TM-0967

A Parallel Copying Garbage Collection for KL1 on a Shared Memory Multiprocessor

by
A. Imai, K. Hirata & K. Taki

November, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Parallel Copying Garbage Collection for KL1 on a Shared Memory Multiprocessor

Akira Imai* Keiji Hirata Kazuo Taki

Institute for New Generation Computer Technology (ICOT)

Abstract

We propose an efficient parallel copying GC which is an extension of Baker's sequential algorithm on a shared memory multiprocessor.

1 Introduction

We are developing a parallel inference machine PIM which has a tightly coupled sub-structure called cluster, where 8 processing elements are connected by shared memory.

KL1, the target language of PIM, is a concurrent logic programming language based on flat-GHC. Its naive implementation consumes memory area rapidly since KL1 does not have side-effect nor backtracking mechanism. As a result, the garbage collection (GC) will occur so frequently that the performance of the total system may be seriously damaged by GC overhead if the GC is not efficient.

To achieve a high performance GC, we designed a parallel copying GC based on a Baker's "stop and copy" algorithm[1]. The advantages of this algorithm are that it is simple and fast because only active cells are accessed.

2 Parallel Execution Algorithm

Reducing Frequent Update of Shared Variables

The bottom of the new heap (*tospace* in [1]) is one of the shared variables of all processors in parallel execution of copying GC. In a naive implementation, this value could be updated with mutual exclusion whenever one data cell is copied to the new heap. The frequent update of this shared variable might be the bottleneck.

To avoid this, the unit of the update of the bottom of the new heap set to a fairly large value (called "heap extension unit", **HEU** in short). This modification however can cause internal fragmentation because any size of data can be allocated.

To solve this, structures are allocated by size which is raised to a value larger nearest 2^n and only the same size of structures are copied in one unit. This strategy does not waste rest of the allocated unit.

Scanning Discontinuous Area

In the parallel algorithm, the new heap is divided and allocated to multiple processors. To scan all new heap which is not continuous from a viewpoint of a processor,

- Pointer *Ss* corresponding to *S (scan)* of sequential algorithm are managed in each size by each processor locally. This pointer represents the scanning point.
- Pointer *Bs* corresponding to *B (bottom)* of sequential algorithm are managed in each size by each processor locally. This pointer represents the bottom of the allocated unit. Also *GlobalB* is managed globally to represent the bottom of the new heap, which represents the bottom of the new heap.
- A global pool is prepared to hold areas to be scanned.

Load Balancing

Initially, each processor copies cells to which his own GC root pointer points. Then each processor scans the unit he allocated. If the two pointers *S* and *B* of the same size are apart farther than the size of load distribution unit (**LDU** in short), the region between two pointers (region to be scanned later) are pushed into global pool. Some processor may turn to be idle where all *S* pointers are reached to *B* pointers. Then idle processor takes out an regions not scanned yet from global pool, which means that load distribution is performed between processors who have many units to be scanned and processors who does not.

*Mita Kokusai Bldg. 21F, 1-4-28, Mita, Minato-ku, Tokyo 108
E-mail: imai@icot.or.jp

Benchmark	average workload	Speedup Size of LDU 256w	64w	32w	ideal
BUP	32K	1.84	3.08	3.23	3.95
MasterMind	39K	2.57	2.65	2.72	2.97
Qlay8	24K	3.97	6.08	6.77	8.00
Queen9	9K	2.50	2.67	2.74	8.00
SemiGroup	56K	3.63	1.88	2.86	8.00
Zebra	179K	6.67	6.80	6.80	8.00
BestPath	121K	5.03	7.08	6.91	8.00

Table 1: Average Workload and Speedup

Termination

GC will be terminated when all pointers of all size / all processors are equal and the global pool is empty.

3 Evaluation

To evaluate load balancing, we defined **Workload** (of a processor) and **Speedup** (of the system) as follows.

$$\text{Workload} = (\#\text{cells copied}) + (\#\text{cells scanned})$$

$$\text{Speedup} = \frac{\sum \text{Workload}}{max(\text{Workload})}$$

We measured these values on the PIM emulator (VPIM) which runs in parallel on Sequent Symmetry using 8 processors, changing the size of LDU (HEU is fixed to 256 words).

Workload is a value which approximates to GC time, and Speedup is based on the idea that a processor of maximum Workload determines GC time[1].

We also calculate "Ideal Speedup" of the program, which is defined

$$\text{Ideal Speedup} = min(\frac{\sum \text{Workload}}{max(\text{Workload for one structure})}, \#\text{processors})$$

Table1 shows the avarage Workload and Speedup.

4 Discussion

Global Pool Usage

In case of Bestpath, Workload for per processor is 15.1K. Avarage number of pushes/pops per processor is 48. That is, a processor pushes into/pop from global pool every 315 workload. This means that global page pool access rarely conflicts. Other benchmark displayed almost same result.

Problems

- In some benchmarks(BUP, MasterMind), Ideal Speedup is limted (2–3).

⇒ Since our system provides separate compilation facility, real application programs consist of many modules. This problem is peculiar to toy programs.

- In some benchmarks (Queen9, SemiGroup), Speedup is much less than Ideal Speedup.

⇒ The benchmarks makes very long flat lists. When copying a long flat list, S and B proceed at the same speed. That is why our load distribution mechanism does not work well. Some other load distribution mechanism should be invented.

5 Conclusion

We proposed a parallel copying GC based on Baker's sequential algorithm. In large application programs such as Zebra or BestPath, almost ideal speedup has been achieved with little overhead. We shall evaluate this GC scheme on PIM in the future.

References

[1] H. G. Baker, "List processing in real time on a serial computer", Comm. ACM, Vol.21, No.4, pp 280-294, 1978.

[1] The value *Speedup* represents just how load balancing is performed and the overhead of load balancing is not taken into consideration.