Machine Models and invariance

Peter van Emde Boas

ILLC, depts. of Mathematics and Computer Science (FWI)
University of Amsterdam

Oct 06 1992

Possible subtitle:

How to survive machine independently in a machine dependent world.

Topics: **Invariance Thesis**
Is the Invariance Thesis true
Machine classes
Two curious 2nd machine-class models

Why is Complexity Theory What ?

-- Complexity measures based on models of computation
-- Results are machine dependent
-- Algorithm Analysis requires specification of model
-- Not needed for Structural theory ; **$P \neq NP$?** etc.

What makes it possible that structural theory doesn't care about models whereas algorithm design does ?

Computation theory: There are many models but only one notion of effective computability. All models can be shown to be equivalent - effective simulations.

THESIS OF CHURCH

Complexity Theory: The simulations above are reasonably efficient.

INVARIANCE THESIS: *Reasonable machine models simulate each other with polynomially bounded overhead in time and constant factor overhead in space .*

An Example of model dependedness in algorithm design

1975: VEB, Kaas & Zijlstra - O(loglog(u)) priority deque for Universe of size u. Initialization time O(u.loglog(u)), Space required O(u.loglog(u)) words.

1977: VEB - improvement: Space O(u) , initialization time O(u)

1978: D.E. Johnson: O(1) Initialization time.

Models used:

VEB : Standard RAM with + and - arithmetic, unit time .
Johnson: Standard RAM with +,-,* and / arithmetic, unit time

Is this really an inprovement ?

IS THIS REALY AN IMPROVEMENT ?

Trivial observation: In the RAM with multiplication and division the original initialization algorithms are unneeded. They are there for supporting the multiplicative address calculations.

Further objections:

-- RAM with multiplication and division in unit time has power of full parallellism (Bertoni ea. STOC 1981)

-- There is no little innocent addition of multiplicative power to the RAM model which doesn't yield about all relevant bit-manipulation routines in constant time. (Schmidt and Siegel 1988)

PURPOSE OF INVARIANCE THESIS

-- **The fundamental complexity hierarchy:**

LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE =
= NPSPACE $\subseteq$ EXPTIME .....

becomes machine independent.

QUESTION: *Is the invariance thesis true ??!*

-- At best true for sequential models only. Compare Parallel Computation thesis :

//PTIME = //NPTIME = PSPACE

so if invariance holds for parallel models => **P = PSPACE**

WHAT ABOUT THE STANDARD SEQUENTIAL MODELS ??

Some problematic cases :

Turing Machine models:

TIME : no problems, see textbooks

SPACE : no problems, but for a case which lacks in the literature:

How to simulate a two dimensional tape on a single dimensional tape with constant factor overhead in space ?

Rectangle method: overhead $O(S^2)$

Address- value records: overhead O(S.log(S))

History method (Hemmerling 1979): overhead O(S)

A result which was presumably known in the late 1960's but which never seems to have made it into the literature.

RAM/RASP models:

Time : No problems for logarithmic time measure

Unit Time is OK, provided Arithmetic restricted to + and -, See textbooks.

Space : OK, **provided correct space definition is used.**

SPACE MEASURES FOR RAMS

General definitions of the form:

$$\sum_{i=0}^{maxaddr} size(i,maxval(i))$$

maxaddr:	largest index used register
maxval(i) :	largest value stored at REG[i]

which size function should be used ??

size(i,x) = 0 for unused REG[i]

$size_s(i,x) = \log(x)$ *standard measure*

$size_b(i,x) = \log(x) + \log(i)$ *correct measure*

-- Space invariance is correct for TM vs. RAM if $size_b$ is used

-- If size $_s$ is used space consumption in underestimated:

$L_0 := \{ w_1\$ \ldots\ \$w_k\$w_0 \mid |w_i| = n \ \&\ w_0 \in \{w_1,\ldots,w_k\} \}$

recognized *on line* in space O(n+k) on RAM with space measure based on $size_s$, but requires space W(k*n) on TM

What happens for off-line processing ?

A MIRACLE

For off-line deterministic processing the set of counterexamples is empty:

Every off-line recognition problem which can be solved on a RAM in space S(n) if measured using $size_S$ can also be recognized in space O(S(n)) on TM .

Problem open for probabilistic and nondeterministic case.

-- Technique based on result on perfect hashing:

THEOREM: [Slot & VEB, 1984] *For every* k-*element set* A *in a universe* U *of size* u , *there exists a perfect hash function* f : U => {0,....,6k} , *such that the program size and evaluation space of* f *are both* O(loglog(u) + k) .

Use U = address range of RAM computation

A = set of addresses actually used

f = Logical-to-physical address translation

-- How to find f for unknown A and unknown parameter values k and n ??

-- Try out all possible f for given k and n

-- Use incremental storage (Savitch)

-- Use backward simulations (Sipser)

THE STORAGE MODIFICATION MACHINE

RAM like device (described by Schönhage) which operates on directed edge coloured bounded out-degree finite graph by creating nodes and (re)directing edges.

-- Very flexible model
-- Based on pointer assignments (paradoxes of assignment)
-- Supports multiplication in linear time

Does this model satisfy the invariance thesis ??

Time: no problem (real time equivalent with sucessor RAM in unit time)
Space: ???

-- Schönhage does not propose a space definition
-- if space = # nodes used in storage space is unlikely to be equivalent up to a constant factor with TM space
-- Argument : the number of n-node out-degeree k space configurations is bounded by :

$$n^{nk-n+1} \leq X(n,k) \leq (kn+1)^{n-1} n^{nk-n+1}$$

For on-line processing counterexample against invariance can be found .

TOTAL DISASTER
For the SMM Space Invariance
Fails.

THEOREM:

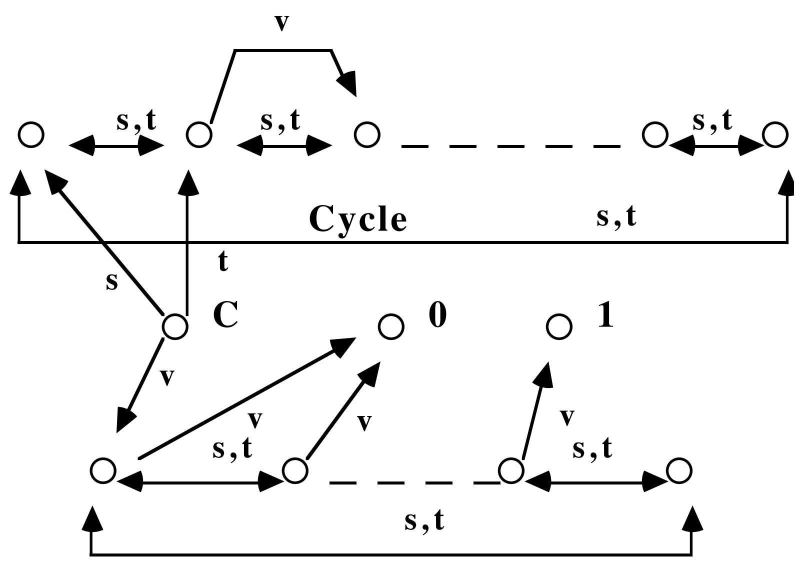
What a Turing Machine can do in Space **S.log(S)** ***can be done on an SMM in space*** **O(S)** **.**

REASON:

The theoretical storage capacity as expressed by the bounds above can be easily used in an effective and efficient way.

PROOF METHOD:

Create a cycle of S nodes ;
have in every node a pointer pointing into this cycle
this system of pointers encodes an array [0..S-1] **of** {0..S-1},
which is equivalent to O(S.log(S)) bits.



Binary counter

By simple cycle traversal the value of a single pointer can be read into a special purpose counter of O(log(S)) bits, which can simulate a Turing Tape segment of this length; after use the value of this counter is stored back into this pointer, and the adjacent segment is fetched.

PROOF METHOD:

This argument does not depend on mode of computation

-- On Line / Off Line

-- Deterministic, Nondeterministic, Probabilistic, Alternating, ...

Similar simulation with improved time simulation overhead (real-time) given by Loui and Luginbuhl (1988/1993).

Based on tree rather than cycle

-- Leaf in tree encodes log(S) bits

-- Path in tree towards leaf encodes contents block log(S) bits and can be stored/read out in time O(log(S))

HOW ABOUT THE KOLMOGOROV USPENSKII MACHINE ?

Same Method doesn't work: the indegree of nodes for the KU-machine is bounded.
KU-Machine can store permutations

O(log(S)) permutations on S/log(S) nodes each provide O(S.log(S)) bits of storage capacity ; each permutation can be encoded and decoded in a counter of O(S) nodes, which can be used for simulating a segment of Turing Machine Tape ....

Requires some method for ranking permutations with P-time bounded encoding & decoding algorithms. Excercise !

WHAT SHOULD WE DO ABOUT IT ?

1) Modify the definition of space on SMM and KUM :

An S-node graph consumes space S.log(S)

Unnatural !

2) Accept that SMM and KUM don't obey Space Invariance

The same logarithmic gain in time efficiency (E.G., the O(n) time multiplication of n-bit numbers) also occurs in Space .
There is more to space invariance than we have believed (or have bothered about).
Halpern ea. (Time vs. Space III) have achieved the same gain as Hopcroft & Paul .

THE FIRST MACHINE CLASS

The *First Machine Class* consists of those models which satisfy the invariance thesis with respect to the Turing machine model: they simulate Turing machines with polynomial time and constant factor space overheads and vice versa.

-- **Orthodox reading** : the simulation should be simultaneous time- and space efficient.
-- **Liberal reading** : two (possibly) different simulations exist, one of which is time efficient and the other is space efficient.

USE ORTHODOX READING

THE SECOND MACHINE CLASS

The *Second machine class* consists of those machine models which satisfy the parallel computation thesis :

//PTIME = //NPTIME = PSPACE

The list of first class machine models:

- -- All Turing machines
- -- RAM & RASP with log-time and $size_b$-based space
- -- RAM & RASP with unit-time and $size_b$-based space, provided +, - arithmetic only.

unlikely to be first machine class members

- -- RAM & RASP with $size_s$-based space
- -- Parallel machine models or other models embodying the power of parallellism (including all second machine class members).

Some models don't fit in this classification

Intermediate models:

-- The Storage Modification Machine

-- Wiedermann's intermediate class (**PTM** and equivalents)

-- Time equivalent with TM up to polynomial overhead , but space-overhead is not known to be constant

Even more powerfull parallel models :

-- **//NPTIME = EXPTIME** or worse

-- Time no longer usable as complexity measure (Parberry)

A SURVEY OF PARALLEL MODELS

Intermediate models

PTM	Wiedermann	83/92

Second machine class members
sequential processing of huge objects in unit time

Vector machine	Pratt & Stockmeyer	74,76
MRAM	Hartmanis & Simon	74,76
MRAM without bit-logic	Bertoni, Mauri & Sabadini	81
EDITRAM	Stegwee, Torenvliet & VEB	85
ASMM	Tromp, van Emde Boas	90,93

Alternating mode of computation

Alternating TM (RAM) Chandra, Stockmeyer & Kozen 76,81

True parallellism

Parallel TM	Savitch	77
PRAM	Savitch & Stimson	76,79
SIMDAG	Goldschlager	78,82
Aggregate	Goldschlager	78,82
Array Processing M.	Wiedermann & van Leeuwen	85

//PTIME = PSPACE , //NPTIME = EXPTIME

MIMD-RAM	Fortune & Wyllie	78
Paralel MRAM	Savitch	82

EXPTIME $\subseteq$ //PTIME

SIMDAG + *,/	N. Blum	83

Everything in constant time

SIMDAG + *,/ + nonuniformity (Parberry) 86

Understanding the power of //

Key idea: Understanding PSPACE

M(x) accepts in space S

<==>

there exists a path in the configuarion graph G(x,M,S) from unique start node to (unique) accepting node

G(x,M,S)

nodes: S-space bounded configurations of M on input x

edges: Legal one-step transitions

determinism: out degree ≤ 1

size bound : $| G(x,M,S) | \leq 2^{(c.S + \log(x))}$

Generic reduction from acceptance to Transitive closure

basic algorithms:

-- Iteration of squaring transition matrix

-- Recursive path splitting (Savitch 1970)

-- Translation of recursive path splitting into logic
(**QBF** is **PSPACE** complete - Meyer & Stockmeyer 74)

Making Model X a Second class member

PSPACE $\subseteq$ **X-PTIME :** Chose one out of three options

-- solve a **PSPACE** complete problem like **QBF** in polynomial time on X

-- implement recursoive path splitting transitive closure algorithm in polynomial time on X (parallel recursion)

-- implement iterated matrix squaring algorithm of M by M adjacency matrix in time O(Log(M)) on X
(PRAM & SIMDAG)

X-PTIME $\subseteq$ **PSPACE :**

-- invent recursive procedure *value(time,device,location)* which yields the value stored at location *location* of device *device* at time *time* during computation.

Additional benefit:

-- If both X and X' are shown to be second machine class members then X and X' simulate each other with polynomially bounded time overhead.

-- Will illustrate this on **EDITRAM** and **ASMM** model

The matrix squaring algorithm

How to decide that M(x) accepts in space S ?

1. Compute relevant sizes of configuration graph G(x,M,S)
   $N := |G(x,M,S)| = 2^{c.S}$.
2. Determine adjacency matrix $A(x,M,S) + I_N$
3. Perform the required number of squarings

Timing analysis on SIMDAG

1. requires time O(S) since SIMDAG needs repeated additions in order to produce large values.
2. Can be done in time O(log(N)) = O(S) on N^2 processors
3. Each squaring can be done in time O(1) on N^3 processors

SAVING TIME on more powerful models:

1. time O(log(S)) if multiplications are used for building large values ; O(1) in non-uniform models.

2. time O(log(S)) if arbitrary shifts and bitwise logic ; time O(1) if also unbounded fan-in and consistent concurrent writes

3. O(log(T)) squarings suffice, where T is run-time of M(x)

Parberry: integrates steps **2. & 3.** into a combined matching of entire path descriptions stored in processor signature.
Yields O(1) time solution provided non-uniformity and unbounded fan-in are allowed.

CONCLUSIONS

-- Fundamental complexity hierarchy

**LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE =
= NPSPACE $\subseteq$ EXPTIME .....**

can be given a machine-independent base using *Invariance Thesis* , but only with respect to a restricted class of models (*first machine class*).

-- Parallellism can be included using *Parallel Computation Thesis* but again only for a subclass of all parallel models (*second machine class*).

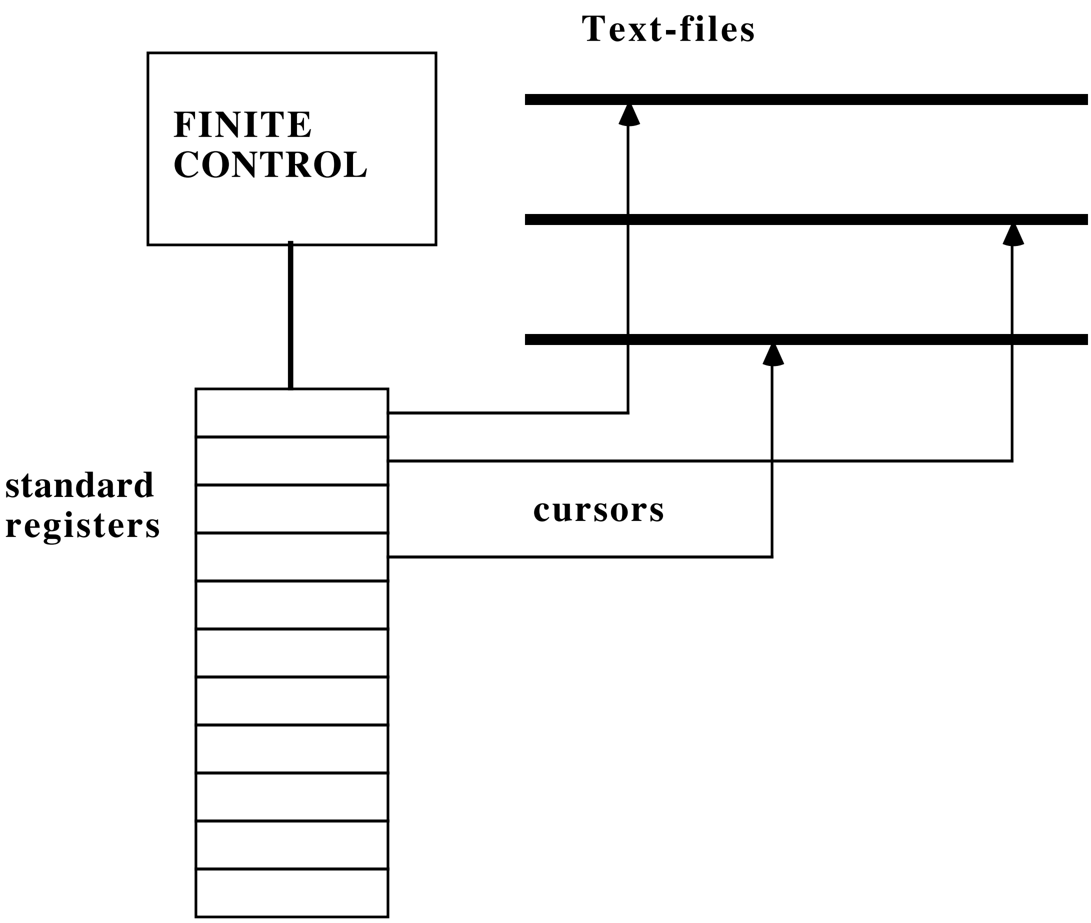
-- Attacks on these theses on the basis of behavior of particular models are invalid, if we don't claim that these classes include all models.

-- Investigation of models outside the classes improves our understanding of these machine classes.

-- Taking Space invariance for granted is an unjustified and unsound way of founding Theoretical Computer Science

Your favourite arithmetic-free second machine class Text Editor

Idea: **Take standard RAM model**
Add fixed finite number of text-file registers
Use standard registers as cursors in text-files



NEW INSTRUCTIONS

- -- **Arithmetic + and - on cursors**
- -- **Move cursor to stored position**
- -- **Move cursor to end of file**
- -- **read/write symbol from/to text-file at cursor**
- -- **copy segement of text-file to text-file (delimited by cursors)**
- -- **replacement of literal strings:**

  c / string1 / string2 / <u>in</u> file_k

  c / string1 / string2 % for file0

  (where string1 **and** string2 **are literals in program text)**

Motivation: the program:

file1 := 'aa'

<u>do</u> c / 'a' / file1 <u>in</u> file1 <u>od</u>

leads to doubly exponential growth of length file1

The EDITRAM is a realistic model of your real-life text editor on your computer

Time complexity: logarithmic time measure with respect to standard registers and cursors

This theorem from Stegwee, Torenvliet and VEB from 1985 shows that your editor has the full power of paralellism.

EDITRAM-NPTIME $\subseteq$ PSPACE

- -- **guess accepting computation**
- -- **write down computation trace, including full record of all standard register values**
- -- **Design recursive procedure Find(time, position, text-file)**

The main difficulty is cuased by the change-instruction; for the remainder the proof is standard.

PSPACE $\subseteq$ EDITRAM-PTIME

Proof by exhibition of P-time algorithm for QBF on the EDITRAM

Input: $Q_1 x_1\ Q_2 x_2 .... Q_k x_k\ [\ F(x_1, x_2, \ldots, x_k)\ ]$

STEP 1: Transform from inside out by transforming:

$\forall x_i\ [\ F(\ldots\ x_i\ \ldots)\ ] \Rightarrow [\ F(\ldots\ 0\ \ldots)\ ] \wedge [\ F(\ldots\ 1\ \ldots)\ ]$

$\exists x_i\ [\ F(\ldots\ x_i\ \ldots)\ ] \Rightarrow [\ F(\ldots\ 0\ \ldots)\ ] \vee [\ F(\ldots\ 1\ \ldots)\ ]$

STEP 2: Evaluate resulting quantifier-free expression using change-operations

STEP 3: Read the answer

Estimates: After STEP 1 the size of the formula is increased by a factor 2^k ; its depth has been increased by k

The program:

c / ' ¬0' / '1' / ; c / ' ¬1' / '0' / ;

c / '0 ∧ 0' / '0' ; c / '0 ∧ 1' / '0' ; c / '1 ∧ 0' / '0' ; c / '1 ∧ 1' / '1' ;

c / '0 ∨ 0' / '0' ; c / '0 ∨ 1' / '1' ; c / '1 ∨ 0' / '1' ; c / '1 ∨ 1' / '1' ;

c / '(0)' / '0' ; c / '(1)' / '1' ;

reduces the depth by 1 ; after depth(F) + k iterations a truth value remains.

How to program the substitutions in STEP 1?

Problem: The identity of the variable x_i is not fixed !

It is easy to locate the innermost quantifier $Q_i x_i$;

copy the variable in a special text-file in order to read its code ; variables are encoded by

x <0110....011>

The program below will replace each occurrence of x_i by the code $x < \$ >$ **leaving all other variables invariant.**

```
c / 'x<' / 'x<#' /  ;                        %  insert marker
for j to L do                                % advance if correct
    if  binrep( i )[ j ] = 0
    then  c / '#0' / '0@' ;  c / '#1'/ '1' ; c / '#>'/ '>' ;  c / '@' / '#'
    else  c / '#1' / '1@' ;  c / '#0'/ '0' ; c / '#>'/ '>' ;  c / '@' / '#'
    fi
od;
c / '#>' / '@>';  c / '#0' / '0';  c / '#1' / '1';      % prepare cleanup
to L do                       % cleanup
    c / '0@' / '#0'  ;  c / '1@' / '#1' ; c / '#' / '@'
od;                                                   %  ready
```