Divisible load theory

Maciej Drozdowski

Maciej.Drozdowski@cs.put.poznan.pl

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

https://schedulingseminar.com/
23.X.2024

Outline of the presentation

1. basic formulation of DLT
2. experimental verification of DLT
3. extending the model
4. computational complexity
5. performance visualization with isolines

Divisible load theory

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

Load – is usually some data to be processed.

In DLT it is assumed that:

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

Divisible load theory

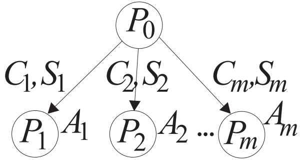
Consequently in divisible computations:
$\Rightarrow$ grain of parallelism is small,
$\Rightarrow$ data dependencies are negligible,
$\Rightarrow$ schedule optimization consists in partitioning the load according to the speeds of communication, computation and other platform features.

Examples of divisible applications:[1]

- distributed searching for patterns in text, audio, graphic etc. files,
- database, measurements, image processing,
- compression,
- some linear algebra algorithms, and simulation,
- MapReduce big data processing.

[1] more on the applications in the following

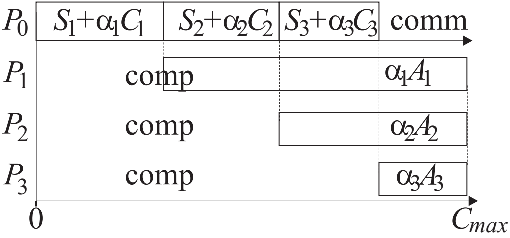
Basic scheduling model



- A single level tree (a.k.a. a star) interconnection
- P_0 - originator, distributes load, does not compute
- $P_1, \ldots, P_m$ - processors (workers) receive and process the load
- V - load size (e.g. in bytes)
- $S_i + \alpha C_i$ - communication delay for sending load α to P_i
- $A_i \alpha$ - computation time for load α 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

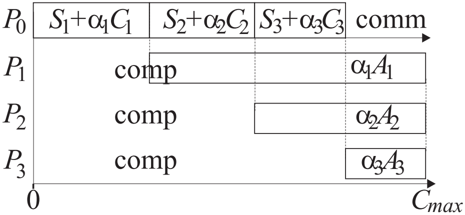


α_i - size of load part sent to processor P_i
C_{max} - schedule length

The challenge: choose α_is such that C_{max} is as short as possible.

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

Solution by a system of linear equations



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

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

The above system of linear equations can be solved in $O(m)$ time due to its special structure:

Closed form solution of a system of linear equations

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

α_i can be expressed as a linear function $k_i \alpha_m + l_i$ of α_m,

$$\begin{aligned} k_i &= k_{i+1}(A_{i+1} + C_{i+1})/A_i \qquad && \textit{for } i = 1, \ldots, m-1 \\ l_i &= S_{i+1}/A_i + l_{i+1}(A_{i+1} + C_{i+1})/A_i \qquad && \textit{for } i = 1, \ldots, m-1 \\ k_m &= 1, l_m = 0, && \end{aligned}$$

Then we have
$$\alpha_m = \frac{V - \sum_{i=1}^{m} l_i}{\sum_{i=1}^{m} k_i}.$$

Closed form solution – observations

Recall:

$$\alpha_m = \frac{V - \sum_{i=1}^m l_i}{\sum_{i=1}^m k_i} \quad l_i = \frac{S_{i+1}}{A_i} + l_{i+1}\frac{A_{i+1}+C_{i+1}}{A_i} \quad i = 1, \ldots, m-1$$

- $\forall i, S_i = 0 \Rightarrow \forall i, l_i = 0$ and $\forall i, \alpha_i > 0$ for arbitrarily large m (strange! unrealistic)
- $S_i > 0 \Rightarrow$ a feasible solution (i.e. with $\forall \alpha_i > 0$) may not exist, because load size V is too small to activate all processors.
- $\Rightarrow$ communication startup S_i is necessary as a practical irreducible yardstick of time.[2]

[2] or some other discrete element

Outline of the presentation

1. basic formulation of DLT
2. **experimental verification of DLT**
3. extending the model
4. computational complexity
5. performance and isolines

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)-(2).
- 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}?

Representing returning of the results

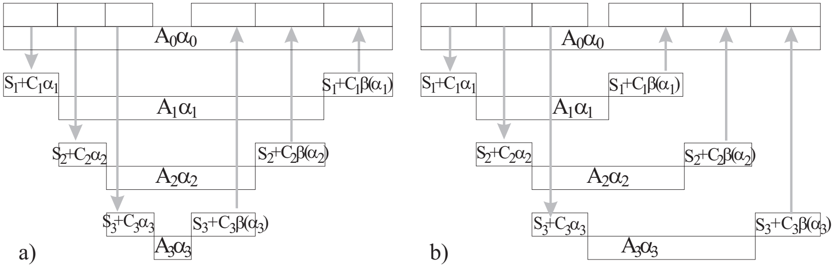
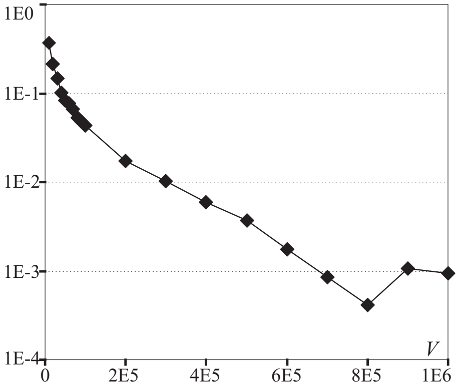


Figure: a)LIFO, b)FIFO orders of returning results.

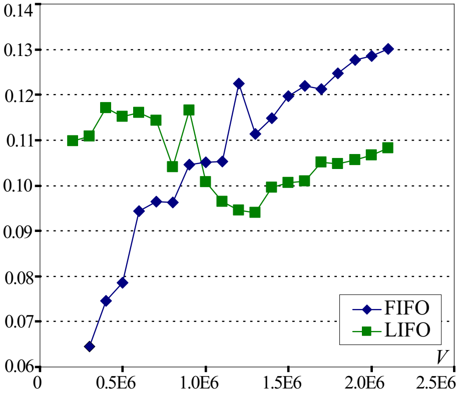
$\beta(\alpha)$ is the size of the results as a function of the input load size.

Model relative error



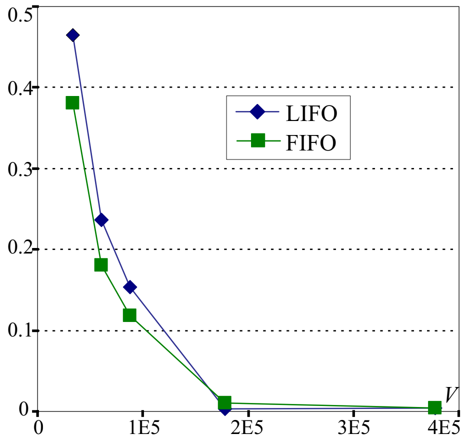
Platform: Transputer system (ca. 1996)
Application: search for a pattern in a text file, LIFO
Error: $< 1\%$ feasible.

Model relative error



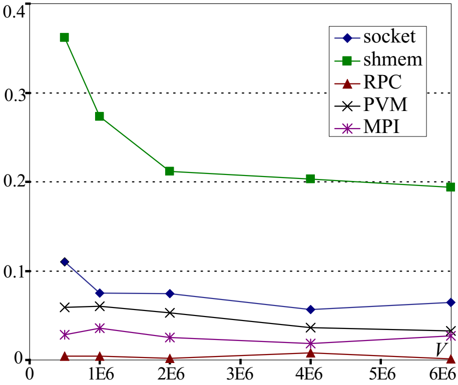
Platform: IBM SP2, PVM (ca. 1997)
Application: LZW compression
Error: 9 – 13% feasible.

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
- accuracy improves with problem size V
- DLT model is practical and relevant.

Outline of the presentation

1. basic formulation of DLT
2. experimental verification of DLT
3. **extending the model**
4. computational complexity
5. performance and isolines

Extending the model

Presented extensions:

- multi-installment processing
- interconnection networks
- and time windows, and cost, and memory
- also hierarchical memory

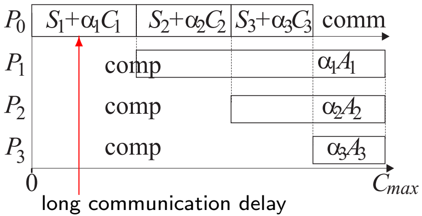
Default assumptions:

- originator P_0 is not computing, but only communicating
- result return time is negligible and is not explicitly scheduled
- worker processors can receive load and compute in parallel
- set of used processors and communication sequence are given[3]

[3] more on this in the complexity section

Multiple Installments

Why multi-installment processing?



Example:
$m = 1$ processor, $V = 10$,
$C_1 = 1, A_1 = 1, S_1 = 0$,

1 installment:
$C_{max} = V(C_1 + A_1) = 20$.

$k = 10$ installments:
$C_{max} = V(C_1/k + A_1) = 11$.

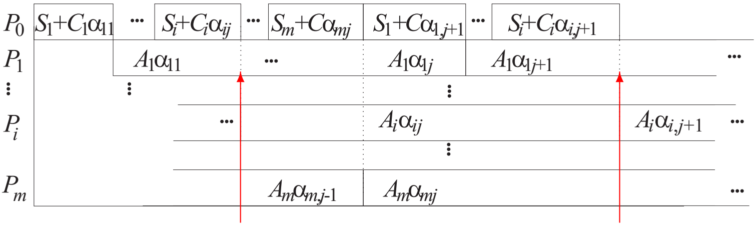
$\Rightarrow$ Multi-installment processing allows to shorten the first communication delay, start computations earlier[4]

Practical question: what should the number k of installments be?

[4] also a method to respect processor limited memory

Multiple Installments - calculating partitions

Partial view of a schedule for multi-installment processing in a star

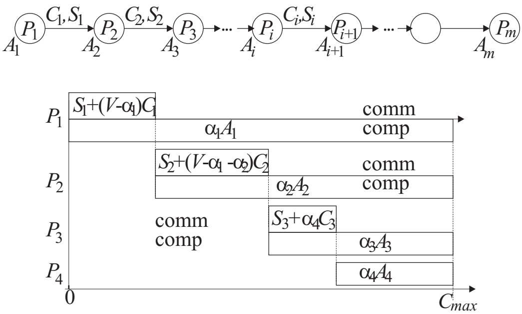


System of linear equations to calculate installments sizes α_{ij}, $i = 1, \dots, m$ — processors, $j = 1, \dots, k$ — installments:

$$\alpha_{ij} A_i = \sum_{\ell=i+1}^{m} (S_\ell + C_\ell \alpha_{\ell,j}) + \sum_{\ell=1}^{i} (S_\ell + C_\ell \alpha_{\ell,j+1}) \quad \text{for } i = 1, \dots, m, j = 1, \dots, k \tag{3}$$

$$V = \sum_{j=1}^{k} \sum_{i=1}^{m} \alpha_{ij} \tag{4}$$

Interconnection Networks – Chain

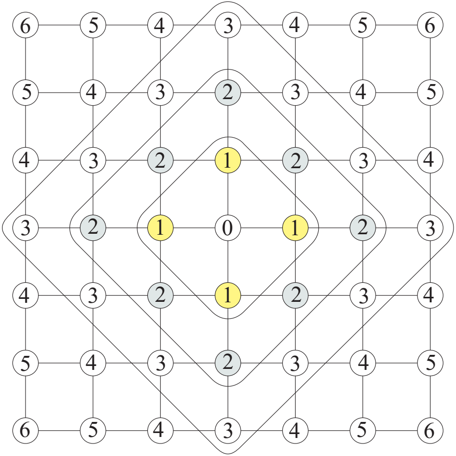


Load partition calculation:

$$\alpha_i A_i = S_i + C_i \sum_{j=i+1}^{m} \alpha_j + \alpha_{i+1} A_{i+1} \qquad \text{for } i = 1, \ldots, m-1 \quad (5)$$

$$\sum_{i=1}^{m} \alpha_i = V \quad (6)$$

Interconnection Networks – 2D-Mesh



Challenge: find a load scattering algorithm in a mesh!

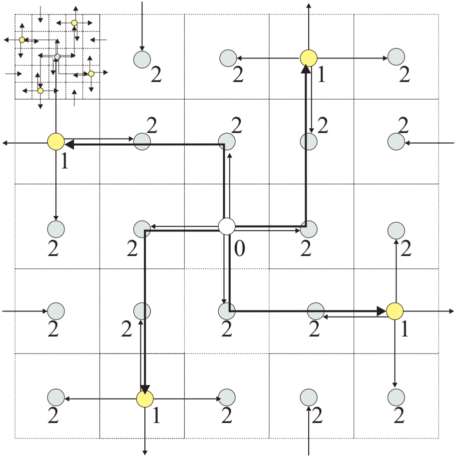
Problem: this solution has asymmetry in layer processor connectivity

Observation: this scattering method assumes communication delay dependence on distance (which needs not be true)

Observation: there are packet routing technologies with weak dependence on distance (e.g. circuit switching, wormhole routing)

Conclusion: load scattering should be done differently

Interconnection Networks – 2D-Mesh



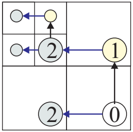
Innovation: scattering with $p = 4$ simultaneously used processor ports

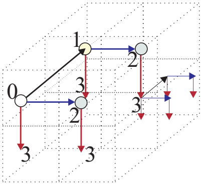
Observation: it works because communication delay only *weakly depends on distance* $\Rightarrow$ *it is advantageous to distribute far away and then locally*

Problem/Question: can this be done with other numbers of ports p?

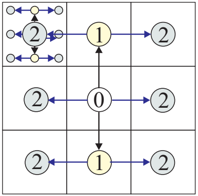
Problem/Question: can this be done in other number of mesh dimensions?

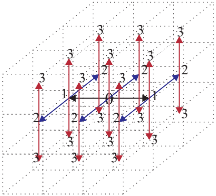
Interconnection Networks – Mesh





Example: scattering with $p = 1, 2$ ports in 2D-, 3D-meshes, but it can be generalized to $p = 1, \ldots, 2 * dimensions$.

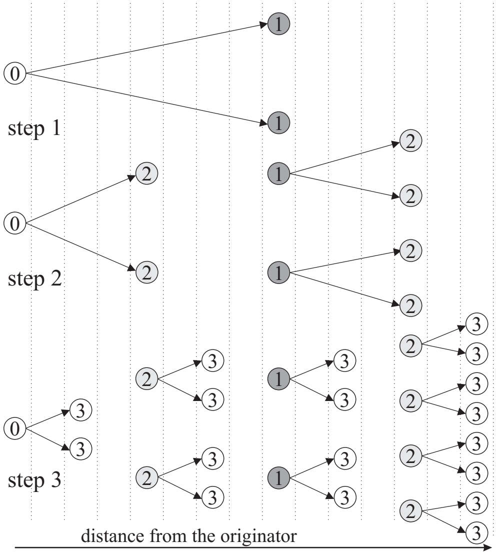




Observation: actually we are embedding some kind of a tree in a communication network

Conclusion: actually we use $p+1$-nomial heap

$p + 1$-nomial heap mode of operation



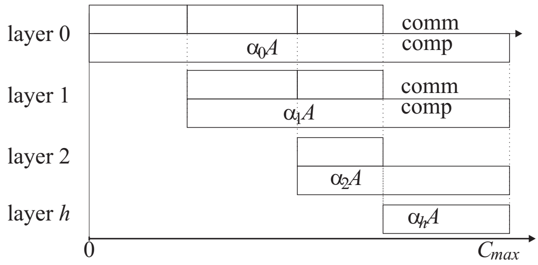
Example: $p = 2$, 3-nomial heap

layer – a set of processors activated in the same scattering step, hence, on the same level of $p + 1$-nomial heap

Observations:

- $(p + 1)^i$ processors are active and computing after step $i = 0, \ldots, h$
- in each step p times new processors are activated
- $p(p + 1)^i$ processors are activated in step $i = 0, \ldots, h - 1$

Processing in a network with $p+1$-nomial heap embedded



Load partition calculation:

$$\alpha_0 A = Sh + C(p+1)^{h-1}\alpha_h + \alpha_h A \quad (7)$$

$$\alpha_i A = S(i-1) + C(p+1)^{i-2}\alpha_{i-1} + \alpha_{i-1} A$$
$$\text{for } i = h, \ldots, 2 \quad (8)$$

$$V = \alpha_0 + p\sum_{i=1}^{h}(p+1)^{i-1}\alpha_i \quad (9)$$

And include also time windows, and memory, and cost

Assumptions:

- $[r_i, d_i]$ – processor P_i availability window,
- B_i – processor P_i memory limit,
- p_i – processor P_i computation startup time,
- $f_i + \ell_i\alpha$ – cost of processing load α on P_i,
- minimize makespan T *subject to cost limit* K, because this is bi-criterion problem,
- plus the previous default assumptions: single level tree (star), originator P_0 is only communicating, result return time is negligible, set of used processors and communication sequence are given.

And include also time windows, and memory, and cost

$$\mathrm{LP}_{\text{time}}(K): \qquad \min T \tag{10}$$

s.t.

$$\sum_{k=1}^{i} (S_k + C_k\alpha_k) + (p_i + A_i\alpha_i) \le T, \qquad i = 1, \dots, m, \tag{11}$$

$$r_i + (p_i + A_i\alpha_i) \le T, \qquad i = 1, \dots, m, \tag{12}$$

$$\sum_{k=1}^{i} (S_k + C_k\alpha_k) + (p_i + A_i\alpha_i) \le d_i, \qquad i = 1, \dots, m, \tag{13}$$

$$r_i + (p_i + A_i\alpha_i) \le d_i, \qquad i = 1, \dots, m, \tag{14}$$

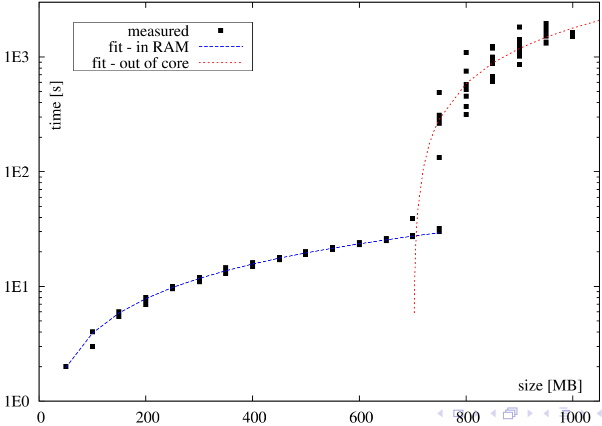
$$0 \le \alpha_i \le B_i, \qquad i = 1, \dots, m, \tag{15}$$

$$\sum_{i=1}^{m} (f_i + \ell_i\alpha_i) \le K, \tag{16}$$

$$\sum_{i=1}^{m} \alpha_i = V. \tag{17}$$

Hierarchical Memory and Energy Cost

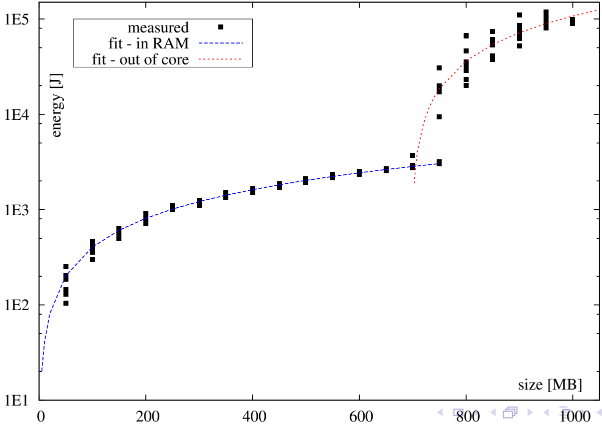
- Contemporary computers have hierarchical memory.
- Out of core memory is virtually unlimited,
- but it is 1-2 orders of magnitude slower.



Pentium IV@2.8 GHz, 1 GB RAM@266 MHz, HDD Caviar WD400, FreeBSD 9.0, image edge detection, ca 2018

Hierarchical Memory and Energy Cost

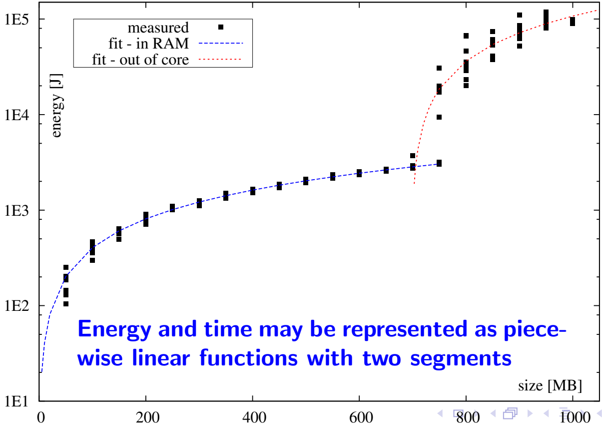
- Contemporary computers have hierarchical memory.
- Out of core memory is virtually unlimited,
- but it is 1-2 orders of magnitude slower.



Pentium IV@2.8 GHz, 1 GB RAM@266 MHz, HDD Caviar WD400, FreeBSD 9.0, image edge detection, ca 2018

Hierarchical Memory and Energy Cost

- Contemporary computers have hierarchical memory.
- Out of core memory is virtually unlimited,
- but it is 1-2 orders of magnitude slower.



Pentium IV@2.8 GHz, 1 GB RAM@266 MHz, HDD Caviar WD400,FreeBSD 9.0, image edge detection, ca 2018

Hierarchical Memory and Energy Cost

$$\mathrm{LP}_{\text{cost}}(T): \qquad \min \textit{Energy} = \sum_{1=1}^{m} E_i \tag{18}$$

s.t.

$$\text{core}: \quad \sum_{k=1}^{i} (S_k + C_k\alpha_k) + (A_{1i}\alpha_i) \leq T, \qquad i = 1, \ldots, m, \tag{19}$$

$$\text{out of core}: \sum_{k=1}^{i} (S_k + C_k\alpha_k) + (p_{2i} + A_{2i}\alpha_i) \leq T, \qquad i = 1, \ldots, m, \tag{20}$$

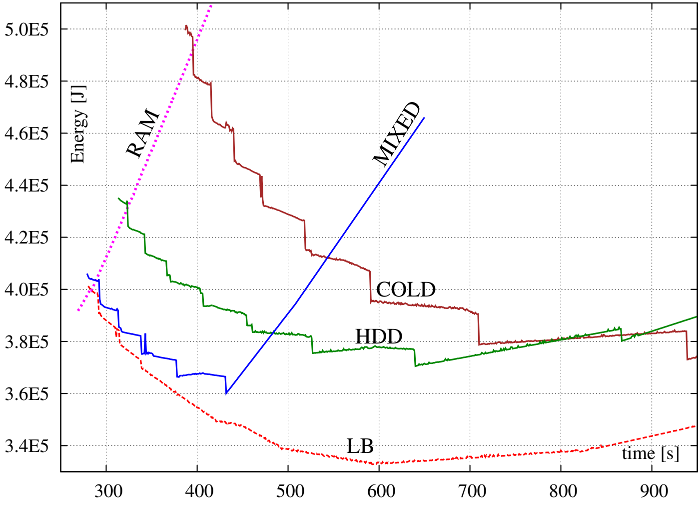
$$\ldots$$

$$\text{core}: \quad \ell_{1i}\alpha_i \leq E_i, \qquad i = 1, \ldots, m, \tag{21}$$

$$\text{out of core}: \quad f_{2i} + \ell_{2i}\alpha_i \leq E_i, \qquad i = 1, \ldots, m, \tag{22}$$

$$\ldots$$

Hierarchical Memory and Energy Cost



Time and energy cost of processing fixed amount of data when starting from various energy saving modes.

Outline of the presentation

1. basic formulation of DLT
2. experimental verification of DLT
3. extending the model
4. **computational complexity**
5. performance and isolines

Computational complexity of DLT

- case: makespan for $S_i = 0, p_i = 0$
- fixed parameter tractability
- first **NP**-hardness proof
- **NP**-hardness for linear communication, computation times and cost

Default assumptions:

- single level tree (star), originator is not computing, result return time is negligible
- availability windows, memory limits, and other features and cost criterion are not binding if not explicitly mentioned

Computational complexity of DLT

The challenges (i.e. scheduling decisions):

1. choose the **subset** of active processors $\mathcal{P}' \subseteq \mathcal{P}$, i.e. performing computation;
2. choose the **sequence** of activating processors in $\mathcal{P}'$
3. calculate load chunk sizes α_i for $P_i \in \mathcal{P}'$

Complexity – case $S_i = p_i = 0$, min. T (makespan)

- **all** processors take part in the computation
- **communication sequence** – activate processors in the order of non-increasing communication *speed*: $C_1 \leq C_2 \leq \cdots \leq C_m$
- **proof:** by interchange argument.

It is rather counterintuitive that:
1) all processors can take part in the computation,
2) processor speed plays no role.

Complexity – fixed parameter tractability for m

In order to:

1. choose the **subset** of active processors $\mathcal{P}' \subseteq \mathcal{P}$ – enumerate all possible 2^m subsets,
2. choose the **sequence** of activating processors – enumerate all possible $m!$ permutations,
3. calculate load chunk sizes α_i by using a linear program with m variables (α_i for $i = 1, \ldots, m$) and $O(m)$ constraints.
4. Hence, for fixed m computational complexity is $O(2^m m! LP(m, O(m)))$.

Complexity – 1st **NP**-hardness proof

Theorem

Divisible load scheduling with memory constraints is **NP***-hard.*

Proof: Reduction from PARTITION:
Given set $E = \{e_1, \ldots, e_q\}$ decide if there set $E' \subset E$, satisfying $\sum_{j \in E'} e_j = \sum_{j \in E - E'} e_j = \frac{1}{2} \sum_{j=1}^{q} e_i = L$ exists.
Without loss of generality we assume that $\forall_{j \in D} e_j > 1$.

Divisible load scheduling instance: $m := q + 1$, $V = L^6 + L$, $C_1 \ldots C_m := 0$, $S_i := e_i$, $A_i := \frac{L}{e_i}$, $B_i := e_i$ for $i := 1, \ldots, q$, $S_m := L$, $C_m := 0$, $A_m := \frac{1}{L^6}$, $B_m := L^6$ for $i := 1, \ldots, q$.

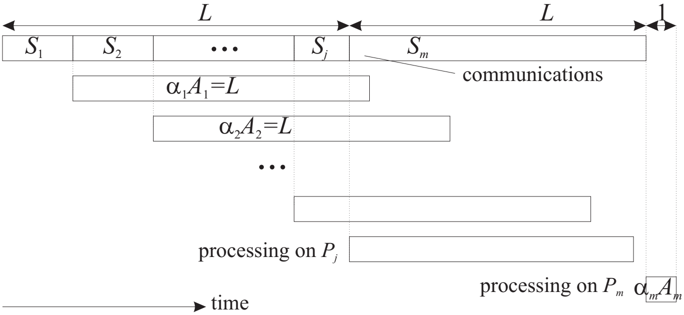
Is possible to process load V in time at most $2L + 1$?

Complexity – 1st NP-hardness proof

makespan, 2 fixed overheads

Divisible load scheduling instance: $m := q+1$, $C_1 \dots C_m := 0$, $S_i := e_i$, $A_i := \frac{L}{e_i}$, $B_i := e_i$ for $i := 1, \dots, q$, $S_m := L$, $C_m := 0$, $A_m := \frac{1}{L^6}$, $B_m := L^6$ for $i := 1, \dots, q$.

Is possible to process volume $V = L^6 + L$ of load on the above network in time at most $2L + 1$?



Complexity – $S_i = p_i = f_i = 0$

makespan, cost, NO fixed overheads

Theorem

Divisible load scheduling for a given makespan and minimum cost is **NP***-hard even for strictly linear processor communication, computation times and cost.*

Complexity – $S_i = p_i = f_i = 0$

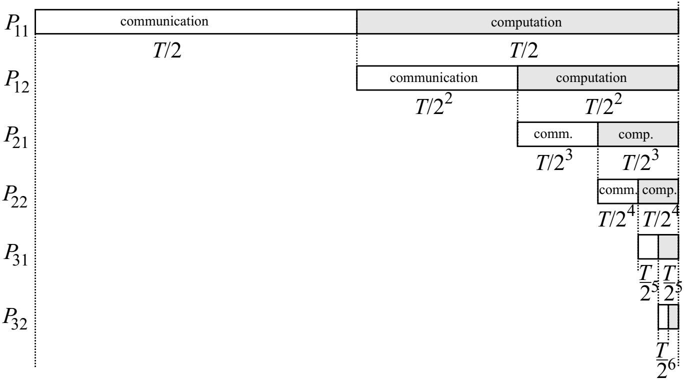
Proof: Reduction from EVEN-ODD PARTITION:
Given set $E = \{e_1, \ldots, e_{2q}\}$ decide if set $E' \subset E$, satisfying $\sum_{j \in E'} e_j = \sum_{j \in E - E'} e_j = \frac{1}{2} \sum_{j=1}^{q} e_i = L$ and such that E' contains exactly one element from pair e_{2i-1}, e_{2i}, for $i = 1, \ldots, n$ exists.

For some arbitrary makespan $T > 0$, divisible load scheduling instance for $i = 1, \ldots, q$:

$$\begin{aligned} A_{2i-1} &= C_{2i-1} = \frac{T}{2^{2i-1}\left(L^{q-i+2} + e_{2i-1}\right)}, \\ \ell_{2i-1} &= \frac{e_{2i-1}}{L^{q-i+2} + e_{2i-1}}, \\ A_{2i} &= C_{2i} = \frac{T}{2^{2i-1}\left(L^{q-i+2} + e_{2i}\right)}, \\ \ell_{2i} &= \frac{e_{2i}}{L^{q-i+2} + e_{2i}}. \end{aligned} \tag{23}$$

Complexity – $S_i = p_i = f_i = 0$

makespan, cost, NO fixed overheads



Is there a schedule of cost $K \leq \frac{3}{2}L$ for load $V = \frac{3}{2}\sum_{i=1}^{q+1} L^i$? ■

Outline of the presentation

1. basic formulation of DLT
2. experimental verification of DLT
3. extending the model
4. computational complexity
5. **performance and isolines**

Isoline maps examples – in meteorology

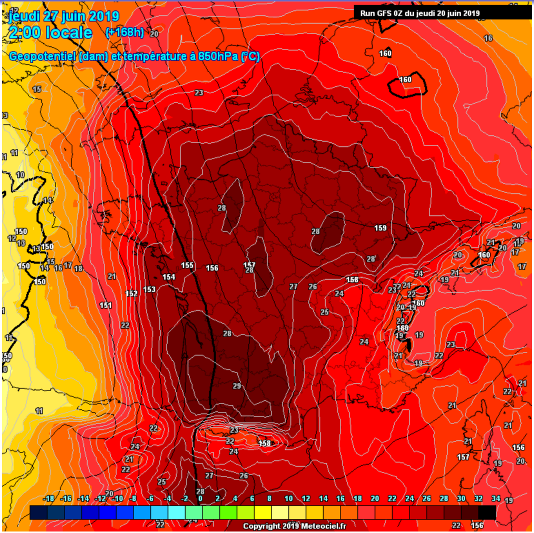


Figure: Isotherms France on 27.VI.2019

Isoline maps examples – in thermodynamics

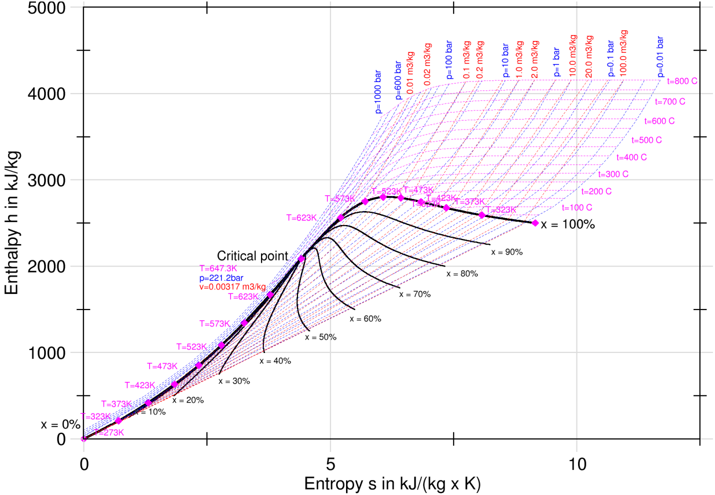


Figure: Enthalpy-entropy chart for water and steam

Why isoefficiency maps? Motivation

- Such visualizations proved effective in building understanding of sensitivities and relationships of complex phenomena in many areas of science and technology (isotherms, isobars, isogons, ...) We want to do the same!

- *Isoefficiency maps* are visual representations of the system parameter interactions by use of *isolines*, i.e. set of points of equal parallel efficiency in 2D projection of system parameters.

- Thus DLT becomes an analytical performance model.

Basic Performance Measures

Classically:

- speedup:

$$\mathcal{S}(m) = \frac{T(1)}{T(m)} \tag{24}$$

- efficiency:

$$\mathcal{E}(m) = \frac{\mathcal{S}}{m} = \frac{T(1)}{m \times T(m)}, \tag{25}$$

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

In the DLT model:

- Efficiency:

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

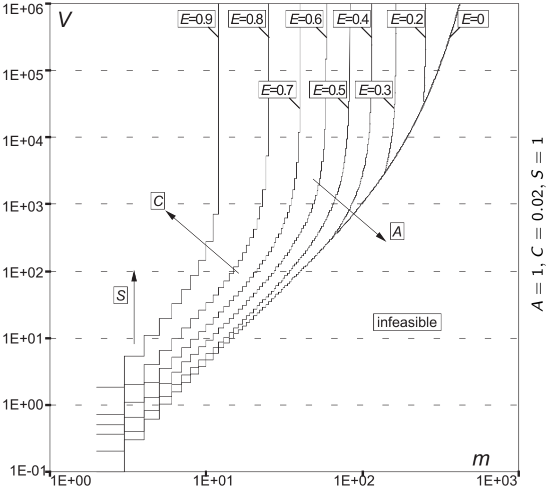
Isoefficiency Map Construction

- Isoefficiency line:

$$\begin{aligned} I(e, X, Y) \;\; = \;\; & \{(x, y) : \mathcal{E}(m, A, C, S, V) = e, \\ & \forall p \in \mathit{Param} \setminus \{X, Y\}\; p = \mathit{const}, \\ & x \in X, y \in Y\}. \end{aligned} \qquad (26)$$

where:
e – efficiency level
X, Y – a pair of interesting parameters to be presented in a 2D map
(x, y) – a pair of particular values of parameters X, Y
Param – set of all model parameters: m, A, C, S, V
$p = \mathit{const}$ – constant value of one particular parameter p in the set $\mathit{Param} \setminus \{X, Y\}$

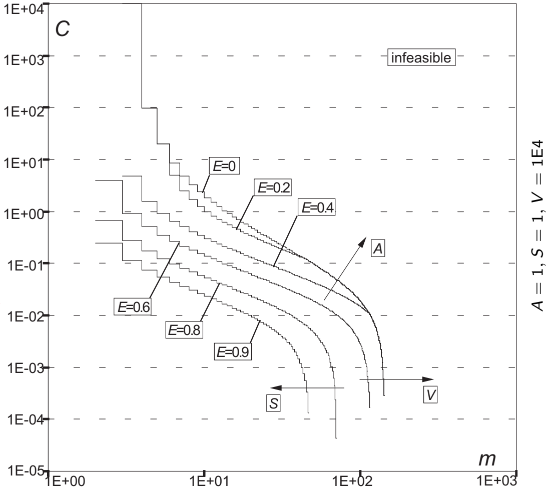
Isoefficiency map: V vs m



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: C vs m

Increasing communication speed.



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, arbitrarily large m cannot be supported by increasing communication speed (because $S > 0$, right).

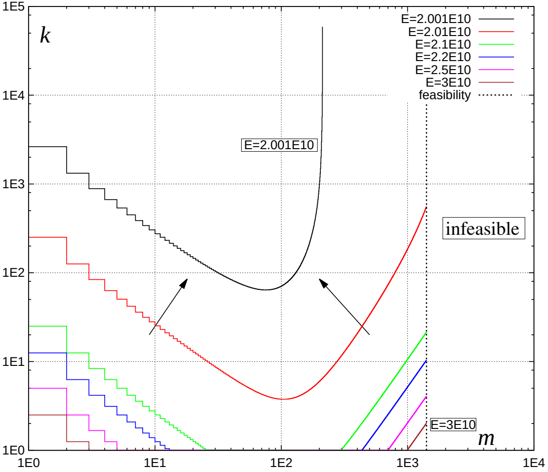
Iso-Maps for other performance measures?

- Such a visualization method can be repeated for other HPC performance indicators.

- For example, for energy – maps of equal energy consumption can be constructed.

Iso-Energy map: k vs m

More aggressive energy saving, harder technologically.



$A = 1E{-3}$, $C = 1E{-8}$, $S = 100$, $V = 1E11$, $P_C = 200$, $P_N = 50$.

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

DLT is an attractive scheduling model because:

- DLT is comprehensive – many details of computing platform can be represented in DLT,
- DLT is a good compromise between complexity and accuracy,
- to some extent DLT is computationally easy,
- DLT is an analytical performance model used to build iso-efficiency and iso-energy maps for understanding of complex relationships between system and application parameters.

Thank you for your attention

A kind Request For Comments:
see `https://arxiv.org/abs/2401.00947`
and tell me what you think

Maciej.Drozdowski@cs.put.poznan.pl

Further reading

- M.Drozdowski, N.V.Shakhlevich, Scheduling divisible loads with time and cost constraints, Journal of Scheduling, 24(5), 2021, 507-521, https://doi.org/10.1007/s10951-019-00626-6, summary of complexity results, open access
- T.Robertazzi, Divisible Load Scheduling, https://www.ece.stonybrook.edu/~tom/dlt.html#THEORY a list of DLT publications up to approx. 2015
- J.Marszalkowski, M.Drozdowski, G.Singh, Time-energy trade-offs in processing divisible loads on heterogeneous hierarchical memory systems, Journal of Parallel and Distributed Computing, 144, 2020, 206-219, https://doi.org/10.1016/j.jpdc.2020.05.015 , a MIP and other algorithms for energy use model with hierarchical memory, open access.
- M.Drozdowski, J.M.Marszalkowski, J.Marszalkowski, Energy trade-offs analysis using equal-energy maps, Future Generation Computer Systems, 36, 2014, 311-321, http://dx.doi.org/10.1016/j.future.2013.07.004 , iso-energy maps for a simpler energy use model.
- M.Drozdowski, Scheduling for Parallel Processing, Springer, 2009. https://doi.org/10.1007/978-1-84882-310-5 , a book on scheduling in general for parallel systems