Parallel Performance Evaluation through Critical Path Analysis

Benno J. Overeinder and Peter M. A. Sloot

University of Amsterdam, Parallel Scientific Computing & Simulation Group
Kruislaan 403, NL-1098 SJ, Amsterdam, The Netherlands
e-mail: {bjo,peterslo}@fwi.uva.nl

Abstract. In this paper we discuss the concept of space-time diagrams as a representation of the execution of an application, and then give a method, based on critical path analysis, for calculating nontrivial upper bounds on the potential parallelism, known as the average parallelism, a complement to the speedup and efficiency.

1 Introduction

Two measures of particular interest in parallel performance evaluation are *speedup* and *efficiency*. Notwithstanding the importance of these two measures, they do not reveal how well the potential parallelism in the application is exploited. Nor do they help to understand why the performance may not be as good as expected.

In this paper we describe a new method called critical path analysis to measure the potential parallelism available in the application. Furthermore, critical path analysis can be used as a tool for bottleneck analysis to improve the potential performance on a parallel system.

The potential parallelism in an application can be quantified in the *average parallelism* measure [1], which is a non-trivial upper bound to the asymptotic speedup (i.e., speedup with infinite resources). The average parallelism reflects the extent to which the software parallelism matches the hardware parallelism. In this respect, the average parallelism measure can be used in the tradeoff between speedup and efficiency that is inherent to the application.

The critical path is the path in the program's execution history that took the longest time to execute. Along this path, we can identify the places where the execution degradation takes place. The knowledge of this path and of the bottleneck(s) along it facilitates an accurate performance evaluation of the problem at hand and gives explicit hints to the possible performance enhancements.

2 Critical Path Analysis

We limit the discussion to programs that adhere to the message-passing model. The parallel program consists of a number of distinct processes that communicate with each other by exchanging messages.

In the following sections the *space-time* model is introduced as an abstract representation of the parallel execution of a program and the global operations of the critical path analysis are described.

2.1 Space-Time Model

Given a particular decomposition of the parallel program, we describe the execution in terms of activities and events. An activity represents a time period of specified length during which some work is performed (calculation or communication). The beginnings and endings of activities are the events in the program (send/receive and process creation/termination events).

The execution of events (the scheduling of the activities) can be ordered according to the fundamental precedence constraints, also known as the causality constraints [4].

Definition 1. *Type 1 constraint*: If e_i and e_j are events in the *same process*, and e_i comes before e_j, then $e_i \rightarrow e_j$. Event e_i is called the *predecessor* of event e_j.

Definition 2. *Type 2 constraint*: If e_i is the sending of a message by one process, and e_j is the receipt of the same message by *another process*, then $e_i \rightarrow e_j$. Event e_i is called the *antecedent* of event e_j.

Now, we introduce *logical clocks* into the system. Generally, a logical clock is just a way of assigning a number to an event, where the number can be thought of as an event particular time stamp. Each process has a logical clock C, which is defined as a function assigning a number $C(e)$ to any event e in the process. A system is considered causally correct if it adheres to the following clock condition:

Definition 3. For any event e_i, e_j, if $e_i \rightarrow e_j$ then $C(e_i) \leq C(e_j)$.

A particular execution run of a parallel program can be described by an acyclic directed graph, also known as a program activity graph or, as we will use, a space-time diagram. Each event in the execution of the parallel program is represented as a vertex in the two dimensional space-time diagram. The two coordinates of each event consist of a spatial coordinate and a temporal coordinate. The spatial coordinate is the process where it is executed. Thus events placed on the same spatial position occur in one process. The temporal coordinate is the (logical) time at which the event occurs, $C(e)$.

The activities between the events are represented by directed edges, adhering to the causality constraints. An example of a space-time diagram is depicted in Fig. 1.

2.2 Critical Times and Critical Paths

In this section, we will use the abstract space-time model to perform the critical path computation.

The activities between the events represented by the directed edges are now labeled with a weight $T(e', e)$, where e' being the immediate ancestor. The weight represents the duration of the activity, i.e., the service time for a calculation or communication.

With the events in the space-time diagram a *critical time* is associated, notation $\mathrm{crit}(e)$. The critical time is related to the service times of the activities, and has no relation with the logical clocks C. The logical clocks are used in the ranking of events and in the space-time diagram construction. Critical time has to do with physical time required to perform activities.

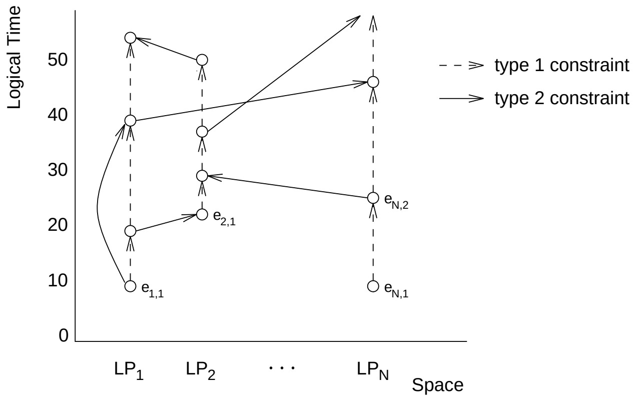


Fig. 1. Space-time diagram depicting the events with their dependency constraints.

We calculate the *critical time* recursively from the predecessors and antecedents as defined in the space-time diagram, with $ANCE$ the set of immediate ancestors we have:

$$\mathrm{crit}(e) = \max_{e' \in ANCE(e)} \{\mathrm{crit}(e') + T(e', e)\}, \tag{1}$$

where $\mathrm{crit}(e)$ is defined zero when $ANCE(e) = \emptyset$. The method for finding the $\mathrm{crit}(e)$ is essentially a topological sort. That is, each $\mathrm{crit}(e)$ can be calculated after all $\mathrm{crit}(e')$ of its immediate ancestors have been computed. The length of the critical path (longest path in the parallel execution) is $\max\{\mathrm{crit}(e)\}$.

Having determined the critical times for each event in bottom-up sequence, the critical path can be defined in top-down order:

– e with $\max\{\mathrm{crit}(e)\}$ is on the critical path;
– if e is on the critical path, then $e' \in ANCE(e)$ resulting in the maximal critical time according to (1) is on the critical path.

The average parallelism can now be defined as the ratio of the total service time required to process the activities, to the length of the critical path.

3 Example: Critical Path Analysis in Parallel Discrete Event Simulation

To assess the general usability of the analysis tool, we apply critical path analysis for the performance evaluation of Parallel Discrete Event Simulation (PDES) protocols.

In performance analysis and evaluation of PDES protocols we need a measure to compare the effectiveness of the different protocols. The key question is: “How much of the potential parallelism is actually realized and what is the lower bound on the execution time?” This lower bound on the execution time is the ultimate goal that the PDES protocols strive to.

The relation with the preceding discussion in Sect. 2 is, that with the construction of the space-time diagram, the logical clock values of the events, $C(e)$, in the execution are assigned through the simulation and not by their execution order. In other words, a simulation is executing the space-time diagram, rather than the space-time diagram generating from the execution of the parallel program.

In the evaluation of the various PDES protocols, we can effectively determine the ability of the protocols to exploit the available parallelism in completing the space-time diagram as given for a particular simulation run. Next, we can use this space-time diagram to abstract a protocol independent measure to quantify the effectiveness of the different protocols.

In the following sections, we will apply the critical path analysis tool to a realistic simulation of a continuous-time Ising spin system.

3.1 Continuous-Time Ising Spin System

The Ising spin system serves as one of the simplest models of interacting particles in statistical physics. It can be used as a general system to model stochastic processes [7]. The system has been used to model ferromagnets, anti-ferromagnetism, and phase separation in binary alloys.

In our case study, we will look into the two-dimensional version of the Ising spin system. The model is comprised of an $n \times n$ square lattice in which each lattice site has an attribute called *spin*, which can take on either of the values 1 (known as up spin) or -1 (known as down spin). Spins on adjacent, nearest-neighbor sites interact in a pair-wise manner with a strength J (known as the exchange energy). There may also be an external field of strength B (known as the magnetic field). The magnetization of the system is the difference between the number of up and down spins on the lattice.

We use a probabilistic model for the computation of the dynamics of the spin values. A lattice site is selected at random and a decision is made on whether or not to flip the site using the well-known Metropolis Algorithm [5].

3.2 Implementation

Ising Spin System with Time Warp. The parallel implementation of the Ising spin system exploits data parallelism available in the system. The lattice is subdivided in sub-lattices of equal sizes, and distributed over the parallel processors. To minimize the communication and synchronization between the processors, shadow boundaries (a copy of the boundaries of its nearest neighbors) are applied to each sub-lattice. When a boundary lattice site makes an actual spin flip, communication takes place to update the shadow boundary of the neighboring processors.

The synchronization protocol we use in the parallel discrete event simulation is known as Time Warp [3, 6]. Its basic synchronization method is process rollback. If the causality constraint (as defined in Def. 3) is violated, the simulation process is rolled back in time, undoing all side effects of the erroneous computation. For a clear introduction of the various PDES protocols, see [2].

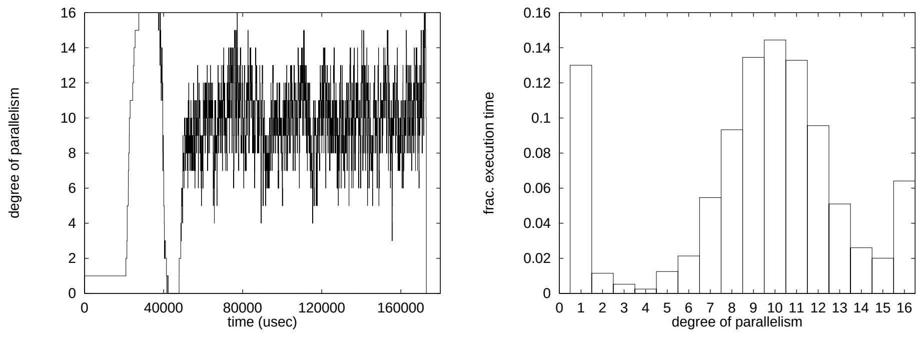
The simulation is performed on a Parsytec GCel-3/512 with the PARIX operating system.

Using the Critical Path Analysis Tool. The critical path analysis tool is an application independent utility. It accepts trace files generated by the execution of a program and constructs a space-time diagram, annotated with the service time of each activity. The critical times of the events are calculated from the space-time diagram according to (1).

The interface between the critical path analysis tool and the simulation is via the Time Warp PDES protocol, and thus transparent to the Ising spin application (or any application implemented with use of the Time Warp protocol). During the execution of the simulation, each processor generates a trace file that is input to the analysis tool.

3.3 Sample Run with Critical Path Analysis Tool

From the execution of the Ising spin system with the Time Warp protocol on sixteen nodes of the Parsytec GCel, we have extracted the following trace data.



(a) Profile of the execution. (b) Shape of the execution.

Fig. 2. Trace output of Ising spin simulation.

In Fig. 2(a) the profile of the execution of the simulation is shown. During the execution, the number of active processors vary between one and sixteen. One can clearly identify an initialization phase in the beginning of the simulation. This corresponds to the exchange of boundary information, required by the Ising spin implementation. After the initialization phase, the simulation continues with randomly updating lattice points, and exchanging message when boundary points are flipped.

Figure 2(b) is a histogram of the degree of parallelism during the execution of the Ising spin simulation. From the shape of the execution, one can see that 13 % of the time only one process is active (most prominent in the initialization phase of the simulation). The other values are clustered around the degree of parallelism of ten. This corresponds with the second phase of the execution in Fig. 2(a). The average parallelism can be found in Table 1 for $p = 16$ and $n = 10$.

In Table 1 the measured *relative* speedup, S, is set against the computed average parallelism, A. The data clearly indicates that for small problem sizes ($n = 10$), the communication and synchronization overhead reduces the attained speedup (40 % of the spin flips result in communication). For problem size $n = 20$, the communication to spin flip

Table 1. Measured speedup versus average parallelism.

	n = 10		n = 20	
	S	A	S	A
p = 4	1.07	2.43	1.78	2.74
p = 16	1.90	8.82	2.77	9.05

ratio is better (20 %), which is also expressed in an increase of the attained speedup relative to the average parallelism. It is clear that the communication to flip ratio behaves as $4/n$, and improves with the increase of the sub-lattice size n. But still, for both problem sizes, the fourfold increase in average parallelism from p = 4 to p = 16, is only translated in a twofold increase in the measured speedup due to communication and protocol overhead.

4 Conclusions

Critical path analysis is shown to be a very useful tool in performance evaluation of parallel systems. Important characterizations of parallel applications, such as the average parallelism, the profile, and the shape of the execution, can be generated from the space-time diagram and the critical path analysis.

The abstraction of the parallel execution to the space-time diagram and the critical path analysis provides information for performance "debugging" and directs users to bottlenecks in the program. This can be very useful in comparing different decompositions of the same problem by showing how much concurrency is available in each decomposition.

In the near future, we will direct our research to the stochastic performance modeling of PDES protocols with modeled stimuli from real applications like the Ising spin system.

References

1. D. L. Eager, J. Zahorjan, and E. D. Lazowska, "Speedup versus efficiency in parallel systems," *IEEE Transactions on Computers*, vol. 38, no. 3, pp. 408–423, Mar. 1989.
2. R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.
3. D. R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, July 1985.
4. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, July 1987.
5. B. D. Lubachevsky, "Efficient parallel simulations of dynamic Ising spin systems," *Journal of Computational Physics*, vol. 75, no. 1, pp. 103–122, Mar. 1988.
6. B. J. Overeinder and P. M. A. Sloot, "Application of Time Warp to parallel simulations with asynchronous cellular automata," in *Proceedings of the 1993 European Simulation Symposium*, (Delft, The Netherlands), pp. 397–402, Oct. 1993.
7. P. M. A. Sloot, "Modelling and simulation," in *Proceedings of the 1994 CERN School of Computing*, (Sopron, Hungary), pp. 177–226, Sept. 1994. ISBN: 92-9083-069-7, ISSN: 0304-2898.