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.

-- Realism of models remains a problem.

ASMM in 2nd Machine Class

Simulation NP-ASMM in PSPACE:

Standard proof : write down computation trace and validate this trace using recursive procedure

Solving QBF in PTIME on ASMM:

-- rather contrieved simulation

-- standard tricks don't seem to work (no pointer jumping etc.)

-- unintuitive representation of truth values (code dependent of context in QBF expression...)

CONCLUSION:

ASMM provides for the right combination of Uniformity and Exponential growth potential.

How ASMM manipulates sets

For w in Δ^* and $\underline{\delta}$ in $\underline{\Delta}$ observe that $p^*(w\underline{\delta})$ denotes the set of nodes having their δ-edge directed to $p^*(w)$

Let sets A, B and C be represented by $p^*(x\underline{\alpha})$, $p^*(y\underline{\beta})$ and $p^*(z\underline{\gamma})$

Set $z\underline{\gamma}$ to ε **# now $p^*(z\underline{\gamma}) = \varnothing$, so C := $\varnothing$**

Set $x\underline{\alpha}\gamma$ to z **# now $p^*(z\underline{\gamma}) = p^*(x\underline{\alpha})$, so C := A**

Set $y\underline{\beta}\gamma$ to z **# similar effect; now C:= A $\cup$ B**

Another example:

Set $z\underline{\gamma}$ to ε **# now $p^*(z\underline{\gamma}) = \varnothing$, so C := $\varnothing$**

Set $x\underline{\alpha}\gamma$ to z **# now $p^*(z\underline{\gamma}) = p^*(x\underline{\alpha})$, so C := A**

Set $y\underline{\beta}\gamma$ to ε **# C looses nodes in B; C:= A \ B**

Set $z\underline{\gamma}\alpha$ to ε **# A := A \ C = A \ (A \ B) = A $\cap$ B**

Exponential growth:

performing (New α ; Set $\alpha\alpha$ to ε) k times creates a set consisting of 2^k nodes.

IMPACT OF REVERSE EDGES

Since w is a path over $(\Delta \cup \underline{\Delta})^*$ p*(w) denotes a set of nodes in the graph.

<u>New</u> w:

-- if $w = \varepsilon$ unchanged; new Center created

p*(ε) always is a singleton set.

-- <u>New</u> wδ: each node u in q*(wδ) = p*(w) spawns a copy u' ; p(u,δ) := u' ; p(u',α) := p(u,δ) for all α

-- <u>New</u> w$\underline{\delta}$: each node u in q*(w$\underline{\delta}$) = p*(wδ) spawns a copy u' ; p(u,δ) := u' ; p(u',α) := p(u,δ) for all α

<u>Set</u> w <u>to</u> v:

-- If $w = \varepsilon$ as before: the Center migrates to p*(v)

-- $w = w'\delta$ or $w = w'\underline{\delta}$: each node u in q*(w) (equaling p*(w) or p*(w$\underline{\delta}$) respectively) has its d-edge redirected towards p*(v)

MEANING OF INSTRUCTIONS

<u>New</u> w:

-- Creates a copy of p*(w) to be inserted in the graph at the end of the last edge on the path w

-- If w = ε the new node becomes the new Center

-- all outgoing edges from the new node are directed to former p*(w)

<u>Set</u> w <u>to</u> v:

-- Redirects last edge on path w to (former) p*(v)

-- If w = ε the Center migrates to p*(v)

<u>If</u> v = w <u>then</u> λ ; <u>If</u> v ≠ w <u>then</u> λ:

-- Conditional tests on equality of p*(v) and p*(w)

BEWARE OF PARADOXES OF ASSIGNMENT:

-- After <u>new</u>(w) the new node can be unequal p*(w)

-- After <u>Set</u> w <u>to</u> v the equality p*(v) = p*(w) may be false

INSTRUCTIONS

Standard instructions for sequential devices:

- **Input** λ_0 , λ_1

  Output β

  Goto λ

  Halt, Accept, Reject

Storage modification instructions

- **New** w

  Set w **to** v

  If v = w **then** λ ; **If** v $\neq$ w **then** λ

For SMM all path-addreses are literals over Δ^*.

For the ASMM the arguments w in New w and

Set w to v may be paths over $(\Delta \cup \underline{\Delta})^*$

Complexity measures:

Time: Uniform : number of instructions

Space: Mass : number of nodes

Capacity: Mass * log(Mass)

SMM and ASMM

-- **Sequential devices, operating on edge-labeled directed graphs <V,E> (Δ-structures)**

-- **Each node has |Δ| outgoing edges labeled by symbols in Δ: $p(a,\delta_1), \ldots, p(a,\delta_k)$**

-- **One node designated to be the Center: c**

-- **Map p*: Δ* -> V , defined by:**

$p^*(\varepsilon) := c$; $p^*(w\delta) := p(p^*(w),\delta)$

-- **ASMM extends p* by allowing reverse edges $\underline{\delta}$ thus yielding a relation p*: $(\Delta \cup \underline{\Delta})^*$ -> V:**

$p^*(\varepsilon) := \{c\}$;

$p^*(w\delta) := \{p(x,\delta) \mid x \in p^*(w)\}$

$p^*(w\underline{\delta}) := \{x \mid p(x,\delta) \in p^*(w)\}$

Starting points of last edges along paths: map q*:

$q^*(\varepsilon) := \varnothing$;

$q^*(w\delta) := p^*(w)$

$q^*(w\underline{\delta}) := p^*(w\underline{\delta})$

WHAT IS THE RIGHT PARALLEL VERSION OF THE SMM?

-- **Parallel versions of SMM / KUM proposed in the early 60-ies (Barzdin 1965) : HMM**
-- **Independently activated nodes manipulating the graph: think of mechanized graph grammars....**

-- **Lack of Uniformity . Hence:**
-- **HMM-NP-time = NEXPTIME (Dymond & Cook / Lam & Ruzzo)**

The ASMM models, by being more Uniform becomes a true 2nd machine class model.

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
```

PSPACE $\subseteq$ EDITRAM-PTIME

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

Input: $Q_1x_1\ Q_2x_2 \ldots\ Q_kx_k\ [\ F(x_1, x_2, \ldots, x_k)\ ]$

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.

EDITRAM-PTIME = EDITRAM-NPTIME = PSPACE

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.

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 / in file_k

c / string1 / string2 % for file0

(where string1 and string2 are literals in program text)

Motivation: the program:

```
file1 := 'aa'
do c / 'a' / file1 in file1 od
```

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

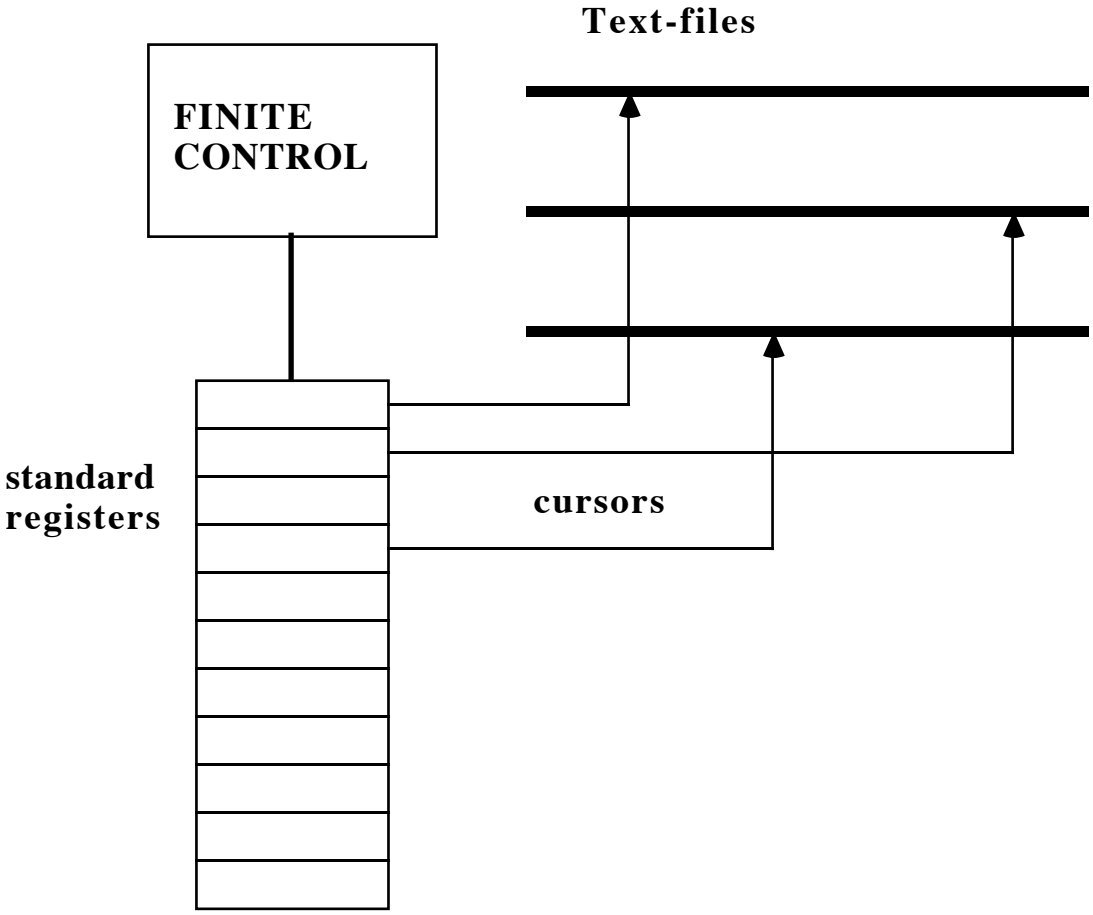
THE EDITRAM

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



Making Model X a Second class member

PSPACE ⊆ X-PTIME : Chose one out of three options

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

-- implement recursive 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 ⊆ 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

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)

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

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)

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

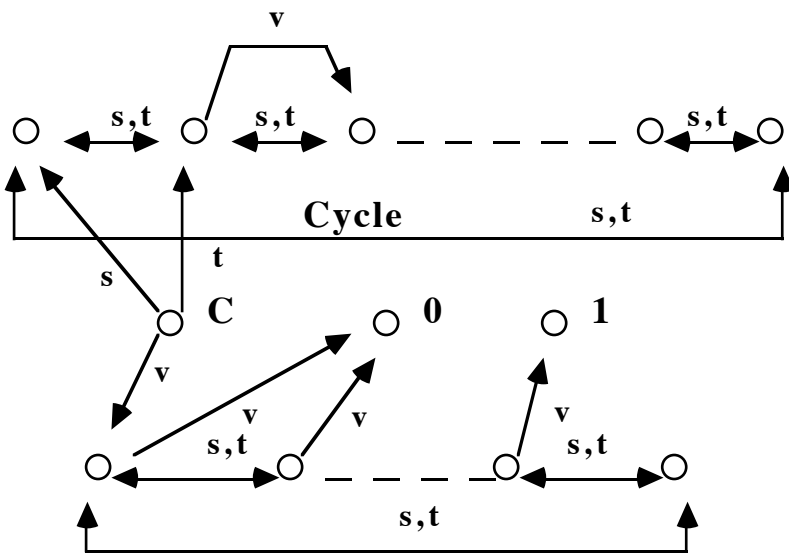
THE SECOND MACHINE CLASS

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

//PTIME = //NPTIME = PSPACE

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

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).

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: ???

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)** **.**

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.**

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 ??

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 ≠ 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 .*

Two sequential models with parallel power

Peter van Emde Boas

ILLC, Logic and Computation Theory Group (FWINS)
University of Amsterdam

May 21 1996

co-authors:
R. Stegwee
L. Torenvliet
J. Tromp

Possible subtitle:

Sequential models are not always what they seem to be

Topics: **Invariance Thesis**
Parallel Computation Thesis
The two Machine Classes
Two sequential 2nd Machine-class Models