5N-8

A Study of Garbage Collection for PIE64

Lu Xu, Kentaro Shimada, Hanpei Koike, and Hidehiko Tanaka
Tanaka Lab., Dept. of Electrical Engineering, Univ. of Tokyo*

Abstract

There are many proposed distributed architectures for efficient execution of programs with potential parallelism. One of these architectures is a loosely-coupled multiprocessor system, in which the processing elements share only the communication medium. PIE64 is such a machine. Here, we will discuss a global garbage collection algorithm suitable for PIE64, with assumption that no real-time constraints is given, but that a fast garbage collector with minimal space, time is required.

Introduction

PIE64 is a parallel inference machine and executes programs described by the committed-choice language referred to as FLENG. In PIE64, there are 64 IUs(inference unit), which are connected by two high-intelligent and high-speed interconnection networks. In each IU, there are 1M word local memory(1 word=4 bytes) and four processors, one of them is UNIRED(unify/reduce) used for inference, one is for control, the other two are for communication using the interconnection networks. In the UNIRED, there is a pipeline for unification.

In our proposal of garbage collection for PIE64, we divide the GC into two phases, one is marking phase, the other is compaction phase. In the marking phase, the system-wide accessible cells will be marked using a combination of parallel breadth-first/depth-first strategies. In compaction phase, the marked cells will be compacted to one end of local memories as much as possible.

Why not the incremental method ?

The global methods are usually disputed because of non-real-time. To meet the real-time constraints, people usually adopt incremental garbage collectors. In particular, it is usually said that in case of committed-choice languages, the memory is consumed up rapidly and garbage collection is required much more frequently. But we choose the conventional method. The reason is that the applications for PIE64 are assumed to be time-consumed and space-consumed problems and there may be no real-time constraints in most of them. Even there are real-time constraints indeed, we think we can guarantee that our GC performance is no bad compared with other methods, because

- for any program in PIE64, on an average, the garbages created by the program must be collected by itself. This is unavoidable. For the incremental methods, this work is distributed in the computation, while for the conventional method, this work is done with halting the computation. But the total amount of the work is same. In this view, the conventional method is much better than incremental methods.

- we can do our best to guarantee that the time for GC is so little that users may not take care of it. By experience, we know in case that the memory is larger than 3M bytes, the users will take the GC's time into account if conventional method is used. For PIE64, there are 4M bytes(1M words) in each IU. But, if we do garbage collection with making full use of the pipeline, the time used will be so little as to almost meet real-time constraints.

It must be admitted that when an incremental method is selected, the computation, may be slowed down to a certain extent. In order to meet real-time constraints but slow down computations, or to expedite computations but neglect real-time constraints, which is better? it's a problem. For PIE64, we choose the latter. So we would obviate from incremental methods as long as possible.

Nevertheless, some other methods may be added in the future to improve the performance of the conventional method.

Object Structure and Object Storage

All objects in PIE64 are distributed in the local memories. As the basic element of objects, cell is presented whose size is 4 bytes. Each cell consists of a two-bit field used for GC, and the data or pointer itself. Because in PIE64, goals are often copied between IUs, there are quite a lot of remote pointers. To manipulate remote pointers efficiently, no difference is made between representation of remote pointers and that of local pointers. But when a remote pointer is created, additional space must be kept for the use of GC in the future.

Global Garbage Collection Specifications

*Univ.of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113

- Synchronization

  We choose the method without Master, because if there is a master, either it is a distinguished IU, or it is any general IU, additional transits are unavoidable. But we can do synchronization without master, by using simple hardware support and choosing properly maximum possible transmission delay of a message in the communication subsystem. With the synchronization mechanism, we can start GC, and decide when the marking phase and compaction phase are completed respectively.

- Marking Phase

  Now, we present the idea of the marking scheme. The space requirement of it is determined and can be guaranteed in advance. The original idea is got from Huak and Keller in their Marking-Tree Collector.

  The key idea is :

  In a conventional collector used before, a stack is used to execute marking. But if a tree structure is used instead of stack, we can get the idea suitable for parallel marking. This tree is referred to as the Marking Tree. So we can imagine a marking task for root is spawned when GC is called. Therefore, a number of sub-mark-tasks are spawned in accordance with its children with the limitation of definite space. In addition, the marking tree is simultaneously built for termination. Termination is detected since each mark task eventually spawns an uptree task, which propagates upward in the marking tree. When an uptree task arrives at a cell, whether all the children of the cell have been marked will be tested. If not, a number of marking tasks will be spawned in accordance with the children that have not been marked, otherwise, a new uptree task will be spawned from this cell and pass over upward the Marking Tree. Spawning an uptree task from the root indicates that marking is complete.

  In fact, there is no need of keeping the pointers form parents to their children. But the pointers form children to their parents are required. In PIE64, because we have no other space to store them, we have to make use of technique of the reverse pointer. So in PIE64, marking objects is performed by two kinds of messages :

  - Mark-object(o,p) : which means that as one child of *p*, *o* is to be marked.
  - Mark-object-Reply(o,p) : which means that as one child of *p*, *o* has been marked.

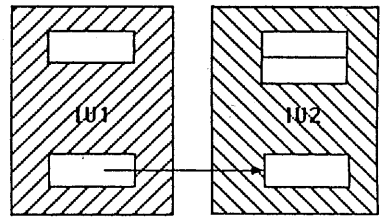
  We will use these two as basic primitives for GC.

- Compaction Phase

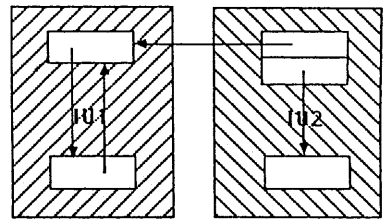
  We plan to use the morris' algorithm.

- About the Remote Pointers

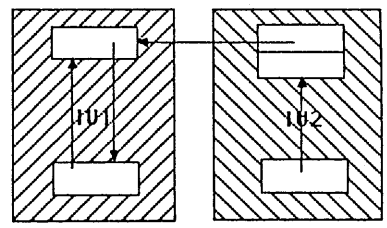
  As said previously, when a remote pointer is created, the space needed by GC must be kept. In the marking phase, when a remote pointer is found, the acknowledgement signal is sent to the corresponding IU, and marking is done there. When compaction is completed, the new address is sent to the cell whose address may also be changed.



a remote pointer in PIE



a remote pointer when it is marked



when a reverse pointer has been created

Future Research

In the global garbage collection, the compaction is very tedious and very time -consumed. How to make use of the pipeline to do the compaction is still a problem. And whether the global garbage collection is enough or not must be tested by means of simulation. These are our future work.

References :

[1]. P.Hudak, R.Keller, " Garbage Collection and Task Deletion in Distributed Applicative Processing Systems", Proceeding of the ACM Symposium on LISP and Function Programming, Aug. 1982

[2]. A.Goto, "Real Time Garbage Collection for Inference Machine By Lazy Reference Count", LPC, 1988

[3]. K.M.Ali, "Global Garbage Collection for Distributed Heap Storage Systems ", Internal IBM Rep. RC 11082(49769)