# JUMP DSM Installation Guide (V1.0 written 16 Aug 2001)

*The Department of Computer Science and Information Systems,*
*The University of Hong Kong*

## 0 Contents



## 1 Introduction

JUMP is a software DSM system written in C code, and appears as a user-level C language UNIX library. It is modified from JIAJIA V1.1, and makes use of the unique migrating-home protocol for maintaining Scope Consistency. Experiments show that under most cases, JUMP runs better than the home-based protocol used by JIAJIA V1.1.

## 2 Prerequisites in Running JUMP DSM

Here are the software and hardware environment needed to run JESSICA 1.0:

- A cluster of maximum 16 network-connected PCs or workstations
- We recommend each PC to have 128MB of main memory or more
- Linux OS version 2.2.x or above (we have tested the DSM system in kernel versions 2.2.1, 2.2.12, 2.2.14 as well as 2.4.2).
- We strongly recommend you to use NFS (especially the latter) in running JUMP, which saves a lot of effort in program execution.
- The package: **`jumpdsm-1.0.tgz`**

## 3 Installing JUMP

It is piece of cake to install the JUMP DSM system. Just unzip the **`jumpdsm-1.0.tgz`** archive by typing **`tar xvfz jumpdsm-1.0.tgz`** in your root directory (**~**). You will find all the needed files in the directory named **`dsm`**. Then, go to the directory **`dsm/jumplib`** and type **`make`**. The library file **`libjia.a`** will be formed, and this is the software library that you need to link when compiling JUMP DSM application programs (see next section for details).

You can also add two optional flags in the **Makefile** in the **dsm/jumplib** directory. The first one is **-DDOSTAT**, and the other one is **-JT**. They provide two different levels of statistics gathered during the application execution, and will be printed out when the application successfully exits. The former flag can provide coarser-grain program statistics, such as the number of locks, unlocks and barriers, the number of page faults and so on. The latter flag provides much finer-grain statistics, which is presented as the time taken in executing each internal DSM function. These statistics are useful for analytical purpose, so as to fine-tune the DSM system. However, gathering these statistics during program execution takes time, and can slow down the program. Hence, most normal users will not need to add these flags.

# 4 Running JUMP Application Programs

We include a number of JUMP application programs for users to try out. They include:

- **mm** – matrix multiplication of two 512x512 integral matrices
- **merge** – merge sort of 2097152 integers
- **radix** – radix sort of 2097152 integers using 4 processors
- **lu** – LU Factorization of a 512x512 integral matrix
- **buck** – Bucket sort of integers (the number of integers to be sorted can be specified in the command line argument. For example, **buck c** stands for sorting 1048576 integers.)

The program sources are put in the **dsm/progsrc** directory. We also include a script called compile in order for you to compile the programs. Simply type **compile [program_name]** to compile the program, and the executable will be located in your root directory **~**.

To execute the program, first you need to edit the file ~/.jiahosts, which a copy has been included when you unzip the archive. You need to edit the file as follows:

```
machine_1       username     dummy_string
machine_2       username     dummy_string
machine_3       username     dummy_string
machine_n       username     dummy_string
```

**machine_1, machine_2, …, machine_n** are *n distinct* machine names in the cluster you are running the application. If you are including *n* machines, all these machines will be used in executing the program. (Therefore, please only specify 4 machines if you want to run the radix sort program provided. For the other sample programs, 2, 4, 8 or 16 machines should work fine.) Moreover, **machine_1** should be the same machine as the one you are going to invoke the program. **username** is the login name of your user account, and **dummy_string** is any non-null string (this field is intended to put in the password in JIAJIA, but it is not implemented.)

Moreover, you also need the **.rhosts** file, so that the **rsh** mechanism used in JUMP can work properly. The **.rhosts** file takes the following format:

```
machine_1       username
machine_2       username
machine_3       username
machine_n       username
```

The **machine_i** and **username** take the same definition as that in the **.jiahosts** file. Next, type **chmod 0600 .rhosts** to set the access right permission of this **.rhosts** file to 0600.

If you are not using NFS, you need to copy the application executable, the `.jiahosts` file and the `.rhosts` file to the root directory of every remote machine which will run the application. Using rcp should be the most convenient way, with the simple command:

```
rcp filename remote_machine_name:~/filename
```

Make sure the access rights of the remote copies do not change. Then, you are able to run the application in `machine_1`. Just type the executable file name and here you go!

# 5 Points to Note when Writing your Own JUMP Application

You may want to write your own JUMP application program. Here we do not intend to give a detailed tutorial on how to write DSM applications (Interested programmers may refer to other related documents or books on DSM programming, and, studying the code of our provided sample programs certainly help a lot). However, we just point out certain things you need to beware of when you write a JUMP application program.

First of all, we strongly recommend you to put your source file into the `dsm/progsrc` directory. Of course, you may not need to follow suit. However, if you are not doing so, you have to be very careful when you include the `jia.h` header file. You need to specify the correct path where this header file can be found (by default, it is in the `~/dsm/jumpsrc` directory). You must include this file with the correct path in your source program. Otherwise the compilation will fail.

Secondly, unlike JIAJIA V1.1 which provides locks (acquire and release), barriers and condition variables, JUMP only provides the former two synchronization facilities. The lock acquire operation is `jia_lock(lock id)`, and the release operation being `jia_unlock(lock id)`. The `lock_id` can range from 0 to 1023, meaning that JUMP provides 1024 locks. Notice that, unlike TreadMarks which makes use of Lazy Release Consistency, JUMP makes use of Scope Consistency, which *only* propagates the updates made within the same lock from one processor to another when the latter acquires that particular lock. Therefore, the lock numbers do matter the correctness of your program. We recommend that throughout the program execution, every access to the same variable should be guarded by the same lock. In this case, the "expected" behaviour of the program can be assured. For barriers, the operation is `jia_barrier()`. There is no arguments needed, since JUMP only has 1 global barrier.

Thirdly, you can also make use of the `jia_clock()` operation for timing your program. By calling `jia_clock()` once at the beginning of the program and once at the end, users are able to time the application execution.

Finally, let me remind you that for each application, you must call `jia_startup()` at the beginning to invoke the initialization of JUMP. Next, you need to call `jia_alloc()` to allocate shared memory space before use. And, at the end of the program, it is a good practice to call `jia_exit()`, so that JUMP is able to do the clean-up and release the resources held.

# 6 Contacts

We shall be grateful if you can find out any bugs about our system. Please send the bug reports to: *wlcheung@csis.hku.hk*

# 7 Appendix – Application Programming Interface of JUMP

(This part is copied and modified from the Programming manual of JIAJIA V1.1)

- **`jia_init(argc, argv)`** – initializes JUMP. It should be called at the beginning of the application. The main task of **`jia_init()`** is to start copies of the application on the hosts specified in the .jiahosts file. Also, **`jia_init()`** initializes the internal data structures of JUMP.

- **`jia_alloc(int size)`** – allocates shared memory. The parameter size indicates the number of bytes allocated. It will be rounded up to the nearest page size. The pages will be allocated in a round-robin fashion. The function returns a pointer which points to the head of the chunk of memory allocated.

- **`jia_lock(int lockid), jia_unlock(int lockid)`** – acquires and releases a lock specified by **`lockid`** (which can range from 0 to 1023 inclusive). These lock operations provide a synchronization mechanism to ensure exclusive access to a critical section. **`jia_lock(), jia_unlock()`** should appear in pairs.

- **`jia_barrier()`** – performs a global barrier. A barrier provides a global synchronization mechanism by preventing any process from proceeding until all processes reach the barrier. Unlike JIAJIA, JUMP allows you to call a barrier between lock/unlock pairs. The barrier will automatically release the locks a process holds before issuing the barrier, and re-acquiring the locks afterwards.

- **`jia_exit()`** – shuts down JUMP.

- **`jia_clock()`** – returns the elapsed time since the start of application in seconds in **`float`** type. It can be used to summarize the time expended by the application.

- **`jia_error(char *str)`** – prints out the error string **`str`** and shuts down all processes started by **`jia_init()`**.