Zehra Sura ([zsura@us.ibm.com](mailto:zsura@us.ibm.com))

Li Zhang ([zhangli@us.ibm.com](mailto:zhangli@us.ibm.com))

Ashish Kundu ([akundu@us.ibm.com](mailto:akundu@us.ibm.com))

Ravi Nair ([nair@us.ibm.com](mailto:nair@us.ibm.com))

1. Invention title

Partial Synchronization Between Compute Tasks Based on Threshold Specification

1. Invention description - Background
2. Problem statement

In parallel or distributed computing, multiple computing tasks (processes or threads) work asynchronously to execute tasks in an application. Except in the case of embarrassingly parallel applications, the multiple computing tasks need to co-ordinate with one another. Barriers are a commonly used synchronization mechanism when multiple tasks need to interlock with one another. Barriers require each task to wait until all tasks have reached the barrier point in the code, and then the barrier releases and all tasks continue execution. A barrier can be configured to apply to only a specified subset of compute processes/threads instead of all processes/threads in the application.

Currently, barriers are not expressive enough to capture algorithmic-level semantics in some application scenarios. For example, the case when *any* x number of processes from a set of y processes need to interlock, or when the barrier release should occur when a certain amount of data is updated in a specified memory region. In such cases, using existing synchronization constructs (e.g. locks with custom code) to encode the application semantics results in unnecessary synchronizations or an inefficient implementation, which renders some class of solutions to be impractical on current compute platforms.

1. Prior art

There are well established synchronization mechanisms in computing, including locks, barriers, semaphores, counting semaphores, mutexes, atomic operations, event traps, and transactions.

There exist efficient implementations for all these synchronization mechanisms, using hardware support and/or system software.

There are also programming paradigms (e.g. bulk synchronous programming and transactional processing) that allow for even more efficient implementation of some of these synchronization mechanisms.

1. Limitations of prior art

All prior synchronization mechanisms cannot directly express synchronization semantics based on thresholds of some measure (e.g. computation done or data processed). This in turn prevents efficient implementations that are practical and that can support end-application semantics.

1. Invention description – Summary
2. Main idea

We propose a flexible barrier-like synchronization mechanism, where the barrier release condition is based on reaching a threshold on some measure. For example, the measure can be the number of processes (as in traditional barriers) or the amount of data updated in a specified memory region. The threshold can be specified as an absolute value (e.g. x of y processes) or as a fraction (e.g. x percent of elements of array A). Such synchronization can then be efficiently implemented in system software, and possibly with hardware support.

1. Advantages

Our proposed synchronization constructs make it possible to directly express some semantics of the algorithm/application in the code implementing it, which enables efficient system-level implementations that are practical and can support a new class of application solutions. Previously, such semantics had to be explicitly encoded and emulated using available mechanisms. This can artificially constrain the behavior and semantics of the application, or can make it too slow/inefficient.

1. Invention description – Details

**Mechanism**

We propose a flexible barrier-like synchronization mechanism, which we refer to as TBarrier, where the barrier release condition is based on reaching a threshold on some measure. In one embodiment, the TBarrier can be specified using three parameters, as in the following notation:

TBarrier ( type, base\_set, threshold )

The “type” parameter identifies one of the different kinds of measures supported for TBarriers on a computing platform. Examples of measures are an absolute number of processes, a fraction of processes, or an absolute/fractional amount of data updated in a memory region. This parameter determines the format and interpretation of the values for the other 2 parameters.

The “base\_set” parameter defines the scope or range of the measure used to determine the barrier release condition. For example, it could be the absolute or symbolic value for the total number of processes involved in the TBarrier, or the program data structure or bounds of the memory region to monitor for updates. It is also possible for this value to be null (unused), for example when the TBarrier waits for an absolute number of processes.

The “threshold” parameter defines an absolute or fractional threshold value that is used to determine the barrier release condition.

**Applications**

An example application for using TBarriers is a parallel implementation of an iterative convergence algorithm such as the Jacobi method. In each iteration of the algorithm, every element of a matrix is updated based on values of elements in neighboring rows and/or columns. The algorithm works correctly even without synchronization between iterations, such that computation in a successive iteration may use updated values from the previous iteration, or stale values from earlier in the program execution, or some combination of updated and stale values. The ratio of updated to stale values represents a tradeoff between speed of computation per iteration and the number of iterations required for convergence. TBarriers can be used to control the minimum ratio of updated to stale values used in the successive iteration by appropriately setting the threshold values for updated array regions. This allows tuning the overall application performance by simply changing the value of the threshold parameter of the TBarrier. Further, different compute platforms support different computation resources and communication latencies/bandwidths. As a result, different compute platforms may exhibit the best execution performance when using different ratios of updated and stale values. In this case, the use of TBarriers makes it easy to tune performance across disparate computing platforms.

A second example application for using TBarriers is partially synchronous weight update in stochastic gradient descent used in training of deep neural networks. The fully synchronous method of stochastic gradient descent requires each learner to wait for updates from all other learners. Previous work has explored using partial synchronization and weight updates: learners are partitioned into subgroups, and each learner only waits for updates from members of its subgroup. Use of TBarriers makes it easy to program different subgroup sizes (in terms of number of processes). It also allows expressing and experimenting with different partial synchronization schemes, e.g. dynamically forming subgroups and waiting for updates from only a certain number of members of the subgroup.

A third example application for using TBarriers is partial data encryption for security. There is a tradeoff between cost of encryption (in terms of time, and compute resources) and degree of protection. A security policy dictates the desired level of protection, and it can be used to encode an expression for the threshold parameter of a TBarrier. The threshold value, expressed as a fraction of the memory region to be encrypted, controls the amount of data that gets encrypted before the barrier is released and encryption terminates.

**Implementation**

Depending on the compute platform and the measure(s) being supported, TBarriers can be implemented in system software, in hardware, or using a combination of both. It is important to note that exposing the semantics to lower levels of the stack allows for efficient and optimized implementations.

As an example, when the measure is the number of processes/threads, the kernel layer or the hypervisor layer can keep track of processes/threads hitting the barrier, signal release of the barrier when the threshold is met, and ensure that any remaining processes/threads are not blocked by the barrier.

As another example, when the measure is the amount of data updated in a memory region, a hardware facility can be built into the memory system that tracks the number of blocks updated within a memory region (with limits on the size of the memory region and the granularity of the blocks), and generates a signal when a threshold is reached. The threshold and bounds of the memory region to be monitored by the hardware facility are programmable in this scenario.

1. Novelty
2. A new barrier-like synchronization mechanism, where the barrier release condition is based on reaching a threshold on some measure.
3. The measure can be number of processes/threads.
4. The measure can be amount of data updated in a memory region.
5. A system can support one or more measures for the new synchronization mechanism, with a “type” parameter to distinguish between them.
6. The threshold can be an absolute value or fractional value, or a symbolic expression in the program for an absolute or fractional value.
7. A “base\_set” parameter can be specified to use as the scope/range in case of fractional threshold values.
8. Specialized hardware and system software can be used to support the new synchronization mechanism.
9. Background/Related Work

* R. Nair and S. Gupta, "Wildfire: Approximate synchronization of parameters in distributed deep learning," in IBM Journal of Research and Development, vol. 61, no. 4, pp. 7:1-7:9, July-Sept. 1 2017
* Jacobi Method: <https://en.wikipedia.org/wiki/Jacobi_method>
* Thread or Process Synchronization in Computer science: https://en.wikipedia.org/wiki/Synchronization\_(computer\_science)
* Leslie G. Valiant, A bridging model for parallel computation, Communications of the ACM, Volume 33 Issue 8, Aug. 1990
* David Abramson and John Rosenberg, Hardware support for program debuggers in a paged virtual memory, SIGARCH Comput. Archit. News 11, 2 (June 1983)