Strict Sequential P-completeness [⋆]

Klaus Reinhardt

Wilhelm-Schickhard Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
e-mail: reinhard@informatik.uni-tuebingen.de

Abstract. In this paper we present a new notion of what it means for a problem in P to be inherently sequential. Informally, a problem L is *strictly sequential P-complete* if when the best known *sequential* algorithm for L has polynomial speedup by parallelization, this implies that *all* problems in P have a polynomial speedup in the parallel setting. The motivation for defining this class of problems is to try and capture the problems in P that are truly inherently sequential. Our work extends the results of Condon who exhibited problems such that if a polynomial speedup of their best known *parallel* algorithms could be achieved, then all problems in P would have polynomial speedup. We demonstrate one such natural problem, namely the *Multiplex-select Circuit Problem* (MCP). MCP has one of the highest degrees of sequentiality of any problem yet defined. On the way to proving MCP is strictly sequential P-complete, we define an interesting model, the *register stack machine*, that appears to be of independent interest for exploring pure sequentiality.

1 Introduction

An important question in parallel complexity theory is whether problems in P have a speedup by parallelization. If we demand the speedup to be 'exponential', this means a speedup to polylogarithmic time, and allowing a polynomial number of processors on a PRAM, we get the famous open problem 'P=NC ?'. This problem can be instantiated to any P-complete problem as for example the circuit value problem or the context-free emptiness problem which means that P = NC iff one of those P-complete problems is in NC (see [Coo85,GHR95]).

We believe an even more realistic and interesting open question is whether problems in P have polynomial speedup by parallelization. This means for every sequential algorithm solving a problem in time $t(n)$ there is a parallel algorithm solving the problem in time $T(n) \in O((t(n))^{1-\varepsilon})$. For practical applications it is more important that the parallelization has a low inefficiency. The *inefficiency* is the quotient of the time processor product and the sequential time. Kruskal *et al.* defined the class EP (AP, SP, respectively) as the class of problems with polynomial speedup by parallelization and constant (polylogarithmic, polynomial, respectively) inefficiency [KRS90].

[⋆] This research has been supported by the DFG Project La 618/3-1 KOMET.

Unfortunately, the question P$\subseteq$EP (AP, SP, respectively) cannot be instantiated to an arbitrary P-complete problem since a lot of them have polynomial speedup by parallelization (see [VS86]).

As a step towards addressing the issues above, Condon defined a problem as *strictly* $T(n)$ *complete* for P, if it has a parallel algorithm with running time $t(n)(\log(t(n)))^{O(1)}$ and if the existence of a parallel algorithm improving this by a polynomial factor, meaning to $O((T(n))^{1-\varepsilon})$, would imply that all problems in P have polynomial speedup [Con92]. She showed that the Square Circuit Value Problem SCVP is strictly $\sqrt{n}$ P-complete.

This was the first result showing that P$\subseteq$SP if a specific problem has a polynomial speedup relative to a known *parallel* running time, i.e., if SCVP has a polynomial speedup relative to $\sqrt{n}$. Since SCVP already has a polynomial speedup of $\sqrt{n}$ relative to its linear sequential time, we do not get the other direction. This means it may still be that P$\subseteq$SP but SCVP has no polynomial speedup relative to $\sqrt{n}$.

What we need is a P complete candidate for not having polynomial speedup at all, that means in P$\setminus$SP if P$\nsubseteq$SP. Thus in this paper we go one step further and look for a problem such that P$\subset$SP iff this specific problem has a polynomial speedup relative to a known *sequential* running time. This means a *strictly sequential P-complete* problem must be strictly $t(n)$ P-complete, where $t(n)$ is now the sequential time. Accordingly, we define a problem with a sequential running time $t(n)$ as *strictly sequential efficient P-complete* (*strictly sequential almost efficient P-complete*, respectively) if the existence of a parallel algorithm improving this by a polynomial factor, meaning to $O((t(n))^{1-\varepsilon})$, with a constant (polylogarithmic, respectively) inefficiency would imply that every problem in P, which has an efficient prediction of termination, is in EP (AP, respectively).

In this paper we use the multiplex select gate in combination with a compressed representation of the wires connecting the gates. We show that the problem MCP to calculate the output of a circuit consisting of such gates which is given by a representing input, is strictly sequential P-complete. Furthermore we are able to prove that MCP is strictly sequential almost efficient P-complete, where we restrict our attention to those problems in P, which have an efficient prediction of termination.

As the sequential model we use the RAM which must read its complete input and needs at least linear time by definition. Additionally, we introduce the *register stack machine* as a new sequential model, which avoids the hidden parallelism in the storage of a RAM. Using this model for sequential algorithms, we can show that MCP is strictly sequential efficient P_{ept}-complete.

As the parallel model we use the PRAM but the same results can be obtained if we use more restricted models such as grids of processors or linear arrays of processors, as long as this model is strong enough to calculate the reduction function $f(x)$ in time $|f(x)|^{1-\varepsilon}$ and an accordingly low inefficiency.

Thus far we have only been able to identify one natural strictly sequential P-complete problem. An interesting open question is to find more problems of this type.

2 Register stack machine

If we consider the simulation of a RAM by a circuit, in some sense the storage of a RAM contains a kind of parallelism on the circuit level: every cell must detect at every step, whether it is the one, where the processor it just writing on. To avoid this, we need a new model, where we have a write once storage and where we are able to simulate normal RAM's with only a logarithmic loss in time. Such a model is interesting on its own right.

A *register stack machine* (RSM) has a constant number c of registers and, in addition, a (non-erasing) stack of registers, i.e., a push-down like managed list of registers, which the machine is only allowed to read. In each step it simultaneously

- pushes the contents of the register number 2 on the stack,
- loads the contents of the stack indexed by the contents of register number 1 to register number 0 (if register number 1 is not negative) and
- all the registers with number >0 can be loaded by a constant or by the sum or the difference between other registers.

The next instruction can depend on a ≤ 0-test of register 3.

Formally an RSM M is a 5-tuple (c, Q, q_0, δ, Q_f) with the transition function $\delta : Q \times \{1, 0\} \mapsto Q \times ((\mathbb{N} \cup \{+, -\} \times \{1, ...c\}^2)^{\{1,...c\}})$. The start configuration is $s_0, ..., s_n, r_0, ..., r_c, q_0$, where $s_0, ..., s_n$ contains the input, $r_4 = n$ and $r_0 = r_1 = r_2 = r_3 = r_5 = ... = r_c = 0$. The configuration transition is

$$s_0, ..., s_m, r_0, ..., r_c, q \vdash_{\overline{M}} s_0, ..., s_{m+1}, r'_0, ..., r'_c, q'$$

where the test $t := 0$ if $r_3 \leq 0$ and $t := 1$ if $r_3 > 0$ determines the transition by $(q', a) := \delta(q, t)$. We set $s_{m+1} := r_2$, $r'_0 := s_{r_1}$, and $r'_i := a(i)$ if $a(i) \in \mathbb{N}$ and $r'_i := r_j t' r_k$ if $a(i) = (t', j, k) \in \{+, -\} \times \{1, ...c\}^2$ for $i > 0$. The input $s_0, ..., s_n$ is accepted if a final state in Q_f is reached.

In the description of an algorithm for an RSM every line has the following shape:

```
$q$: if $r_3 \leq 0$ then $r_1$:=...,$r_2$:=... , ... $r_c$:=... goto $q'$
                  else $r_1$:=...,$r_2$:=... , ... $r_c$:=... goto $q''$
```

For convention we may add and subtract constants (which means add or subtract an additional register, which was loaded with this constant before), if a register r_i is not mentioned, it means $r_i := r_i$ (which means $r_i := r_i + 0$) and if a goto is missing in line q_i it means goto q_{i+1}. Also 'if $r_3 \leq 0$ then ... else' may be missing, if the same is done in both cases.

Proposition 1. *[Wie90] A RAM with running time $t(n)$ under logarithmic cost measure can be simulated in time $O(t(n))$ by a RAM using integers of length $O(\log(t(n)))$ and addresses of value in $O((t(n))^2)$.*

Lemma 2. *A RAM with running time $t(n)$ under logarithmic cost measure can be simulated by a RSM in time $O(t(n)\log(t(n)))$ using integers of length $O(\log(t(n)))$.*

Proof. The registers R_j which are accessed by the RAM directly (constantly many) are simulated by the RSM directly in registers $r_{j'}$. The indirect addressed storage of the RAM is simulated as a binary tree. The address of the root node is stored in r_5, r_4 contains the address of the top of the stack (it is incremented in every step). Each inner node is represented as 3 consecutive register contents on the stack: one number containing the smallest address in the subtree with the bigger addresses and two pointers to subtrees. The leave nodes are represented as 2 consecutive register contents on the stack: one marking (by -1 or -2), which tells that this is a leave node and whether this is one of the (constantly many) directly accessed cells (-2) and the contents of the register of the simulated RAM.

Because of Proposition 1, the tree has logarithmic depth and we do not have to do balancing.

In order to simulate an indirect reading of the RAM indexed by register R_j to register r_k, the RSM walks down the tree in the following way: first the pointer to the root node is loaded to r_1, then, as long as r_0 does not contain a leave marking, it sets $r_3 := r_{j'} - r_0$ which allows to set $r_1 := r_1 + 1$ or $r_1 := r_1 + 2$ in the two possible following states depending whether $r_3 \leq 0$ and continue with $r_1 := r_0$. If the leave node is found, it sets $r_1 := r_1 + 1$ (if R_{R_j} is not one of the directly accessed cells) and in the next step r_0 contains the contents of R_{R_j}.

```
$q_0$: $r_1 := r_5$, $r_4 := r_4 + 1$
$q_1$: $r_3 := r_{j'} - r_0$, $r_4 := r_4 + 1$
$q_2$: if $r_3 \leq 0$ then $r_1 := r_1 + 1$, $r_4 := r_4 + 1$
                 else $r_1 := r_1 + 2$, $r_4 := r_4 + 1$
$q_3$: $r_1 := r_0$, $r_4 := r_4 + 1$
$q_4$: $r_3 := r_0 + 1$, $r_4 := r_4 + 1$
$q_5$: if $r_3 \leq 0$ then $r_3 := r_3 + 1$, $r_4 := r_4 + 1$
                 else $r_3 := r_{j'} - r_0$, $r_4 := r_4 + 1$, goto $q_2$
$q_6$: if $r_3 \leq 0$ then $r_k := r_0$, $r_4 := r_4 + 1$
                 else $r_k := r_{k'}$, $r_4 := r_4 + 1$
```

This takes $O(\log(t(n)))$ many steps.

In order to simulate an indirect writing of register r_k of the RAM indexed by register R_j, the RSM first checks whether R_{R_j} is not one of the directly accessed cells. Then the complete path to the corresponding node in the binary tree is copied on top of the stack in reverse order by replacing the pointers on the path by the new ones which are the current position in r_4 plus a constant. This constant is the number of steps until the RSM is copying the child node. Then it writes the new leave node on top of the stack. Again this takes $O(\log(t(n)))$ many steps. ■

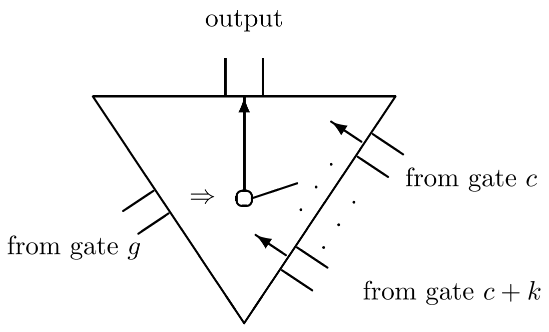
Remark 3. It is easy to see that an RSM with running time $t(n)$ can be simulated by a RSM in time $O(t(n))$.

In [LN93] K.-J. Lange and R. Niedermeier introduce the notion of write data-independence, which means that it is determined only by the length of the input, which memory cell is written at each time. This notion can be sharpened to *strict write data-independence*, which means that at any time for any memory cell the time, when this cell was written the last time, can be calculated in constantly many steps. It is easy to see that such a strict write data-independent RAM can be simulated by an RSM without a logarithmic loss of time. (Of course any RSM is strictly write data-independent.) In this manor for example Merge-sort can be performed on an RSM as fast as on an RAM.

3 The corresponding circuit

Analogously to the CVP we define MCP as the problem to calculate the output of a circuit, which has multiplex select gates like the ones which are used in [FLR96] to characterize OROW-PRAM's.

Such a multiplex select gate has two kinds of input signals: One bundle of $O(\log(n))$ steering signals and up to n bundles of $O(\log(n))$ data signals. The number which is binary encoded by the steering signals is the number of the bundle of data signals which is switched through to the output.



A multiplex select gate can be described by the (binary) encoding of $(\star, g, c, k)$ where g is the number of the gate whose output bundle is connected to the steering input bundle, k is the number of data input bundles and $c+j$ is the number of the gate whose output is connected to the j'th data input bundle for $j \leq k$. This means that the encoding can still have a logarithmic size although the number of input bundles can be linear.

Analogously we have gates for and $(\wedge)$, or $(\vee)$, addition $(+)$ and subtraction (-) which are encoded by a symbol and two numbers of gates (for example $(+, d_1, d_2)$), gates for the >0 test and negation $(\neg)$ which have one number of a gate in the encoding and gates having no input and a fixed output o, where the encoding is $(\$, o)$.

(All those gates could be simulated by polylogarithmically many multiplex select gates with independently connected steering signals and no bundles.)

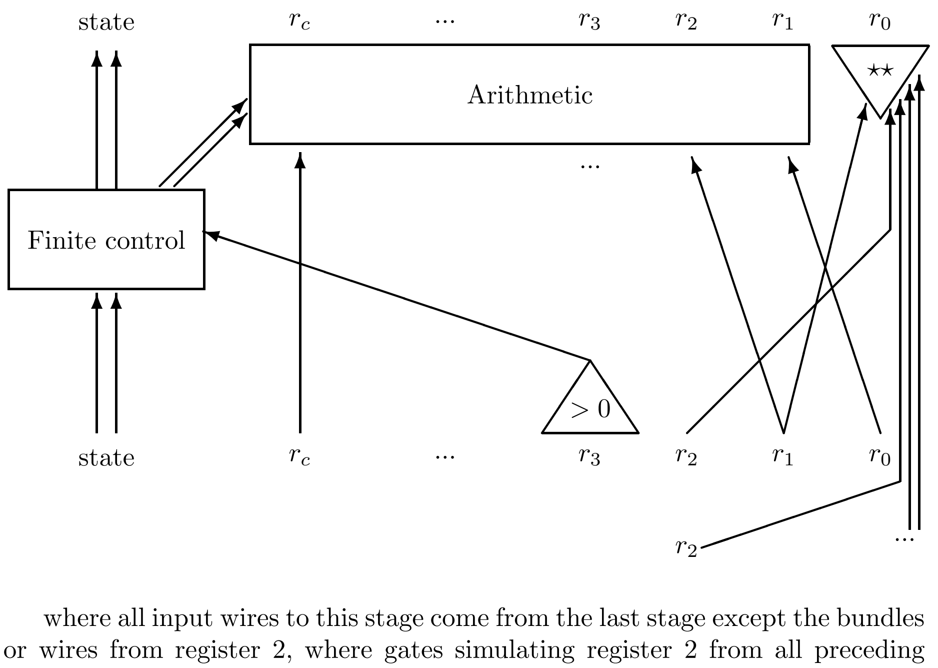
MCP is the language of inputs encoding a circuit with multiplex select gates, where each gate has only inputs from gates with smaller numbers and where the last gate has the value 1.

Remark 4. To simulate one multiplex select gate with (unbounded fan-in) $\wedge$, $\neg$ and $\vee$ gates would need a number of gates, which is linear in the size of the circuit.

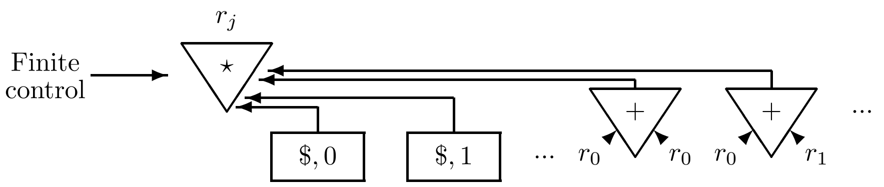
Lemma 5. *An RSM with running time $t(n)$ using only integers having the length $O(\log(t(n)))$ with a general log cost measure (that means that every step costs $O(\log(t(n)))$ time units) can be simulated by a uniform family of multiplex circuits of size $O(t(n) + n)$.*

Proof. Given an RSM and an input $s_0, ..., s_m$, we construct a circuit starting with m stages for the input having the form $(\$, 0)^{p-3}(\$, s_i)(\$, 0)(\$, 0)$ for $i < m$ and one stage having the form $(\$, 0)^{p-5}(\$, m)(\$, 0)(\$, s_m)(\$, 0)(\$, 0)$, where the period p of the circuit is the number of gates in one stage simulating one step of the RSM. Thus p is determined by the simulated RSM. We may assume that p is a power of 2. (Furthermore we may assume that the starting state is encoded by zeros.)

Since one step costs $O(\log(t(n)))$ time units, the circuit must simulate $\frac{t(n)}{\log(t(n))}$ steps of the RSM. This is done by $\frac{t(n)}{\log(t(n))}$ stages having the form
$...(+, p(i-1) - \log p - 1, p(i-1) - \log p - 1)(+, pi - \log p, pi - \log p)$
$(+, pi - \log p + 1, pi - \log p + 1)...(+, pi - 2, pi - 2)(\star, pi - 1, p - 2, p(i-1))$
for $i > m$ being shown by the picture



where all input wires to this stage come from the last stage except the bundles or wires from register 2, where gates simulating register 2 from all preceding stages are connected as data input to the multiplex select gate simulating register 0. The $\star\star$-gate in the picture is a shorting for a $\star$-gate and $\log p$ addition-gates, which multiply the value for the steering input by p. Hence $(p-2)+jp$ is the number of the gate whose output is switched through to the output of gate number pi if $j < i$ is the output of gate number $p(i-1) - \log p - 1$, which simulates register r_1 in the previous stage. This simulates indexed reading from the stack which contains the input $s_0, ..., s_m$ as well as all contentses of register 2. For each register with a number $j > 0$, the arithmetic looks like



which preserves every possible assignment to the register depending on the state. The encodings of the stages look almost the same, except gate numbers appearing in them, which are linear dependent from the number of the stage. Thus the only difficulty to calculate the encoding of the circuit is to calculate $t(n)$.

The last stage contains only some logic gates such that the output of the last gate is 1 iff a final state in Q_f is reached. ■

Lemma 6. *MCP$\in$DTIME$(O(n))$ with log cost measure.*

Proof. The input $s_0, ..., s_m$ on the stack of an RSM (or in the register of a RAM) contains the encoding of the circuit, where the specifying symbol and the numbers encoding one gate are contained in consecutive registers. An RSM can evaluate one gate in a constant k number of steps. (For example $k = 16$ should be sufficient.) Thus it can write the output of gate number j to the position $m + kj$ on the stack.

To evaluate a gate, the RSM first reads the specifying symbol to detect the kind of gate. To evaluate a multiplex select gate, the machine first reads the number g, which is in the next position in the input, multiplies it by k in $\log k$ steps and loads $m+kg$ to register r_1. Then r_0 contains the number l of the input bundle, which is switched through to the output. The next position in the input contains c, which has to be added to get the number $l + c$ of the gate whose output is connected to this input bundle. Also the RSM has to check $l \leq k$, which is in the following position on the input. Now $m + k(l + c)$ is loaded to register r_1 (again in $\log k$ steps). Then r_0 contains the output value of the gate, which is loaded to r_2 to store it on the stack, if k steps took place since the evaluation of the last gate. The evaluation of the other kinds of gates works in analogous way.

In this way an RSM can evaluate the circuit in linear time. The same holds for a RAM. ■

For proving the next theorem we need the following lemma which was proved by J. Hoover.

Lemma 7. *[Con94] Let l be an eventually non-decreasing function such that $l(n) \in \Omega(n)$ and $l(n) \in n^{O(1)}$. For all $\delta > 0$, there is a rational number σ and a natural number n_0 such that*

- *$l(n) \leq n^{\sigma}$ for $n \geq n_0$ and*
- *$n^{\sigma} \in O(l(n)n^{\delta})$.*

Theorem 8. *MCP is strictly sequential P-complete.*

Proof. According to Lemma 6, MCP$\in$DTIME($O(n)$) with log cost measure. We have to show that if MCP has polynomial speedup, this means if it can be solved by a PRAM in time $O(n^{1-2\delta})$ for a $\delta > 0$, then every problem in P has polynomial speedup.

Let L be a language in P which is recognized by a RAM with running time $t(n) \in \Omega(n)$. According to Lemma 2, it can be recognized by a RSM in time $l(n) \in O(t(n)\log(t(n)))$ using only integers of length $O(\log(t(n)))$. Applying Lemma 7 it can also be recognized by a RSM in time n^{σ}. Thus according to Lemma 5 it can be reduced to MCP by a function f, which generates on input x the encoding of the circuit, which simulates the RSM on input x. For $|x| = n$ we have $|f(x)| \in O(n^{\sigma})$. Obviously the reduction $f(x)$ can be calculated by a PRAM in polylogarithmic time since n^{σ} can be calculated fast. The same PRAM can afterwards test in $O(n^{\sigma(1-2\delta)})$ steps whether $f(x)$ $\in$MCP. Thus this PRAM can recognize L in time

$$O(n^{\sigma(1-2\delta)}) \subseteq O((l(n)n^{\delta})^{1-2\delta}) \subseteq O((l(n)^{1+\delta})^{1-2\delta})$$
$$\subseteq O((l(n)^{1-\delta-2\delta^2}) \subseteq O((t(n)^{1-\delta}).$$

This means that L $\in$SP. ∎

Corollary 9. *If MCP$\in$SP then P$\subseteq$SP.*

4 Using prediction of termination

The inefficiency of the polynomial speedup of a problem in P in Theorem 8 is caused by the inefficiency for the speedup of MCP and by n^{δ}, which we have to sacrifice in order to get a a time limit, which can be calculated fast.

For practical reasons we are interested in those problems L in P having an optimal $t(n)$ time bounded sequential algorithm (this means no other algorithm recognizes L in time $o(t(n))$ with *efficient prediction of termination* which means $t(n)$ can be calculated in sublinear time.

Theorem 10. *MCP is strictly sequential almost efficient P-complete.*

Proof. According to Lemma 6, MCP$\in$DTIME($O(n)$) with log cost measure. We have to show that if MCP has almost efficient polynomial speedup, this means if it can be solved by a PRAM in time $O(n^{1-\varepsilon})$ with $\widetilde{O}(n^{\varepsilon})$ processors, then every problem in P_{ept} has almost efficient polynomial speedup. Let $\widetilde{O}(t(n)) := t(n)(\log(t(n)))^{O(1)}$.

Let L be a language in P which is recognized by a RAM with running time $t(n) \in \Omega(n)$, which can be calculated in sublinear time. According to Lemma 2 it can be recognized by a RSM in time $\widetilde{O}(t(n))$ using only integers of length $O(\log(t(n)))$. Thus according to Lemma 5 it can be reduced to MCP by a function f, which generates on input x the encoding of the circuit, which simulates the RSM on input x. For $|x| = n$ we have $|f(x)| \in \widetilde{O}(t(n))$. The reduction $f(x)$ can be calculated by a PRAM in $O(|f(x)|^{1-\varepsilon})$ steps and $O(|f(x)|^{\varepsilon})$ processors. The same PRAM can afterwards test whether $f(x) \in$MCP in time $\widetilde{O}(|f(x)|^{1-\varepsilon})$ and $O(|f(x)|^{\varepsilon})$ processors. This means that $L \in$AP. ■

Corollary 11. *If MCP$\in$AP, then every problem in P, which has an efficient prediction of termination, is in AP.*

The inefficiency of the polynomial speedup of a problem in P in Theorem 8 is caused by the inefficiency for the speedup of MCP and by the $\log n$ factor for the simulation of a RAM by an RSM. Thus in analogous way we get the following result:

Theorem 12. *If we regard the RSM with a general log cost measure as the model to define sequential time complexity, then, MCP is strictly sequential efficient P-complete.*

Corollary 13. *If we regard the RSM with a general log cost measure as the model to define sequential time complexity and if MCP$\in$EP, then every problem in P, which has an efficient prediction of termination, is in EP.*

Remark 14. We can use the uniformity of the stages in the construction of the circuit to invent more compressed representations which can be padded afterwards. In this way we can create artificial variants of MCP which are also strictly sequential P-complete complete but which are in DTIME(n^{σ})\DTIME($n^{\sigma-\varepsilon}$) for $\sigma > 1$ and $\varepsilon > 0$.

Open Problems: Is CVP strictly sequential P-complete? Is CVP$\in$ SP? Are there other natural strictly sequential P-complete problems?

Acknowledgment: I thank Henning Fernau, Ray Greenlaw and Klaus-Jörn Lange for helpful remarks.

References

[Con92] A. Condon. A theory of strict P-completeness. In *Proc. of 9th STACS*, number 577 in LNCS, pages 33–44. Springer-Verlag, 1992.

[Con94] A. Condon. A theory of strict P-completeness. *Computational Complexity*, 4, 1994.

[Coo85] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[FLR96] H. Fernau, K.-J. Lange, and K. Reinhardt. *Advocating Ownership*. Manuscript, 1996.

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to parallel computation: P-completeness theory*. New York u.a., Oxford Univ. Pr., 1995.

[KRS90] C. P. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, 71:95–132, 1990.

[LN93] K.-J. Lange and R. Niedermeier. Data-independences of parallel random access machines. In R. K. Shyamasundar, editor, *Proc. of 13th FST&TCS*, number 761 in LNCS, pages 104–113, Bombay, India, December 1993. Springer-Verlag.

[VS86] J. S. Vitter and R. A. Simons. New classes for parallel complexity: A study of unification and other complete problems for P. *IEEE Transactions on Computers*, C-35(5):403–418, 1986.

[Wie90] Juraj Wiedermann. Normalizing and accelerating RAM computations and the problem of reasonable space measures. In M.S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (Warwick University, England, July 1990)*, LNCS 443, pages 125–138. EATCS, Springer-Verlag, 1990.