Fast and Optimal Simulations between CRCW PRAMs

TORBEN HAGERUP

Max-Planck-Institut für Informatik
W-6600 Saarbrücken, Germany

We describe new simulations between different variants of the CRCW PRAM. On a minimum PRAM, the minimum value is written in the event of a write conflict, whereas concurrent writing to the same cell is without effect on a tolerant PRAM. Compared to other variants of the CRCW PRAM, the minimum PRAM is very strong, and the tolerant PRAM is very weak. In fact, the two variants are near opposite ends of the spectrum of CRCW PRAM variants commonly considered. We show that one step of a (randomized) minimum PRAM with \( n \) processors can be simulated with high probability in \( \Omega((\log^* n)^3) \) time on a randomized tolerant PRAM with \( O(n/(\log^* n)^3) \) processors. The simulation is optimal, in the sense that the product of its slowdown and the number of simulating processors is within a constant factor of the number of simulated processors. It subsumes most previous work on randomized simulations between \( n \)-processor CRCW PRAMs with infinite memories.

1 Introduction

A CRCW PRAM is a synchronous parallel machine with processors numbered 1, 2, \ldots and with a global memory that supports simultaneous access to a single cell by arbitrary sets of processors. Whereas the meaning of simultaneous reading is clear, researchers have studied several variants of the CRCW PRAM, each distinguished by a different write conflict resolution rule. The following write conflict resolution rules and corresponding variants are relevant to the present paper: Fetch-and-Add (Gottlieb et al., 1983): The effect of concurrent writing is as if the processors involved access the memory cell in question sequentially in some order, each processor adding its value to the contents of the cell after first reading the previous contents; Minimum (Cole and Vishkin, 1986): The smallest value that some processor attempts to store in a given cell in a given step gets stored in the cell; Priority (Goldschlager, 1982): If two or more processors attempt to write to a given cell in a given step, then the lowest-numbered processor among them succeeds and writes its value; Arbitrary (Shiloach and Vishkin, 1982): If two or more processors attempt to write to a given cell in a given step, then one of them succeeds, but there is no rule assumed to govern the selection of the successful processor; Common (Shiloach and Vishkin, 1981): All processors writing to a given cell in a given step must be writing the same value, which then gets stored in the cell; Collision (Fich et al., 1988b): If two or more processors attempt to write to a given cell in a given step, then a special collision symbol gets stored in the cell; Collision+ (Chlebus et al., 1988): If the processors attempting to write to a given cell in a given step all attempt to write the same value, then that value gets stored in the cell; if at least two values differ, a collision symbol gets stored in the cell;

Supported in part by the Deutsche Forschungsgemeinschaft, SFB 124, TP B2, VLSI Entwurfsmethoden und Parallelität, and in part by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).
TOLERANT (Grolmusz and Ragde, 1987): If two or more processors attempt to write to a given cell in a given step, then the contents of that cell do not change;

ROBUST (Hagerup and Radzik, 1990): A cell subjected to concurrent writing afterwards contains a well-defined but unknown value.

A CRCW PRAM working according to the PRIORITY (ARBITRARY, etc.) rule will be called a PRIORITY (ARBITRARY, etc.). When no explicit processor bounds are mentioned in a statement about a particular simulation, we always intend the number of processors of the simulated machine to be denoted by $n$.

A machine $M_2$ simulates another machine $M_1$ with slowdown $T$ if at most $T$ steps of $M_2$ are needed to simulate a single step of $M_1$. As has been done before, we use "$A \leq B^*$", where $A$ and $B$ are names of variants, to indicate that an $n$-processor machine of type $B$ can simulate an $n$-processor machine of type $A$ with constant slowdown. The following hierarchy is easy to establish.

$$ROBUST \leq TOLERANT \leq COLLISION \leq COLLISION^+ \leq ARBITRARY \leq PRIORITY \leq Minimum.$$ Simulations of strong variants of the CRCW PRAM (i.e., variants near the right end of the above chain) on weaker variants have been the object of intensive study in later years (Kučera, 1982, 1983; Fich et al., 1988a, 1988b; Chlebus et al., 1988, 1989; Boppana, 1989; Ragde, 1989; Hagerup and Radzik, 1990; Matias and Vishkin, 1990a, 1990b; Gil, 1990). In the interest of brevity, we limit the discussion below to randomized simulations without an increase in the number of processors (i.e., the simulating machine is allowed no more processors than the simulated machine).

If we are concerned only with the slowdown of a simulation, the best previous results are: $O(\log n)$ for the simulation of any of the above variants on any other variant (or even on a bounded-degree processor network), $O(\log \log n)$ for the simulation of Minimum on Tolerant (Gil, 1990), and $O(\log \log n)$ for the simulation of PRIORITY on ROBUST (Hagerup and Radzik, 1990). Speed is not everything, however, and the simulations mentioned so far use $n$ processors and nonconstant time to simulate one step of an $n$-processor machine, although the sequential cost of this computation is only $O(n)$. A desirable goal, as in other areas of parallel computation, is clearly to attain optimality, which in the present case means to obtain optimal simulations, for which the product of slowdown and number of simulating processors is within a constant factor of the number of simulated processors. The only previous optimal simulations known to the author were given by Matias and Vishkin (1990a; revised and corrected: 1990b), who showed that MINIMUM and FETCH-AND-ADD can be simulated optimally on ARBITRARY with expected slowdown $O(\log n)$ (by a convention introduced above, $n$ denotes the number of processors of the simulated MINIMUM or FETCH-AND-ADD PRAM; since the simulation is optimal, the simulating ARBITRARY PRAM has $O(n/\log n)$ processors). This result assumes that a prime larger than the size of the memory of the simulated machine is given for free; following previous authors, we shall ignore the difficulty of obtaining such a prime. We are not aware of any lower bounds on the slowdown of randomized simulations between variants of the CRCW PRAM.

For some of the simulations discussed above, the bounds on the slowdown are bounds on the expected value of this parameter only. The probability that an actual run of the simulation exceeds these bounds, which we (over-dramatizing slightly) call the error probability, may therefore be nonnegligible. More satisfactory simulations have bounds on the error probability that depend on the number $n$ of simulated processors and converge rapidly towards zero as $n$ increases. Our simulations are of this kind.

A final problem with which the simulation of (Hagerup and Radzik, 1990) is afflicted is the so-called "memory blowup": A machine with $n$ processors and $m$ global memory cells is simulated by a machine with as many as $\Theta(nm)$ global memory cells. Ideally, we would like the simulation to use no more memory than the $m$ cells of the simulated machine. The simulations of (Matias and Vishkin, 1990a, 1990b) and (Gil, 1990) come close to this ideal, using $O(n + m)$ global memory cells.

The present paper contributes essentially two new simulations, both of which are optimal. One is of ARBITRARY on TOLERANT, the other one of MINIMUM on COLLISION$^+$. In both cases, the slowdown is $O((\log^* n)^2)$ with high probability. The simulations can be combined to yield an optimal simulation of MINIMUM on TOLERANT whose slowdown is $O((\log^* n)^2)$ with high probability. In terms of speed, our simulations are a long way better than all previous results. For PRAMs with infinite memories, they subsume all previous results except one algorithm of (Matias and Vishkin, 1990a, 1990b), which allows the FETCH-AND-ADD PRAM to be simulated, and one algorithm of (Hagerup and Radzik, 1990), which