Divisible load theory - from basics to applications

Maciej Drozdowski

Maciej.Drozdowski@cs.put.poznan.pl

Institute of Computing Science, Poznań University of Technology,
Poznań, Poland

Outline of the presentation

1. basic formulation of DLT
2. experimental verification of DLT
3. DLT in MapReduce
4. time and energy minimization
5. isoefficiency maps

Divisible load theory

Divisible load theory (DLT) – is a performance and optimization (scheduling) model of data-parallel applications.

In DLT it is assumed that:

1. computations can be divided into parts of arbitrary sizes,
2. these parts can be executed independently in parallel.

Load – is usually some data to be processed.

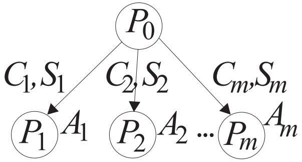
Divisible load theory

Consequently in divisible computations:
$\Rightarrow$ grain of parallelism is small,
$\Rightarrow$ data dependencies are negligible,
$\Rightarrow$ schedule optimization consists in adjusting sizes of the processed load parts to the speeds of communication and computation.

Examples of divisible applications:
• distributed searching for patterns in text, audio, graphic etc. files,
• data retrieval systems,
• database, measurements, image processing,
• some linear algebra algorithms, and simulation.[1]

[1] more on the applications in the following

Basic scheduling model

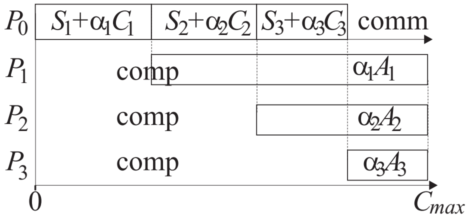


We assume a single level tree (a.k.a. star) interconnection.

- P_0 - originator (master), distributes load, does not compute,
- $P_1, \dots, P_m$ - processors (workers) receive and process the load
- V - load size (e.g. in bytes)
- $S_i + xC_i$ - communication delay for sending load x to P_i
- $A_i x$ - computation time for load x on P_i

For the simplicity of the exposition let us assume (for a moment) that results return time is negligible.

A schedule with negligible return times



α_i - size of load part sent to processor P_i
C_{max} - schedule length

The challenge: choose α_is such that C_{max} is the shortest possible.

Optimality principle: since result return time is negligible, all computations finish at roughly the same time.

Solution by solving a system of linear equations

$$\alpha_i A_i = S_{i+1} + \alpha_{i+1}(A_{i+1} + C_{i+1}) \qquad \text{for } i = 1, \ldots, m-1 \quad (1)$$

$$\sum_{i=1}^{m} \alpha_i = V \quad (2)$$

The above system of linear equations can be solved in $O(m)$ time due to its special structure. Closed-form solutions exist.*

Note that if $S_i > 0$, then a feasible solution (i.e. with $\forall \alpha_i > 0$) may not exist, because load size V is too small to activate all processors.

* For example, α_i (for $i = m, \ldots, 1$) can be expressed 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 have

$$\alpha_m = \frac{V - \sum_{i=1}^{m} l_i}{\sum_{i=1}^{m} k_i}. \quad (3)$$

Why is DLT an attractive scheduling model?

There are many models for scheduling parallel applications.
Is DLT any better?

The advantages of DLT are as follows:

- DLT is comprehensive – many details of computing platform can be represented in DLT, e.g.:
  – alternative load scattering algorithms,
  – alternative communication interconnection topologies,
  – heterogeneity of the system,
  – memory limitations.
- DLT is a good compromise between complexity and accuracy.
- DLT is computationally easy.[2]

[2] Easy in the basic formulations, but NP-hard in heterogeneous cases with fixed time and cost overheads.

Outline of the presentation

1. basic formulation of DLT
2. **experimental verification of DLT**
3. DLT in MapReduce
4. time and energy minimization
5. isoefficiency maps

DLT validity

Is DLT correctly representing real-world applications?

Let us consider the following verification framework:

- Measure system, and application parameters A_i, S_i, C_i for machines $i = 1, \ldots, m$.
- Split the load size V into parts of sizes $\alpha_1, \ldots, \alpha_m$ according the model formulas (1)-(3).
- Calculate expected (theoretical) execution time C^T_{max}.
- Execute the application with the calculated work split $\alpha_1, \ldots, \alpha_m$ and measure real schedule length C^R_{max}.
- How far is C^R_{max} from C^T_{max}?

How to represent returning of the results

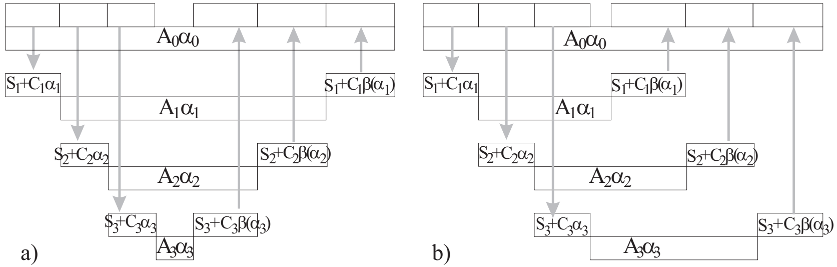
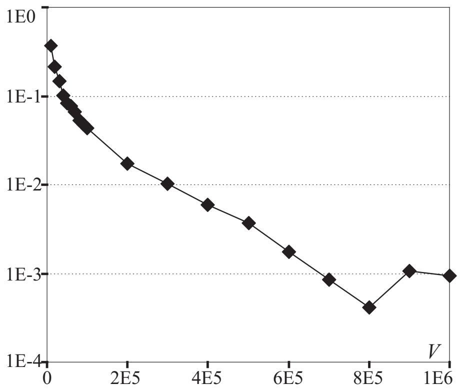


Figure: a)LIFO, b)FIFO orders of returning results.

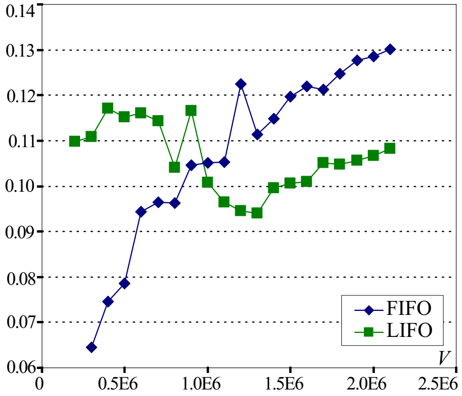
$\beta(\alpha)$ is the size of the results as a function of the input load size.

Model relative error



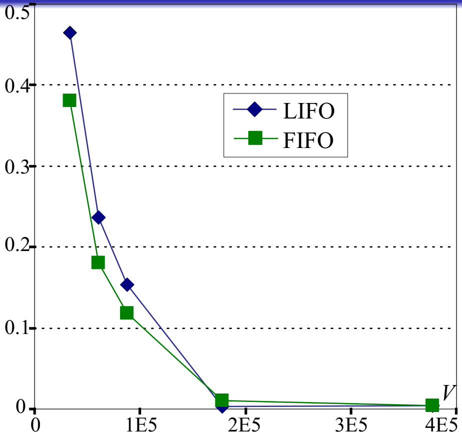
Platform: Transputer system (ca. 1996)
Application: search for a pattern in a text file, LIFO
Error: $< 1\%$ feasible.

Model relative error



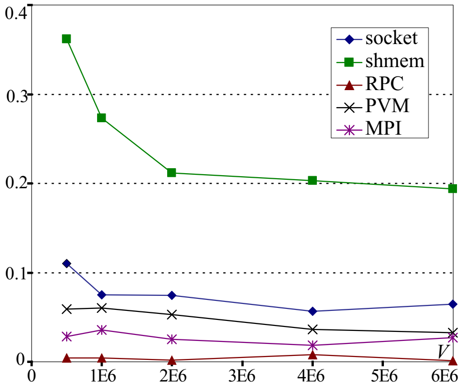
Platform: IBM SP2, PVM (ca. 1997)
Application: LZW compression
Error: 9 – 13% feasible.

Model relative error



Platform: Windows NT, MPI (ca. 1999)
Application: database join
Error: $< 10\%$ feasible.

Model relative error



Platform: Silicon Graphics Origin 3000, various communication technologies (ca. 2003)
Application: search for pattern in a text file
Error: $< 5\%$ feasible.

Conclusion on model accuracy

Conclusion:

- overall accuracy of DLT model is good
- usually accuracy improves with problem size V

Outline of the presentation

1. basic formulation of DLT
2. experimental verification of DLT
3. **DLT in MapReduce**
4. time and energy minimization
5. isoefficiency maps

Why MapReduce?

In this part of the presentation we intend to show, on the example of MapReduce, that DLT can be applied to analyze a wide class of Big Data applications.

What is MapReduce

- *MapReduce* is a programming model for processing large data sets on big numbers of computers, introduced by Google [DG04].
- MapReduce exploits data parallelism.
- Popular MapReduce implementations is Apache Hadoop.
- MapReduce is a component of the NoSQL databases (CouchDB, HBase, MongoDB) and libraries for big data processing (Apache Giraph, MRQL).

What is MapReduce

MapReduce applications have two steps:

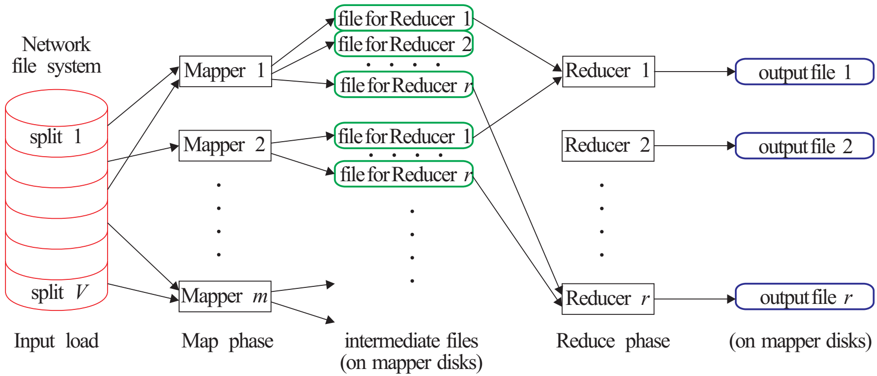
- In the *Map* step a *Map* function transforms the input dataset, into a set of intermediate $(key1, value1)$ pairs.
- In the *Reduce* step:
  - the intermediate values are sorted by $key1$,
  - a *Reduce* function merges the intermediate pairs with equal value of $key1$, to produce pairs $(key1, value2)$.

Examples of MapReduce Application

Example 1: Word frequency in a set of documents.
- *Map* function emits intermediate pair $(word, 1)$ for each *word* in the input file(s).
- *Reduce* function sorts intermediate pairs by *word*, sums the 1s for a given *word*, and produces pairs $(word, count)$.

Example 2: Inverted index.
In the inverted index computation all documents comprising certain *word* must be identified.
- *Map* function emits pairs $(word, docID)$, where *docID* is a document identifier (e.g. a URL).
- *Reduce* function sorts all $(word, docID)$, and emits a pair $(word, list\_docIDs)$, where *list_docIDs* is a list of *docIDs*.

Execution Overview of MapReduce



- Load is split into load units (size *lu*).
- The output of the *Map* function is partitioned into *r* files.
- Each reducer processes certain range of $(key1, \ldots)$ values.

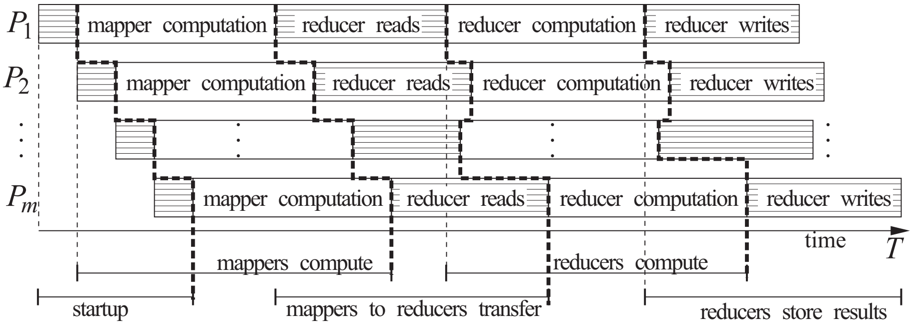
Usually something like $hash(key1) \bmod r$ is used.

Assumptions & Notation

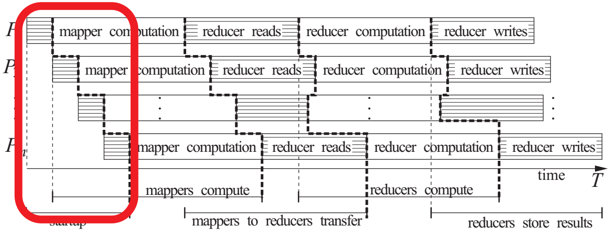
Here the input files are *divisible*

P_i	processor i, processors are identical
ℓ	*bisection width* - maximum number of communication channels concurrently in use
$1/C$	communication speed for two processors in the "empty" network
ℓ/C	maximum total bandwidth for concurrent channels in the whole network
m	number of mappers
r	number of reducers
V	the total size of load to be processed (e.g. in bytes)
lu	size of a load unit (16-64MB [DG04])

MapReduce Schedule Structure



Startup

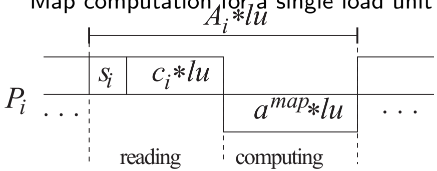


Mapper and the reducer codes are uploaded together.

S - computation startup time for each processor (mS for all).

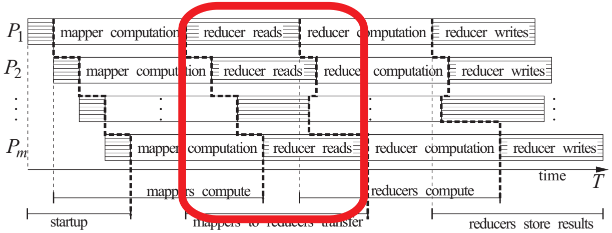
Mapping

Map computation for a single load unit



a^{map}, c_i, s_i	'microscopic' computing rate, communication rate, communication startup time for processor P_i; in seconds per byte (a^{map}, c_i), and seconds (s_i);
A_i	$= \frac{s_i}{lu} + a^{map} + c_i$ 'macroscopic' computing rate of processor P_i executing a mapper application;
α_i	size (e.g. in bytes) of load assigned to P_i;
γ	mapper result multiplicity fraction, $\gamma\alpha_i$ is size of P_i output;

Mapper-Reducer load transfer (shuffle)



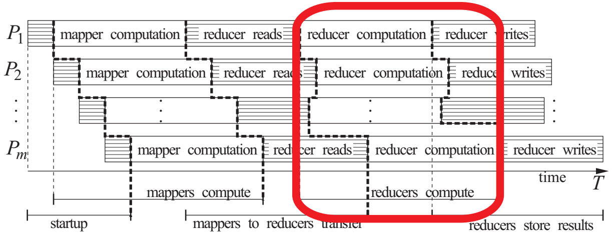
$\gamma\alpha_i/r$ size of reducers' input from mapper P_i, $i = 1, \dots, m$,

$\gamma V/r$ total size of the input for each reducer,

$\gamma\alpha_i C/r$ time of transferring load α_i from mapper P_i to a reducer without bandwidth limitation.

Only one channel can be opened from a mapper, and to a reducer.

Reducing



s^{red}	reducer computation startup time
a^{red}	reducer computing rate (includes storing results)
$\tau(x) = a^{red}(x \log x)$	reducer computing time vs input size x
$t_R = s^{red} + \tau(\gamma V / r)$	reducers running time

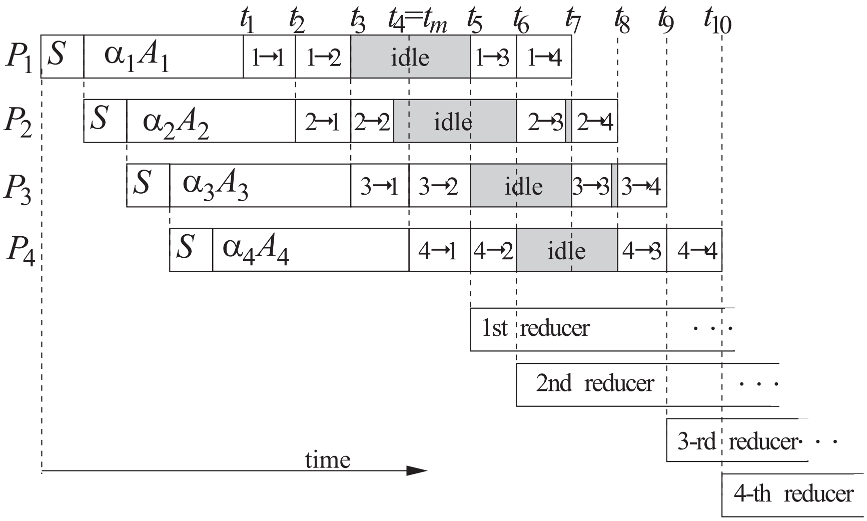
Storing reducer results in the network file system is contention-free.

Observations

Our goal is to:

- partition input load V into mapper chunks $\alpha_1, \ldots, \alpha_m$
- schedule mapper to reducer communications (shuffle phase),
- so that the whole schedule length T is as short as possible.

Communication Schedule



$m = r = 4, \ell = 2$

Load Partitioning

Partitioning of the load can be calculated from the linear program:

$$\textit{minimize} \quad t_{itv(m,r)+1} \tag{4}$$

$$iS + A_i\alpha_i = t_i \quad \textit{for} \quad i = 1, \ldots, m \tag{5}$$

$$\frac{\gamma C}{r}\alpha_k \leq t_{i+1} - t_i \ \textit{for}\ i = 1, \ldots, itv(m, r), k \in vti(i) \tag{6}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{7}$$

where:

$itv(i,j)$	- the number of the interval in which mapper i and reducer j communicate (can be tabulated in $O(mr)$)
$vti(i)$	- the set of mappers sending messages in interval i

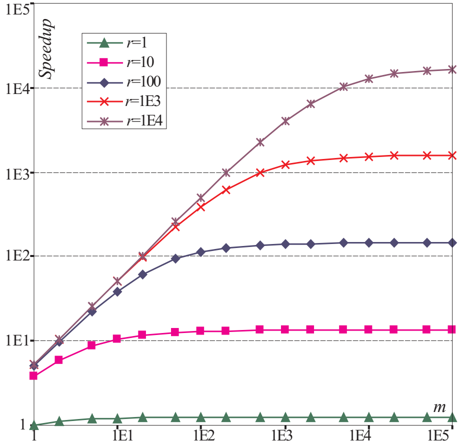
Modeling Setting

In this section we give examples of MapReduce performance analysis using the above scheduling model.

Reference application and system parameters:

lu =16E6	(load unit size 16 MB),
$m = 1000$	(1000 mappers),
$r = \ell = 100$	(100 reducers, at most 100 concurrent channels),
$S = 1$	(application startup 1s per processor),
$C = c^{map}$ =1E-6	(communication speed 1MB/s),
$\gamma = 0.1$	(mapper output is 10% of the input size),
$a^{map} = a^{red}$ =1E-5	(computing rate 10μs/Byte),
$s^{map} = s^{red}$ =1E-2	(computation startup time for each load unit on the mappers, and for the reducers are 10ms),
V = 1E15	(load size is 1PB).

Speedup vs m, r

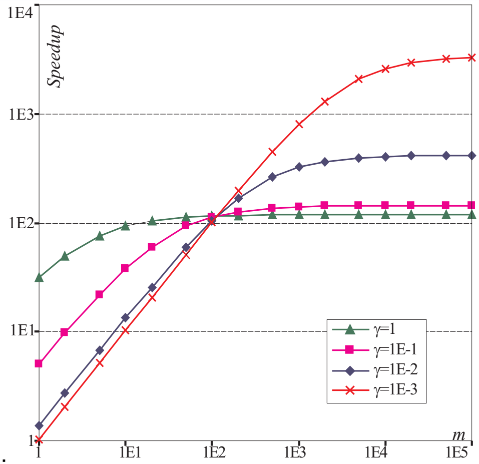


Observation:

- with growing m speedup levels-off around r,

- speedup grows with increasing r.

Speedup vs m, γ



$r = 100$

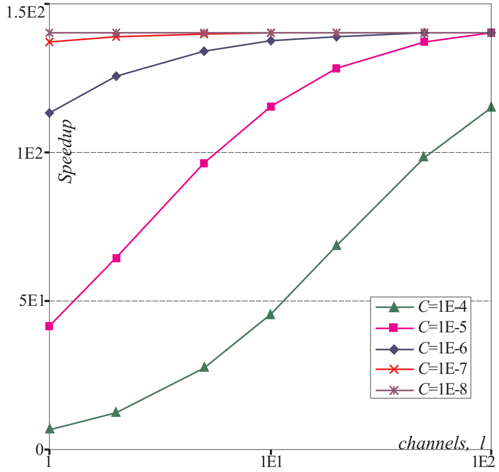
Observation:

- the smaller γ is, the less results are transferred to the reducers,

$\Rightarrow$ when γ is small, systems with $m \gg r$ can be effectively used,

$\Rightarrow$ γ is a key parameter for scalability of MapReduce computations.

Speedup vs bisection width ℓ and communication rate C



Observation:

- the faster the communication is (the smaller C is) the smaller the impact of the bisection width ℓ.

Observations on DLT & MapReduce

- DLT can be applied to analyze MapReduce computations.

Observations on DLT & MapReduce

- DLT can be applied to analyze MapReduce computations.
- The amount of results γV produced by the mappers is a key parameter controlling performance of MapReduce.

Observations on DLT & MapReduce

- DLT can be applied to analyze MapReduce computations.
- The amount of results γV produced by the mappers is a key parameter controlling performance of MapReduce.
    - $\gamma \approx 1 \Rightarrow$ number of mappers m need not be greater than the number of reducers r.

Observations on DLT & MapReduce

- DLT can be applied to analyze MapReduce computations.
- The amount of results γV produced by the mappers is a key parameter controlling performance of MapReduce.
  - $\gamma \approx 1 \Rightarrow$ number of mappers m need not be greater than the number of reducers r.
  - $\gamma \approx 1 \Rightarrow$ MapReduce computations scale well with the number of reducers r.

Observations on DLT & MapReduce

- DLT can be applied to analyze MapReduce computations.
- The amount of results γV produced by the mappers is a key parameter controlling performance of MapReduce.
    - $\gamma \approx 1 \Rightarrow$ number of mappers m need not be greater than the number of reducers r.
    - $\gamma \approx 1 \Rightarrow$ MapReduce computations scale well with the number of reducers r.
- Increasing bisection width ℓ has similar effect as increasing communication speed $1/C$.

Outline of the presentation

1. basic formulation of DLT
2. experimental verification of DLT
3. DLT in MapReduce
4. **time and energy minimization**
5. isoefficiency maps

Why analyzing energy consumption?

Example

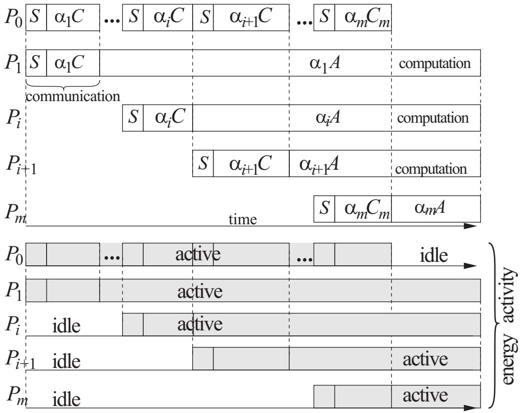
Average power usage of the first 9 supercomputers from the current (November 2021) top500 list is over 10MW.

At the current prices in Poland this would cost $\approx$ 31k EUR daily, and $\approx$11.5M EUR annually.

Why analyzing energy consumption?

- Energy consumption is a factor limiting growth of contemporary data centers, and supercomputing facilities.
- Economizing on energy use is an indispensable element of the future high performance computing.
- Divisible Load Theory can be used as an analytical tool to model and economize on energy usage.

Simple model of energy usage in DLT



Originator is not computing here.

Schedule length $T(m)$ and load split $\alpha_1, \ldots, \alpha_m$ can be computed as explained in the "Basics" Section.

Definitions, energy use features

We assume:

- the shortest schedule $\Rightarrow$ we know $T(m), \alpha_1, \dots, \alpha_m$
- energy is split into:

$\rightarrow$ idle state energy

$\rightarrow$ running state energy used beyond the idle state.

P_C	power consumed by active processors,
P_N	power consumed by active network equipment,
k	reduction in energy consumption in the idle state, i.e. P_C/k, and P_N/k is the power consumed in the idle state,
E_I	idle state energy,
E_{RC}	energy beyond the idle state consumed when processors are running,
E_{RN}	energy beyond the idle state consumed when network is running.

Deriving energy consumption, originator *not* computing

Idle state - base energy consumption

During the whole schedule of length $T(m)$, the originator, m idle processors and the idle network consume energy

$$E_I = T(m)((m+1)P_C + P_N)/k. \tag{8}$$

Running state - network

Processor activation and load distribution time is

$$T_{comm} = \sum_{i=1}^{m} (S + C\alpha_i) = mS + CV. \tag{9}$$

The energy consumed above the network idle state is

$$E_{RN} = P_N \frac{k-1}{k} T_{comm} = P_N \frac{k-1}{k}(mS + CV). \tag{10}$$

Deriving energy consumption, originator *not* computing ...

Running state - processors

$\rightarrow$ Originator is active during load distribution time $mS + CV$.

$\rightarrow$ Other processors: processor P_i is active for time $S + \alpha_i(C + A)$.

The computation energy consumption beyond the idle state is

$$\begin{aligned} E_{RC} &= P_C \frac{k-1}{k}\left(mS + CV + \sum_{i=1}^{m}(S + (C + A)\alpha_i)\right) \\ &= P_C \frac{k-1}{k}(2mS + (2C + A)V). \end{aligned} \tag{11}$$

Total energy consumption

$$E = E_I + E_{RN} + E_{RC}. \tag{12}$$

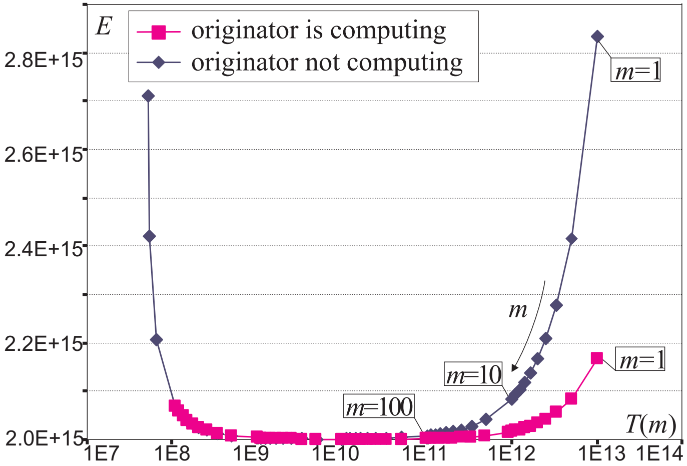
Do power usage assumptions hold?

Table: Power versus problem size.

Size α_i / App.	50MB P_C[W]	50MB cov[%]	200MB P_C[W]	200MB cov[%]	800MB P_C[W]	800MB cov[%]
quicksort	126.8	0.6	127.1	0.9	127.4	1.2
string search	125.7	0.7	126.0	0.5	125.6	0.5
md5	127.4	0.4	126.2	0.6	126.2	0.7
edge detection	131.9	0.5	131.2	0.7	130.7	0.6
matrix transpose	128.9	0.7	128.8	0.6	128.9	0.7
idle	$P_C/k = 73.0$ W, $cov = 2.7\%$, $k \approx 1.8$					
hibernation	$P_C/k = 6.3$ W, $cov = 8.4\%$, $k \approx 20.2$					

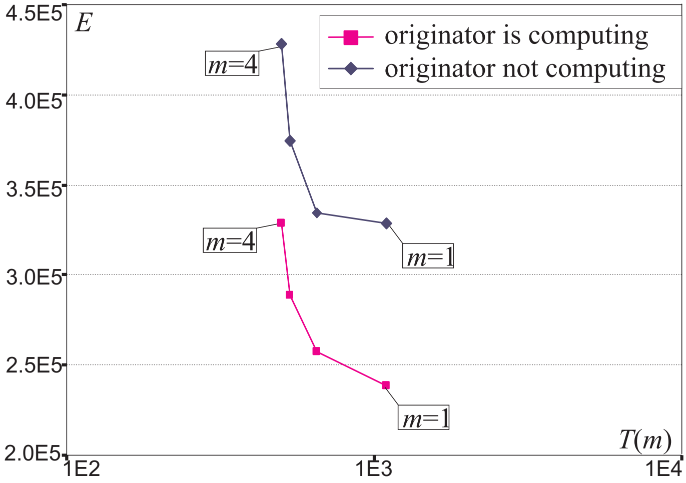
AMD Phenom II X4 945 3.00GHz, 8GB RAM DDR2 800MHz, FreeBSD 8.1 (ca 2012)

Energy E vs. schedule length $T(m)$, $V = 1E13$



$A = 1$, $C = $ 1E-6, $S = $ 1E2,$P_N = 50$, $P_C = 200$, $k = 3$, $V = $ 1E13

Energy E vs. schedule length $T(m)$, $V = 1E3$



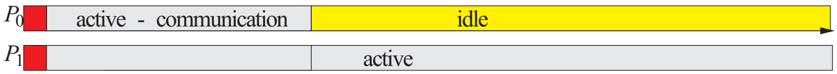
$A = 1$, $C = 1E\text{-}6$, $S = 1E2$, $P_N = 50$, $P_C = 200$, $k = 3$, $V = 1E3$.

Energy E vs. schedule length $T(m)$, conclusions

Intuitively, it could be expected that shorter schedules engage more processors, and hence, shorter schedules should use more energy.

Surprisingly, E as a function of $T(m)$ has a *minimum*.

With growing processor number m schedule length $T(m)$ is decreasing, idle times decrease, energy E (initially) decreases.



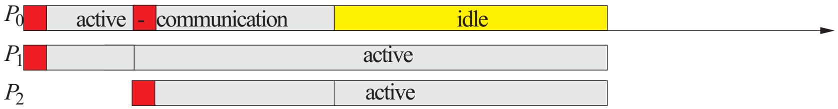
.

Energy E vs. schedule length $T(m)$, conclusions

Intuitively, it could be expected that shorter schedules engage more processors, and hence, shorter schedules should use more energy.

Surprisingly, E as a function of $T(m)$ has a *minimum*.

With growing processor number m schedule length $T(m)$ is decreasing, idle times decrease, energy E (initially) decreases.



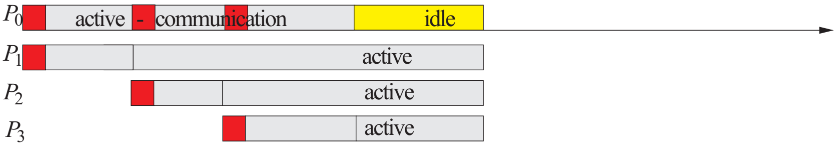
.

Energy E vs. schedule length $T(m)$, conclusions

Intuitively, it could be expected that shorter schedules engage more processors, and hence, shorter schedules should use more energy.

Surprisingly, E as a function of $T(m)$ has a *minimum*.

With growing processor number m schedule length $T(m)$ is decreasing, idle times decrease, energy E (initially) decreases.



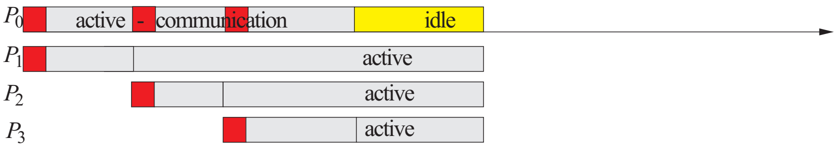
.

Energy E vs. schedule length $T(m)$, conclusions

Intuitively, it could be expected that shorter schedules engage more processors, and hence, shorter schedules should use more energy.

Surprisingly, E as a function of $T(m)$ has a *minimum*.

With growing processor number m schedule length $T(m)$ is decreasing, idle times decrease, energy E (initially) decreases.



We can save energy by parallel processing!

Outline of the presentation

1. basic formulation of DLT
2. experimental verification of DLT
3. DLT in MapReduce
4. time and energy minimization
5. **isoefficiency maps**

Why isoefficiency maps? Key idea

In this part of the presentation DLT applicability in tracing complex performance interactions is demonstrated.

Thus DLT becomes analytical performance model.

Isoefficiency maps are visual representations of complex interactions by use of *isolines*, i.e. as set of points of equal value of some measure (e.g. performance) in 2D projection of system and application parameters.

Isoline maps examples – in cartography

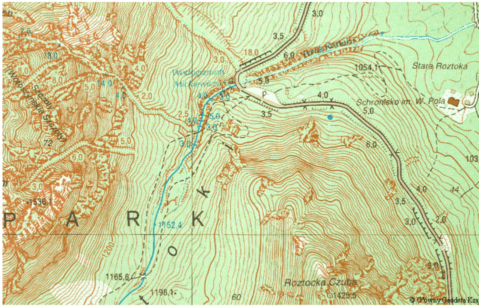


Figure: Contour lines join points of equal elevation above sea level

Isoline maps examples – in meteorology

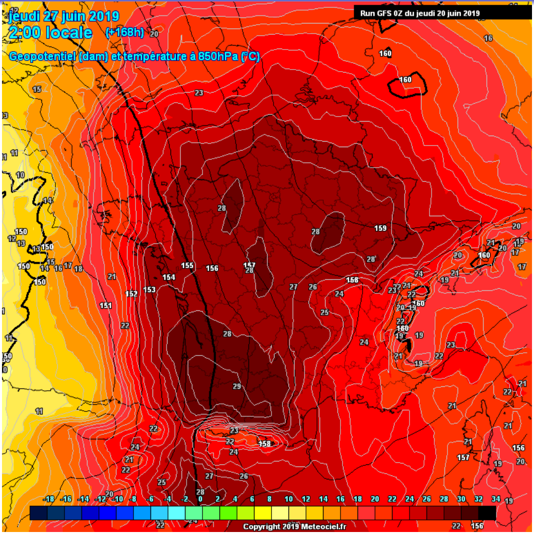


Figure: Isotherms France on 27.VI.2019

Isoline maps examples – in thermodynamics

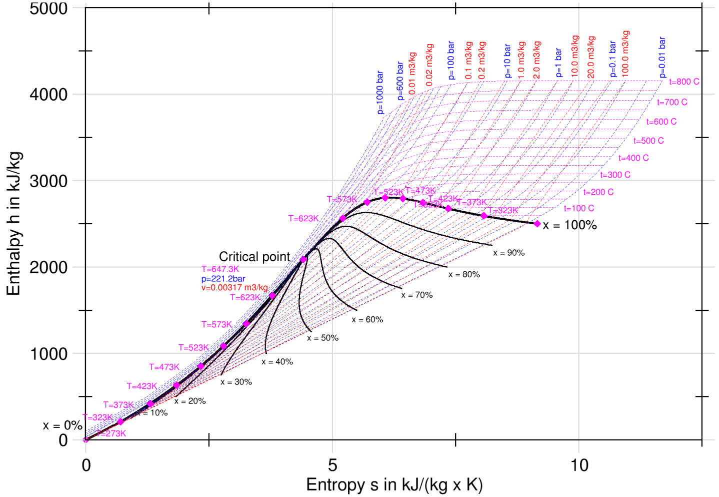


Figure: Enthalpy-entropy chart for water and steam

Why isoefficiency maps? Motivation

• Thus, such visualizations proved very effective in building understanding of sensitivities and relationships of complex phenomena in other areas of science and technology (isotherms, isobars, isogons, . . .)

• We want to do the same! To grasp quickly and communicatively the impacts of system and application parameters on HPC performance.

• While earlier approaches to the HPC performance analysis studied scalability with machine number (speedup) at fixed problem size.

Basic Performance Indexes

Ability to scale parallel computations is measured by

- speedup:

$$\mathcal{S}(m) = \frac{T(1)}{T(m)} \tag{13}$$

- efficiency:

$$\mathcal{E}(m) = \frac{\mathcal{S}}{m} = \frac{T(1)}{m \times T(m)}, \tag{14}$$

where $T(i)$ is execution time on i machines.

$\mathcal{S}$ should grow with m (preferably linearly).

$\mathcal{E}$ should be as close to 1 as possible.

Yet, in most cases speedup saturates at certain number of machines m, efficiency decreases with m.

Isoefficiency Function

Parallel performance depends on the problem size – bigger problems allow to exploit more processors with higher efficiency. *Isoefficiency function* was invented [GK93] to grasp this relationship.

Definition

Isoefficiency function $I(e, m)$ is size of the problem required to maintain efficiency $\mathcal{E}(m) = e$ on m processors.

Isoefficiency Function

Example

A parallel algorithm finding MST in a graph with v vertices has complexity:
$T(m, v) = c_1 v^2/m + c_2 v \log m$
Efficiency is $\mathcal{E}(m, v) = e = c_1 v^2/(c_1 v^2 + c_2 vm \log m)$.
Isoefficiency function is $I(e, m) = c_2 em \log m/(c_1(1 - e))$.

- Thus, classic isoefficiency function $I(e, m)$ determines relationship between problem size and machine number.

- We will extend it to arbitrary pairs of parameters.

Isoefficiency Map Construction

- $T(m, A, C, S, V)$ – schedule length obtained from (1)-(3) for: m machines, A, C, S system & application parameter, and problem size V.
- $T(1, A, C, S, V) = S + CV + AV$ – schedule length on a single machine.
- Efficiency:

$$\mathcal{E}(m, A, C, S, V) = T(1, A, C, S, V)/(m \times T(m, A, C, S, V)).$$

Isoefficiency Map Construction

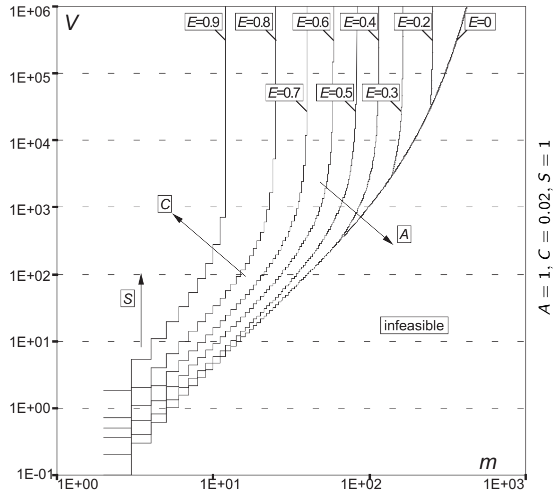
- Isoefficiency function:

$$I(e,x,y) = \{(x,y) : \mathcal{E}(m,A,C,S,V) = e, \quad (15)$$
$$\forall p \in Param \setminus \{X,Y\}\ p = const\}. \quad (16)$$

where:
X, Y - a pair of interesing parameters to be presented in a 2D map
(x, y) - a pair of particular values of parameters X, Y
$Param$ - set of all system parameters
$p = const$ - constant value of one particular parameter p in the set $Param \setminus \{X, Y\}$

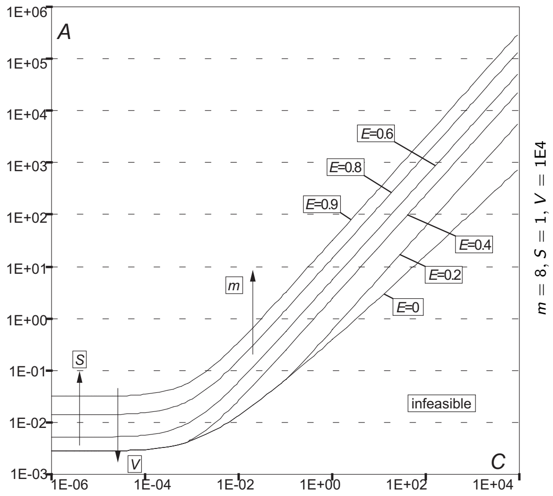
- Due to the complex nature of (1)-(3) closed-form formulas of $I(e,x,y)$ exist only for some pairs for parameters, for the others $I(e,x,y)$ was found numerically.

Isoefficiency map: V vs m



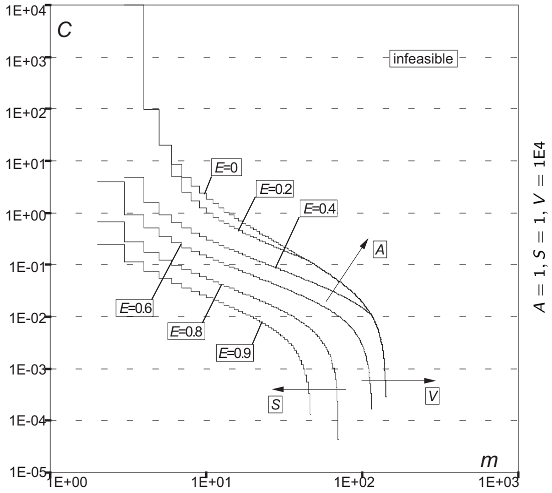
When m grows also V should grow for constant efficiency. But not all machine numbers m can be feasibly used even for very large V (because $S > 0$).

Isoefficiency map: A vs C



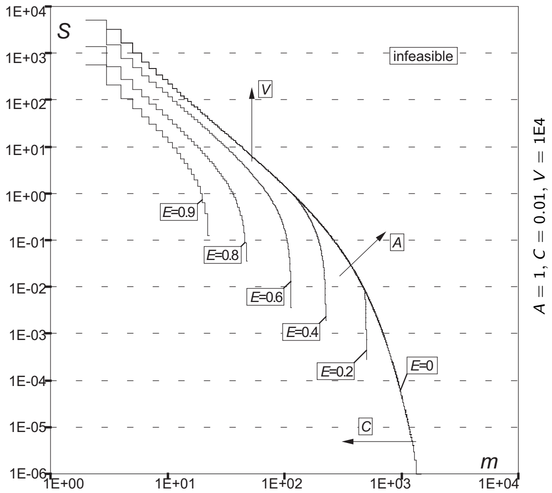
Computing speed $1/A$ and communication speed $1/C$ can compensate each other, but only if they both are very slow. In typical conditions they are disconnected.

Isoefficiency map: C vs m



When m is small even slow communication allows for good efficiency (left). In typical conditions speed of communication must increase (C decreases) to use big numbers of machines (center). Ultimately, use of arbitrarily large m cannot be allowed by increasing communication speed (because $S > 0$, right).

Isoefficiency map: S vs m



Startup (fixed overhead) S must quickly decrease with machine number m for constant efficiency. Ultimately, use of arbitrarily large m cannot be allowed by decreasing startup time S (because $C > 0$).

Can there be Iso-Maps for other performance measures?

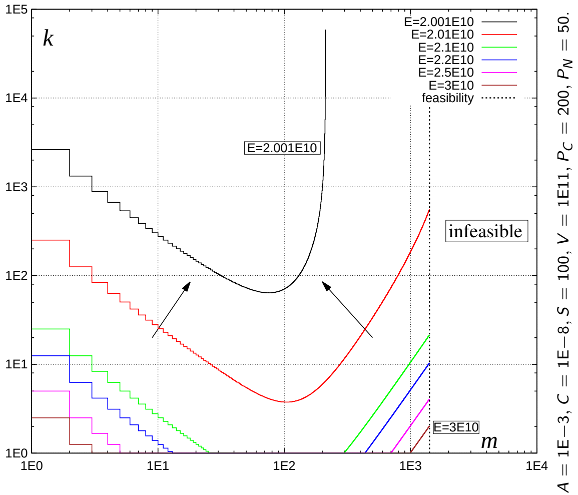
• We constructed isoefficiency maps for one measure of HPC performance: the *efficiency*.

• Can this be repeated for other HPC performance indicators?

• Yes! For example, for energy – maps of equal energy consumption can be constructed.

• We already know how to calculate energy usage, so it is doable to calculate what one system parameter should be as a function of another parameter, to obtain certain energy consumption while the remaining parameters are fixed.

Iso-Energy map: k vs m



k – reduction in electric power consumption when idle.
When increasing processor number m we reduce overheads and energy consumption, this can be wasted by less effective machine idle states (k decreases, left). Yet, ultimately for large machine numbers, constant energy consumption cannot be achieved by just more effective idle state (k is growing, right).

Conclusions

1. In this presentation Divisible Load Theory (DLT) was introduced
2. DLT was applied to analyze Big Data applications (MapReduce)
3. DLT was applied to analyze energy consumption in parallel processing
4. DLT was used to build iso-efficiency maps and facilitate understanding of complex relationships between system and application parameters

Conclusions

1. In this presentation Divisible Load Theory (DLT) was introduced
2. DLT was applied to analyze Big Data applications (MapReduce)
3. DLT was applied to analyze energy consumption in parallel processing
4. DLT was used to build iso-efficiency maps and facilitate understanding of complex relationships between system and application parameters

Indeed, DLT is a very versatile theory.

The End

Thank you for your attention.

Please remember about Bharadwaj Veeravalli talk this afternoon