Simulating Asynchronous Parallel Circuit Optimization Algorithms

Árpád Bűrmen, Janez Puhan, Iztok Fajfar, Andrej Nussdorfer, and Tadej Tuma
Faculty of Electrical Engineering
University of Ljubljana
Tržaška 25, 1001 Ljubljana, Slovenia
arpadb@fides.fe.uni-lj.si

Abstract

The idea of asynchronous parallel optimization occured during the efforts to develop efficient parallel optimization algorithms. Since asynchronous parallel algorithms are more difficult to analyse than their synchronous counterparts, the development of such an algorithm benefits greatly from practical experience. The only way to assess the practical value of a particular algorithm is to try it out on established test function suites and real-world problems. In order to test an algorithm it must be implemented and run on a cluster of workstations. Since development and debugging for such platforms is not an easy task, we expect to accelerate the process of developement by utilizing a simulation tool where the developer can concentrate on the algorithm itself without wasting time on various practical implementation issues. We present a simple model of a parallel optimization environment. Based on the model the simulation technique is presented. The simulator is implemented in MATLAB and its use is illustrated with an example.

1 Introduction

In order to accelerate the search for a minimum of a cost function in the process of optimization, the idea to divide the work among several workers occurred. First attempts resulted in synchronous parallel algorithms, where workers exchanged information after reaching so called synchronisation points in the algorithms they were executing. Since the time required to evaluate the cost function can vary greatly from point to point in the search space, several workers that finished sooner had to wait for the slower ones to reach the syncronisation point (synchronisation penalty). The idle time of individual workers resulted in an overall smaller acceleration.

The variations of evaluation time are not the only cause for degraded performance of the system. Communication among workers introduces delays and when these delays become comparable to the time between successive syncronisation points, their impact on the performance becomes significant. A solution for the latter problem is to assign multiple cost function evaluations to individual workers which moves the synchronisation points further appart and reduced the effect of communication delays [1].

The aforementioned solution can't cope with the synchronisation penalty arising from the variations of the evaluation time. Therefore an idea occured to remove the syncronisation points from the algorithm and communicate the information as soon as it is available [2]. The approach resulted in asynchronous parallel optimization algorithms. Such algorithms are more difficult to design and analyze and practical experience plays a great role in their evolution [4]. In order to test a particular algorithm it must be implemented and run on a cluster of workstations. The developement cycles for parallel distributed programs are generally longer than for single-processor programs since one has to consider several practical issues that are of little or no relevance to the algorithm itself. Obtaining a cluster of workstation is also associated with relatively high costs. These facts all speak in favor of a simulation-based approach.

The remainder of this paper is organized as follows. First a simple model of a parallel optimization system is presented. The principles of event driven simulation [3] are explained and associated with the problem of simulation of asynchronous parallel optimization algorithms. Implementation details regarding the simulator are given and its use is illustrated by simulating optimization runs of the asynchronous parallel pattern search (APPS) algorithm [4, 5].

2 A Model of a Parallel Optimization System

Let $\mathcal{U}_M$ denote the set of all possible messages M that can be communicated by any of the workers. A message consists of information such as the position in the search space (member of $\mathbb{R}^n$), cost function value (member of $\mathbb{R}$), etc.

Every worker W_i from the set of available workers $\mathcal{W} = \{W_1, W_2, ..., W_p\}$ in such an optimization system has a state S_i associated with it. The state represents the memory of a worker. Let $\mathcal{U}_S^i$ denote the set of all possible states worker W_i can be in.

A combination of time, source, destination, and message is called an event ($E = (t, s, d, M)$, $t \in \mathbb{R}$, $s \in \{1, ..., p\}$, $d \in \{0, 1, ..., p\}$, $M \in \mathcal{U}_M$). Destination is the number of the worker that will receive the message. 0 has special meaning. Events with destination 0 are sent to all workers (broadcast). Let $\mathcal{U}_E$ denote the set of all possible events (eq. (1)).

$$\mathcal{U}_E = \mathbb{R} \times \{1, ..., p\} \times \{0, 1, ..., p\} \times \mathcal{U}_M. \qquad (1)$$

Let $\mathcal{U}_\mathcal{F}$ denote the set of all possible subsets of $\mathcal{U}_E$. A worker (eq. (2)) is a transformation that for some incoming event $E_{in} = (t_{in}, s_{in}, d_{in}, M_{in})$ and some state S_{old} produces a set of outgoing events $\mathcal{F}_{out}$ and a new state S_{new}:

$$\begin{gathered} W_i : (E_{in}, S_{old}) \mapsto (\mathcal{F}_{out}, S_{new}), \qquad (2) \\ E_{in} \in \mathcal{U}_E, \ \ \mathcal{F}_{out} \in \mathcal{U}_\mathcal{F} \ \ S_{old}, S_{new} \in \mathcal{U}_S^i. \end{gathered}$$

S_{new} becomes the new state of worker W_i.

Generally $t_{in} \leq t_{out}$ (causality) and $s_{out} = i$ for all $E_{out} = (t_{out}, s_{out}, d_{out}, M_{out}) \in \mathcal{F}_{out}$ are required from W_i. A worker can respond to an incoming event with an empty set $\mathcal{F}_{out}$, meaning that there is no respone to the incoming event.

$t_{out} - t_{in} = \delta_W$ represents the evaluation time and can be modelled either deterministically or stochastically. One could even use a simulator and measure the evaluation time (δ_W) for a particular incoming event in order to accurately simulate the behaviour of the optimization system.

Up to now we have dealt with the model of a worker. This is the part that has to be provided by the algorithm developer. Next comes the modelling of the communication system. This part depends only on the characteristics of the platform (e.g. tightly coupled multiprocessor machine, cluster of workstations, etc.).

All events from the set $\mathcal{F}_{out}$ are delivered by the communication system to their destinations. The communication system is modelled by a transformation of an individual event $E = (t, s, d, M)$ (which is a member of $\mathcal{F}_{out}$ produced by W_i) to a set of distributed events $\mathcal{F}'$ (eq. (3)).

$$\begin{gathered} T : E \mapsto \mathcal{F}', \qquad (3) \\ E \in \mathcal{U}_E, \ \ \mathcal{F}' \in \mathcal{U}_\mathcal{F}. \end{gathered}$$

If $d \neq 0$, $F' = \{E'\}$ and $E' = (t', s', d', M')$ is composed as follows:

$$t' \geq t, \ \ s' = s, \ \ d' = d, \ \ M' = M. \qquad (4)$$

If $d = 0$ (broadcast), $F' = \{E'_1, E'_2, ..., E'_p\}$ and $E'_i = (t'_i, s'_i, d'_i, M'_i)$ is composed as follows:

$$t'_i \geq t, \ \ s'_i = s, \ \ d'_i = i, \ \ M'_i = M. \qquad (5)$$

The delay introduced by the communication system when delivering an event to a worker ($\delta_T^i = t'_i - t$) can be modelled deterministically or stochastically.

Now we have a model of a worker and a model of the communication system. In the following section we present the simulation algorithm based on an event queue.

3 Simulation Algorithm

An event queue $\mathcal{Q}$ is a set of events ordered by time. Enqueueing an event E means that the event is added to $\mathcal{Q}$. Dequeueing an event $E \in \mathcal{Q}$ means that the event is removed from $\mathcal{Q}$.

The oldest event E (with smallest t) in the queue is dequeued and processed by the simulation algorithm. Processing the event E means sending it to its destination worker (W_d). The state of worker W_d is updated and the resulting events from the set $\mathcal{F}_{out}$ are processed by the model of the communication system T. The resulting events are enqueued in $\mathcal{Q}$. Then the next oldest event is dequeued and the process described above is repeated.

4 Initialization

The simulation is initialized by enqueueing p initialization events with $t = 0$ in the event queue (one for every worker). These events carry all the information needed by the workers for responding to incoming events. The initialization events are dequeued by the simulator and distributed to the workers as the simulation starts Worker W_i responds to the initialization event by copying the information from the event into its state S_i. In response outgoing events are generated that are processed by the model of the communication system T and then enqueued by the simulator.

5 Simulator Implementation

The simulator is implemented in MATLAB. The workers are described by a function similar to the following one:

```
function [Eout, Sout]=worker(Ein, Sin)
```

The worker is responsible for setting the output event's time by adding the evaluation time δ_W to the input event's time. In the example the evaluation time is modelled stochasticaly by choosing a random value from the interval $[t_W^{low}, t_W^{high}]$. Events (E) and worker's state (S) are MATLAB structures. S_i is manipulated only by W_i and can contain arbitrary fields. All events must have 3 fields: source (s), destination (d), and time (t). The rest is arbitrary and represents the message (M) carried by the event.

There is only one worker function. In case the optimization system is homogeneous, this is all one needs for describing the worker's behaviour. In case the system consists of different workers, the worker's number can be obtained from the destination field of the input event. This is how the simulator choses the corresponding input state for the worker function.

The simulator is responsible for holding the worker's state and managing the event queue. It sends all events produced by the workers to the model of the communication system. The resulting events are then enqueued in $\mathcal{Q}$.

The model of the communication system (eqns. (3), (4), and (5)) is responsible for adding the communication delay to the events. In case the destination is the same as the source, the delay is chosen randomly from interval $[t_T^{low_1}, t_T^{high_1}]$. Otherwise the delay is chosen from the interval $[t_T^{low_2}, t_T^{high_2}]$. The multiplication of broadcast events is also the job of the communication system model.

The simulator function's arguments are: the name of the worker function, the intervals from which the communication delay is chosen, time when to stop the simulation, and the set of initialization events $\{E_1, E_2, ..., E_p\}$ that are enqueued at the beginning of the simulation. The number of workers is determined from the number of initialization events.

6 Example

The APPS from the convergence analysis paper [5] was implemented for the proposed simulator. A simplified stepsize control was used. All workers were running the same algorithm. Worker W_i holds as its state S_i the best point found in the process of the search ($x_{best}^i \in \mathbb{R}^n$), its associated cost function value ($f_{best}^i = f(x_{best}^i)$), the stepsize parameter (Δ^i), and the search direction $d^i \in \mathbb{R}^n$.

An *init* event carries the search direction (d), the initial point (x), and the initial stepsize (Δ). The same initial stepsize and the same initial point are sent to all workers. The search directions must positively span $\mathbb{R}^n$ [6]. A *point* event carries a point (x), its associated cost function value (f), and the stepsize (Δ). The cost function evaluator receives an *eval* event carrying a point of origin (x), a direction (d), and a stepsize parameter (Δ). It evaluates the cost function at point $x_e = x + \Delta \cdot d$. After the evaluation an *evalEnd* event is sent back with the point of origin (x), the point of cost function evaluation (x_e), the computed cost function value (f_e) at x_e, and the stepsize parameter (Δ). See Listing 1 for the APPS algorithm.

APPS was simulated for the square 2D function, Rosenbrock 2D function, and Woods 4D function for various numbers of workers. The initial point for the square function was $[5, 5]$, for the Rosenbrock 2D function $[-1.2, 1.0]$, and for the Woods 4D function $[-3, -3, -3, -3]$. Initial stepsize ($\Delta_{initial}$) was set to 0.03. Δ_{min} was set to $\Delta_{initial}/2^{20}$ and Δ_{max} was set to $\Delta_{initial} \cdot 2^7$. The evaluation time was a uniformly distributed random variable from $[0.1s, 0.5s]$. The communication delay was also uniformly distributed from $[5ms, 20ms]$ ($[1\mu s, 5\mu s]$ for messages with $s = d$).

```
/* APPS, i-th worker's algorithm. */
/* E is the received event. */
/* $\Delta_{min} = \Delta_{initial} \cdot 2^{c_{min}} < \Delta_{initial}$ */
/* $\Delta_{max} = \Delta_{initial} \cdot 2^{c_{max}} > \Delta_{initial}$ */
/* $c_{min} < c_{max};\ c_{min}, c_{max} \in \mathbb{Z}$ */

if $E$ is an init event then
    $d^i := E.d$;
    $x_{best}^i := E.x$;
    $f_{best}^i := +\infty$;
    $\Delta^i := E.\Delta$;
    Send an eval event to the local evaluator
        with ($x := x_{best}^i, d := \mathbf{0}, \Delta := \Delta^i$);
else if $E$ is a point event then
    if $E.f < f_{best}^i$ then
        $x_{best}^i := E.x$;
        $f_{best}^i := E.f$;
        $\Delta^i := E.\Delta$;
    end
else if $E$ is an evalEnd event then
    if $E.f_e < f_{best}^i$ then
        $x_{best}^i := E.x_e$;
        $f_{best}^i := E.f_e$;
        $\Delta^i := E.\Delta \cdot 2$;
        if $\Delta^i < \Delta_{min}$ then
            $\Delta^i := \Delta_{min}$;
        end
        if $\Delta^i > \Delta_{max}$ then
            $\Delta^i := \Delta_{max}$;
        end
        Broadcast a point event
            with ($x := x_{best}^i, f := f_{best}^i, \Delta := \Delta^i$);
    else if $x_{best}^i = E.x$ then
        $\Delta^i := \Delta^i/2$;
    end
    Send an eval event to the local evaluator
        with ($x = x_{best}^i, d = d^i, \Delta = \Delta^i$);
end
```

Listing 1: The simplified APPS algorithm.

In the first set of simulation runs the simulation was stopped after t_{stop}. The results are listed in Table 1. The performance with regard to the obtained cost function value significantly improves as the number of workers increases.

Test function	p	t_{stop} [s]	$f(x)$
Square 2D	4	15	$0.3125 \cdot 10^{-6}$
Square 2D	8	15	$0.1953 \cdot 10^{-8}$
Rosenbrock 2D	4	100	$0.1250 \cdot 10^{+0}$
Rosenbrock 2D	8	100	$0.8236 \cdot 10^{-3}$
Woods 4D	8	100	$0.6079 \cdot 10^{+1}$
Woods 4D	16	100	$0.2294 \cdot 10^{+1}$
Woods 4D	24	100	$0.4066 \cdot 10^{-1}$

Table 1: Simulation results for the APPS algorithm. Simulation was stopped after t_{stop} was reached.

In the second set of simulation runs the simulation was stopped after the cost function dropped below f_{stop}. The results are listed in Table 2. The time required to reach cost function value f_{stop} dramatically drops with the increasing number of workers, indicating that a fair amount of acceleration can be expected from a parallel asynchronous optimization system.

Test function	p	f_{stop}	Time [s]
Square 2D	4	$1.0 \cdot 10^{-5}$	12.2676
Square 2D	8	$1.0 \cdot 10^{-5}$	10.4614
Rosenbrock 2D	4	$0.5 \cdot 10^{+0}$	70.3586
Rosenbrock 2D	8	$0.5 \cdot 10^{+0}$	12.2444
Woods 4D	8	$0.8 \cdot 10^{+0}$	166.228
Woods 4D	16	$0.8 \cdot 10^{+0}$	29.7512
Woods 4D	24	$0.8 \cdot 10^{+0}$	10.3047

Table 2: Simulation results for the APPS algorithm. Simulation was stopped after $f(x)$ dropped below f_{stop}.

From Figure 1 it can be seen that although the system is asynchronous, the workers are very strongly coupled. The cost function value and the stepsize may take different paths on different workers, but they tend to become equal across all processors approximately every 0.5s.

Kolda and Torczon prooved [5] that such "round-ups" are ensured by stepsize limiting. The APPS convergence proof is based on this observation. An interesting point (observed in Figure 1 and other optimization runs) is, that the stepsize parameter does not even come close to its bounds (Δ_{min}, Δ_{max}), so there must be some other mechanism in the algorithm (that at least assists convergence if not guarantees it) than the one found by Kolda and Torczon [5].

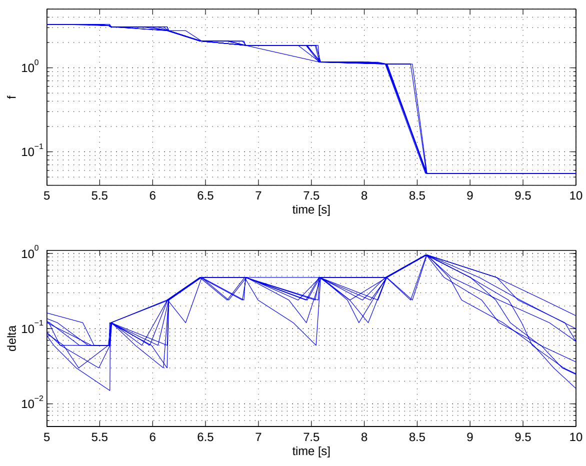

Figure 1: Cost function and stepsize with respect to time for all workers (Rosenbrock 2D function, APPS run with 8 workers).

7 Conclusions

The importance of simulation in the design and testing of asynchronous parallel optmization algorithms was outlined. A simple model of a parallel optimization system was created and a simulation algorithm for such systems was developed. The simulator was implemented in MATLAB and tested with several APPS runs involving various numbers of workers and three different benchmark cost functions. In APPS the "round-up" effect was demonstrated and we observed that it cannot be completely explained by the arguments given by Kolda and Torczon in their APPS convergence analysis.

References

[1] J.E. Dennis, Jr. and V.J. Torczon. Direct search methods on parallel machines. SIAM Journal on Optimization, 1:448–474, 1991.

[2] D. Bertsekas and J. Tsitsiklis. Parallel and Distributed Computation: Numerical Methods. New Jersey, NJ, Prentice-Hall, 1989.

[3] A. Law and D. Kelton. Simulation Modeling and Analysis. New York, NY, McGraw-Hill, 2000

[4] P.D. Hough, T.G. Kolda, V.J. Torczon. Asynchronous parallel pattern search for nonlinear optimization. SIAM Journal on Scientific Computing, 23:134–156, 2001.

[5] T.G. Kolda, V.J. Torczon. On the convergence of asynchronous parallel pattern search. Submitted to SIAM Journal on Optimization, December 2001.

[6] C. Davis. Theory of positive linear dependence. American Journal of Mathematics, 76:83–108, 1954.