines		
tile	MIP	SSC	GSS	MIP	SSC	GSS
Q1	0.0001	0.4736	0.3530	0.0001	0.2507	0.3530
Q2	0.0326	1	0.5342	0.0313	0.4505	0.5342
Q3	0.1124	1	0.7177	0.1061	0.6825	0.7177

worse both in solution quality and runtime (when referring to time the medians) than the original GSS methods. The super-GSS is only marginally better in solution quality than the original GSS methods. It can be concluded that the GSS methods dominate other heuristics.

Let us now analyze the sources of heuristic algorithm inefficiency with respect to solution quality. In Tab. 6.3 quartiles Q1, Q2, Q3 of the fractions of the schedule length spent in busy-waiting or idle are shown. The fractions were collected over a population of 30 instances with $m = 10$ compared to the minimum length reference solution. In the case of heuristic algorithms all processor sorting rules were considered. This statistic has been collected for all available machines (even if some of them were not used), and separately, only for the machines which indeed took part in the computation. It can be seen that the solutions from the MIP model have very little idle time. The results for all machines and for the activated machines do not differ much. This signifies that MIP schedules almost always use all available processors, idle times are short if any, communications and computations are very well coordinated to avoid idle times. These quality results come at twofold price: computational complexity of solving the MIPs, and benchmarking the application and the platform to obtain precise data used in the model. Conversely, SSC quite often has long idle time. In particular, values 1 mean that some processors are not activated at all. This is confirmed by the fact that overall amount of idle time in SSC schedules decreases if only the active processors are considered. Thus, SSC has a potential

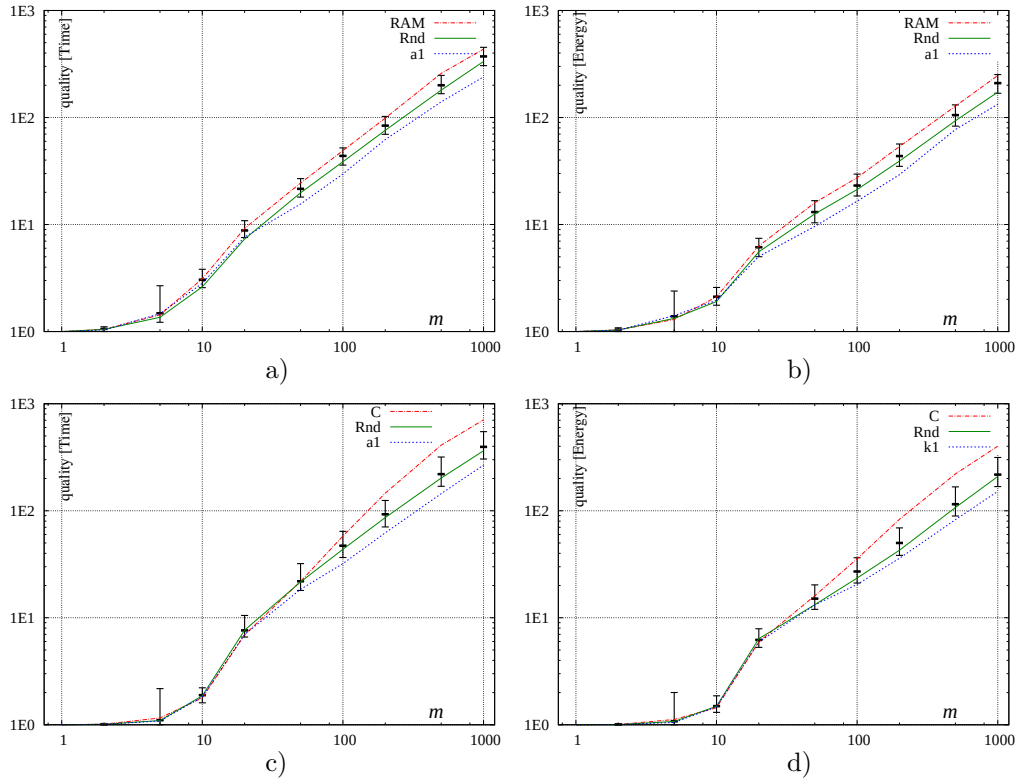


Figure 6.11: Time and energy quality among the heuristics vs processor number m. a) SSC-heuristics schedule length, b) SSC-heuristics energy; c) GSS-heuristics schedule length, d) GSS-heuristics energy. Continuous lines show quality medians of the worst, *Rnd*, and the best heuristics. Logarithmic axes.

for improvement by tuning the set of used processors. GSS schedules involve almost all processors which is signified by equal values both when all machines, and if only the active ones are taken into consideration.

In Fig. 6.11 evaluation of heuristic solutions quality is extended to bigger processor numbers m. Relative distance from the lower bound is shown along the vertical axes. The lower bound of schedule length is

$$LB(T) = S_{\min} + O_{\min} + \rho_{\min} * C_{\min} + V/(\sum_{i=1}^{m} 1/a_{1i}), \qquad (6.29)$$

where $S_{\min}$, $O_{\min}$, $\rho_{\min}$, $C_{\min}$, are minimum machine startup time, minimum communication overhead, minimum RAM size and communication rate in the processor set, respectively. In (6.29) it is assumed that the shortest possible

communication is done and then, whole load V is processed on all machines in parallel. The energy lower bound is

$$LB(E) = S_{\min} P^S_{\min} + (O_{\min} + VC_{\min}) P^N_{\min} + V k_{1\,\min} + P^N_0 LB(T), \quad (6.30)$$

where $P^S_{\min}$ is minimum machine starting power, $P^N_{\min}$ is minimum networking power, and $k_{1\,\min}$ is the minimum energy per load unit in the processor set. In (6.30) it is assumed that the least energy-costly machine startup is executed, all the load is transfered over the least energy-consuming communication link and the load is processed on the most energy-efficient processor. In Fig. 6.11 quartiles Q1, Q2, Q3 for the population of all the method solutions are shown. Moreover, the medians (Q2) of the best, the worst, and Rnd processor sorting rules are shown as continuous lines. It can be seen that the median of all method does not differ much from the performance of Rnd order. Some methods slightly distinguish themselves both in positive and in the negative sense. For example, the order of increasing on-core computing rates a_{1i} (that is decreasing computing speeds) allows for a bit shorter schedules (Fig. 6.11a, Fig. 6.11c), and consequently more energy-efficient solutions among the SSC-heuristics (Fig. 6.11b). For these cases sorting rule $a1$ improves schedule length and energy quality measures by roughly 30% related to the Rnd median quality. Depending on the number of processors, results of $a1$ are in the lowest 10-30% of the results population of Rnd sorting rule. Similar observations can be done for GSS algorithm with the $k1$ rule (Fig. 6.11d). It can be concluded that the quality of solutions generated by the heuristics is similar, but PSRs $a1$, $k1$ have some modest advantage.

6.5 Conclusions

In this section time- and energy-performance of processing data-parallel computations in heterogeneous systems with hierarchical memory has been studied. Hierarchical memory subsystems incur complex dependence of the running time and energy consumption on the size of solved problem. These dependencies have been represented by piecewise-linear functions. The computation scheduling problem has been rendered as an optimization problem consisting in selecting the set of activated processors, the sequence of processor communications, and sizing the load chunks. Two approaches have been proposed: solving a MIP formulation, or applying fast heuristics. The results obtained indicate that due to the existence of idle processors which still consume some power, there are sharp local optima in the energy vs schedule length trade-offs. Hence, short schedules, even though compute- and power-intensive, can be more effective than long schedules with some machines in low-energy computing mode and some other in the idle mode. Moreover, holding machines in a diverse set of energy modes is advantageous because the machines in shallow suspension can quickly start the computations, while simultaneously starting the machines in deeper suspension modes. It has been also established that limited use of the out-of-core memory may be beneficial by limiting communications or activating fewer machines. The performance of the scheduling algorithms is determined by various factors. The schedules obtained by solving an MIP are almost always the best, but this dominance comes at cost: MIP runtime and the need for information on the model parameters. The fast heuristics proposed here build solutions approx. 3 times worse than the MIP's with respect to solution quality, but in 4–5 orders of magnitude shorter time. Among the fast heuristics, GSS methods offer good trade-off between solution quality and runtime, sending the load to the fastest or the least-energy consuming processors is moderately advantageous.

7 Summary and Final Remarks

In this thesis problems of scheduling divisible computations in systems with hierarchical memory, for the criteria of energy and time performance were analyzed. The computation time and energy models for machines with hierarchical memory were not assumed but constructed on the basis of measurements on practical algorithms in real computer systems. These models are piecewise-linear functions of the size of load (data) to be processed. Divisible load processing both in homogeneous and heterogeneous systems was considered. Two types of scheduling algorithms for load processing were proposed: fast greedy algorithms and mixed integer linear programming-based methods. Results of the performance modeling confirm existence of a trade-off between time and energy criteria. However, it was also shown that in many cases both energy usage and schedule length can be reduced by increasing parallelism of the computations. In heterogeneous systems frequent irregularities of schedules construction were observed. The impact of various computing platform components on time and energy performance in processing divisible loads was analyzed by use of isoenergy and isoefficiency maps. These two-dimensional visualizations helped to expose complex connections between seemingly independent computing subsystems and the parameters representing them. The two types of algorithms solving divisible load scheduling problems, that is the greedy heuristics and the MIP-based problems, also expose a trade-off between computational complexity and solution quality (both in energy and in time performance). Thus, high

quality solutions come at the cost of computational time, but also benchmarking of the data-parallel application on a specific computing platform necessary to obtain specific scheduling model parameters.

We believe that this thesis opens options for further studies on energy performance in parallel processing. The reported above trade-off between the benchmarking and complexity costs and quality of the solutions calls for further investigation of the algorithms that can possibly offer better quality solutions and require less information on the scheduled computation. The concept of isoline maps also can be extended to other performance models and scheduling problems.

A Summary of Notations

Table A.1: Chapter 4 summary of notations.

α_i	load assigned to machine M_i (e.g. [bytes])
a	computing rate (e.g. [s/byte])
C	communication rate (e.g. [s/byte])
E	sum of all forms of consumed energy (e.g. [J])
E^I	energy consumed in the idle state (e.g. [J])
E^{RN}	energy above the idle state consumed in communication
E^{RC}	energy above the idle state consumed in computation
f	size of parallel part of the computation (Section 4.1)
k	power reduction factor for the idle state
m	number of processors
P^C	processor power in active state (e.g. [W])
P^N	network power in active state (e.g. [W])
S	startup time (e.g. [s])
V	size of load (e.g. [bytes])

Table A.2: Chapter 5 summary of notations

α_{ij}	the amount of load sent to M_i in the jth communication;
a_1	processing rate on-core, e.g. [s/MB]
a_2, b_2	parameters of computation time out-of-core, e.g. [s/MB], [s]
C	communication rate (1/bandwidth) [s/MB]
$\varepsilon(\alpha)$	$= \max\{k_1\alpha, k_2\alpha + l_2\}$, energy consumed in processing load of size α in a hierarchical memory system, cf. Section 3.3
E	schedule energy [J]
E_i^I	idle state energy on machine M_i [J]
E_i^N	networking energy on machine M_i [J]
E_i^S	energy consumed by the starting machine M_i [J]
E_i^R	running M_i machine energy [J]
k_1	energy rate per data unit on-core [J/MB]
k_2, l_2	parameters of energy consumed out-of-core, per data unit, e.g. [J/MB], [s]
m	number of machines
O	fixed communication overhead, Section 5.2.3 [s]
P^I	idle state power of the machines [W]
P^N	networking power [W]
P^R	running power [W] of the machines
ρ	size of RAM available to store data
S	startup time (e.g. [s])
$\tau(\alpha)$	$= \max\{a_\alpha, a_2\alpha + b_2\}$, time of processing load of size α in a hierarchical memory system, cf. Section 3.3
T	schedule length [s]
t_i^I	idle time of machine M_i [s]
t_i^N	networking time of machine M_i [s]
t_i^R	running time of machine M_i [s]
x_i	decision variable =1 if computer i is activated; =0 otherwise
V	size of load to process [MB]

Table A.3: Chapter 6 summary of the main notations.

	MIP variables
α_{ij}	the amount of load sent to M_i in the jth communication;
E	sum of all forms of the consumed energy;
E_0	energy consumed by the originator;
E^I	total energy consumed in idle waiting before starting the processors and later in busy-waiting;
E_{ij}	energy consumed by machine M_i in the computations on the jth load chunk;
E^N	total energy consumed in the networking;
E^R	total energy consumed in the computations;
E^S	total energy consumed in staring the processors;
$idleStart_i$	the time until starting machine M_i;
$idle_i$	the total time when machine M_i is busy-waiting;
q_{ij}	a variable equal to the product $t_i x_{ij}$;
t_j	the time when the jth communication begins;
τ_{ij}	the duration of computation on the jth load chunk on machine M_i;
u_{ij}	a binary variable equal 1 if $\tau_{ij} = a_{2i}\alpha_{ij} + b_{2i}$, 0 if $\tau_{ij} = a_{1i}\alpha_{ij}$;
x_{ij}	a binary variable equal 1 if machine M_i receives load in the jth communication, 0 otherwise;
y_{ij}	a binary variable equal 1 if machine M_i received some load in the communication jth or earlier, 0 otherwise;
	constants
a_{1i}, a_{2i}, b_{2i}	parameters of machine M_i piecewise-linear computing time function;
C_i	communication rate of machine M_i (inverse of bandwidth);
k_{1i}, k_{2i}, l_{2i}	parameters of machine M_i piecewise-linear energy function;
m	size of the set of available machines
n	number of communications
O_i	fixed communication overhead of machine M_i;
P_i^I	idle state power of machine M_i;
P_i^N	networking power of machine M_i;
P_i^S	starting state (power up) power of machine M_i;
S_i	startup time from idle of machine M_i;
ρ	size of RAM on machine M_i available to store data
T	schedule length;
V	size of load to process;
Z, Z', Z''	big constants.

B Streszczenie w języku polskim

Wstęp

Przetwarzanie dużych ilości danych jest jednym z najważniejszych zastosowań technologii informatycznych we współczesnych nauce i przemyśle. W celu sprostania wyzwaniom związanym z przetwarzaniem dużych ilości danych opracowane zostało wiele nowych technologii: platformy i frameworki do przetwarzania danych [8, 12, 27], biblioteki programistyczne [10, 11], systemy zarządzania bazami danych [7, 9, 65]. Jednak ogromne centra danych służące przetwarzaniu dużych ilości informacji obciążają sieci energetyczne. W związku z tym, dostawy energii i jej koszty narzucają ograniczenia w dalszym rozwoju centrów danych i superkomputerów. Ograniczenia energetyczne są ważnym zagadnieniem również w przypadku sieci sensorów, lotnictwa i kosmonautyki czy aplikacji internetu rzeczy. Praca ta jest poświęcona analizie i optymalizacji czasu oraz wydajności energetycznej w przetwarzaniu dużych ilości danych. Efektywne zaplanowanie przetwarzania równoległego będzie sposobem na optymalizację jego wydajności. Platformy obliczeniowe, zwłaszcza w kontekście dużych ilości danych, nakładają szereg ograniczeń. Na przykład, dystrybucja danych do zdalnego przetwarzania nie jest natychmiastowa, a opóźnienia w komunikacji stanowią znaczącą część czasu wykonania aplikacji. Dlatego opóźnienia w komunikacji muszą być brane

pod uwagę przy planowaniu wykonania równoległych obliczeń. Współczesne komputery mają hierarchiczną strukturę pamięci, począwszy od rejestrów CPU, poprzez pamięci podręczne procesora, pamięć operacyjną, czyli tzw. RAM, aż po zewnętrzne pamięci masowe (sieciowe, SDD, dyski twarde). Rozmiar pamięci rośnie, ale prędkość dostępu maleje w miarę „oddalania" się od procesora. Także energochłonność aplikacji pracujących na różnych poziomach pamięci znacznie się różni. Wielkości danych, które muszą być przetwarzane w dzisiejszych aplikacjach, z łatwością przekraczają wielkości dostępnych pamięci operacyjnych współczesnych komputerów. Dlatego należy unikać korzystania z pamięci zewnętrznych lub liczyć się ze spadkiem szybkości przetwarzania wynikającym z korzystania z niższych poziomów pamięci. Praktyczny plan równoległego wykonywania aplikacji powinien wykorzystywać opcję oszczędzania energii poprzez włączanie niezbędnego komputera tylko w razie potrzeby. Wreszcie, heterogeniczność platform obliczeniowych, czy to w postaci mieszania CPU z obliczeniami na GPU, stosowania różnych instancji obliczeniowych, takich jak Amazon EC2 [1], czy też centralnych serwerów przetwarzania ze zdalnymi czujnikami, jest jeszcze jedną rzeczywistością obecnych aplikacji, które powinny być reprezentowane w metodach szeregowania obliczeń. W pracy tej zamierzamy zaproponować metody szeregowania zadań równoległych uwzględniające opóźnienia komunikacyjne, hierarchiczne poziomy pamięci, istnienie trybów oszczędzania energii oraz heterogeniczność systemu.

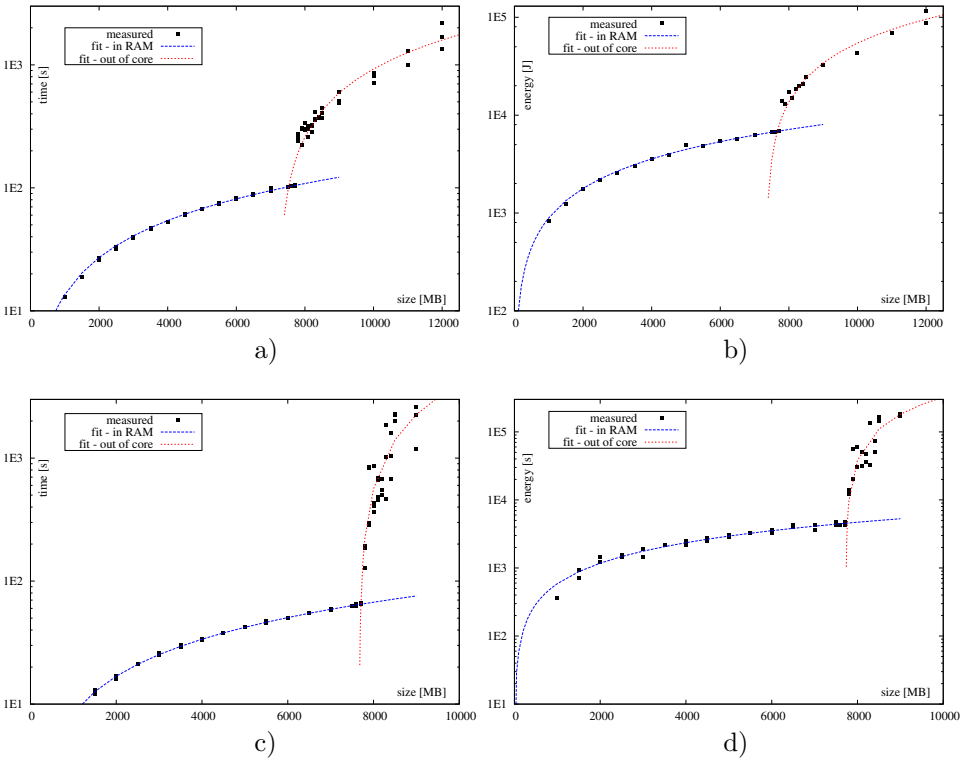
Cel i zakres pracy

Głównym celem tej pracy jest zwiększenie efektywności energetycznej przez lepsze zarządzanie przetwarzaniem równoległym i zasobami systemowymi. Kolejnym istotnym celem jest zrozumienie ograniczeń przetwarzania równoległego oraz zależności pomiędzy czasem i energią jako miarami wydajności. Celem niższego poziomu jest stworzenie modeli szeregowania i wydajności dla platform i aplikacji przetwarzania rozproszonego. Jako podstawowe narzędzie modelowania wykorzystana została idea obliczeń jednorodnie podzielnych (ang. divisible

load theory). Zakłada ona, że obliczenia można dowolnie dzielić na części i wykonywać w sposób równoległy. Odpowiada to równoległemu przetwarzaniu dużych zbiorów danych. Następnym celem jest zaproponowanie algorytmów rozwiązujących te modele, tak aby móc konstruować efektywne harmonogramy dla aplikacji równoległych i przewidywać ich wydajność. Dzięki analizie jakości skonstruowanych harmonogramów, poznane zostaną determinanty wydajności i relacje między nimi. Zbadany zostanie wpływ platformy obliczeniowej i parametrów aplikacji na kryteria czasu i energii. Aby osiągnąć powyższe cele opracowane zostaną, modele czasu wykonywania obliczeń i zużycia energii na jednym komputerze w zależności od wielkości przetwarzanych danych. Modele optymalizacji obejmą systemy z nieograniczoną pamięcią, systemy homogeniczne z pamięcią hierarchiczną i systemy heterogeniczne z pamięcią hierarchiczną. Dla sformułowanych problemów optymalizacji opracowane zostaną algorytmy, a ich koszty obliczeniowe zostaną również ocenione. Oznaczenia użyte w dalszej części polskiego streszczenia objaśniono w załączniku A.

Modele matematyczne czasu obliczeń i zużycia energii

W celu stworzenia modeli zużycia energii w czasie obliczeń, wykonano pomiary na różnych komputerach z przykładowymi aplikacjami: quicksort, wyszukiwanie ciągu w tekście, obliczanie wartości skrótu md5 (tablice tęczowe), wykrywanie krawędzi na obrazach bitmapowych i transpozycja macierzy. Aplikacje te zostały zaimplementowane w gcc i uruchomione pod FreeBSD 8.1 i Ubuntu 14.04 LTS. Do pomiarów zużycia energii zastosowano watomierz Lutron DW-6090 o rozdzielczości mocy 1W i rozdzielczości czasowej 1s. W niektórych z mierzonych komputerów prędkości wentylatorów zależne były od temperatury procesora i systemu. Natomiast zmiana prędkości wentylatora powodowała kilka watów różnicy w pomiarze. Aby uniezależnić wyniki od temperatury otoczenia lub wyników termicznych wcześniejszych eksperymentów, konieczne było zasile-



Rysunek B.1: Zależność czasu i energii od wielkości obciążenia. quicksort: a) czas, b) energia; Wyszukiwanie ciągów znaków: c) czas, d) energia, w zależności od rozmiaru problemu. Osie pionowe są logarytmiczne. Linie ciągłe zostały dopasowane za pomocą regresji liniowej.

nie tych wentylatorów z zewnętrznego źródła zasilania. Watomierz Lutron jest podłączony złączem szeregowym do innego komputera, który rejestruje dane. Schematy stanowiska pomiarowego pokazano na rys. 3.3.

Czas i energia potrzebne do obliczeń porcji danych (obciążenia) zależą od wielkości porcji α. Ważnym wyznacznikiem jest to, jak duża jest paczka danych α w porównywaniu z wielkością dostępnej dla procesu pamięci operacyjnej. Aby zweryfikować związek między czasem obliczeń, zużytą energią i rozmiarem problemu, przeprowadzono szereg testów obliczeniowych. Na podstawie przeprowadzonych testów (Rys. B.1) zaobserwowaliśmy, że czas obliczeń można zapisać jako funkcję maksimum dwóch funkcji liniowych:

$$\tau_i(\alpha) = \max\{a_{1i}\alpha, a_{2i}\alpha + b_{2i}\}.$$

Składnik $a_{1i}\alpha$ odpowiada obliczeniom w pamięci operacyjnej ze współczynnikiem a_{1i} (odwrotność prędkości). Drugi składnik reprezentuje obliczenia poza pamięcią operacyjną. Funkcje τ_i mają dwie właściwości: $\tau_i(0) = 0$ i $\tau_i(\rho_i) = a_{1i}\rho_i = a_{2i}\rho_i + b_{2i}$, dla $i = 1, \ldots, m$, gdzie ρ_i to rozmiar pamięci dostępnej dla aplikacji na maszynie M_i (niekoniecznie całej pamięci RAM sprzętu). Dla rozmiarów problemu α większych od ρ_i maszyna zaczyna korzystać z pamięci dyskowej. Energia zużywana w obliczeniach jest określana przez analogiczną funkcję:

$$\varepsilon_i(\alpha) = \max\{k_{1i}\alpha, k_{2i}\alpha + l_{2i}\}$$

spełniającą również analogiczne wymagania: $\varepsilon_i(0) = 0, \varepsilon_i(\rho_i) = k_{1i}\rho_i = k_{2i}\rho_i + l_{2i}$. Dla rozmiaru dostępnej pamięci ρ_i, obie funkcje τ_i oraz ε_i spełniają:

$$\rho_i = b_{2i}/(a_{1i} - a_{2i}) = l_{2i}/(k_{1i} - k_{2i}).$$

Zauważmy, że ponieważ dla przetwarzania poza pamięcią operacyjną (ang. out-of-core) czas i energia rosną znacznie szybciej niż przy przetwarzaniu on-core, możemy założyć że: $a_{1i} < a_{2i}, b_{2i} < 0$ $k_{1i} < k_{2i}, l_{2i} < 0$, dla wszystkich maszyn M_i. Zauważmy, że współczynniki $a_{1i}, a_{2i}, b_{2i}, k_{1i}, k_{2i}, l_{2i}$ zależą zarówno od parametrów maszyny, jak i od uruchomionej aplikacji. Przykłady zależności $\tau_i(\alpha), \varepsilon_i(\alpha)$ przedstawiono na Rys. B.1

Izomapy

Przedstawiamy tu koncepcję map izolinii, które zostaną wykorzystane jako wizualne pomoce w pokazaniu zjawisk i zależności czasowych i energetycznych.

Koncepcja graficznego przedstawiania punktów o równej wartości danego kryterium na dwuwymiarowych obrazach jest szeroko stosowana w nauce i technice. Przykładami dwuwymiarowych przedstawień złożonych obiektów fizycznych są mapy konturowe elewacji w kartografii, izobary, izotermy, mapy izohietowe w meteorologii, mapy entalpii w termodynamice [94]. Powodem tak

szerokiego wykorzystania izomap jest fakt, że takie wizualizacje okazały się bardzo skuteczne w budowaniu zrozumienia związków i podatności na zmiany dla złożonych zjawisk w wielu dziedzinach nauki i techniki. Pewnym rodzajem mapy izolinii są mapy izoefektywności wprowadzone w [33], rozważane także w rozdziale 5.2.4. Mapy izoenergetyczne stanowią ogólną metodę wizualizacji przetargów w zakresie charakterystyki energetycznej. Każdy model szeregowania lub model wykorzystania energii może posługiwać się mapami izoenergetycznymi jako metodą wizualnej prezentacji. W złożonych relacjach pomiędzy wieloma czynnikami mapy izoenergetyczne dają holistyczny obraz i orientację w kierunku działań optymalizacyjnych.

Izomapy w obliczeniach jednorodnie podzielnych

Przykładowa mapa (V, C) została pokazana na rys. 4.4. Taka izomapa pokazuje jak powiązane ze sobą są parametry rozmiaru rozwiązywanego problemu V i odwrotności prędkości C. Rys. 4.4 pokazuje, czy zużycie energii spowodowane rosnącą wielkością problemu może być zrekompensowane przez szybszą komunikację. Konfiguracje w lewym górnym rogu, gdzie komunikacja jest szybka, a rozmiar problemu niewielki, są niemożliwe do zrealizowania, gdyż wykonanie obliczeń w sposób równoległy na wszystkich procesorach założonej konfiguracji wymagałoby nieplanowanego przestoju. Wraz z postępem w wysokowydajnych obliczeniach rozmiary V rozwiązywanych problemów nieuchronnie rosną. Mapa (V, C) pokazuje, że zużycie energii rośnie głównie z V. Tylko w przypadku C większego niż określony próg, zużycie energii zależy również od C. Wartość progowa C jest określona przez relację pomiędzy energią zużytą w obliczeniach VaP^C, a energią zużytą przez procesory czekające w stanie bezczynności na rozpoczęcie obliczeń, która wynosi w przybliżeniu $VC(P^C m + P^N)/k$. Tak więc dla $P^C m \gg P^N$ i C większego niż około ak/m =3E-6 te dwa parametry mogą wzajemnie się kompensować. Oznacza to, że komunikacja musi być szybsza, aby równoważyć rosnące V i utrzymać stałe zużycie energii. Alternatywnie, można powiedzieć, że dla $C > ak/m$ komunikacja jest zbyt wolna i powoduje niepo-

trzebne koszty energii poprzez utrzymywanie procesorów w stanie bezczynności. Zastosowanie mapy izoenergetycznej (V, C) w realnym scenariuszu może wyglądać następująco: Planowane jest przetwarzanie rosnących rozmiarów problemu V przy użyciu istniejącej aplikacji i centrum danych. Co należy zrobić, aby przygotować się do takiej zmiany i ograniczyć zużycie energii? Zużycie energii nieuchronnie wzrośnie wraz z V. Głównym składnikiem są obliczenia, iloczyn VaP^C musi być zminimalizowany. Można to osiągnąć dzięki lepszym algorytmom i programowaniu (minimalizacja a), lepszemu sprzętowi (P^C). Dalsze zmiany zależą od interakcji pomiędzy a, C, m, k, P^C, P^N. Jeśli $P^C m \gg P^N$ i $C < ak/m$ nie są potrzebne żadne zmiany w podsystemie sieciowym.

Systemy homogeniczne z pamięcią hierarchiczną

W celu uszeregowania obliczeń jednorodnie podzielnych w homogenicznych systemach komputerowych (tj. składających się z takich samych komputerów) zastosowano dwie metody dystrybucji pracy do wykonania. Pierwsza metoda polega na jednorazowej komunikacji z komputerem. Rozwiązanie problemu dystrybucji obliczeń zostało zapisane jako zadanie mieszanego programowania liniowego (ang. mixed integer linear programming, MIP). Druga metoda zakłada dystrybucję wieloetapową. Rozwiązanie uzyskano metodami heurystycznymi oraz przez sprowadzenie do mieszanego programowania liniowego (MIP).

W przypadku przetwarzania jednoetapowego harmonogram komunikacji i obliczeń przedstawiono na rys.5.1a. Na początku całość danych do obliczeń znajduje się na inicjatorze czyli komputerze oznaczonym dalej jako M_0. Inicjator jest połączony ze wszystkimi maszynami podrzędnymi $M_1, \ldots, M_m$, przy pomocy pewnej sieci o wysokiej przepustowości $1/C$. Inicjator jedynie dzieli i rozsyła dane. Komunikacja jest wykonywana tylko pomiędzy M_0 a maszynami podrzędnymi, nie ma komunikacji pomiędzy $M_1, \ldots, M_m$. Wolumen V jest wysyłany w kawałkach o rozmiarze $\alpha_1, \ldots, \alpha_m$ odpowiednio do maszyn

$M_1, \ldots, M_m$. Maszyny potrzebują niezerowego czasu na rozruch S zanim będą w stanie odebrać dane. Czas rozruchu S jest ważnym elementem modelu obliczeń, ponieważ bez niego może być aktywowana dowolna liczba procesorów, co nie ma praktycznego sensu [22, 28]. Po zakończeniu odbierania danych przez maszynę M_i, M_i zaczyna je przetwarzać, podczas gdy inicjator aktywuje maszynę M_{i+1} w celu wysłania do niej danych o rozmiarze α_{i+1}. Procedura jest powtarzana do momentu rozpoczęcia obliczeń na wszystkich procesorach m.

W teorii obliczeń jednorodnie podzielnych zazwyczaj zakłada się, że czas zwracania danych z maszyn do inicjatora jest pomijalny. Przy takim założeniu można wykazać [22, 26, 74], że optymalnym rozwiązaniem jest takie, w którym obliczenia na wszystkich maszynach kończą się równocześnie. Rozwiązanie problemu uszeregowania obliczeń zapisano w zadaniu mieszanego programowania liniowego (5.5)-(5.11).

W przypadku przetwarzania wieloetapowego paczki danych do maszyn rozsyłane mogą być więcej niż raz. Do rozwiązania tego problemu proponujemy proste algorytmy zachłanne oraz metody programowania liniowego dające rozwiązania optymalne. Przedstawione heurystyki to:

Simple static chunk (SSC) algorytm ten zakłada, że rozmiary paczek danych są równe rozmiarowi dostępnej pamięci RAM, tj. $\alpha_{SSC} = \rho$. W ten sposób SSC unika korzystania z pamięci zewnętrznej. Wadą takiego podejścia jest to, że na końcu uszeregowania są „wystające" komputery, nierówno kończące obliczenia. To znaczy, że w ostatniej iteracji dystrybucji obciążenia niektóre procesory mogą pozostać bezczynne.

Static chunk with underload (SCU) algorytm ten przyjmuje $\alpha_{SCU} = V/(\lceil V/\rho m \rceil m)$ gdzie m to liczba homogenicznych maszyn z pamięcią o rozmiarze ρ. Tak więc, algorytm SSU zaokrągla w dół rozmiar paczki danych którą uzyskuje każdy komputer tak, aby unikąć niezrównoważenia obciążeń komputerów kosztem ewentualnej dodatkowej iteracji.

Static chunk with overload (SCO) zaokrągla liczbę iteracji w dół, kosztem ewentualnego wykorzystania przetwarzania w pamięci zewnętrznej.

Guided Self-Scheduling Adaptation (GSS) algorytm wyznacza rozmiary paczek danych jako $\alpha_{GSS} = \min\{V',\max\{1,\min\{V'/m,\rho\}\}\}$, gdzie V' jest ilością danych pozostającą do przetworzenia. Rozmiary paczek zmniejszają się w trakcie realizacji uszeregowania. Zakładając, że $V > \rho$, algorytm początkowo wysyła paczki o rozmiarach równych rozmiarowi dostępnej pamięci RAM. Kiedy $V' < \rho$, GSS stopniowo zmniejsza rozmiary paczek, a tym samym minimalizuje różnice w czasach ukończenia poszczególnych maszyn. GSS nie wysyła paczek o rozmiarze mniejszym niż pewien stały rozmiar, który jest tu oznaczony jako 1. Dla $V \gg m\rho$ liczba użytych procesorów w GSS jest taka sama jak w poprzednich algorytmach, ponieważ początkowe kawałki obciążenia mają rozmiar ρ. Jednakże, jeśli $V/\rho > m$ GSS od razu wysyła paczki mniejsze niż ρ, rozmiary paczek zmniejszają się, a komunikacja jest coraz krótsza. W takiej sytuacji GSS jest w stanie uruchomić więcej maszyn niż SSC, SSU, SCO, jednocześnie nie zwiększając czasów bezczynności niektórych maszyn na końcu uszeregowania.

Uzyskane wyniki symulacyjnych badań wydajności pokazały, że istnieje przetarg pomiędzy energią a czasem uszeregowania. Wydajność czasowa i energetyczna są regulowane przez: i) wielkości wysyłanych paczek danych, które determinują prowadzenie obliczeń w pamięci wewnętrznej lub zewnętrznej, ii) liczbę procesorów możliwych do równoległego użycia , co daje obniżenie kosztów dzięki skróceniu uszeregowania, iii) czas bezczynności, który ma wpływ na ilość zmarnowanej energii. Mówiąc ogólniej, o wydajności decyduje złożona zależność między szybkością i poborem mocy elektrycznej obliczeń w pamięci wewnętrznej i zewnętrznej, kosztami uruchomienia nowych maszyn, opóźnieniami w komunikacji oraz wielkością rozwiązanego problemu. Można zauważyć, że w szerokim zakresie parametrów systemowych przetwarzanie równoległe ma wpływ na energię i czas uszeregowania: możliwe jest zaoszczędzenie na obu kryteriach poprzez

dodanie nowych maszyn. Zjawisko to jest jednak ograniczone do obliczeń dużych rozmiarów i krótkich czasów rozruchu maszyn. Większe czasy rozruchu szybko zmniejszają szanse na skrócenie czasu uszeregowania i oszczędność energii przez zastosowanie równoległości obliczeń. Tak więc wszystkie parametry wpływające na możliwość uzyskania efektywnych obliczeń równoległych będą miały również wpływ na optymalizację długości uszeregowań i zużycia energii. Co więcej, można zaobserwować, że oszczędności energii uzyskane w wyniku zmiany jednego parametru lub jednej części systemu są zazwyczaj ograniczone. Dlatego do zmniejszania zużycia energii elektrycznej używanej do zasilania wysokowydajnych obliczeń i dużych centrów danych niezbędny jest postęp we wszystkich podsystemach tak platformy, jak i aplikacji równoległej.

Systemy heterogeniczne z pamięcią hierarchiczną

Celem badań nad systemami heterogenicznymi była analiza wpływu różnorodności komputerów na możliwe do uzyskania wyniki wydajnościowe. W przypadku systemów heterogenicznych ograniczono się do wieloetapowej metody dystrybucji danych. Do konstrukcji uszeregowań użyte zostały proste algorytmy heurystyczne wykorzystujące pewne zasady preferencji procesorów wg ich parametrów wydajnościowych, a także zaprezentowano rozwiązania używając mieszanego programowania liniowego. Algorytmy zachłanne zaproponowane do szeregowania zdań w systemach heterogenicznych to adaptacje wcześniej wykorzystanych algorytmów Simple Static Chunk (SSC) i Guided Self-Scheduling (GSS). Dla tych algorytmów rozważamy ustawianie kolejności użycia procesorów, np. wg prędkości komunikacji, czasów rozruchu, mocy w stanach spoczynku, komunikacji, rozruchu, rozmiaru pamięci. Stosowano również uszeregowanie w losowej kolejności dystrybucji danych stosowane jako punkt odniesienia. Możliwe jest zastosowanie wszystkich zasad sortowania równocześnie i wybranie z nich najlepszego rozwiązania. Czas przeszukiwania rozwiązań dla

wszystkich zasad sortowania nadal będzie dalece krótszy niż znalezienie rozwiązań metodą programowania liniowego, a dzięki różnorodności metod wyboru kolejności dystrybucji danych unikamy powtarzania złych decyzji. Tą drogą zwiększamy niezawodność metod zachłannych i osłabiamy znaczenie ich najgorszych przypadków. Taką metodę będziemy nazywać odpowiednio SuperSSC i SuperGSS. Zarówno szeregowanie zadań jednorodnie podzielnych, jak i mieszane programowanie liniowe w liczbach całkowitych w ogólności, są zagadnieniami **NP**-trudnymi. Oznacza to, że zgodnie z aktualnym stanem wiedzy (chyba że **P**=**NP**), do optymalnego rozwiązania tych problemów wymagane są wykładnicze czasy działania algorytmów. W naszym problemie, w najgorszym przypadku złożoność obliczeniowa rośnie wykładniczo wraz z liczbą procesorów m i liczbą paczek danych n. Jednak dla rozsądnych rozmiarów problemów sformułowania mieszanego programowania liniowego (MIP) mogą być rozwiązane dość dobrze przez współczesne solvery.

W pracy dokonano porównania jak różne typy algorytmów szeregowania wymieniają czas swojego działania na jakość tworzonych rozwiązań. Przetarg pomiędzy jakością rozwiązań a czasem działania algorytmu pokazany jest na rys. 6.10. Złożoność obliczeniowa rozwiązania problemu zależy od m, n, a w przypadku algorytmów heurystycznych również od V. W celu uniknięcia zamazania zależności jakości rozwiązania od czasu działania algorytmu przez zależność złożoności obliczeniowej od m, n, V, parametry te zostały ustawione na $m = 10, n = 12, V = 24000$MB. W prostokątach na wykresie 6.10 przedstawiono czasy działania algorytmu (w poziomie) oraz względną odległość od najlepiej uzyskanego rozwiązania (w pionie) na populacji 30 instancji testowych. Prostokąty te rozciągają się od kwartyla Q1 do Q3 w czasie (poziomo). Rozpiętość jakościowa jest reprezentowana analogicznie wzdłuż wymiaru pionowego. Na rys. 6.10 zaznaczone są również mediany (Q2) czasu trwania i jakości. Chociaż istnieje 14 zasad sortowania procesorów dla heurystyk SSC i GSS, to analiza statystyczna (ANOVA) wykazała, że ani w czasach wykonania algorytmu, ani w jakości rozwiązań nie ma żadnej reguły sortowania, która miałaby statystycz-

nie uzasadnioną przewagę dla rozpatrywanego m, n, V. Tak więc, aby uniknąć bałaganu na wykresie, wyniki wszystkich reguł sortowania są umieszczane w ramkach z rozróżnieniem tylko na metody SSC i GSS (oznaczone odpowiednio jako All SSC, All GSS). Wyniki dla metod SuperSSC i SuperGSS, które wybierają najlepsze rozwiązanie spośród wszystkich reguł sortowania procesora, są przedstawiane odpowiednio jako SuSSC i SuGSS. Jak widać na rys. 6.10, rozwiązania skonstruowane poprzez użycie modelu MIP są zawsze najlepsze w odniesieniu do jakości rozwiązania. Ale ta gwarancja jakości została uzyskana kosztem czasu obliczeń, najwyższego spośród wszystkich badanych metod. Osłabienie zadanych gwarancji jakości rozwiązania, które tworzy solver MIP do 10% odległości od optimum poprawia czas wykonania z niewielką utratą jakości rozwiązania. Podejście to ma jednak ograniczoną skalowalność, ponieważ przy $m = 20$ również osłabiony model MIP przekracza limit czasowy 1200s co widać na rys. 6.9. Rozwiązania heurystyczne są średnio 1,7 – 3 razy gorsze w długości uszeregowania i 1,5 – 2,3 razy gorsze w kryterium energii. I odwrotnie, metody heurystyczne są o 4 – 5 rzędów wielkości szybsze od rozwiązania modelu MIP. Można zauważyć, że algorytm GSS jest lepszy niż SSC, ale jest nieco wolniejszy (ok. 50% dłuższy czas działania). Super-SSC poprawia jakość rozwiązania w porównaniu z oryginalnymi metodami SSC, ale jest i tak gorszy zarówno pod względem jakości rozwiązania jak i czasu działania (w odniesieniu do mediany czasu) niż oryginalne metody GSS. Super-GSS jest tylko nieznacznie lepszy w jakości rozwiązania niż oryginalne metody GSS. Można stwierdzić, że metody GSS dominują nad innymi heurystykami.

Podsumowanie i uwagi końcowe

W niniejszej pracy analizowano problemy związane z planowaniem obliczeń jednorodnie podzielnych w systemach z pamięcią hierarchiczną, dla kryteriów wydajności energetycznej i czasowej. Nie założono modeli wydajności czasowych i energetycznych, lecz skonstruowano je w oparciu o badania właściwości rzeczy-

wistych algorytmów w rzeczywistych systemach komputerowych. Modele te są funkcjami odcinkowo liniowymi dla wielkości obciążenia (danych) do przetworzenia. Rozważano przetwarzanie danych w układzie homogenicznym i heterogenicznym. Zaproponowano dwa rodzaje algorytmów szeregowania dla przetwarzania obciążenia: szybkie algorytmy zachłanne i metody korzystające z programowania liniowego. Wyniki modelowania wydajności aplikacji równoległych potwierdzają istnienie przetargu pomiędzy kryteriami czasu i energii. Wykazano jednak również, że w wielu przypadkach zarówno zużycie energii, jak i długość uszeregowania można zredukować poprzez zrównoleglenie obliczeń. W systemach heterogenicznych zaobserwowano częste nieregularności w budowie uszeregowań. Analiza wpływu różnych składników platformy obliczeniowej na czas i wydajność energetyczną w przetwarzaniu zadań jednorodnie podzielnych została wsparta zastosowaniem map izoenergetycznych i izoefektywności. Te dwuwymiarowe wizualizacje pozwoliły na wyeksponowanie złożonych powiązań pomiędzy pozornie niezależnymi podsystemami obliczeniowymi. Dwa rodzaje algorytmów konstruujących rozwiązania problemów szeregowania obliczeń jednorodnie podzielnych, czyli heurystyki zachłanne i modele oparte na MIP, również cechują się przetargiem pomiędzy złożonością obliczeniową a jakością rozwiązania. Tak więc, wysokiej jakości rozwiązania powstają kosztem czasu obliczeniowego, ale także benchmarkingu aplikacji na konkretnej platformie obliczeniowej, niezbędnego do uzyskania wymaganych parametrów modelu uszeregowania. Wierzymy, że ta praca otwiera możliwości dla dalszych badań nad wydajnością energetyczną w przetwarzaniu równoległym.

Bibliography

[1] *Amazon, Amazon EC2 Instance Types, year=2019, howpublished = [online]* `https://aws.amazon.com/ec2/instance-types/`.

[2] R. Agrawal and H. Jagadish, *Partitioning techniques for large-grained parallelism*, IEEE Transactions on Computers, 37 (1988), pp. 1627–1634.

[3] S. G. Aki, *The design and analysis of parallel algorithms*, Old Tappan, NJ (USA); Prentice Hall Inc., 1989.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, *A scalable, commodity data center network architecture*, in Proceedings of the ACM SIGCOMM'08, ACM, 2008, pp. 63–74.

[5] S. Albers, *Energy-efficient algorithms*, Communications of the ACM, 53 (2010), pp. 86–96.

[6] G. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, in Proceedings of the AFIPS '67 Spring joint computer conference, ACM, 1967, pp. 483–485.

[7] Apache Software Foundation, *Hbase and MapReduce.* [on-line] `https://hbase.apache.org/book/mapreduce.html`, 2014.

[8] ——, *Welcome to apache hadoop.* [on-line] `http://hadoop.apache.org/`, 2014.

[9] ——, *Apache CouchDB.* [on-line] `http://couchdb.apache.org/`, 2015.

[10] ——, *Apache MRQL.* [on-line] `https://mrql.incubator.apache.org/`, 2015.

[11] ——, *Giraph - Welcome To Apache Giraph!* [on-line] `http://giraph.apache.org/`, 2015.

[12] ——, *Apache spark lightning-fast unified analytics engine.* [on-line] `https://spark.apache.org/`, 2018.

[13] F.-A. Armenta-Cano, A. Tchernykh, J. M. Cortés-Mendoza, R. Yahyapour, A. Y. Drozdov, P. Bouvry, D. Kliazovich, A. Avetisyan, and S. Nesmachnow, *Min_c: Heterogeneous concentration policy for energy-aware scheduling of jobs with resource contention*, Programming and Computer Software, 43 (2017), pp. 204–215.

[14] A. Aryania, H. S. Aghdasi, and L. M. Khanli, *Energy-aware virtual machine consolidation algorithm based on ant colony system*, Journal of Grid Computing, 16 (2018), pp. 477–491.

[15] K. Barr and K. Asanović, *Energy-aware lossless data compression*, ACM Transactions on Computer Systems, 24 (2006), pp. 250–291.

[16] O. Beaumont, H. Larchevêque, and L. Marchal, *Non linear divisible loads: There is no free lunch*, in 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, IEEE, 2013, pp. 863–873.

[17] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, *Independent and divisible tasks scheduling on heterogeneous star-shaped platforms with limited memory*, in 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing, IEEE, 2005, pp. 179–186.

[18] L. Benini and G. de Micheli, *System-level power optimization: techniques and tools*, ACM Transactions on Design Automation of Electronic Systems, 5 (2000), pp. 115–192.

[19] J. Berlińska and M. Drozdowski, *Scheduling divisible mapreduce computations*, Journal of Parallel and Distributed Computing, 71 (2011), pp. 450–459.

[20] J. Berlińska and M. Drozdowski, *Comparing load-balancing algorithms for mapreduce under zipfian data skews*, Parallel Computing, 72 (2018), pp. 14–28.

[21] J. Berlińska, M. Drozdowski, and M. Lawenda, *Experimental study of scheduling with memory constraints using hybrid methods*, Journal of Computational and Applied Mathematics, 232 (2009), pp. 638–654.

[22] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi, *Scheduling divisible loads in parallel and distributed systems*, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[23] A. Bounceur, M. Bezoui, R. Euler, N. Kadjouh, and F. Lalem, *Brogo: A new low energy consumption algorithm for leader election in wsns*, in 2017 10th International Conference on Developments in eSystems Engineering (DeSE), IEEE, 2017, pp. 218–223.

[24] J. Carter and K. Rajamani, *Designing energy-efficient servers and data centers*, Computer, 43 (2010), pp. 76–78.

[25] S. Charcranoon, T. G. Robertazzi, and S. Luryi, *Parallel processor configuration design with processing/transmission costs*, IEEE Transactions on Computers, 49 (2000), pp. 987–991.

[26] Y. Cheng and T. Robertazzi, *Distributed computation with communication delay*, IEEE Transactions on Aerospace and Electronic Systems, 24 (1988), pp. 700–712.

[27] J. Dean and S. Ghemawat, *MapReduce: Simplified data processing on large clusters*, in OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004, pp. 137–150.

[28] M. Drozdowski, *Scheduling for Parallel Processing*, Springer-Verlag New York Inc, 2009.

[29] ——, *Energy considerations for divisible load processing*, in Proceedings of the 8th international conference on Parallel Processing and Applied Mathematics (PPAM 2010), Part II. LNCS 6068, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, eds., Springer, 2010, pp. 92–101.

[30] M. Drozdowski and M. Lawenda, *The combinatorics in divisible load scheduling*, Foundations of Computing and Decision Sciences, 30 (2005), pp. 297–308.

[31] M. Drozdowski, J. Marszałkowski, and J. Marszałkowski, *Isoenergy maps*, Tech. Rep. RA-17/2011, Institute of Computing Science, Poznan University of Technology, 2011.

[32] M. Drozdowski, J. M. Marszałkowski, and J. Marszałkowski, *Energy trade-offs analysis using equal-energy maps*, Future Generation Computer Systems, 36 (2014), pp. 311–321.

[33] M. Drozdowski and L. Wielebski, *Isoefficiency maps for divisible computations*, IEEE Transactions on Parallel and Distributed Systems, 21 (2010), pp. 872–880.

[34] M. Drozdowski and P. Wolniewicz, *Experiments with scheduling divisible tasks in clusters of workstations*, in Proceedings of the 6th International Euro-Par Conference on Parallel Processing, LNCS 1900, A. Bode, T. Ludwig, W. Karl, and R. Wismuller, eds., Springer-Verlag, 2000, pp. 311–319.

[35] ——, *Divisible load scheduling in systems with limited memory*, Cluster Computing, 6 (2003), pp. 19–29.

[36] ——, *Out-of-core divisible load processing*, IEEE Transactions on Parallel and Distributed Computing, 14 (2003), pp. 1048–1056.

[37] ——, *Optimum divisible load scheduling on heterogeneous stars with limited memory*, European Journal of Operational Research, 172 (2006), pp. 545–559.

[38] N. Farrington, E. Rubow, and A. Vahdat, *Data center switch architecture in the age of merchant silicon*, in 17th IEEE Symposium on High Performance Interconnects (HOTI), IEEE, 2009, pp. 93–102.

[39] D. Fernández-Cerero, A. Jakóbik, D. Grzonka, J. Kołodziej, and A. Fernández-Montes, *Security supportive energy-aware scheduling and energy policies for cloud environments*, Journal of Parallel and Distributed Computing, 119 (2018), pp. 191–202.

[40] ——, *Security supportive energy-aware scheduling and energy policies for cloud environments*, Journal of Parallel and Distributed Computing, 119 (2018), pp. 191–202.

[41] S. Fuller and L. Millett, *Computing performance: Game over or next level?*, Computer, 44 (2011), pp. 31–38.

[42] M. R. Garey and D. S. Johnson, *Computers and intractability*, vol. 29, WH Freeman New York, 2002.

[43] I. Goiri, J. Berral, J. Oriol Fito, F. Julia, R. Nou, J. Guitart, R. Gavalda, and J. Torres, *Energy-efficient and multifaceted resource management for profit-driven virtualized data centers*, Future Generation Computer Systems, 28 (2012), pp. 718–731.

[44] A. Grama, A. Gupta, and V. Kumar, *Isoefficiency: Measuring the scalability of parallel algorithms and architectures*, IEEE Parallel & Distributed Technology, 1 (1993), pp. 12–21.

[45] A. Gupta and V. Kumar, *Performance properties of large scale parallel systems*, Journal of Parallel and Distributed Computing, 19 (1993), pp. 234–244.

[46] J. Gustafson, *Reevaluating Amdahl's law*, Communications of the ACM, 31 (1988), pp. 532–533.

[47] R. Hockney, *The Science of Computer Benchmarking*, SIAM, Philadelphia, 1996.

[48] H. Hunter, L. Lastras-Montano, and B. Bhattacharjee, *Adapting server systems for new memory technologies*, Computer, 47 (2014), pp. 78–84.

[49] R. Katz, *Tech titans building boom*, IEEE Spectrum, 46 (2009), pp. 40–54.

[50] T. Kaur and I. Chana, *Energy efficiency techniques in cloud computing: A survey and taxonomy*, ACM Computing Surveys (CSUR), 48 (2015), p. 22.

[51] P. Kogge, *The tops in the flops*, IEEE Spectrum, 48 (2011), pp. 48–54.

[52] R. Kothiyal, V. Tarasov, P. Sehgal, and E. Zadok, *Energy and performance evaluation of lossless file data compression on server systems*, in Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, ACM, Article No. 4, 2009.

[53] J. Lang, G. Rünger, and P. Stöcker, *Towards energy-efficient linear algebra with an atlas library tuned for energy consumption*, in 2015 International Conference on High Performance Computing & Simulation (HPCS), IEEE, 2015, pp. 63–70.

[54] K. Li, *Optimal task execution speed setting and lower bound for delay and energy minimization*, Journal of Parallel and Distributed Computing, 123 (2019), pp. 13–25.

[55] X. Li, V. Bharadwaj, and C. Ko, *Processing divisible loads on single-level tree networks with buffer constraints*, IEEE Transactions on Aerospace and Electronic Systems, 36 (2000), pp. 1298–1308.

[56] X. Li, B. Veeravalli, and C. Ko, *Distributed image processing on a network of workstations*, International Journal of Computers and Applications, 25 (2003), p. 10.

[57] S. Lin and K. Banerjee, *A design-specific and thermally-aware methodology for trading-off power and performance in leakage-dominant cmos technologies*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 16 (2008), pp. 1488–1498.

[58] N. Maheshwari, R. Nanduri, and V. Varma, *Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework*, Future Generation Computer Systems, 28 (2012), pp. 119–127.

[59] M. Malik, K. Neshatpour, S. Rafatirad, R. V. Joshi, T. Mohsenin, H. Ghasemzadeh, and H. Homayoun, *Big vs little core for energy-efficient hadoop computing*, Journal of Parallel and Distributed Computing, 129 (2019), pp. 110–124.

[60] T. Maqsood, N. Tziritas, T. Loukopoulos, S. A. Madani, S. U. Khan, C.-Z. Xu, and A. Y. Zomaya, *Energy and communication aware task mapping for mpsocs*, Journal of Parallel and Distributed Computing, 121 (2018), pp. 71–89.

[61] J. Marszałkowski, D. Mokwa, M. Drozdowski, Ł. Rusiecki, and H. Narożny, *Fast algorithms for online construction of web tag clouds*, Engineering Applications of Artificial Intelligence, 64 (2017), pp. 378–390.

[62] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, *Cloud computing: Survey on energy efficiency*, Acm Computing Surveys (CSUR), 47 (2015), p. 33.

[63] X. Mei, K. Zhao, C. Liu, and X. Chu, *Benchmarking the memory hierarchy of modern GPUs*, in Network and Parallel Computing, Springer, 2014, pp. 144–156.

[64] M. Moges, L. Ramirez, C. Gamboa, and T. Robertazzi, *Monetary cost and energy use optimization in divisible load processing*, in Proceedings of the 2004 Conference on Information Sciences and Systems, Princeton University, 2004, p. 6.

[65] MongoDB Inc., *Mongo db manual 2.4*. [on-line] `http://docs.mongodb.org/manual/core/map-reduce/`, 2014.

[66] S. Nesmachnow, B. Dorronsoro, J. E. Pecero, and P. Bouvry, *Energy-aware scheduling on multicore heterogeneous grid computing systems*, Journal of Grid Computing, 11 (2013), pp. 653–680.

[67] L. Newcombe, Z. Limbuwala, P. Latham, and V. Smith, *Data centre fixed to variable energy ratio metric DC-FVER*, tech. rep., BCS Data Centre Specialist Group, 2012. [on-line] `http://dcsg.bcs.org/data-centre-fixed-variable-energy-ratio-metric-dc-fver`.

[68] K. O'brien, I. Pietri, R. Reddy, A. Lastovetsky, and R. Sakellariou, *A survey of power and energy predictive models in hpc systems and applications*, ACM Computing Surveys (CSUR), 50 (2017), p. 37.

[69] F. Pan, V. Freeh, and D. Smith, *Exploring the energy-time tradeoff in high-performance computing*, in Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS05), IEEE, 2005, p. 9.

[70] J. Park and W. Baek, *Analyzing and optimizing the performance and energy efficiency of transactional scientific applications on large-scale numa systems with htm support*, Journal of Parallel and Distributed Computing, 127 (2019), pp. 1–17.

[71] C. Patel, R. Sharma, C. Bash, and S. Graupner, *Energy aware grid: Global workload placement based on energy efficiency*, Tech. Rep. HPL-2002-329, HP Laboratories Palo Alto, 2002.

[72] J.-M. Pierson, *Large-scale Distributed Systems and Energy Efficiency: A Holistic View*, John Wiley & Sons, 2015.

[73] H. Plattner, *Changes in hardware*, in A Course in In-Memory Data Management, Springer, 2014, pp. 23–32.

[74] T. Robertazzi, *Ten reasons to use divisible load theory*, Computer, 36 (2003), pp. 63–68.

[75] T. Robertazzi, *Divisible load scheduling.* [on-line] `http://www.ece.sunysb.edu/~tom/dlt.html`, 2011.

[76] N. V. Shakhlevich, *Scheduling divisible loads to optimize the computation time and cost*, in International Conference on Grid Economics and Business Models, Springer, 2013, pp. 138–148.

[77] L. Sharifi, L. Cerdà Alabern, F. Freitag, and L. Veiga, *Energy efficient cloud service provisioning: keeping data center granularity in perspective*, Journal of Grid Computing, 14 (2016), pp. 299–325.

[78] H. Shi, W. Wang, and N. Kwok, *Energy dependent divisible load theory for wireless sensor network workload allocation*, Mathematical Problems in Engineering, 2012 (2012).

[79] A. Shokripour, M. Othman, H. Ibrahim, and S. Subramaniam, *New method for scheduling heterogeneous multi-installment systems*, Future Generation Computer Systems, 28 (2012), pp. 1205–1216.

[80] S. Singh and I. Chana, *A survey on resource scheduling in cloud computing: Issues and challenges*, Journal of Grid Computing, 14 (2016), pp. 217–264.

[81] J. Sohn, T. Robertazzi, and S. Luryi, *Optimizing computing costs using divisible load analysis*, IEEE Transactions on Parallel and Distributed Systems, 9 (1998), pp. 225–234.

[82] S. Song, C. Su, R. Ge, A. Vishnu, and K. Cameron, *Iso-energy-efficiency: an approach to power-constrained parallel computation*, in Proceedings of International Parallel & Distributed Processing Symposium (IPDPS), IEEE, 2011, pp. 128–139.

[83] SPEC, *SPEC: Standard performance evaluation corporation.* [on-line] `http://www.spec.org/power_ssj2008`, 2009.

[84] R. Springer, D. Lowenthal, B. Rountree, and V. Freeh, *Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster*, in Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), ACM, 2006, pp. 230–238.

[85] B. Subramaniam and W. Feng, *Statistical power and performance modeling for optimizing the energy efficiency of scientific computing*, in 2010 IEEE/ACM International Conference on Green Computing and Communications (GreenCom) & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing (CPSCom), IEEE, 2010, pp. 139–146.

[86] G. Sun, *Exploring Memory Hierarchy Design with Emerging Memory Technologies*, vol. 267 of Lecture Notes in Electrical Engineering, Springer International Publishing, 2014.

[87] S. Swanson and A. Caulfield, *Refactor, reduce, recycle: Restructuring the I/O stack for the future of storage*, Computer, 46 (2013), pp. 52–59.

[88] A. Tchernykh, J. E. Pecero, A. Barrondo, and E. Schaeffer, *Adaptive energy efficient scheduling in peer-to-peer desktop grids*, Future Generation Computer Systems, 36 (2014), pp. 209–220.

[89] V. T'kindt and J.-C. Billaut, *Multicriteria scheduling: theory, models and algorithms*, Springer Science & Business Media, 2006.

[90] R. van der Wijngaart and M. Frumkin, *NAS grid benchmarks version 1.0*, Tech. Rep. NAS-02-005, NASA Advanced Supercomputing Division, 2002. [on-line] `http://www.nas.nasa.gov/assets/pdf/techreports/2002/nas-02-005.pdf`.

[91] L. Wang, G. Von Laszewski, J. Dayal, and F. Wang, *Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS*, in Proceedings of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2010, pp. 368–377.

[92] Wikipedia contributors, *Advanced configuration and power interface, wikipedia, the free encyclopedia.* [on-line] `https://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface`, 2019.

[93] ——, *Contour line.* [on-line] `https://en.wikipedia.org/wiki/Contour_line`, 2019.

[94] ——, *Enthalpy-entropy chart*. [on-line] `https://en.wikipedia.org/wiki/Enthalpy%E2%80%93entropy_chart`, 2019.

[95] D. Woo and H. Lee, *Extending Amdahl's law for energy-efficient computing in the many-core era*, Computer, 41 (2008), pp. 24–31.

[96] R. Xu, D. Mossé, and R. Melhem, *Minimizing expected energy consumption in real-time systems through dynamic voltage scaling*, ACM Transactions on Computer Systems, 25 (Article No.9, 2007), p. 40.

[97] G. D. y Alvarez, F. Favaro, F. Lecumberry, Á. Martín, J. P. Oliver, J. Oreggioni, I. Ramírez, G. Seroussi, and L. Steinfeld, *Wireless eeg system achieving high throughput and reduced energy consumption through lossless and near-lossless compression*, IEEE Transactions on Biomedical Circuits and Systems, 12 (2018), pp. 231–241.

[98] Y. Yang, H. Casanova, M. Drozdowski, M. Lawenda, and A. Legrand, *On the complexity of multi-round divisible load scheduling*, (2007).

[99] M. H. N. Yousefi and M. Goudarzi, *A task-based greedy scheduling algorithm for minimizing energy of mapreduce jobs*, Journal of Grid Computing, 16 (2018), pp. 535–551.

[100] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, *Performance assessment of multiobjective optimizers: An analysis and review*, IEEE Transactions on Evolutionary Computation, 7 (2003), pp. 117–132.

[101] A. Y. Zomaya and Y. C. Lee, *Energy-efficient distributed computing systems*, vol. 88, John Wiley & Sons, 2012.