プログラミング 8−11
(1996. 8. 28)

並列分散プログラミング言語 Sushi のアノテーションによる負荷分散の評価

菅野博靖　山中英樹
{suga,yamanaka}@iias.flab.fujitsu.co.jp
(株) 富士通研究所　情報社会科学研究所
〒261 千葉市美浜区中瀬1−9−3

並列分散プログラミング言語 Sushi (Smartly User Schedulable HIgh-level Language) のアノテーション機構を用いた負荷分散の効果について，N-体問題を解く Barnes-Hut アルゴリズムを題材として行った評価の結果について報告する．Sushi は，ワークステーションクラスタ上において動的な資源管理をユーザに扱い易く提供することを目的として設計された言語であり，簡単なアノテーションの付加によってワークステーションクラスタ上でのプロセスの分散による性能向上が実現できることが確認できた．しかし，同時に言語デザインと実装の両面からの問題も明確になりつつある．

An Evaluation of Parallel/Distributed Programming Language Sushi

Hiroyasu Sugano　Hideki Yamanaka
{suga,yamanaka}@iias.flab.fujitsu.co.jp
Institute for Social Information Science,
FUJITSU LABORATORIES LTD.
1-9-3 Nakase, Mihama-ku, Chiba 261 Japan.

We developed a parallel/distributed programming language Sushi (Smartly User Schedulable HIgh-level Language) with the programmable annotation designed to provide a flexible resource management facility on Workstation clusters. In this paper, we report an evaluation of current Sushi specification and implementation by applying it to well known parallel programming benchmark Barnes-Hut method, one of most important hierarchical N-body algorithms. While attaching a simple annotation can realize a performance improvement by process distribution among the clusters, we found some problems on Sushi language and implementation.

1 Introduction

Developing parallel/distributed software is a hard task not only for novice programmers but even for experts in parallel programming. The reason is that there is no established methodology for resource management to utilize parallel/distributed computing environments effectively. In a conventional style, programmers writing parallel/distributed programs have to specify the suitable strategy for resource allocation in the program itself in a try-and-error manner. While parallel computers and workstation clusters on LAN are getting very popular these days, those possibly high-performance computing environment are not often effectively exploited. It is an urgent demand to develope an easy-to-use parallel/distributed computing environment.

Parallel/distributed programming language Sushi (Smartly User Schedulable HIgh-level Language) [5, 3] has been developed for meeting such needs. One of the novel feature of Sushi is a separation of the computation part, in which problem specific algorithm is described, and the annotation part, which is programmable annotations specifying resource management. Annotations are provided as a library in Sushi distribution, so users can use those built-in annotations or default annotations to exploit the computing and network resource effectively without much difficulty. Moreover, performance-concious programmers or experts for parallel programming can write best-fit annotations for problems they actually try to solve.

In this paper, we report an evaluation of current Sushi specification and implementation by applying it to well known parallel programming benchmark Barnes-Hut method[1, 2], one of most important hierarchical N-body algorithms. The Barnes-Hut method is widely used as an benchmark of shared-memory multiprocessor machines. The reason is in its characteristics; it deals with dynamically changing system of particles or bodies, and the communication pattern of the program depends on the distribution pattern of particles, and thus is quite unstructured. Most of the computing time is spent in the force computing of each particle, which is actually traverse of the hierarchical tree once per particle.

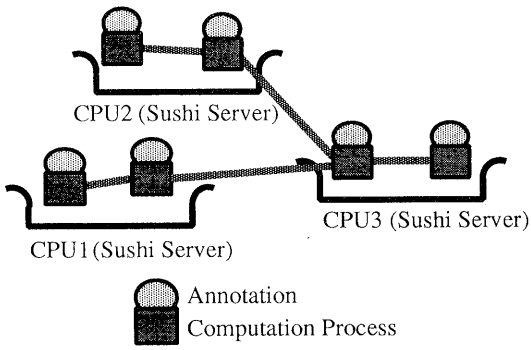

Figure 1: Sushi processes

The language Sushi is implemented on clusters of workstations with Solaris 2. 4 and SunOS 4. 1. 3 operating systems, and it provodes network transparent streams and arrays as communition facility. In our experiments, we built two models of Barnes-Hut algorithm based on different communication structures, one is based on shared array and the other is based on stream communication. As a load balancing strategy, we tried simple process distribution, which migrates particle processes at random, and CPU-based process distribution, which migrates particle manager processes one for each CPU. While the results we have at present are still intermediate, we found some problems on language design and implementation of the current version of Sushi.

After presenting a brief overview of the language Sushi in the next section, in Section 3, we present the characteristics of hierarchical N-body method Barnes-Hut. In Section 4, we show how the annotations of Sushi can be utilized to realize the load-balancing of the algorithm and give preliminary evaluation. In Section 5, we make a discussion on the current research status of Sushi, and conclude in Section 6.

2 Overview of Programming Language *Sushi*

Parallel/distributed programming language Sushi is designed to effectively utilize distributed memory parallel computers and the workstation cluster on a local network. One of

```
/* {"hostname", SPEC-INT} */
CPU[]={{"olive",88},{"purple",88},{"ivory",88},
       {"cyan",40},{"magenta",60}};

/* {"networkname", bandwidth (Mbps)} */
NET[]={{"fddi0",100},{"ether1",10}};

/* {"absolute-lib-path",...} */
ANN[]={"/usr/local/lib/sushi/ann/cooperate.so",
       "/usr/local/lib/sushi/ann/another.so"};
```

Figure 2: Example of Configuration file

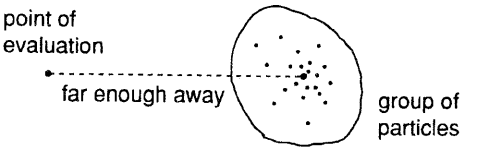

Figure 3: Approximation of particle group

the novel features of Sushi is a separation of the computation processes, in which problem specific algorithm is described, and the annotation, which is programmable annotations specifying resource management such as process migration under the dynamic control of annotations. It appears that such dynamic and programmable annotations can express a variety of balancing strategies which are beyond the reach of static compiler-based methods.

The computation processes (or Sushi processes) is actually a set of concurrent sequential processes communicating each other via streams and associative arrays, the former realizes asynchronous inter-process communication. Sushi processes can be spawned by other Sushi processes explicitly, and each Sushi process can be attached with one annotation at the time of the creation. The figure 1 shows pairs of computation processes and annotations are distributed on some CPUs.

Annotations provide facilities to make an access to information managed by the Sushi runtime system and the operating system. They are also used to modify the location and priority of the procees exection. Table 1 shows some of useful system variables and functions available in the annotations of the current version of Sushi. They can be used to know the CPU load, communication traffic, the current computer and network environment. The allow us to specify the process priority and dynamic process migration.

Annotations are usually exploited as shared libraries independent with Sushi programs. Users can bind annotation library among several libraries at the execution time, and this will make performance tuning effective. The binding of annotation library can be specified by a configuration file, which is used to give global or physical information to Sushi run time system. The Sushi system reads the configuration file at the execution time and determins available CPUs and annotation libraries. Figure 2 shows an example of a configuration file `sushi.conf`. `CPU[]` is an array of pairs of host names and its SPEC-INT value.

The advantage of the program structure of Sushi can be summarized as follows. Software developers can concentrate on the problem to solve itself on the one hand, and performance tuning depending on computing environment on the other hand. This will reduce the complicatedness of parallel/distributed programming, make parallel/distributed programs tractable, and increase portability and reusability. But, as those advantages are somewhat subjective, we have to make an evaluation of Sushi language based on some benchmarks.

3 Hierarchical N-body method Barnes-Hut

Classical N-body problem simulates the evolution of systems composed of bodies or particles under the influence of Newtonian gravitation. As it can be apllied to many problems in physical domain, it is one of the most important scientific computation. If all pairwise forces are computated directly for a system with n particles, its time complexity is $O(n^2)$. The hierarchical method reduces it to $O(n \log n)$.

The principle of Barnes-Hut method is very simple. It assumes that the effect of the group of particles may be approximated by that of single equivalent particle if the group is far enough away from the point at which the effect is being evaluated (Figure 3). Barnes-Hut method divides 3 dimensional space into 8 subspaces recursively. Thus it builds an octree of subspaces (cells) rooted at the whole space containing all particles (Figure 4).

System Variables	MY_CPU	Current CPU ID
	MY_PID	Process ID of the process
System Functions	spawn_at(CPU)	spawn the procees at CPU
	migrate(CPU)	migrate the process to CPU
	set_priority(PRI)	modify the process priority to PRI
	neighbours()	array of neighbouring CPU in the network
	load(CPU)	the load of CPU
	traffic(CPU)	network traffic between CPU the current cpu
	get_cpu(PID)	CPU where the process PID is

Table 1: System variables and functions

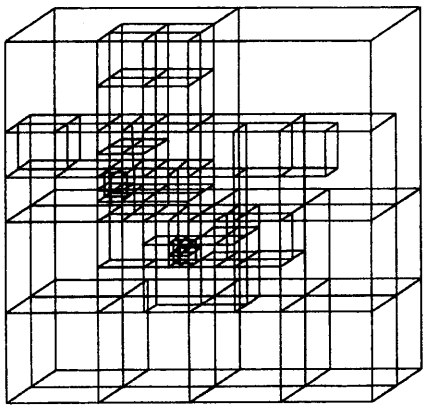

Figure 4: Hierarchy of cells

Barnes-Hut method consists of four phases: (1) building an octree of cells, (2) calculating center of mass of each cell, (3) computing force on each particle, (4) updating the position and the velocity of each particle. Here, we just simply call (1) and (2) *cell hierarchy building* phase, and (3) and (4) *force computing* phase. The octree is traversed once for each particle to calculate the force acting on the particle. If the center of mass of a cell is far enough away from the particle, the effect caused by all the particles contained in the cell are approximated by that of the cell's center of mass. A cell's center of mass is determined to be far enough away from the particle if the following inequation is satisfied:

$$l/d < \theta,$$

where l is the length of a side of the cell, d is a distance of the cell's center of mass from the particle, and θ is a user defined accuracy parameter, which is usually between 0.4 and 1.2.

The characteristics of Barnes-Hut algorithm is that the distribution patterns of particles are nonuniform and dynamically changing. As the communication patterns are dependent on the distribution, it is quit unstructured. This is the reason Barnes-Hut method is adopted by a number of benchmark test of shared memory parallel computers [4].

4 Parallelizing Barnes-Hut in Sushi

Parallelization of Barnes-Hut algorithm is mainly exploited across particle force computation and cell hierarchy building. As the scalability of this algorithm is based on the number of particles, we adopt a parallelization of force computating phase here. Each particle can be realized by Sushi process communicating with the particle manager and they are distributed among several CPUs.

Sushi programming language realizes network transparent streams and arrays among processes distributed on the cluster. So, we have two directions to implement Barnes-Hut communication structure in Sushi: array-based communication and stream-based communication. Actual communication structures we tried are two different models based on those extreams.

The main components of Barnes-Hut in Sushi are the particle manager and the cell hierarchy manager syncronizing the phases of cell hierarchy building and force computing. As those phases come in turns and they must have barrier syncronization, these two component processes should be on the same machine. Our two models differ in the communication style of particle processes and cell building processes via manager processes.

- Shared array model
  Communications among processes are mostly done via shared arrays. All particle processes and cell building processes share the same arrays for communication.

- Stream model
  There are no shared array for communication between particle processes and cell building processes.

Another axis on our experimentation is strategy of process distribution with annotations. We choose the following two strategies in this paper: a) simple distribution, and b) CPU-based distribution. Simple distribution strategy is to exploit the possible parallelism in particle force computing phase simply by migrating particle processes to CPUs whose load are relatively less. CPU-based distribution is a bit more intelligent strategy, which is to copy particle managers to CPUs to decrease the bottleneck at the manager. The former uses the annotation `locate_if_possible` and the latter uses the annotation `div_conq` shown in Figure 5.

At the time of writing, we only have intermediate results on the evaluation with Barnes-Hut, and we can show the case of simple distribution here. Figure 6 shows relative speedup to the number of CPUs in the case of simple distribution in the shared array and the stream model. What we can see from the graphs and other data is summarized as follows;

1. We have only limited performance improvement for both cases. In the case of stream model, overall performance gets worse when the number of CPU is 4 and 5.

2. Stream communication between different machines may cause significant overhead.

3. We use stream-as-data for broadcasting for both cases. We found that this may be one major cause of overhead.

Note: we used 1 to 6 SparcStations with Solaris 2.4 operating system connected by a CDDI network. Sushi processes are realized as Solaris threads with Sushi intermediate code interpreters. The data are generated at random in advance, and we used 100 particles.

```
annotation locate_if_possible(arg) {
  init : {
    if (load(arg % $MAX_CPU)<2000)
       spawn_at($MY_CPU);
    else spawn_at(arg % $MAX_CPU);
  }

  if (load($MY_CPU)>10000)
    for (i in $neighbour[])
       if (load($neighbour[i])<5000) {
           migrate($neighbour[i]);
          break;
       }
}

const THRESHOLD=5;
annotation div_conq(arg) {
  init : {
    if (arg>0) threshold=arg;
    else threshold=THRESHOLD;
    des=n_descendant();
    sib=n_sibling();
    if (des < threshold ) {
       spawn_at((des+sib)%$MAX_CPU);
    }
  }

  if (load($MY_CPU)>5000*cpu_speed($MY_CPU)/88)
    for (i in $neighbour[])
      if ( load($neighbour[i]) <
                1000*cpu_speed($MY_CPU)/88 ) {
        migrate($neighbour[i]);
        break;
      }
}
```

Figure 5: Annotations

5 Discussions

Although the language Sushi is designed to provide a useful framework on parallel/distributed programming, Barnes-Hut method is our first trial to implement a scientific computation algorithm in Sushi. The overall performance is not high compared with other parallel programming languages bacause of the current implementation based on the interpreter. While the performance improvement is not sufficient either, this improvement is realized by simply attaching annotations at one line of the original program. This is the partial realization of the facility for tractable parallel/distributed programming, one of our motivations.

However, we are not satisfied with the current results at all in the following reasons. One is that the performance improvement is too low. One of the reasons will be partly in the current communication overhead in the network and operating system and partly in the program struc-

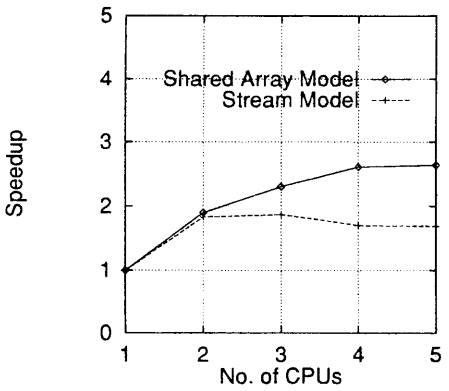

Figure 6: Speedup in Simple Distribution

ture which give overhead to manager processes. To make the problem clear, we have to analyze the acquired data more thoroughly. We will continue our experiment on Barnes-Hut making our communication model sophisticated.

The other point is in annotation design which we found is very restrictive. We would sometimes like to write a communication between computation processes and annotations, but it is quite restricted at present. We are now reconsidering our annotation design to get more flexibility.

6 Concluding remarks

In this paper, we report intermediate result of an evaluation of current Sushi specification and implementation by applying it to well known parallel programming benchmark Barnes-Hut method[2], one of most important hierarchical N-body algorithms. While we do not have sufficient result, we found several problems of current Sushi language and implementation.

We expect that annotation style load balancing can provide an flexible resource management on parallel/distributed computing environment. It will have portability and reusability as well. To make the problems clearer and improve Sushi language, we are continuing its evaluation from both of language design and performance point of view.

As another large scale problem, we are now implementing the scalable communication server on the Internet cyberspace application with Sushi. This is designed to serve large amount of participants communicating each other. Such an Internet application is a good example of Sushi's scalable parallel/distribution programming. We are also examining building Sushi system on Fujitsu AP3000.

Acknowledgement

The authors would like to thank Youji Kohda, Haruyasu Ueda, and Takanori Ugai for their fruitful discussions and helpful comments.

References

[1] J. E. Barnes and P. Hut. A hierarchical $O(N \log N)$ force calculation algorithm. *Nature*, 324(4):446–449, 1986.

[2] J. P. Singh, J. L. Hennessy, and A. Gupta. Implications of hierarchical N-body methods for multiprocessor architectures. *ACM Transactions on Computer Systems*, 13(2):141–202, May 1995.

[3] H. Sugano, H. Yamanaka, and T. Ugai. Annotation programming in Sushi. *IPSJ 95-PRO-2 (in Japanese)*, 95(2):113–120, August 1995.

[4] S. C. Woo, M. Ohara, E. Torrie, J. P. Signh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerarions. In *Proc. 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

[5] H. Yamanaka, T. Ugai, H. Ueda, H. Sugano, and Y. Wada. Sushi - a Parallel and Distributed Programming Language with Voluntary Process Migration - (in Japanese). In *Proc. of 49th Conference of IPSJ*, volume 4, pages 319–320. IPSJ, September 1994.