BMDFM

Comprehensive Manual

2015

Oleksandr Pochayevets

This page is intentionally left blank.
Contents of the document begins on the next page.

Table of Contents

Table of Contents (cont.)

List of Figures

List of Figures (cont.)

About This Manual

This manual gives an overview of the BMDFM product. The BMDFM architecture, installation procedure and programming issues are described.

This page is intentionally left blank.

Chapter 1
Introduction

What is BMDFM?

BMDFM (Binary Modular DataFlow Machine) is software, which enables running an application in parallel on shared memory symmetric multiprocessors (SMP) using the multiple processor cores to speed up the execution of single applications.

BMDFM automatically identifies and exploits parallelism due to the static and mainly DYNAMIC SCHEDULING of the data flow instruction sequences derived from the formerly sequential program ensuring unique parallel correctness.
No directives for parallel execution are required!
No highly knowledgeable parallel programmers are required!

What does BMDFM provide to a user?

A user understands BMDFM as a virtual machine, which runs every statement of an application program in parallel having all parallelization and synchronization mechanisms fully transparent. The statements of an application program are normal operators, which any singlethreaded program might consist of - they are variable assignments, conditional executions, loops, function calls, etc. BMDFM has a rich set of standard operators/functions, which can be extended by user functions written in C/C++.

In comparison with the recent general methodology of sequential code parallelization, which is based on static analysis, BMDFM uses dynamic scheduling to define and to run code fragments in parallel. It means that data computed at run time will define further branches for parallel processing

(DataFlow principle). It also means that loops of an application program will be dynamically unrolled to process several iterations in parallel.

Which granularity of parallelism is used in BMDFM?

BMDFM exploits fine-grain parallelism. All instructions of an application will be processed in parallel. In addition, it is possible to exploit coarse-grain parallelism that will decrease costs spent on dynamic scheduling. In order to achieve this a portion of C code can be defined as a user function, which will be treated by the dynamic scheduler as one seamless instruction.

Which platforms may run BMDFM?

Every machine supporting ANSI C and POSIX/SVR4-IPC may run BMDFM.
Obviously, BMDFM is able to accelerate the execution time of an application only when installed on a multiprocessor computer implementing an SMP paradigm (hardware mapping of distributed memory into virtual shared memory, cache coherent non-uniform memory access ccNUMA, UMA!, etc.)

BMDFM is provided as compiled multi-threaded versions for:

* *x86:* **Linux/32, FreeBSD/32, MacOS/32, SunOS/32, UnixWare/32,**
* *x86:* **Win-UWIN/32, Win-SFU/32;**
* *x86-64:* **Linux/64, FreeBSD/64, MacOS/64, SunOS/64;**
* *VAX:* **Ultrix/32;**
* *Alpha:* **Tru64OSF1/64, Linux/64, FreeBSD/64;**
* *IA-64:* **HP-UX/32, HP-UX/64, Linux/64;**
* *XeonPhiMIC:* **Linux/64;**
* *MCSTelbrus:* **Linux/32, Linux/64;**
* *PA-RISC:* **HP-UX/32, HP-UX/64, Linux/32;**
* *SPARC:* **SunOS/32, SunOS/64, Linux/32, Linux/64;**
* *MIPS:* **IRIX/32, IRIX/64, Linux/32, Linux/64;**
* *MIPSel:* **Linux/32, Linux/64;**
* *PowerPC:* **AIX/32, AIX/64, MacOS/32, MacOS/64, Linux/32, Linux/64;**
* *PowerPCle:* **Linux/32, Linux/64;**

* *S390:* **Linux/32, Linux/64;**
* *M68000:* **Linux/32;**
* *ARM:* **Linux/32, Linux/64;**
* *ARMbe:* **Linux/64;**
* and a limited single-threaded version for *x86:* **Win/32**[1].

A machine with one CPU can be used for development and test purposes only as it is not possible to get real acceleration on one CPU. But as soon the application program has reached a certain state of maturity it can be moved to BMDFM running on a wide range of multicore/many-core computers (from tiny embedded devices to multiprocessor big iron mainframes) as shown in Figure 1-1.

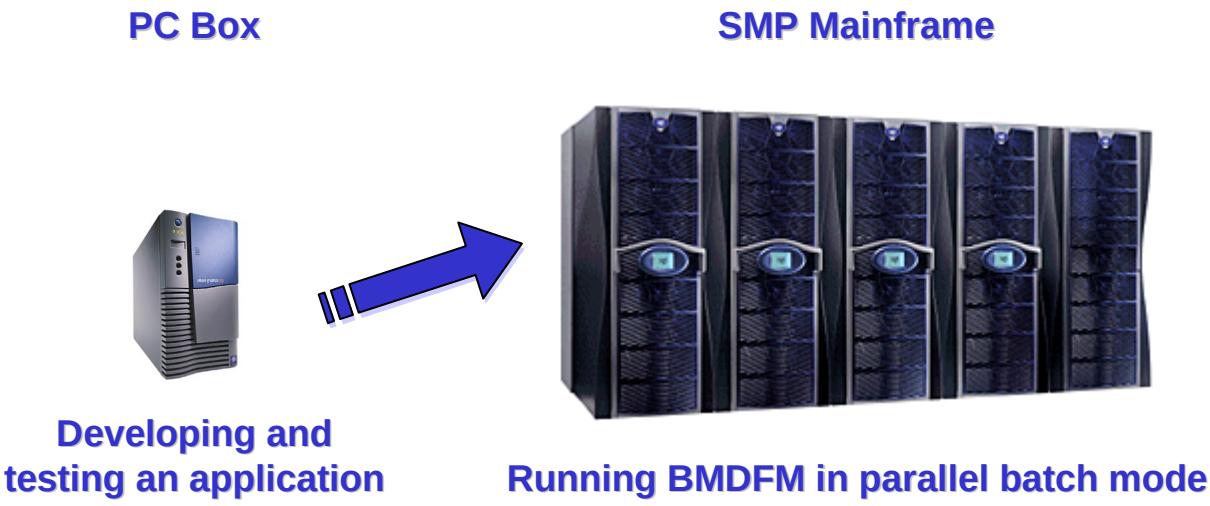

Figure 1-1. Running BMDFM on different machines

BMDFM Official Web Site: ***http://bmdfm.com***

BMDFM Support: email to: ***bmdfm@bmdfm.de***

1. Actual BMDFM installation factory package can provide more platform specific compiled versions.

This page is intentionally left blank.

Chapter 2
Architectural Overview

BMDFM uses both highly efficient dynamic and static scheduling combining SMP (Shared Memory Symmetric Multi Processing), MIMD (Multiple Instruction Stream, Multiple Data Stream) and DFM (DataFlow Machine) paradigms. The BMDFM architecture is shown in Figure 2-1.

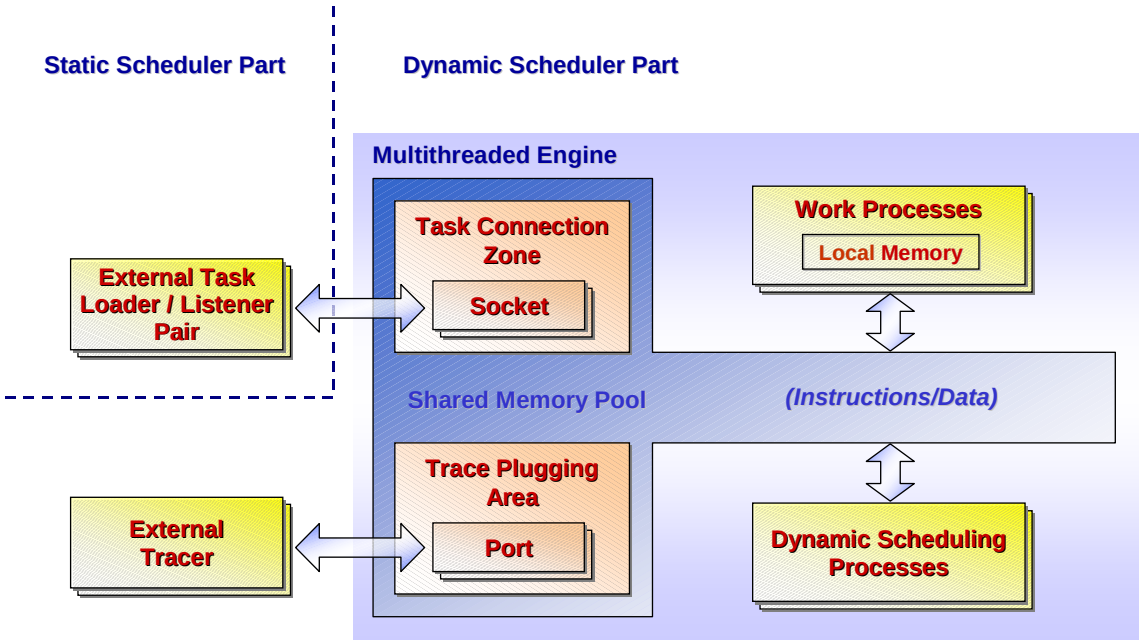

Figure 2-1. BMDFM architecture

A pool of processes is divided into two subsets: Work Processes, which execute parallel instruction streams, and Dynamic Scheduling Processes, which automatically convert sequential instruction streams into parallel ones. **Running under an SMP OS the processes will occupy all available real machine processors.**

All processes share the Shared Memory Pool containing Instructions and Data. Each Work Process also has its own Local Memory, which may contain user subroutines to implement additional coarse-grain levels of parallelization. The External Loader/Listener Pair performs preprocessing and static scheduling of the input program instructions and stores them clustered in the Task Connection Zone. The Listener is responsible for the ordered output after the out-of-order processing in the Multithreaded Engine. Clustered instructions and data are fetched by the Dynamic Scheduling Processes into the Shared Memory Pool. Additionally, Dynamic Scheduling Processes release (garbage collect) resources after the data contexts and speculative branches are processed. Lastly, the External Tracers assist in debugging of the multithreaded out-of-order processing of the input program. The External Tracers are connected via the Ports of the Trace Plugging Area. The Tracer can operate in various modes of full/partial and master/slave debugging.

2.1 Static Scheduler

Figure 2-2 shows the static scheduling part of BMDFM. An application program (Input Sequential Program) is processed in three stages: preliminary code reorganization (Code Reorganizer), static scheduling of the statements (Static Scheduler) and compiling/loading (Compiler). The output after the static scheduling stages is a **Multiple Clusters Flow** that feeds the Multithreaded Engine via the Interface designed in a way to avoid bottlenecks. For some special cases, such as development of loaders, the Interface can be published. Multiple Clusters Flow can be understood as a compiled input program split on the marshaled clusters, in which all addresses are resolved and extended with context information. Splitting on the marshaled clusters enables loading them multi-threadedly. Context information lets iterations be processed in parallel.

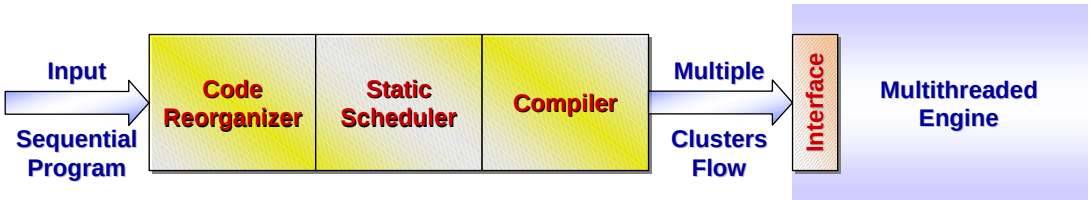

Figure 2-2. Architecture of static scheduler

2.2 Dynamic Scheduler

Figure 2-3 shows the part of BMDFM responsible for the dynamic scheduling in more detail. The BMDFM dynamic scheduling subsystem performs an efficient **SMP emulation of the Tagged-Token Dataflow Machine** as described below.

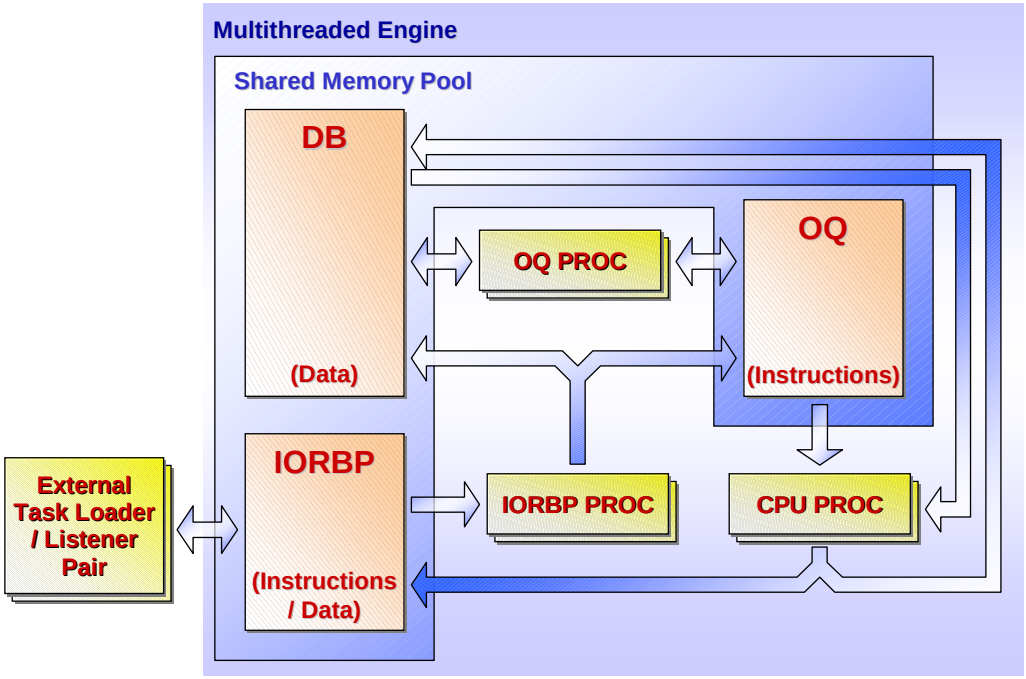

Figure 2-3. Architecture of dynamic scheduler

The Shared Memory Pool is divided in three main parts: Input/Output Ring Buffer Port (IORBP), Data Buffer (DB) and Operation Queue (OQ).

The external static scheduler (External Task Loader/Listener Pair) puts clustered instructions and data of an input program into the IORBP. The ring buffer service processes (IORBP PROC) move data into the DB and instructions into the OQ. The operation queue service processes (OQ PROC) tag the instructions as ready for execution if the required operands' data is accessible. The execution processes (CPU PROC) execute instructions, which are tagged as ready and output computed data into the DB or to the IORBP. Additionally, IORBP PROC and OQ PROC are responsible for freeing memory after contexts have been processed. The context is a special unique identifier representing a copy of data within

different iteration bodies. This allows the dynamic scheduler to handle several iterations in parallel.

In order to allow several processes accessing the same data concurrently, BMDFM locks objects in the Shared Memory Pool via POSIX/SVR4 semaphore operations. Locking policy provides multiple read-only access and exclusive access for modification.

2.3 Configuration

Caution

Please, ensure suitable amount of SVR4 IPC resources when running BMDFM (note that BMDFM can be configured with POSIX/SVR4-IPC-synchronization).

Example for SunOS:

```
/etc/system
set shmsys:shminfo_shmmax=2147483647
set semsys:seminfo_semmni=128
set semsys:seminfo_semmsl=256
set semsys:seminfo_semmns=32768
```

Configuration parameters are gathered in the BMDFM configuration profile. The configuration parameters are explained below:

SHMEM_POOL_SIZE defines maximal shared memory pool size. Bigger values ensure that BMDFM will operate for larger amounts of data. Note that 2147483647 (2GB) shmmax value is normally a limit for 32-bit mode. Run 64-bit BMDFM that allows you to configure more shared memory space.

SHMEM_POOL_MNTADDR defines explicit mounting address of the shared memory pool. By default, the mounting address is chosen by the BMDFM Server and the OS automatically.

SHMEM_POOL_PERMS defines permissions of the shared memory pool in a form of traditional Unix-like and otherwise POSIX-compliant system permissions (e.g. decimal 432 is equal to octal 0660 and means "rw-rw----").

SHMEM_POOL_BANKS defines the number of banks in shared memory pool. Several banks together work faster than one, however, the memory bank restricts maximal memory block size that can be allocated.

POSIX_SEMA4_SYNC defines whether POSIX semaphores should be used instead of SVR4 semaphores.

ARRAYBLOCK_SIZE defines the policy of the memory allocation. Memory is allocated in chunks. Bigger values cause less intensive and faster memory allocation, however at the same time, cause more inefficient memory usage.

OQ_FUNC_ARG_COUNT defines the default number of the function arguments statically allocated in the OQ. In case where the actual number of arguments exceeds, they will be allocated dynamically.

Q_OQ defines OQ size. Bigger values allow the running of tasks with more complex data dependencies, however, a big OQ requires additional memory space, an additional number of semaphores and can slow down associative searches in the dynamic scheduling subsystem.

Q_DB defines DB size. Bigger values allow running tasks with more variables, however, a big DB requires additional memory space and an additional number of semaphores.

Q_IORBP defines IORBP size. Bigger values allow more intensive loading of data via the Task Connection Zone, however, a big IORBP requires additional memory space and an additional number of semaphores.

N_IORBP defines the number of IORBPs, thus the number of tasks, which can be processed in parallel. Processing several tasks simultaneously uses system resources more efficiently.

N_TRACEPORT defines how many tracers can be attached at the same time. Bigger values allow one to have more tracers working in separate windows displaying different resources.

N_CPUPROC defines the number of CPUPROC processes. Usually, it makes sense to set this value equal or doubled to the number of system logical processors (processors * cores * threads_per_core). An additional tuning can be done after analysis of STALL WARNINGS in the generated log files.

N_OQPROC defines the number of OQPROC processes. Usually, it makes sense to set this value equal or doubled to the number of system logical processors (processors * cores * threads_per_core). An additional tuning can be done after analysis of STALL WARNINGS in the generated log files.

N_IORBPPROC defines the number of IORBPROC processes. Usually, it makes sense to set this value equal or doubled to the number of system logical processors (processors * cores * threads_per_core). An additional tuning can be done after analysis of STALL WARNINGS in the generated log files.

CPUPROC_MTHREAD specifies multithreading model (if switched on) or multi-process model (if switched off) for the CPUPROC processes.

OQPROC_MTHREAD specifies multithreading model (if switched on) or multi-process model (if switched off) for the OQPROC processes.

IORBPPROC_MTHREAD specifies multithreading model (if switched on) or multi-process model (if switched off) for the IORBPROC processes.

BMDFMLDR_MTHREAD specifies multithreading model (if switched on) or multi-process model (if switched off) for the BMDFMldr static scheduler processes.

MTHREAD_TLS_CHECK switches verification for Thread-Local Storage (TLS) on/off. The verification is done at startup.

ALLOW_CPUPROC_ASLR allows CPUPROC processes to use Address Space Layout Randomization (ASLR) provided by the OS (if switched on).

T_STATISTIC defines time interval between attempts of collecting statistics. Less values make the statistics more precise but increase costs for this.

PROC_HEARTBEATS switches process heartbeats on/off. The heartbeats are sent between CPUPROC, OQPROC and IORBPROC processes in order to detect whether the processes are alive.

DFSTLHAZARD_DETECT switches detection of dataflow stall hazards on/off. All stalled dataflow instructions will be purged after a stall hazard is detected.

ALLOW_DROP_NONPROD allows dropping nonproductive instructions (if switched on). Nonproductive instructions are those that do not influence any execution path for achieving results of an application program (the results of an application program are VM native inputs and outputs of this application program).

PROC_CPU_LOGS switches Data Flow Logging Facility on/off. This facility allows logging the CPUPROC and IORBPROC process activities into log files.

HARD_ARRAY_SYNCHRO ensures correct array processing where the multiple assignments are applied to the same array members. As a rule, the BMDFM system asks to switch this option on if it is necessary.

EXT_IN_OUT_SYNCHRO synchronizes console messages that are generated by the Loader/Listener Pair. If this configuration parameter is switched on, the Loader always waits until the Listener releases console.

OQ_DB_SEM_LIMIT defines the maximal allowed number of SVR4 semaphores in the OS kernel that are owned by the BMDFM instance. By default, no limitation is set. Note that BMDFM can be configured with POSIX/SVR4-IPC-synchronization.

RELAXED_CNSTN_SM_MODEL compensates relaxed consistency model of shared memory (if switched on). The compensation mechanisms are activated by default. It is strongly recommended to keep them activated if the consistency model of SMP machine is not clear enough.

DEFOP configures user-defined functions to be loaded into CPUPROC local memory.

Figure 2-4 shows configuration example for a 1024-way SMP machine.

```
SHMEM_POOL_SIZE     = 10995116277760 # (10TB) Shared memory pool size    [Bytes]
SHMEM_POOL_MNTADDR =               0 # ShMemPool mount address (0=auto)
SHMEM_POOL_PERMS    =             432 # ShMemPool permissions (0660=="rw-rw----")
SHMEM_POOL_BANKS    =             500 # Number of banks in pool
POSIX_SEMA4_SYNC    =       RW+Count # Replace None/RW/RW+Count SVR4 with POSIX sema4
ARRAYBLOCK_SIZE     =              64 # Array block size                  [Entities]
OQ_FUNC_ARG_COUNT   =              32 # OQ functions arguments count      [Entities]

Q_OQ                =           50000 # Operation Queue (OQ) size         [Entities]
Q_DB                =           10000 # Data Buffer (DB) size             [Entities]
Q_IORBP             =            1000 # I/O Ring Buffer Port (IORBP) size [Entities]
N_IORBP             =              10 # Number of the IORBPs
N_TRACEPORT         =               5 # Number of the Trace Ports (TPs)

N_CPUPROC           =            2048 # Number of the CPU PROCs
N_OQPROC            =            2048 # Number of the OQ PROCs
N_IORBPPROC         =            2048 # Number of the IORBP PROCs

CPUPROC_MTHREAD     =              No # CPU PROC is multithreaded
OQPROC_MTHREAD      =              No # OQ PROC is multithreaded
IORBPPROC_MTHREAD   =              No # IORBP PROC is multithreaded
BMDFMLDR_MTHREAD    =              No # BMDFMldr is multithreaded

MTHREAD_TLS_CHECK   =              No # Check for Thread-Local Storage (TLS)
ALLOW_CPUPROC_ASLR =               No # Allow CPU PROC Address Space Layout
                                      # Randomization (ASLR)

T_STATISTIC         =               1 # Time to scan DFM for statistic    [Seconds]
PROC_HEARTBEATS     =             Yes # Heartbeats for the CPU, OQ && IORBP PROCs
DFSTLHAZARD_DETECT  =             Yes # Detection of dataflow stall hazards
ALLOW_DROP_NONPROD  =              No # Allow dropping nonproductive instructions
PROC_CPU_LOGS       =              No # Logs registration for the CPU && IORBP PROCs
HARD_ARRAY_SYNCHRO  =              No # Hard synchronization of the arrays
EXT_IN_OUT_SYNCHRO  =             Yes # I/O synchronization of external task
OQ_DB_SEM_LIMIT     =               0 # Max number of OQ&&DB semaphores (0=unlim.)


DEFOP               =    (defun true  (progn 1))
                         (defun false (progn 0))
                        # (defun ...)
```

Figure 2-4. Configuration example for a 1024-way SMP machine

2.4 Top Screen of the Running System

The ***top*** screen of the running BMDFM is shown in Figure 2-5. It clearly demonstrates that BMDFM is built according to the **MIMD architecture** and has **negligible dynamic scheduling overhead**!

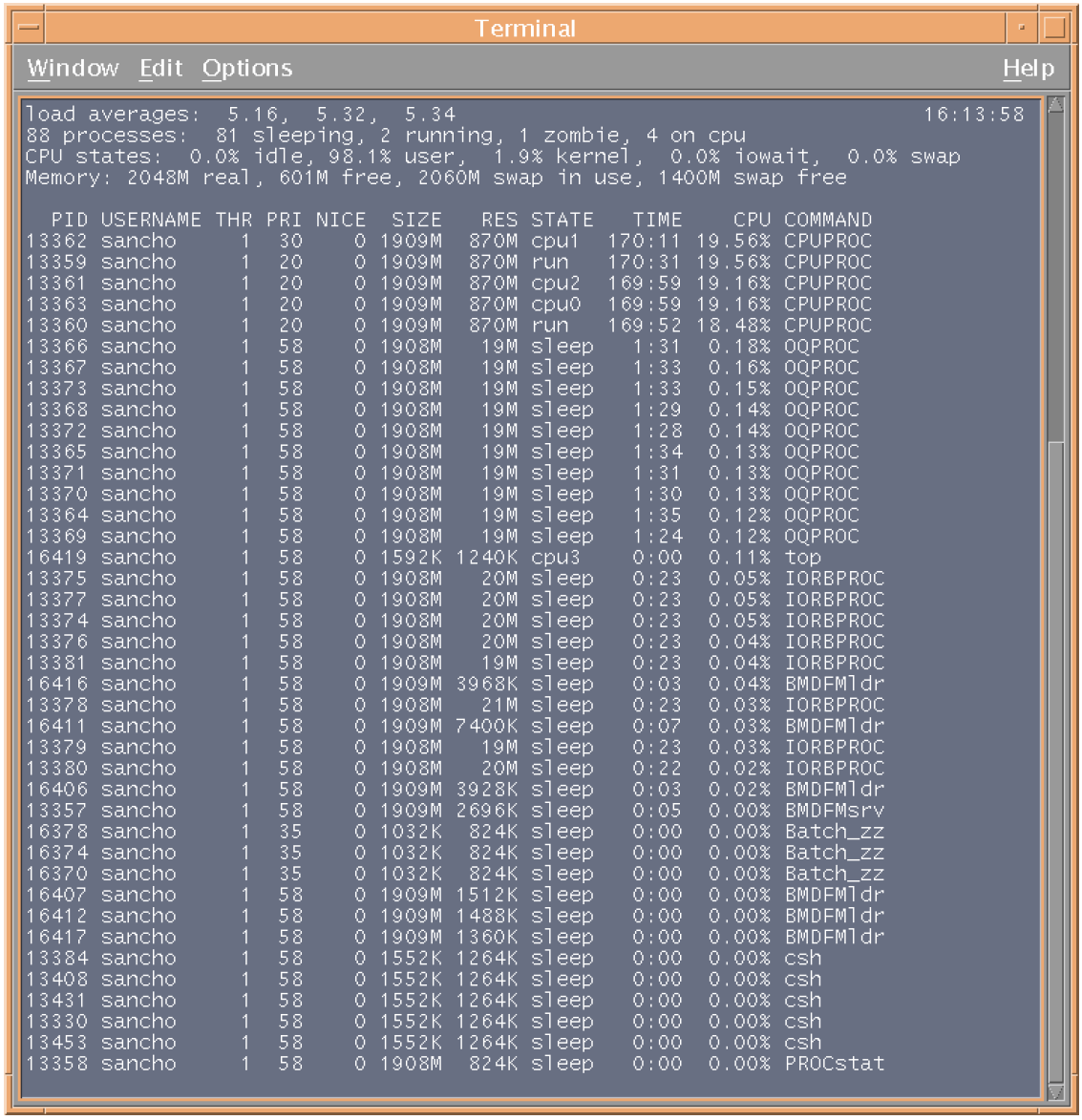

Figure 2-5. BMDFM running on a 4-way SMP machine

This page is intentionally left blank.

Chapter 3
Installation and Use

3.1 Structure of Modules on the Disk

BMDFM modules shown in Figure 3-1. All modules are given as executables and some of them additionally as object files and sources. Recompilation is necessary only if a new application (interface and implementation) is written in C/C++. Singlethreaded version of BMDFM consists of only one module, all the rest belongs to the multithreaded engine. The multithreaded BMDFM starts with the BMDFMsrv server, which automatically starts multiple copies of the daemons (CPUPROC, OQPROC, IORBPROC and PROCstat). The BMDFMldr, BMDFMtrc and freeIPC (as well as the fastlisp and BMDFMsrv themselves) are standalone utilities.

```
cflp_udf.h cflp_udf.c  User applications interface and implementation
Makefile               Build file
fastlisp.o fastlisp    BMDFM singlethreaded engine
BMDFMsrv.o BMDFMsrv    BMDFM multithreaded engine server unit
CPUPROC.o  CPUPROC     CPU PROC daemon
           OQPROC      OQ PROC daemon
           IORBPROC    IORBP PROC daemon
           PROCstat    Daemon that collects statistic information
BMDFMldr.o BMDFMldr    External task unit (Loader/Listener Pair)
           BMDFMtrc    External Tracer unit
           freeIPC     Utility that releases system IPC resource after crash
```

Figure 3-1. BMDFM modules

Normally, the BMDFM files are gathered in a ***$BMDFM_HOME/*** working directory. The directory structure is shown in Figure 3-2. Choose a BMDFM build by linking Bin directory to the right target directory (default link is **Bin -> Build/x86_Linux_32/**), e.g.: ***rm Bin; ln -s Build/x86-64_Linux_64/ Bin***

```
$BMDFM_HOME/
    EULA.txt
    READ_ME.1ST
    Doc/
    Bin/ -> Build/x86_Linux_32/
    Build/
       x86_Win_32/              SPARC_SunOS_32/
       x86_Win-UWIN_32/         SPARC_SunOS_64/
       x86_Win-SFU_32/          SPARC_Linux_32/
       x86_Linux_32/            SPARC_Linux_64/
       x86_FreeBSD_32/          MIPS_IRIX_32/
       x86_MacOS_32/            MIPS_IRIX_64/
       x86_SunOS_32/            MIPS_Linux_32/
       x86_UnixWare_32/         MIPS_Linux_64/
       x86-64_Linux_64/         MIPSel_Linux_32/
       x86-64_FreeBSD_64/       MIPSel_Linux_64/
       x86-64_MacOS_64/         PowerPC_AIX_32/
       x86-64_SunOS_64/         PowerPC_AIX_64/
       VAX_Ultrix_32/           PowerPC_MacOS_32/
       Alpha_Tru64OSF1_64/      PowerPC_MacOS_64/
       Alpha_Linux_64/          PowerPC_Linux_32/
       Alpha_FreeBSD_64/        PowerPC_Linux_64/
       IA-64_HP-UX_32/          PowerPCle_Linux_32/
       IA-64_HP-UX_64/          PowerPCle_Linux_64/
       IA-64_Linux_64/          S390_Linux_32/
       XeonPhiMIC_Linux_64/     S390_Linux_64/
       MCSTelbrus_Linux_32/     M68000_Linux_32/
       MCSTelbrus_Linux_64/     ARM_Linux_32/
       PA-RISC_HP-UX_32/        ARM_Linux_64/
       PA-RISC_HP-UX_64/        ARMbe_Linux_64/
       PA-RISC_Linux_32/        <Arch>_<OS>_<WordWidth>/
           fastlisp     Makefile
           BMDFMsrv     cflp_udf.h
           CPUPROC      cflp_udf.c
           OQPROC       fastlisp.o
           IORBPROC     BMDFMsrv.o
           PROCstat     CPUPROC.o
           BMDFMldr     BMDFMldr.o
           BMDFMtrc     fastlisp.cfg
           freeIPC      BMDFMsrv.cfg
    fastlisp      -> Bin/fastlisp
    BMDFMsrv      -> Bin/BMDFMsrv
    CPUPROC       -> Bin/CPUPROC
    OQPROC        -> Bin/OQPROC
    IORBPROC      -> Bin/IORBPROC
    PROCstat      -> Bin/PROCstat
    BMDFMldr      -> Bin/BMDFMldr
    BMDFMtrc      -> Bin/BMDFMtrc
    freeIPC       -> Bin/freeIPC
    fastlisp.cfg -> Bin/fastlisp.cfg
    BMDFMsrv.cfg -> Bin/BMDFMsrv.cfg
```

Figure 3-2. BMDFM directory tree

3.2 Programming and Compilation

Normally, the life cycle of a BMDFM user application has two major steps as shown in Figure 3-3. At first the application is developed and tested using the BMDFM singlethreaded engine, then if it works properly it can be moved without any modifications to the BMDFM multithreaded engine.

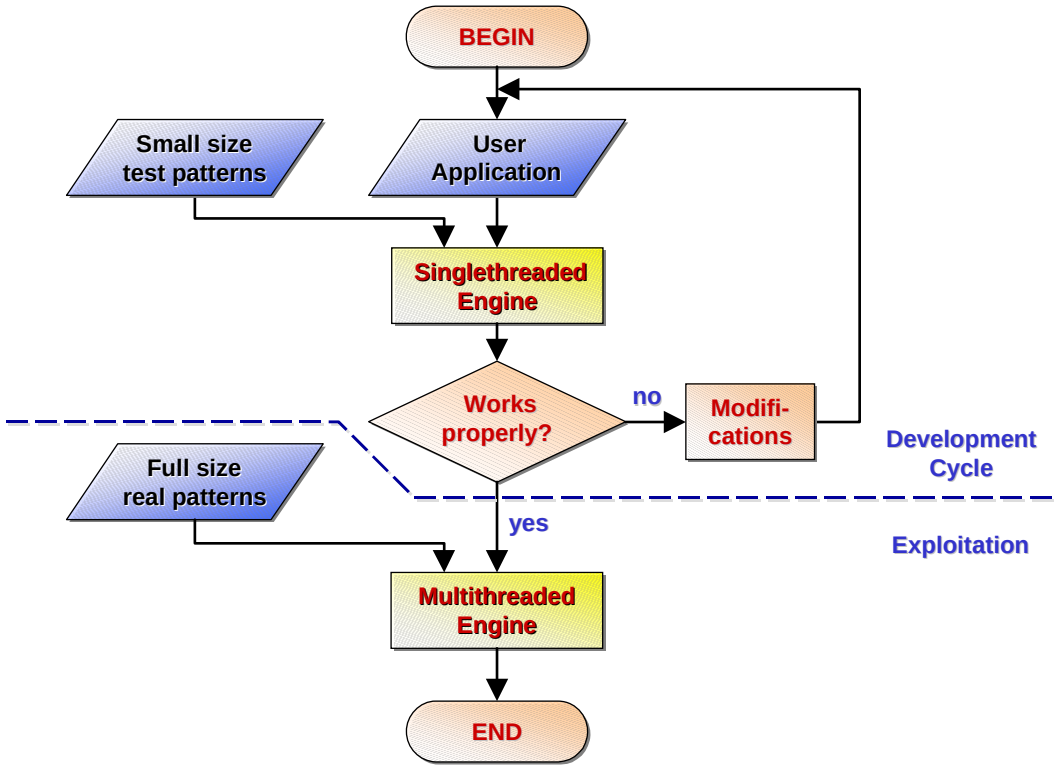

Figure 3-3. BMDFM user application life cycle

A BMDFM user application itself can be built according to the three schemes in Figure 3-4 (actually, the application can be structured as any combination of these three schemes).

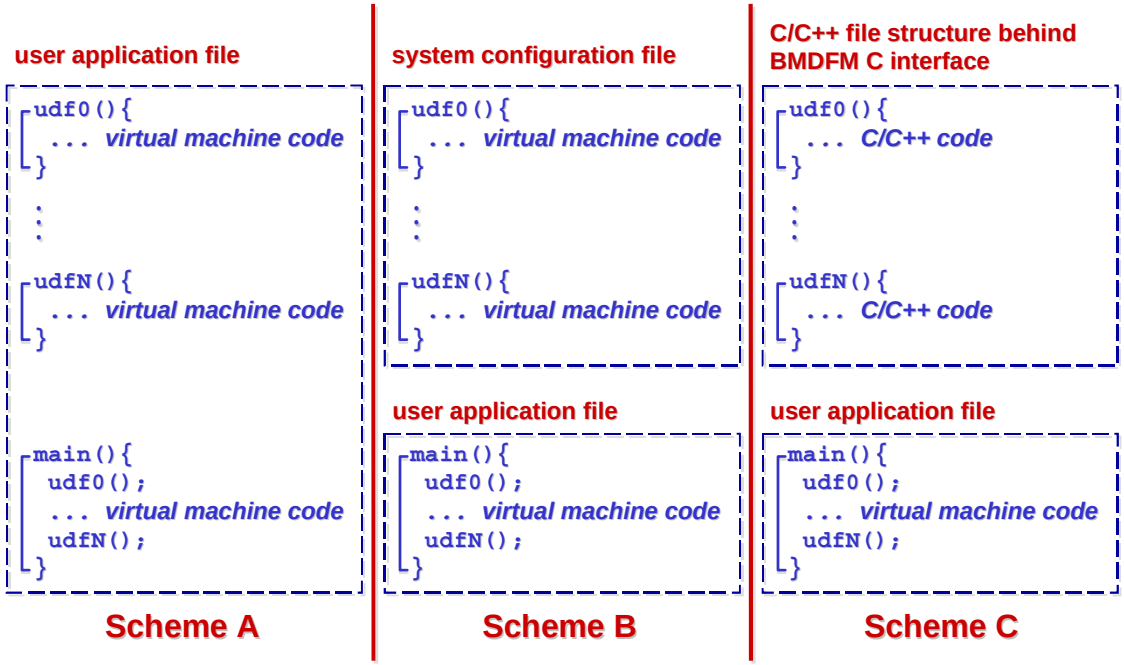

Figure 3-4. BMDFM user application structure

Scheme A. A complete application is written in pure virtual machine language. In this case BMDFM will exploit fine-grain parallelism, thus BMDFM will try to unroll the loops and to execute all statements in parallel. If it runs on a non-UMA (non-Uniform Memory Access) machine the dynamic scheduling can be expensive.

Scheme B. According to this scheme some UDFs (User Defined Functions) are located in the configuration profile, thus BMDFM will upload them into CPU PROCs Local Memory and their bodies will be prevented from scheduling for parallel processing (such a UDF will be treated as one seamless statement). In this case less time is obviously spent on dynamic scheduling.

Scheme C. This scheme enables using the C code directly instead of the virtual machine code. Of course the C code compiled and optimized by a local C compiler is faster than virtual machine code. In this case some BMDFM modules should be recompiled as shown in Figure 3-5.

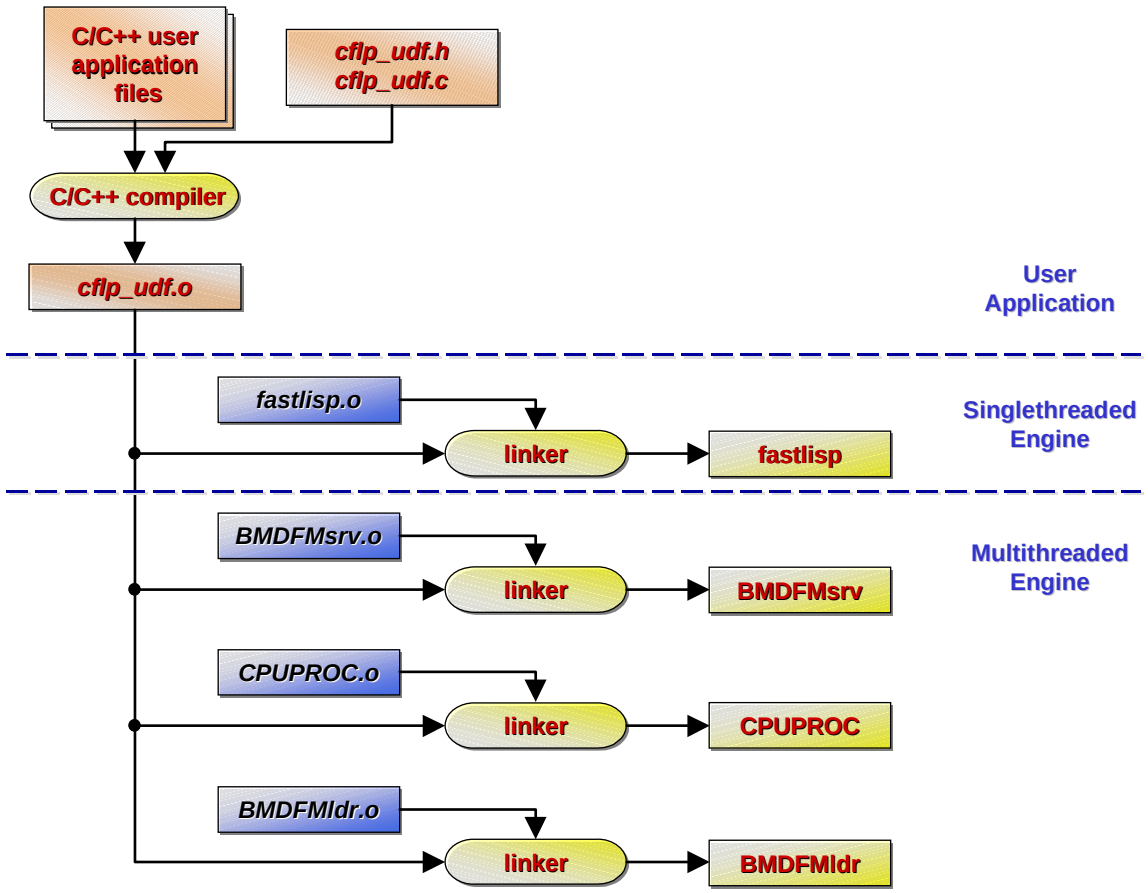

Figure 3-5. Recompilation of the BMDFM modules

3.3 Singlethreaded Engine

The BMDFM singlethreaded engine can be used from a command line according to the syntax shown in Figure 3-6. The help option creates a documentation file and a set of examples on the disk. The compile2disk option creates a machine dependent compiled code of the application on the disk. Optionally, the environment variable defines location of the configuration profile.

```
  Usage0: fastlisp -h|--help
  Usage1: fastlisp -V|--versions
  Usage2: fastlisp [-q|--quiet] <FastLisp_file_name> [args...]
  Usage3: fastlisp [-c|--compile2disk] <FastLisp_file_name> [args...]
  Usage4: fastlisp [-q|--quiet] <Precompiled_FastLisp_file_name>

The following environment variable:
  FAST_LISP_CFGPROFILE_path="fastlisp.cfg"
specifies a configuration profile that can be used for the Global FastLisp
function definitions. The format of the configuration profile is:
  <(DEFUN ...)>{ <(DEFUN ...)>}  # <EOF>.
```

Figure 3-6. BMDFM singlethreaded engine command line syntax

The BMDFM singlethreaded engine compiles, links and runs a user application in a standalone mode as shown in Figure 3-7.

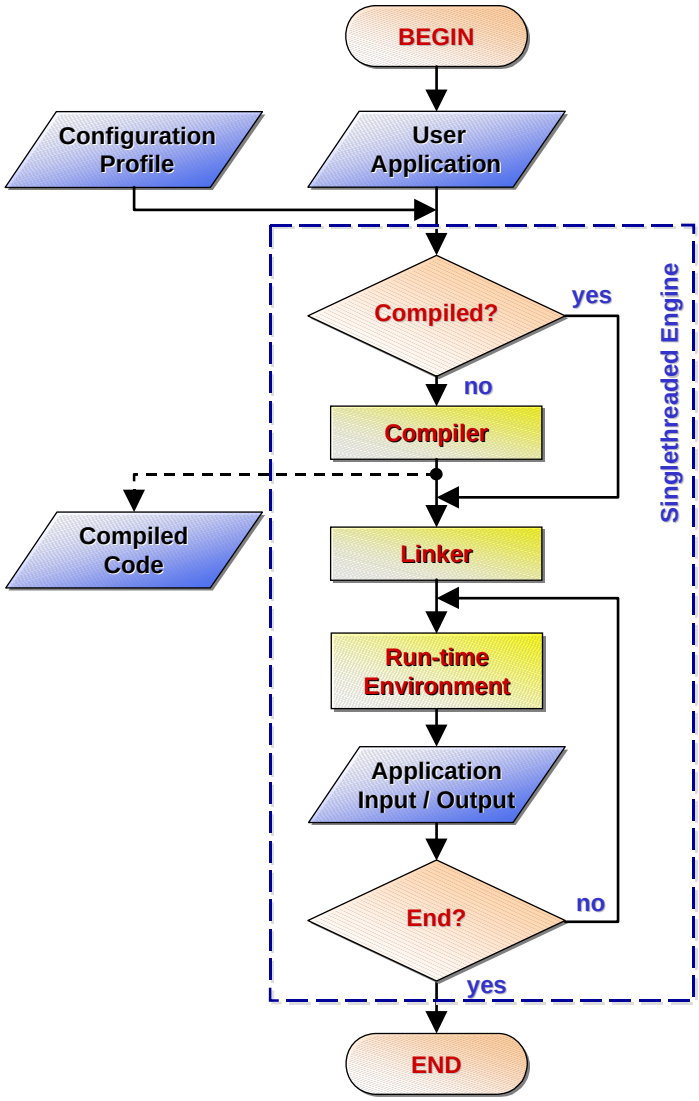

Figure 3-7. BMDFM singlethreaded engine work flow

3.4 Server Unit

There is only one way to start the BMDFM multithreaded engine correctly. It should be started by the BMDFM Server from a command line according to the syntax shown in Figure 3-8. The BMDFM Server may also run as a daemon. Additional logfile options enable/disable logging of console information on the disk. Optionally, the environment variables define locations of all the daemons, which will be started in the background, and further configuration parameters.

```
  Usage0: BMDFMsrv [-d|--daemonize]
  Usage1: BMDFMsrv -h|--help
  Usage2: BMDFMsrv [-d|--daemonize] -n|--no-logs
  Usage3: BMDFMsrv [-d|--daemonize] -l|--logfile <log_file_name>

Runtime environment variable dump:
  BM_DFM_CFGPROFILE_path="./BMDFMsrv.cfg";
  BM_DFM_PROCstat_path="./PROCstat";
  BM_DFM_CPUPROC_path="./CPUPROC";
  BM_DFM_OQPROC_path="./OQPROC";
  BM_DFM_IORBPROC_path="./IORBPROC";
  BM_DFM_CONNECTION_FILE_path="/tmp/.BMDFMsrv";
  BM_DFM_CONNECTION_NPIP_path="/tmp/.BMDFMsrv_npipe";
  BM_DFM_EMERGENCY_IPC_FILE_path="./freeIPC.inf";
  BM_DFM_LOGFILE_KEEP_NxSIZE="10x10000000";
  BM_DFM_PROCLOGFILE_KEEP_NxSIZE="10x10000000";
  BM_DFM_PROCLOGFILE_path="./PROCs.log";
```

Figure 3-8. BMDFM Server command line syntax

The BMDFM Server unit reads the configuration profile, initializes the Shared Memory Pool, starts multiple copies of the daemons in the background and enters a console mode. The BMDFM Server unit is also responsible for shutting down the whole multithreaded engine correctly. This procedure is illustrated in Figure 3-9. A screen shot of the BMDFM Server console, when it operates typical routines, is shown in Figure 3-10.

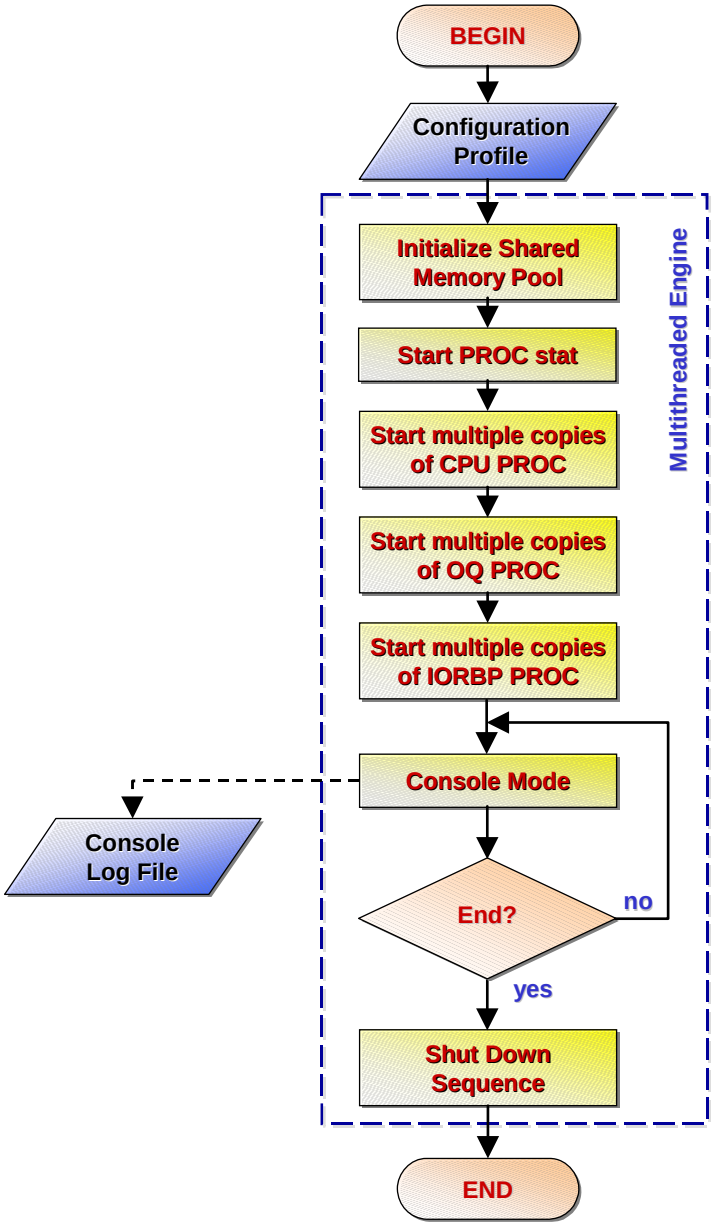

Figure 3-9. BMDFM Server unit work flow

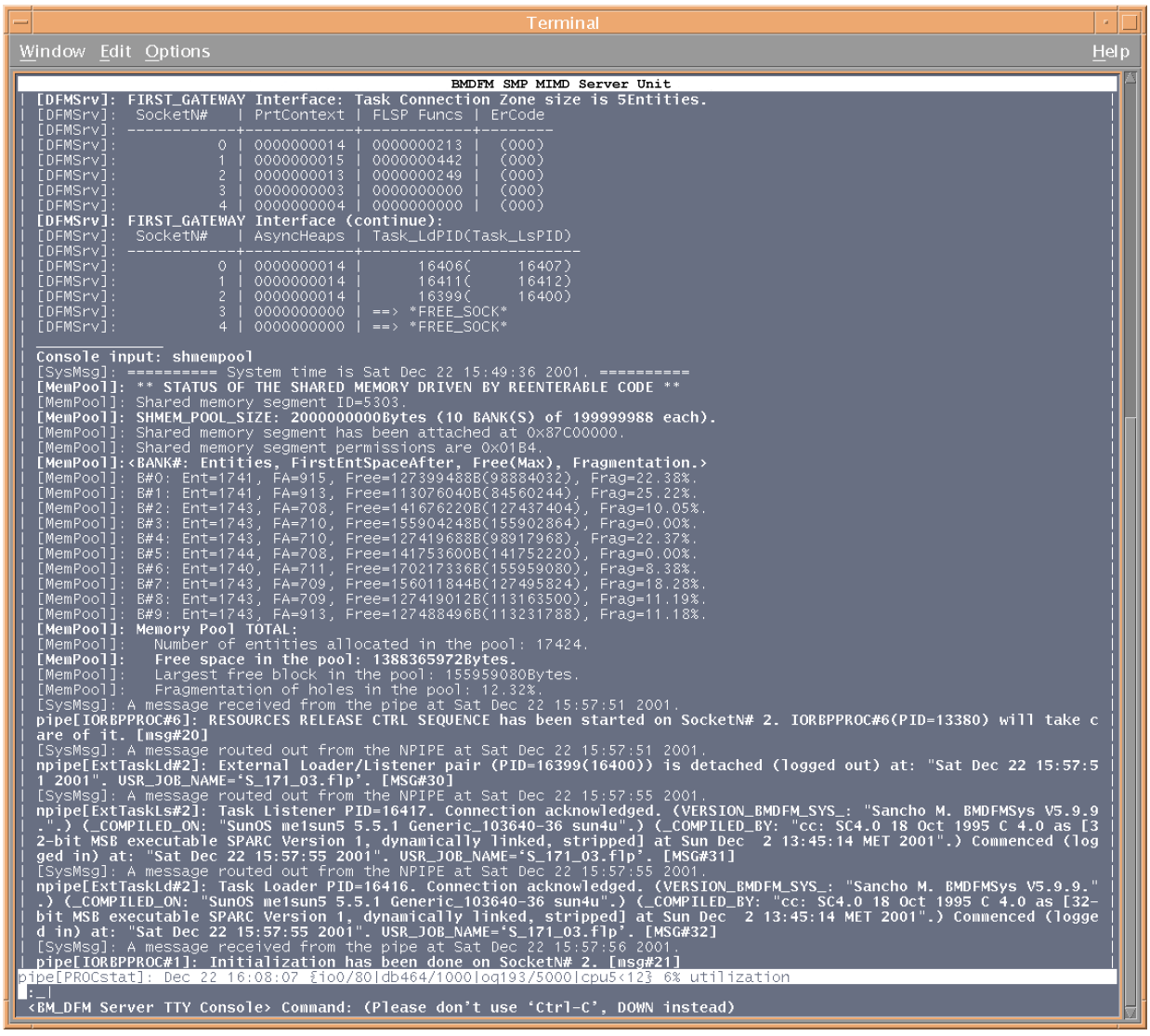

Figure 3-10. BMDFM Server console view

3.5 External Task Unit

The external task unit (Loader/Listener Pair) can be used from a command line according to the syntax shown in Figure 3-11. The compile2disk option creates a machine dependent compiled code of the application on the disk. Optionally, the environment variables define connection pathes to a BMDFM Server running instance.

```
  Usage0: BMDFMldr -h|--help
  Usage1: BMDFMldr -V|--versions
  Usage2: BMDFMldr [-q|--quiet] <FastLisp_file_name> [args...]
  Usage3: BMDFMldr [-c|--compile2disk] <FastLisp_file_name> [args...]
  Usage4: BMDFMldr [-q|--quiet] <Precompiled_FastLisp_file_name>

Runtime environment variable dump:
  BM_DFM_CONNECTION_FILE_path="/tmp/.BMDFMsrv";
  BM_DFM_CONNECTION_NPIP_path="/tmp/.BMDFMsrv_npipe";
```

Figure 3-11. External task unit command line syntax

The external task unit reorganizes the user code, makes static scheduling, compiles, connects multithreaded engine, links, starts listener thread and uploads the user application into the multithreaded engine as shown in Figure 3-12.

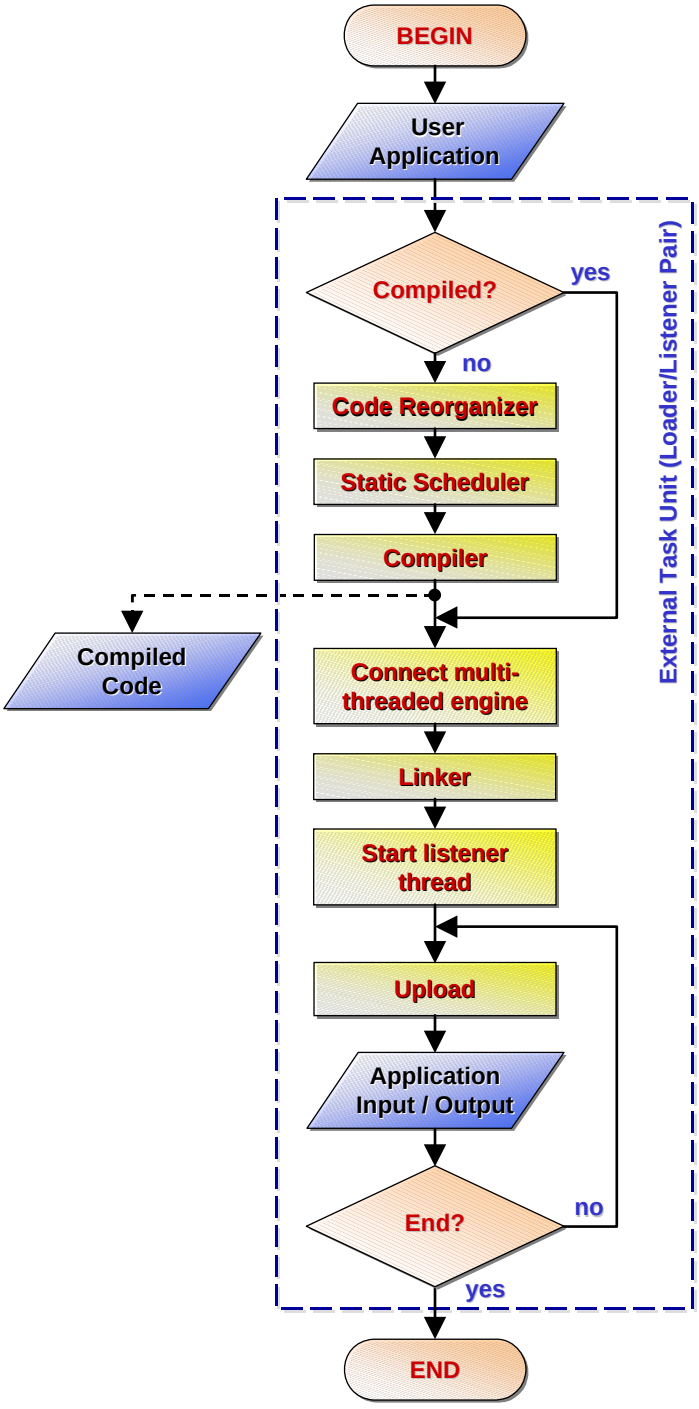

Figure 3-12. External task unit work flow

3.6 External Tracer Unit

The external tracer unit can be used from a command line according to the syntax shown in Figure 3-13. Additional logscreen options enable/disable information logging on the disk. Optionally, the environment variables define connection pathes to a BMDFM Server running instance.

```
  Usage0: BMDFMtrc
  Usage1: BMDFMtrc -h|--help
  Usage2: BMDFMtrc -l|--log-screen [<log_file_name>]

Runtime environment variable dump:
  BM_DFM_CONNECTION_FILE_path="/tmp/.BMDFMsrv";
  BM_DFM_CONNECTION_NPIP_path="/tmp/.BMDFMsrv_npipe";
```

Figure 3-13. External tracer unit command line syntax

The external tracer unit connects the multithreaded engine and enters the tracer console as shown in Figure 3-14. A screen shot of the external tracer console is shown in Figure 3-15.

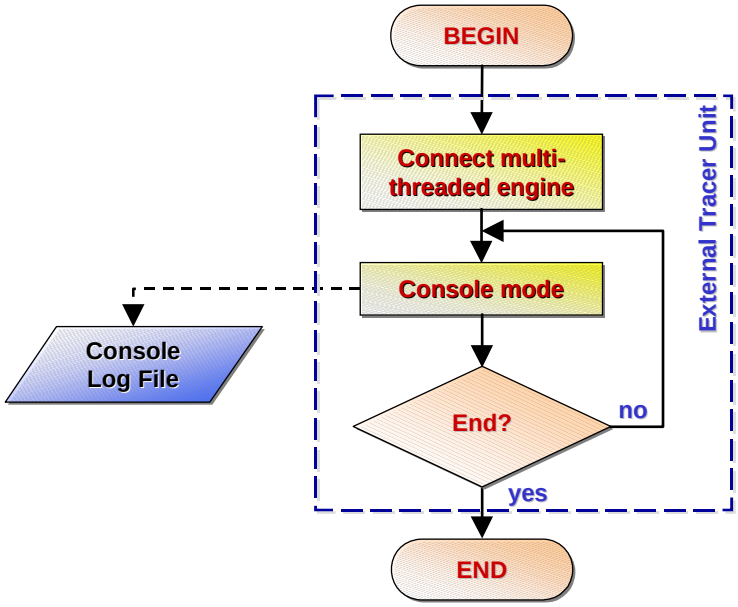

Figure 3-14. External tracer unit work flow

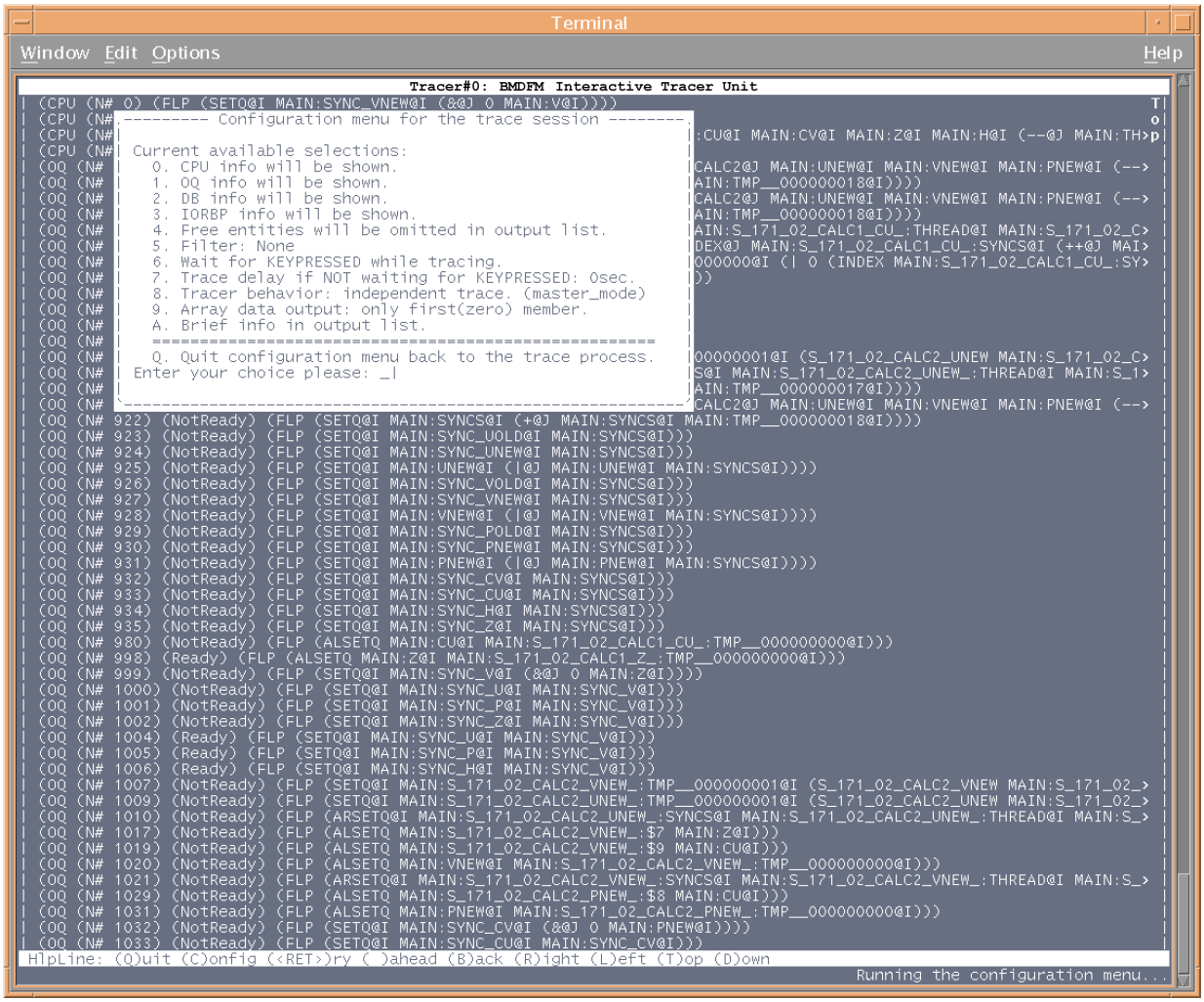

Figure 3-15. External tracer console view

Appendix A
Programming Language Reference

The BMDFM virtual machine language is based on a prefix form syntax and uses **transparent dataflow semantics**.

A.1 Program

A program is a function:

```
"<FuncName>"
   (<FuncName> <Val_argument1> <Val_argument2> ...  <Val_argumentN>)
     --> <Val_calculated_value>
```

It returns its calculated value as a result. It is possible to specify a formal argument, a constant value, a variable name or some other function at the place of the function argument. Formal arguments appear like $1 ... $N. All argument types will be automatically converted to the required types when the function is called.

`#' is a comment symbol. It indicates that all text following the `#' symbol (until the carriage return) is a comment. All symbols in the same line after the `#' symbol will be ignored.

A.2 Constant definition

Integer constants:

```
[<+|->]<digit>[{<digit>}]
```

Float constants:

```
[<+|->][{<digit>}][.{<digit>}][e[<+|->]{<digit>}]
```

String constants:

```
"<char>{<char>}"
Special symbols are:
  \a    - beep;
  \b    - back space;
  \e    - escape;
  \f    - page feed;
  \n    - new line;
  \r    - return;
  \t    - tab;
  \v    - vertical tab;
  \\    - \;
  \"    - ";
  \0xNN - symbol by its hex code (e.g.: \0x25 == `%', \0x5C == `\').
```

Special void type constant:

```
nil
```

A.3 Variables

Variable names are not case sensitive. A variable name starts with a letter followed by letters, numbers and other symbols except `@' and `:' symbols. Variables change their types dynamically among integers, floats, strings and nil.

Boolean `TRUE' is considered to be an integer non-zero value.

Boolean `FALSE' is considered to be an integer zero value.

A variable can also be an array. An array changes its size (number of members) dynamically. Array members can be integers, floats, strings, NIL's and arrays themselves that allows one creating list and tree structures.

First(zero) array member cannot be an array itself.

Predefined terminal control variables (same names like the corresponding terminal capabilities functions) are:

Variable Name	Type	Value
TERM_TYPE	String	<`TERM'_environment>
LINES_TERM	Integer	<termcap(li)>
COLUMNS_TERM	Integer	<termcap(co)>
CLRSCR_TERM	String	<termcap(cl)>
REVERSE_TERM	String	<termcap(mr)>
BLINK_TERM	String	<termcap(mb)>
BOLD_TERM	String	<termcap(md)>
NORMAL_TERM	String	<termcap(me)>
HIDECURSOR_TERM	String	<termcap(vi)>
SHOWCURSOR_TERM	String	<termcap(ve)>
GOTOCURSOR_TERM	String	<termcap(cm)>

Variable assignment and index functions:

```
.-------------------------------------------------------------------------.
| "setq" "alsetq"                                                         |
|   (setq <VarName> <Val_to_be_assigned_single_value_or_array>)           |
|     --> <Val_assigned_single_value_or_array>                            |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "arsetq"                                                                |
|   (arsetq <VarName> <IVal_index>                                        |
|           <Val_to_be_assigned_single_value_or_array>)                   |
|     --> <Val_assigned_single_value_or_array>                            |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "index"                                                                 |
|   (index <VarName> <IVal_index>)                                        |
|     --> <Val_array_member_value_by_array_member_index>                  |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "alindex"                                                               |
|   (alindex <VarName> <IVal_indices>)                                    |
|     --> <Val_array_of_values_by_number_of_indices>                      |
`-------------------------------------------------------------------------'
```

A.4 Special construction functions

Argument types will be casted.

Ordered grouping:

```
"progn"
  (progn <Val_to_be_evaluated_value1> <Val_to_be_evaluated_value2>
         ... <Val_to_be_evaluated_valueN>)
    --> <Val_evaluated_valueN>
```

Conditional execution:

```
"if"
  (if <IVal_condition> <Val_to_be_evaluated_if_true>
                       <Val_to_be_evaluated_if_false>)
    --> <Val_evaluated_on_true_or_false>
```

While-loop:

```
"while"
  (while <IVal_condition> <Val_to_be_evaluated_while_true>)
    --> <IVal_0>
```

For-loop:

```
"for"
  (for <VarName_control_variable> <IVal_start_value>
       <IVal_increment> <IVal_end_value>
       <Val_to_be_evaluated_iteratively>)
    --> <IVal_0>
```

User Defined Function (UDF). UDF names are not case sensitive. A UDF name may consist of letters, numbers and other symbols except `@' and `:' symbols. A UDF can be nested inside of another UDF. Formal arguments appear like $1 ... $N in the UDF body. A UDF may access only its private local variables, or in other words, all variables that are referenced in the UDF body are private local variables of this UDF.

```
"defun"
  (defun <FuncName> <Val_to_be_evaluated>)
    --> <SVal_empty_string>
```

Break iteration loop or UDF execution of the current nested level:

```
"break"
  (break)
    --> <SVal_empty_string>
```

Cancel program execution and exit:

```
"exit"
  (exit)
    --> <SVal_empty_string>
```

A.5 Input/Output functions

Argument types will be casted:

```
"accept"
  (accept <SVal_prompt_message_for_console_or_empty_for_stdin>)
    --> <SVal_input_string>
```

```
"scan_console"
  (scan_console <IVal_wait_key_forever_if_1_or_useconds_if_positive>)
    --> <SVal_keypressed_string>
```

```
"outf"
  (outf <SVal_printf_format> <Val_value>)
    --> <SVal_printf_formatted_string_that_was_printed_to_stdout>
```

```
"file_create"
  (file_create <SVal_file_name>)
    --> <IVal_file_descriptor_or_-1>
```

```
"file_open"
  (file_open <SVal_file_name>)
    --> <IVal_file_descriptor_or_-1>
```

```
"file_write"
  (file_write <IVal_file_descriptor> <SVal_string_to_be_written>)
    --> <IVal_number_of_bytes_written_or_-1>
```

```
"file_read"
  (file_read <IVal_file_descriptor> <IVal_number_of_bytes_to_be_read>)
    --> <SVal_string_read_or_empty_string>
```

```
.-------------------------------------------------------------------------.
| "file_seek_beg"                                                         |
|   (file_seek_beg <IVal_file_descriptor>                                 |
|                  <IVal_offset_in_bytes_from_file_beginning>)            |
|     --> <IVal_offset_in_bytes_from_file_beginning_or_-1>                |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "file_seek_cur"                                                         |
|   (file_seek_cur <IVal_file_descriptor>                                 |
|                  <IVal_offset_in_bytes_from_file_current_offset>)       |
|     --> <IVal_offset_in_bytes_from_file_beginning_or_-1>                |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "file_seek_end"                                                         |
|   (file_seek_end <IVal_file_descriptor>                                 |
|                  <IVal_offset_in_bytes_from_file_end>)                  |
|     --> <IVal_offset_in_bytes_from_file_beginning_or_-1>                |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "file_close"                                                            |
|   (file_close <IVal_file_descriptor>)                                   |
|     --> <IVal_0_or_-1>                                                  |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "file_remove"                                                           |
|   (file_remove <SVal_file_name>)                                        |
|     --> <IVal_0_or_-1>                                                  |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "user_io"                                                               |
|   (user_io <IVal_user_defined_integer> <IVal_user_defined_string>)      |
|     --> <SVal_string_returned_by_user_defined_io_c_function>            |
`-------------------------------------------------------------------------'
```

A.6 Built-in comparison functions

Second argument type will be casted to first argument type:

```
.-------------------------------------------------------------------.
| "==" "equal"                                                      |
|   (== <Val_value1> <Val_value2>)                                  |
|     --> <IVal_true_if_value1_and_value2_are_equal_otherwise_false>|
`-------------------------------------------------------------------'
.-------------------------------------------------------------------.
| "!=" "notequal"                                                   |
|   (!= <Val_value1> <Val_value2>)                                  |
|     --> <IVal_true_if_value1_and_value2_are_not_equal_otherwise_false>|
`-------------------------------------------------------------------'
.-------------------------------------------------------------------.
| "<" "less"                                                        |
|   (< <Val_value1> <Val_value2>)                                   |
|     --> <IVal_true_if_value1_is_less_than_value2_otherwise_false> |
`-------------------------------------------------------------------'
.-------------------------------------------------------------------.
| ">" "greater"                                                     |
|   (> <Val_value1> <Val_value2>)                                   |
|     --> <IVal_true_if_value1_is_greater_than_value2_otherwise_false>|
`-------------------------------------------------------------------'
.-------------------------------------------------------------------.
| "<=" "lessorequal"                                                |
|   (<= <Val_value1> <Val_value2>)                                  |
|     --> <IVal_true_if_value1_is_less_than_value2_or_              |
|                  _if_value1_and_value2_are_equal_otherwise_false> |
`-------------------------------------------------------------------'
.-------------------------------------------------------------------.
| ">=" "greaterorequal"                                             |
|   (>= <Val_value1> <Val_value2>)                                  |
|     --> <IVal_true_if_value1_is_greater_than_value2_or_           |
|                  _if_value1_and_value2_are_equal_otherwise_false> |
`-------------------------------------------------------------------'
```

A.7 Built-in boolean functions

Argument types will be casted:

```
.-------------------------------------------------------------------------.
| "&&" "and"                                                              |
|   (&& <IVal_boolean1> <IVal_boolean2_short_circuit_evaluation>)         |
|     --> <IVal_boolean_AND>                                              |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "||" "or"                                                               |
|   (|| <IVal_boolean1> <IVal_boolean2_short_circuit_evaluation>)         |
|     --> <IVal_boolean_OR>                                               |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "!" "not"                                                               |
|   (! <IVal_boolean>)                                                    |
|     --> <IVal_boolean_NOT>                                              |
`-------------------------------------------------------------------------'
```

A.8 Built-in integer functions

Argument types will be casted:

```
"ival"
  (ival <Val_value>)
    --> <IVal_integer>
```

```
"indices"
  (indices <Val_value>)
    --> <IVal_number_of_array_indices>
```

```
"irnd"
  (irnd <IVal_maxrange_or_negative_to_reset_random_generator>)
    --> <IVal_random_value_within_the_range_of_0_to_maxrange>
```

```
"+" "iadd"
  (+ <IVal_integer1> <IVal_integer2>)
    --> <IVal_addition_integer1+integer2>
```

```
"-" "isub"
  (- <IVal_integer1> <IVal_integer2>)
    --> <IVal_subtraction_integer1-integer2>
```

```
"*" "imul"
  (* <IVal_integer1> <IVal_integer2>)
    --> <IVal_multiplication_integer1*integer2>
```

```
"/" "idiv"
  (/ <IVal_integer1> <IVal_integer2>)
    --> <IVal_division_integer1/integer2>
```

```
"*+" "ima"
  (*+ <IVal_integer1> <IVal_integer2> <IVal_integer3>)
    --> <IVal_MultiplyAccumulateOperation_integer1*integer2+integer3>
```

```
"%" "imod"
  (% <IVal_integer1> <IVal_integer2>)
    --> <IVal_modulo_integer_remainder_integer1%integer2>
```

```
"++" "iincr"
  (++ <IVal_integer>)
    --> <IVal_increment>
```

```
"--" "idecr"
  (-- <IVal_integer>)
    --> <IVal_decrement>
```

```
"0-" "ineg"
  (0- <IVal_integer>)
    --> <IVal_additive_inverse>
```

```
"iabs"
  (iabs <IVal_integer>)
    --> <IVal_absolute_value>
```

```
"&" "iand"
  (& <IVal_integer1> <IVal_integer2>)
    --> <IVal_bitwise_AND>
```

```
"|" "ior"
  (| <IVal_integer1> <IVal_integer2>)
    --> <IVal_bitwise_OR>
```

```
"^" "ixor"
  (^ <IVal_integer1> <IVal_integer2>)
    --> <IVal_bitwise_exclusive_OR>
```

```
"~" "inot"
  (~ <IVal_integer>)
    --> <IVal_bitwise_inversion>
```

```
.-------------------------------------------------------------------------.
| ">>" "ishr"                                                             |
|   (>> <IVal_integer> <IVal_shift_positions>)                            |
|     --> <IVal_bitwise_right_shift>                                      |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "<<" "ishl"                                                             |
|   (<< <IVal_integer> <IVal_shift_positions>)                            |
|     --> <IVal_bitwise_left_shift>                                       |
`-------------------------------------------------------------------------'
```

A.9 Built-in float functions

Argument types will be casted:

```
"fval"
  (fval <Val_value>)
    --> <FVal_float>
```

```
"frnd"
  (frnd <FVal_maxrange_or_negative_to_reset_random_generator>)
    --> <FVal_random_value_within_the_range_of_0_to_maxrange>
```

```
"+." "fadd"
  (+. <FVal_float1> <FVal_float2>)
    --> <FVal_addition_float1+float2>
```

```
"-." "fsub"
  (-. <FVal_float1> <FVal_float2>)
    --> <FVal_subtraction_float1-float2>
```

```
"*." "fmul"
  (*. <FVal_float1> <FVal_float2>)
    --> <FVal_multiplication_float1*float2>
```

```
"/." "fdiv"
  (/. <FVal_float1> <FVal_float2>)
    --> <FVal_division_float1/float2>
```

```
"*+." "fma"
  (*+. <FVal_float1> <FVal_float2> <FVal_float3>)
    --> <FVal_MultiplyAccumulateOperation_float*float2+float3>
```

```
"fabs"
  (fabs <FVal_float>)
    --> <FVal_absolute_value>
```

```
.-------------------------------------------------------------------------.
| "fint" "int"                                                            |
|   (fint <FVal_float>)                                                   |
|     --> <FVal_integer_part>                                             |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "fround" "round"                                                        |
|   (fround <FVal_float>)                                                 |
|     --> <FVal_rounded_value>                                            |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "fcos" "cos"                                                            |
|   (fcos <FVal_radians>)                                                 |
|     --> <FVal_cosine>                                                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "fsin" "sin"                                                            |
|   (fsin <FVal_radians>)                                                 |
|     --> <FVal_sine>                                                     |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "fcas" "cas"                                                            |
|   (fcas <FVal_radians>)                                                 |
|     --> <FVal_sine+cosine>                                              |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "fatn" "atn"                                                            |
|   (fatn <FVal_float>)                                                   |
|     --> <FVal_arctangent_radians>                                       |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "fexp" "exp"                                                            |
|   (fexp <FVal_float>)                                                   |
|     --> <FVal_exponential>                                              |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "fpow" "pow" "^." "**"                                                  |
|   (fpow <FVal_base> <FVal_exponent>)                                    |
|     --> <FVal_base_raised_to_the_power_of_exponent>                     |
`-------------------------------------------------------------------------'
```

```
.-------------------------------------------------------------------------.
| "fln" "ln"                                                              |
|   (fln <FVal_float>)                                                    |
|     --> <FVal_natural_logarithm_base_E>                                 |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "fsqrt" "sqrt" "sqr"                                                    |
|   (fsqrt <FVal_float>)                                                  |
|     --> <FVal_square_root>                                              |
`-------------------------------------------------------------------------'
```

A.10 Built-in string functions

Argument types will be casted:

```
"str"
  (str <Val_value>)
    --> <SVal_string>
```

```
"chr"
  (chr <IVal_integer>)
    --> <SVal_one_character_string>
```

```
"asc"
  (asc <SVal_string>)
    --> <IVal_code_of_first_character>
```

```
"type"
  (type <Val_value>)
    --> <SVal_type_among_I_F_S_Z>
```

```
"dump_i2s"
  (dump_i2s <IVal_integer>)
    --> <SVal_memory_dump_of_integer>
```

```
"dump_f2s"
  (dump_f2s <FVal_float>)
    --> <SVal_memory_dump_of_float>
```

```
"dump_s2i"
  (dump_s2i <SVal_string>)
    --> <IVal_integer_dumped_from_string>
```

```
"dump_s2f"
  (dump_s2f <SVal_string>)
    --> <FVal_float_dumped_from_string>
```

```
.-------------------------------------------------------------------------.
| "notempty"                                                              |
|   (notempty <SVal_string>)                                              |
|     --> <IVal_true_if_not_empty>                                        |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "len"                                                                   |
|   (len <SVal_string>)                                                   |
|     --> <IVal_length>                                                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "at"                                                                    |
|   (at <SVal_substring_to_be_found> <SVal_string_to_be_searched_in>)     |
|     --> <IVal_found_first_occurrence_position_from_left_or_zero>        |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "rat"                                                                   |
|   (rat <SVal_substring_to_be_found> <SVal_string_to_be_searched_in>)    |
|     --> <IVal_found_first_occurrence_position_from_right_or_zero>       |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "cat"                                                                   |
|   (cat <SVal_string1> <SVal_string2>)                                   |
|     --> <SVal_concatenation_string1+string2>                            |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "space"                                                                 |
|   (space <IVal_length>)                                                 |
|     --> <SVal_empty_string_filled_with_spaces>                          |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "replicate"                                                             |
|   (replicate <SVal_pattern> <IVal_number_of_copies>)                    |
|     --> <SVal_string_filled_with_patterns>                              |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "left"                                                                  |
|   (left <SVal_string> <IVal_position_from_left>)                        |
|     --> <SVal_left_part_of_string>                                      |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "leftr"                                                                 |
|   (leftr <SVal_string> <IVal_position_from_right>)                      |
|     --> <SVal_left_part_of_string>                                      |
`-------------------------------------------------------------------------'
```

```
.-------------------------------------------------------------------------.
| "right"                                                                 |
|   (right <SVal_string> <IVal_position_from_right>)                      |
|     --> <SVal_right_part_of_string>                                     |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "rightl"                                                                |
|   (rightl <SVal_string> <IVal_position_from_left>)                      |
|     --> <SVal_right_part_of_string>                                     |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "substr"                                                                |
|   (substr <SVal_string> <IVal_position_from_left> <IVal_length>)        |
|     --> <SVal_substring_derived_from_string>                            |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "strtran"                                                               |
|   (strtran <SVal_string> <SVal_pattern> <SVal_substitution>)            |
|     --> <SVal_string_where_patterns_are_replaced_with_substitutions>    |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "str_raw"                                                               |
|   (str_raw <SVal_string>)                                               |
|     --> <SVal_string_with_no_escape_characters_for_special_symbols>     |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "str_unraw"                                                             |
|   (str_unraw <SVal_string>)                                             |
|     --> <SVal_string_with_escape_characters_for_special_symbols>        |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "str_dump"                                                              |
|   (str_dump <SVal_string>)                                              |
|     --> <SVal_semihexadecimal_string_dump>                              |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "str_fmt"                                                               |
|   (str_fmt <SVal_printf_format> <Val_value>)                            |
|     --> <SVal_printf_formatted_string>                                  |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "ltrim"                                                                 |
|   (ltrim <SVal_string>)                                                 |
|     --> <SVal_string_with_no_leading_blanks>                            |
`-------------------------------------------------------------------------'
```

```
.-------------------------------------------------------------------------.
| "rtrim"                                                                 |
|   (rtrim <SVal_string>)                                                 |
|     --> <SVal_string_with_no_ending_blanks>                             |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "alltrim"                                                               |
|   (alltrim <SVal_string>)                                               |
|     --> <SVal_string_with_neither_leading_nor_ending_blanks>            |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "pack"                                                                  |
|   (pack <SVal_string>)                                                  |
|     --> <SVal_string_with_no_redundant_blanks>                          |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "head"                                                                  |
|   (head <SVal_string>)                                                  |
|     --> <SVal_first_token>                                              |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "tail"                                                                  |
|   (tail <SVal_string>)                                                  |
|     --> <SVal_remaining_tokens>                                         |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "lsp_head"                                                              |
|   (lsp_head <SVal_string>)                                              |
|     --> <SVal_first_FastLisp_token>                                     |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "lsp_tail"                                                              |
|   (lsp_tail <SVal_string>)                                              |
|     --> <SVal_remaining_FastLisp_tokens>                                |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "upper"                                                                 |
|   (upper <SVal_string>)                                                 |
|     --> <SVal_upper_case_string>                                        |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "lower"                                                                 |
|   (lower <SVal_string>)                                                 |
|     --> <SVal_lower_case_string>                                        |
`-------------------------------------------------------------------------'
```

```
"rev"
  (rev <SVal_string>)
    --> <SVal_reverse_ordered_string>
```

```
"padl"
  (padl <SVal_string> <IVal_length>)
    --> <SVal_left_justified_string>
```

```
"padr"
  (padr <SVal_string> <IVal_length>)
    --> <SVal_right_justified_string>
```

```
"padc"
  (padc <SVal_string> <IVal_length>)
    --> <SVal_centered_string>
```

```
"time"
  (time)
    --> <SVal_current_system_time>
```

```
"getenv"
  (getenv <SVal_environment_variable_name>)
    --> <SVal_environment_variable_value>
```

A.11 Built-in asynchronous memory heap functions

Argument types will be casted:

```
"asyncheap_create"
  (asyncheap_create <IVal_size_bytes>)
    --> <IVal_descriptor_or_0>
```

```
"asyncheap_getaddress"
  (asyncheap_getaddress <IVal_descriptor>)
    --> <IVal_address>
```

```
"asyncheap_putint"
  (asyncheap_putint <IVal_descriptor> <IVal_offset> <IVal_integer>)
    --> <IVal_1>
```

```
"asyncheap_getint"
  (asyncheap_getint <IVal_descriptor> <IVal_offset>)
    --> <IVal_integer>
```

```
"asyncheap_putfloat"
  (asyncheap_putfloat <IVal_descriptor> <IVal_offset> <FVal_float>)
    --> <IVal_1>
```

```
"asyncheap_getfloat"
  (asyncheap_getfloat <IVal_descriptor> <IVal_offset>)
    --> <FVal_float>
```

```
"asyncheap_putstring"
  (asyncheap_putstring <IVal_descriptor> <IVal_offset> <SVal_string>)
    --> <IVal_1>
```

```
"asyncheap_getstring"
  (asyncheap_getstring <IVal_descriptor> <IVal_offset> <IVal_length>)
    --> <SVal_string>
```

```
.-------------------------------------------------------------------------.
| "asyncheap_reallocate"                                                  |
|   (asyncheap_reallocate <IVal_descriptor> <IVal_new_size_bytes>)        |
|     --> <IVal_new_descriptor>                                           |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "asyncheap_replicate"                                                   |
|   (asyncheap_replicate <IVal_descriptor>)                               |
|     --> <IVal_new_descriptor>                                           |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "asyncheap_delete"                                                      |
|   (asyncheap_delete <IVal_descriptor>)                                  |
|     --> <IVal_1>                                                        |
`-------------------------------------------------------------------------'
```

A.12 Built-in mapcar function

Argument types will be casted:

```
"mapcar"
  (mapcar <SVal_FastLisp_function_or_compiled_bytecode>)
    --> (<SVal_preprinted_info>
         <Val_result_of_execution_that_can_be_list_or_tree_as_well>
         <IVal_syntax_error_code>
         <SVal_syntax_error_message>
         <IVal_runtime_error_code>
         <SVal_runtime_error_message>
         <SVal_processed_FastLisp_function_code>
         <SVal_processed_FastLisp_function_compiled_bytecode>
         <SVal_processed_FastLisp_function_linked_bytecode>
         <FVal_time_spent_seconds>
        )
```

A.13 Built-in terminal capabilities functions

Argument types will be casted:

```
"reinit_terminal"
  (reinit_terminal <SVal_terminal_type_or_empty_for_default_terminal>)
    --> <SVal_terminal_capabilities_status>
```

```
"term_type"
  (term_type)
    --> <SVal_TERM_environment_configured_terminal_name>
```

```
"lines_term"
  (lines_term)
    --> <IVal_terminal_capability_li>
```

```
"columns_term"
  (columns_term)
    --> <IVal_terminal_capability_co>
```

```
"clrscr_term"
  (clrscr_term)
    --> <SVal_terminal_capability_cl>
```

```
"reverse_term"
  (reverse_term)
    --> <SVal_terminal_capability_mr>
```

```
"blink_term"
  (blink_term)
    --> <SVal_terminal_capability_mb>
```

```
"bold_term"
  (bold_term)
    --> <SVal_terminal_capability_md>
```

```
.-------------------------------------------------------------------------.
| "normal_term"                                                           |
|   (normal_term)                                                         |
|     --> <SVal_terminal_capability_me>                                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "hidecursor_term"                                                       |
|   (hidecursor_term)                                                     |
|     --> <SVal_terminal_capability_vi>                                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "showcursor_term"                                                       |
|   (showcursor_term)                                                     |
|     --> <SVal_terminal_capability_ve>                                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "gotocursor_term"                                                       |
|   (gotocursor_term <IVal_y_coordinate> <IVal_x_coordinate>)             |
|     --> <SVal_filled_with_coordinates_terminal_capability_cm_           |
|          _or_unfilled_terminal_capability_cm_if_input_is_negative>      |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "gotocursor1_term"                                                      |
|   (gotocursor1_term <SVal_unfilled_terminal_capability_cm>              |
|                     <IVal_y_coordinate> <IVal_x_coordinate>)            |
|     --> <SVal_filled_with_coordinates_terminal_capability_cm>           |
`-------------------------------------------------------------------------'
```

A.14 Built-in constant and info functions

```
"ee"
  (ee)
    --> <FVal_base_of_natural_logarithm_e>
```

```
"gamma"
  (gamma)
    --> <FVal_Euler_Mascheroni_constant_gamma>
```

```
"phi"
  (phi)
    --> <FVal_golden_ratio_constant_phi>
```

```
"pi"
  (pi)
    --> <FVal_constant_pi>
```

```
"prn_integer_fmt"
  (prn_integer_fmt)
    --> <SVal_preconfigured_printf_format_for_integer>
```

```
"prn_float_fmt"
  (prn_float_fmt)
    --> <SVal_preconfigured_printf_format_for_float>
```

```
"prn_string_fmt"
  (prn_string_fmt)
    --> <SVal_preconfigured_printf_format_for_string>
```

```
"version_fstlisp"
  (version_fstlisp)
    --> <SVal_FastLisp_version>
```

```
.-------------------------------------------------------------------------.
| "version_termcap"                                                       |
|   (version_termcap)                                                     |
|     --> <SVal_termcap_library_version>                                  |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "version_strglib"                                                       |
|   (version_strglib)                                                     |
|     --> <SVal_string_library_version>                                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "version_mempool"                                                       |
|   (version_mempool)                                                     |
|     --> <SVal_memory_pool_library_version>                              |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "compiled_on"                                                           |
|   (compiled_on)                                                         |
|     --> <SVal_compilation_related_machine_os_kernel_specific_info>      |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "compiled_by"                                                           |
|   (compiled_by)                                                         |
|     --> <SVal_compilation_related_machine_compiler_specific_info>       |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "n_cpuproc"                                                             |
|   (n_cpuproc)                                                           |
|     --> <IVal_number_of_configured_parallel_processing_units>           |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "id_cpuproc"                                                            |
|   (id_cpuproc)                                                          |
|     --> <IVal_id_of_current_parallel_processing_unit>                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "n_taskjob"                                                             |
|   (n_taskjob)                                                           |
|     --> <IVal_number_of_maximal_parallel_task_jobs>                     |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "id_taskjob"                                                            |
|   (id_taskjob)                                                          |
|     --> <IVal_id_of_current_task_job>                                   |
`-------------------------------------------------------------------------'
```

```
.-------------------------------------------------------------------------.
| "am_I_in_the_fastlisp_module"                                           |
|   (am_I_in_the_fastlisp_module)                                         |
|     --> <IVal_true_if_running_in_the_fastlisp_module>                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "am_I_in_the_BMDFMldr_module"                                           |
|   (am_I_in_the_BMDFMldr_module)                                         |
|     --> <IVal_true_if_running_in_the_BMDFMldr_module>                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "am_I_in_the_BMDFMsrv_module"                                           |
|   (am_I_in_the_BMDFMsrv_module)                                         |
|     --> <IVal_true_if_running_in_the_BMDFMsrv_module>                   |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "am_I_in_the_CPUPROC_module"                                            |
|   (am_I_in_the_CPUPROC_module)                                          |
|     --> <IVal_true_if_running_in_the_CPUPROC_module>                    |
`-------------------------------------------------------------------------'
.-------------------------------------------------------------------------.
| "am_I_in_the_multithreaded_module"                                      |
|   (am_I_in_the_multithreaded_module)                                    |
|     --> <IVal_true_if_running_in_the_multithreaded_module>              |
`-------------------------------------------------------------------------'
```

A.15 Built-in rise runtime error functions

Argument types will be casted:

```
"rise_error"
  (rise_error)
```

```
"rise_error_info"
  (rise_error_info <IVal_error_code> <SVal_error_text>)
```

A.16 C Interface

See cflp_udf.h/cflp_udf.c for details.

In order to simplify declaration constructions the following abbreviations are used for the standard types:

```
#define CHR char
#define UCH unsigned char
#define SCH signed char
#define USH unsigned short int
#define SSH signed short int
#define ULO unsigned long int
#define SLO signed long int
#define DFL double
```

Each variable is stored in a universal structure that enables it to change data types dynamically and to have a single value or an array with different types of members, thus supporting lists and trees.

The declaration of a variable of the universal structure type allocates a single value on the stack and an array in the heap that is very convenient assuming most variables store only single values. No memory overhead is needed for storing arrays with members of the same type.

```
struct fastlisp_data{
  UCH disable_ptr; /* 1 stores value, 0 ptr to a variable possible */
  UCH single;      /* 1 single value, 0 array                      */
  UCH type;        /* 0 undef, 'I'int, 'F'float, 'S'string, 'Z'nil */
  UCH arraytype;   /* 0 undef, 'I'int, 'F'float, 'S'string, 'Z'nil */
  union{
    SLO ival;      /* integer value */
    DFL fval;      /* float value */
  } value;
  CHR *svalue;     /* string value */
  ULO indices_numb;/* number of indices in the array */
  UCH *aready_tags;/* member flags '0IFSZ' for arraytype!=0 */
  union{
    struct fastlisp_data *mix;/* array members of mixed types */
    SLO *ival;                /* array members of integer type */
    DFL *fval;                /* array members of float type */
    CHR **svalue;             /* array members of string type */
  } array;
};
```

A user C function can be defined through the following type declaration:

```
typedef void (*fcall)(const ULO*, struct fastlisp_data*);
```

The first argument is a pointer to the passed function arguments and the second argument is a pointer to the result structure.

Passed arguments can be obtained from inside the function via the following set:

```
/* get universal data structure */
  void (*fcall)(const ULO*, struct fastlisp_data*);

/* get integer or pointer value */
  void ret_ival(const ULO *dat_ptr, SLO *targ);

/* get float value */
  void ret_fval(const ULO *dat_ptr, DFL *targ);

/* get string value */
  void ret_sval(const ULO *dat_ptr, CHR **targ);
```

Additionally, there are:

- two helper functions to be called from each created user thread;
- two functions for copying and deleting the universal data structure;
- three callback functions;
- four info functions for the processing units and task jobs;
- five info functions for the modules;
- five functions to handle runtime errors.

```
/* helper functions to be called from each created user thread
    (if this thread uses BMDFM functionality) */

  void setup_bmdfm_thread_safe_functionality_for_local_thread(void);

  void cleanup_bmdfm_thread_safe_functionality_for_local_thread(void);
```

```
/* copy the universal data structure */
  void copy_flp_data(struct fastlisp_data *dest,
                     const struct fastlisp_data *source,
                     ULO indices_numb);

/* delete the universal data structure */
  void free_flp_data(struct fastlisp_data *ret_dat);
```

```
/* called at startup */
  void startup_callback(void);

/* called at task job end */
  void taskjob_end_callback(ULO id_taskjob);

/* (user_io <IVal> <SVal>) callee */
  void user_io_callback(SLO usr_id, CHR **usr_buff);
```

```
/* get the number of configured parallel processing units
   ('N_CPUPROC' configuration parameter) */
  ULO get_n_cpuproc(void);

/* get ID of the current parallel processing unit
   (within the range of [0; 'N_CPUPROC'[) */
  ULO get_id_cpuproc(void);

/* get the number of maximal parallel task jobs
   ('N_IORBP' configuration parameter) */
  ULO get_n_taskjob(void);

/* get ID of the current task job
   (within the range of [0; 'N_IORBP'[) */
  ULO get_id_taskjob(void);
```

```
/* info functions for the modules */

  UCH am_I_in_the_fastlisp_module(void);

  UCH am_I_in_the_BMDFMldr_module(void);

  UCH am_I_in_the_BMDFMsrv_module(void);

  UCH am_I_in_the_CPUPROC_module(void);

  UCH am_I_in_the_multithreaded_module(void);
```

```
/* check whether no runtime error occurred */
  UCH noterror(void);

/* rise runtime error */
  void rise_error(void);

/* rise runtime error info */
  void rise_error_info(UCH errcode, const CHR *errtext);

/* get runtime error code */
  UCH get_error_code(void);

/* get runtime error text */
  CHR *get_error_text(CHR **errtext);

/* Reserved runtime error codes:
  ECODE_RT__INT_DIVZERO        1
  ECODE_RT__INT_MODZERO        2
  ECODE_RT__FLOAT_DIVZERO      3
  ECODE_RT__FLOAT_POWER        4
  ECODE_RT__FLOAT_LOGARITHM    5
  ECODE_RT__FLOAT_SQRT         6
  ECODE_RT__AHEAP_WRONGDESCR   7
  ECODE_RT__AHEAP_OUTOFRANGE   8
  ECODE_RT__WRONG_FMT_STRING   9
  ECODE_RT__VAR_NOTINIT       10
  ECODE_RT__ARR_NOTINIT       11
  ECODE_RT__ARR_MEMBSNOTINIT  12
  ECODE_RT__ARR_NEGINDEX      13
  ECODE_RT__ARR_ZEROMEMB      14
  ECODE_RT__ARR_WRONGINDICES  15
  ECODE_RT__ARR_TYPEMISMATCH  16
  ECODE_RT__RESERVED6        249
  ECODE_RT__RESERVED5        250
  ECODE_RT__RESERVED4        251
  ECODE_RT__RESERVED3        252
  ECODE_RT__RESERVED2        253
  ECODE_RT__RESERVED1        254
  ECODE_RT__RESERVED0        255
*/
```

The final step, which should be performed after a user C function is defined, is to fill the instruction database according to the following structure:

```
typedef struct{
  const CHR *fnc_name;  /* function name */
  const SSH operands;   /* number of arguments */
  const UCH ret_type;   /* result type: 'I', 'F', 'S', 'Z' */
  const UCH *op_type;   /* flags 'IFSZ' for every argument */
  const fcall func_ptr; /* pointer to the function */
} INSTRUCTION_STRU;
```

See an example of a user defined C function below. Passed function arguments are obtained sequentially through the incremented "dat_ptr". Internal calls to "ret_ival", "ret_fval" and "ret_sval" provide dynamic type casting if required. A direct "fcall" function invocation omits the dynamic casting and returns a universal data structure.

Finally, "my_function" is registered in the instruction database. The corresponding record states that the function has five arguments and returns an integer value. Argument types are integer, integer, float, string and integer respectively.

```
"my_function"
  (my_function <IVal> <IVal> <FVal> <SVal> <IVal>)
    --> <IVal>
```

```
void my_function(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  const ULO *tmp_ptr;
  SLO n,result=0;
  CHR *str=NULL;
  DFL *f_array,koef;
  struct fastlisp_data idat={0,1,0,0,{0},NULL,1,NULL,{NULL}};

  ret_ival(dat_ptr,&n);                  /* arg0: integer */
  ret_ival(dat_ptr+1,(SLO*)&f_array);    /* arg1: ptr to floats */
  ret_fval(dat_ptr+2,&koef);             /* arg2: float */
  ret_sval(dat_ptr+3,&str);              /* arg3: string */
  tmp_ptr=*((ULO**)(dat_ptr+4));         /* arg4: integer as */
  (*(fcall)*tmp_ptr)(tmp_ptr+1,&idat);   /*   universal data struct */

  if(noterror()){
    /* data processing to compute `result': */
    ret_dat->single=1;
    ret_dat->type='I';
    ret_dat->value.ival=result;

    /* or an error occurred: */
    rise_error_info(USER_DEFINED_UNRESERVED_ERRCODE,"ERROR_TEXT");
  }

  if(idat.disable_ptr)
    free_flp_data(&idat);
  free_string(&str);

  return;
}

INSTRUCTION_STRU INSTRUCTION_SET[]={
  /* ... */
  {"MY_FUNCTION",5,'I',(UCH*)"IIFSI",&my_function},
  /* ... */
};
const ULO INSTRUCTIONS=
          sizeof(INSTRUCTION_SET)/sizeof(INSTRUCTION_STRU);
```

And last but not least is string processing. Internally, strings are stored in the following format:

```
.-------------------------------------------------------------------------.
| <ULO string_size><string><zero_char><zero_char_alignment_to_ULO_size>    |
|                  ^                                                      |
|                  |                                                      |
| CHR *string_ptr -'                                                      |
`-------------------------------------------------------------------------'
```

The implemented set of the string processing functions is basically equal to the same set on the FastLisp level:

```
.-------------------------------------------------------------------------.
| CHR *mk_std_buff(CHR **buff, ULO size);                                 |
| CHR *mk_std_buff_secure(CHR **buff, ULO size);                          |
| CHR *mk_fst_buff(CHR **buff, ULO size);                                 |
| CHR *mk_fst_buff_secure(CHR **buff, ULO size);                          |
| CHR *get_std_buff(CHR **targ, const CHR *buff);                         |
| CHR *get_std_buff_secure(CHR **targ, const CHR *buff);                  |
| UCH notempty(const CHR *string);                                        |
| ULO len(const CHR *string);                                             |
| ULO at(const CHR *pattern, const CHR *among);                           |
| ULO rat(const CHR *pattern, const CHR *among);                          |
| UCH cmp(const CHR *string1, const CHR *string2);                        |
| SCH cmp_s(const CHR *string1, const CHR *string2);                      |
| CHR *equ(CHR **targ, const CHR *source);                                |
| CHR *equ_secure(CHR **targ, const CHR *source);                         |
| CHR *equ_num(CHR **targ, SLO num);                                      |
| CHR *equ_fnum(CHR **targ, DFL fnum);                                    |
| CHR *cat(CHR **targ, const CHR *source);                                |
| CHR *lcat(CHR **targ, const CHR *source);                               |
| CHR *space(CHR **targ, ULO pos);                                        |
| CHR *replicate(CHR **targ, const CHR *source, ULO num);                 |
| CHR *left(CHR **targ, const CHR *source, ULO pos);                      |
| CHR *leftr(CHR **targ, const CHR *source, ULO posr);                    |
| CHR *right(CHR **targ, const CHR *source, ULO pos);                     |
| CHR *rightl(CHR **targ, const CHR *source, ULO posl);                   |
| CHR *substr(CHR **targ, const CHR *source, ULO from, ULO pos);          |
| CHR *strtran(CHR **targ, const CHR *source, const CHR *pattern,         |
|              const CHR *subst);                                         |
| CHR *ltrim(CHR **targ, const CHR *source);                              |
| CHR *rtrim(CHR **targ, const CHR *source);                              |
| CHR *alltrim(CHR **targ, const CHR *source);                            |
| CHR *pack(CHR **targ, const CHR *source);                               |
| CHR *head(CHR **targ, const CHR *source);                               |
| CHR *tail(CHR **targ, const CHR *source);                               |
| CHR *lsp_head(CHR **targ, const CHR *source);                           |
| CHR *lsp_tail(CHR **targ, const CHR *source);                           |
`-------------------------------------------------------------------------'
```

```
CHR *upper(CHR **targ, const CHR *source);
CHR *lower(CHR **targ, const CHR *source);
CHR *rev(CHR **targ, const CHR *source);
CHR *padl(CHR **targ, const CHR *source, ULO width);
CHR *padr(CHR **targ, const CHR *source, ULO width);
CHR *padc(CHR **targ, const CHR *source, ULO width);
CHR *strraw(CHR **targ, const CHR *source);
CHR *strunraw(CHR **targ, const CHR *source);
CHR *strdump(CHR **targ, const CHR *source);
CHR *string_time(CHR **targ);
CHR *strings_version(CHR **targ);
CHR *sch2str(CHR **targ, SCH num);
CHR *ssh2str(CHR **targ, SSH num);
CHR *slo2str(CHR **targ, SLO num);
CHR *ptr2str(CHR **targ, void *ptr);
CHR *dfl2str(CHR **targ, DFL num);
SCH str2sch(const CHR *string);
SSH str2ssh(const CHR *string);
SLO str2slo(const CHR *string);
void *str2ptr(const CHR *string);
DFL str2dfl(const CHR *string);
CHR *free_string(CHR **targ);
```

An example of string processing is given below:

```
CHR *str0=NULL,*str1=NULL,*str2=NULL;
get_std_buff(&str0,"To be or not to be");
get_std_buff(&str1,"be");
get_std_buff(&str2,"compute");
upper(&str0,strtran(&str0,str0,str1,str2));
printf("`%s'\n",str0);  /* Result: `TO COMPUTE OR NOT TO COMPUTE' */
free_string(&str0);
free_string(&str1);
free_string(&str2);
```

This page is intentionally left blank.

Appendix B
Example of Application Programming

This is a simple application example, which computes direct and inverse 2D nonseparative discrete Hartley transforms (DHT and IDHT) according to the computation flow shown in Figure B-1.

$$H[p,q]=\sum_{n=0}^{N-1}\sum_{m=0}^{M-1}x[n,m]\cdot cas\left(\frac{2\cdot\pi}{N}\cdot p\cdot n+\frac{2\cdot\pi}{M}\cdot q\cdot m\right),$$

$$p=0..N-1,\ q=0..M-1$$

$$H_{inv}[p,q]=\frac{1}{N\cdot M}H[p,q]$$

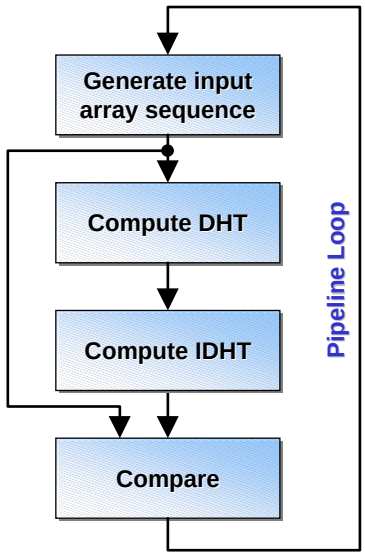

Figure B-1. Computation example of discrete Hartley transforms

Four functions from Figure B-1 can be implemented in C as follows:

```
void func__dhtpipe0_generate(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  SLO i,j,n,m;
  DFL *array;
  ret_ival(dat_ptr,(SLO*)&array);
  ret_ival(dat_ptr+1,&n);
  ret_ival(dat_ptr+2,&m);
  if(noterror()){
    for(i=0;i<n;i++)
      for(j=0;j<m;j++)
        *(array+i*m+j)=1.*rand()/RAND_MAX;
    ret_dat->single=1;
    ret_dat->type='I';
    ret_dat->value.ival=(SLO)array;
  }
  return;
}

void func__dhtpipe0_dht(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  SLO i,j,p,q,n,m;
  DFL *target_array,*source_array,pi,c1,s1,sum,tmp;
  ret_ival(dat_ptr,(SLO*)&target_array);
  ret_ival(dat_ptr+1,&n);
  ret_ival(dat_ptr+2,&m);
  ret_ival(dat_ptr+3,(SLO*)&source_array);
  if(noterror()){
    pi=3.1415926535897932;
    c1=2*pi/n;
    s1=2*pi/m;
    for(p=0;p<n;p++)
      for(q=0;q<m;q++){
        sum=0;
        for(i=0;i<n;i++)
          for(j=0;j<m;j++)
            sum+=(*(source_array+i*m+j)*(cos(tmp=c1*p*i+s1*q*j)+sin(tmp)));
        *(target_array+p*m+q)=sum;
      }
    ret_dat->single=1;
    ret_dat->type='I';
    ret_dat->value.ival=(SLO)target_array;
  }
  return;
}
```

```
void func__dhtpipe0_idht(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  SLO i,j,p,q,n,m;
  DFL *target_array,*source_array,pi,c1,s1,sum,tmp;
  ret_ival(dat_ptr,(SLO*)&target_array);
  ret_ival(dat_ptr+1,&n);
  ret_ival(dat_ptr+2,&m);
  ret_ival(dat_ptr+3,(SLO*)&source_array);
  if(noterror()){
    pi=3.1415926535897932;
    c1=2*pi/n;
    s1=2*pi/m;
    for(p=0;p<n;p++)
      for(q=0;q<m;q++){
        sum=0;
        for(i=0;i<n;i++)
          for(j=0;j<m;j++)
            sum+=(*(source_array+i*m+j)*(cos(tmp=c1*p*i+s1*q*j)+sin(tmp)));
        *(target_array+p*m+q)=sum/n/m;
      }
    ret_dat->single=1;
    ret_dat->type='I';
    ret_dat->value.ival=(SLO)target_array;
  }
  return;
}

void func__dhtpipe0_compare(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  SLO result=1,n,m;
  DFL *array0,*array1;
  ret_ival(dat_ptr,(SLO*)&array0);
  ret_ival(dat_ptr+1,(SLO*)&array1);
  ret_ival(dat_ptr+2,&n);
  ret_ival(dat_ptr+3,&m);
  if(noterror()){
    if((fabs(*array0-*array1)>1.e-10)
       ||(fabs(*(array0+n*m-1)-*(array1+n*m-1))>1.e-10))
      result=0;
    ret_dat->single=1;
    ret_dat->type='I';
    ret_dat->value.ival=result;
  }
  return;
}

INSTRUCTION_STRU INSTRUCTION_SET[]={
  {"DHTPIPE0_GENERATE",3,'I',(UCH*)"III", &func__dhtpipe0_generate},
  {"DHTPIPE0_DHT",     4,'I',(UCH*)"IIII",&func__dhtpipe0_dht},
  {"DHTPIPE0_IDHT",    4,'I',(UCH*)"IIII",&func__dhtpipe0_idht},
  {"DHTPIPE0_COMPARE", 4,'I',(UCH*)"IIII",&func__dhtpipe0_compare}
};
const ULO INSTRUCTIONS=sizeof(INSTRUCTION_SET)/sizeof(INSTRUCTION_STRU);
```

After recompilation of the virtual machine the following trivial implementation of the function main() will work properly. Please notice two things:

- Some speculative SMP RISC architectures require aligned memory addresses for allocated float arrays in the shared memory pool.
- Access to the asynchronous heaps can be synchronized e.g. via some additional synchronization helper variables that create artificial dataflow dependencies enabling parallel function calls where possible:
  - every pointer variable "*_addr*" has a corresponding synchronization helper variable "*_sync*";
  - an "*_addr*" pointer variable to a read-only area can be passed directly to a function call;
  - an "*_addr*" pointer variable to a modified area is passed to a function call being dependent on the corresponding "*_sync*" helper variable;
  - after the function call is finished, both "*_sync*" helper variables (for all pointers involved in the call) and "*_addr*" pointer variables (only for pointers to the modified areas) are re-assigned.

```
# dhtpipe0.flp

# Pipeline calculation of the 2D nonseparative Hartley transform.
#
#  Direct Hartley transform:
#
#            N-1  M-1               2Pi      2Pi
#   H[p,q] =  E    E  x[n,m] cas ( --- pn + --- qm ), p=0..N-1, q=0..M-1.
#            n=0  m=0               N        M
#
#  Inverse Hartley transform:
#
#                1
#   Hinv[p,q] =  --- H[p,q].
#               N*M

(progn # main()

  (outf
    "Pipeline calculation of the 2D nonseparative Hartley transform.\n\n" 0)
```

```
(setq m (ival (accept "M-value of M*N-matrix: ")))
(setq n (ival (accept "N-value of M*N-matrix: ")))
(setq numb (ival (accept "How many input data packs: ")))
(setq arrays_size (* (++ (* m n)) (setq floatlen (len (dump_f2s 0.)))))

(for i 1 1 numb (progn  # main pipeline loop

  (outf "Sequence %ld:" i)

  ######################################################
  # 1. This is the way how we create an input array. #
  # 1.1. This is the way how we get address.         #
  ######################################################
  (setq inp_array_addr (setq inp_array_addr_
    (asyncheap_create arrays_size)))

  ######################################################################
  # 1.2. This is the way how we align the address.                     #
  #      (optional: float might be misaligned on some architectures). #
  ######################################################################
  (setq inp_array_sync (& 0
    (setq inp_array_addr
      (+ inp_array_addr (- floatlen (iabs (% inp_array_addr floatlen)))))
  ))


  ######################################################
  # 2. This is the way how we generate input sequence. #
  ######################################################
  (setq inp_array_sync (& 0
    (setq inp_array_addr
      (dhtpipe0_generate (| inp_array_sync inp_array_addr) n m))
  ))

  ################################################
  # 3. This is the way how we create a DHT array. #
  # 3.1. This is the way how we get address.      #
  ################################################
  (setq dht_array_addr (setq dht_array_addr_
    (asyncheap_create arrays_size)))

  ######################################################################
  # 3.2. This is the way how we align the address.                     #
  #      (optional: float might be misaligned on some architectures). #
  ######################################################################
  (setq dht_array_sync (& 0
    (setq dht_array_addr
      (+ dht_array_addr (- floatlen (iabs (% dht_array_addr floatlen)))))
  ))
```

```
    ########################################
    # 4. This is the way how we compute DHT. #
    ########################################
    (setq dht_array_sync (setq inp_array_sync (& 0
      (setq dht_array_addr
        (dhtpipe0_dht (| dht_array_sync dht_array_addr) n m inp_array_addr))
    )))

    ################################################
    # 5. This is the way how we create an IDHT array. #
    # 5.1. This is the way how we get address.        #
    ################################################
    (setq idht_array_addr (setq idht_array_addr_
      (asyncheap_create arrays_size)))

    ##########################################################################
    # 5.2. This is the way how we align the address.                         #
    #      (optional: float might be misaligned on some architectures). #
    ##########################################################################
    (setq idht_array_sync (& 0
      (setq idht_array_addr
        (+ idht_array_addr (- floatlen (iabs (% idht_array_addr floatlen)))))
    ))

    ########################################
    # 6. This is the way how we compute IDHT. #
    ########################################
    (setq idht_array_sync (setq dht_array_sync (& 0
      (setq idht_array_addr
        (dhtpipe0_idht (| idht_array_sync idht_array_addr) n m dht_array_addr))
    )))

    ##########################################################
    # 7. This is the way how we compare input_sequence with #
    #    IDHT(DHT(input_sequence)).                          #
    ##########################################################
    (setq inp_array_sync (setq idht_array_sync (& 0
      (setq cmp_res (dhtpipe0_compare inp_array_addr idht_array_addr n m))
    )))
    (outf " %s.\n" (if cmp_res "Ok" "Fail"))

    ############################################
    # 8. This is the way how we release memory. #
    ############################################
    (asyncheap_delete (| inp_array_sync inp_array_addr_))
    (asyncheap_delete (| dht_array_sync dht_array_addr_))
    (asyncheap_delete (| idht_array_sync idht_array_addr_))

  ))  # "end progn" of main pipeline loop; "end for" of main pipeline loop
  ""

)  # "end progn" of main()
```

The above version of the application exploits a coarse-grain parallelism of the iterations of the main pipeline loop. In the next version of the same application the computation loops written in C are interleaved. This modification enables using a coarse-grain parallelism within each iteration of the main pipeline loop.

```
void func__dhtpipe1_generate(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  SLO step,interleave,i,j,n,m;
  DFL *array;
  ret_ival(dat_ptr,(SLO*)&array);
  ret_ival(dat_ptr+1,&step);
  ret_ival(dat_ptr+2,&interleave);
  ret_ival(dat_ptr+3,&n);
  ret_ival(dat_ptr+4,&m);
  if(noterror()){
    for(i=step;i<n;i+=interleave)
      for(j=0;j<m;j++)
        *(array+i*m+j)=1.*rand()/RAND_MAX;
    ret_dat->single=1;
    ret_dat->type='I';
    ret_dat->value.ival=0;
  }
  return;
}

void func__dhtpipe1_dht(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  SLO step,interleave,i,j,p,q,n,m;
  DFL *target_array,*source_array,pi,c1,s1,sum,tmp;
  ret_ival(dat_ptr,(SLO*)&target_array);
  ret_ival(dat_ptr+1,&step);
  ret_ival(dat_ptr+2,&interleave);
  ret_ival(dat_ptr+3,&n);
  ret_ival(dat_ptr+4,&m);
  ret_ival(dat_ptr+5,(SLO*)&source_array);
  if(noterror()){
    pi=3.1415926535897932;
    c1=2*pi/n;
    s1=2*pi/m;
    for(p=step;p<n;p+=interleave)
      for(q=0;q<m;q++){
        sum=0;
        for(i=0;i<n;i++)
          for(j=0;j<m;j++)
            sum+=(*(source_array+i*m+j)*(cos(tmp=c1*p*i+s1*q*j)+sin(tmp)));
        *(target_array+p*m+q)=sum;
      }
    ret_dat->single=1;
    ret_dat->type='I';
    ret_dat->value.ival=0;
  }
  return;
}
```

```
void func__dhtpipe1_idht(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  SLO step,interleave,i,j,p,q,n,m;
  DFL *target_array,*source_array,pi,c1,s1,sum,tmp;
  ret_ival(dat_ptr,(SLO*)&target_array);
  ret_ival(dat_ptr+1,&step);
  ret_ival(dat_ptr+2,&interleave);
  ret_ival(dat_ptr+3,&n);
  ret_ival(dat_ptr+4,&m);
  ret_ival(dat_ptr+5,(SLO*)&source_array);
  if(noterror()){
    pi=3.1415926535897932;
    c1=2*pi/n;
    s1=2*pi/m;
    for(p=step;p<n;p+=interleave)
      for(q=0;q<m;q++){
        sum=0;
        for(i=0;i<n;i++)
          for(j=0;j<m;j++)
            sum+=(*(source_array+i*m+j)*(cos(tmp=c1*p*i+s1*q*j)+sin(tmp)));
        *(target_array+p*m+q)=sum/n/m;
      }
    ret_dat->single=1;
    ret_dat->type='I';
    ret_dat->value.ival=0;
  }
  return;
}

INSTRUCTION_STRU INSTRUCTION_SET[]={
  // ...
  {"DHTPIPE1_GENERATE",5,'I',(UCH*)"IIIII", &func__dhtpipe1_generate},
  {"DHTPIPE1_DHT",     6,'I',(UCH*)"IIIIII",&func__dhtpipe1_dht},
  {"DHTPIPE1_IDHT",    6,'I',(UCH*)"IIIIII",&func__dhtpipe1_idht},
  // ...
};
const ULO INSTRUCTIONS=sizeof(INSTRUCTION_SET)/sizeof(INSTRUCTION_STRU);
```

In the function main() all interleaved functions are called in a way of the threaded loops:

```
# dhtpipe1.flp

# Pipeline calculation of the 2D nonseparative Hartley transform.
#
#  Direct Hartley transform:
#
#            N-1  M-1              2Pi      2Pi
#   H[p,q] =  E    E  x[n,m] cas ( --- pn + --- qm ), p=0..N-1, q=0..M-1.
#            n=0  m=0               N        M
#
#  Inverse Hartley transform:
#
#                 1
#   Hinv[p,q] =  --- H[p,q].
#               N*M

(progn  # main()

  (outf
    "Pipeline calculation of the 2D nonseparative Hartley transform.\n\n" 0)
  (setq m (ival (accept "M-value of M*N-matrix: ")))
  (setq n (ival (accept "N-value of M*N-matrix: ")))
  (setq numb (ival (accept "How many input data packs: ")))
  (setq arrays_size (* (++ (* m n)) (setq floatlen (len (dump_f2s 0.)))))
  (setq threads (if (< n (n_cpuproc)) n (n_cpuproc)))

  (for i 1 1 numb (progn  # main pipeline loop

    (outf "Sequence %ld:" i)

    ######################################################
    # 1. This is the way how we create an input array. #
    # 1.1. This is the way how we get address.         #
    ######################################################
    (setq inp_array_addr (setq inp_array_addr_
      (asyncheap_create arrays_size)))

    ######################################################################
    # 1.2. This is the way how we align the address.                     #
    #      (optional: float might be misaligned on some architectures). #
    ######################################################################
    (setq inp_array_sync (& 0
      (setq inp_array_addr
        (+ inp_array_addr (- floatlen (iabs (% inp_array_addr floatlen)))))
    ))
```

```
#####################################################
# 2. This is the way how we generate input sequence. #
#####################################################
(setq inp_array_addr (| inp_array_sync inp_array_addr))
(for thread 1 1 threads
  (setq inp_array_sync (+ inp_array_sync
    (dhtpipe1_generate inp_array_addr (-- thread) threads n m)))
)
(setq inp_array_addr (| inp_array_sync inp_array_addr))

################################################
# 3. This is the way how we create a DHT array. #
# 3.1. This is the way how we get address.      #
################################################
(setq dht_array_addr (setq dht_array_addr_
  (asyncheap_create arrays_size)))

#########################################################################
# 3.2. This is the way how we align the address.                        #
#      (optional: float might be misaligned on some architectures). #
#########################################################################
(setq dht_array_sync (& 0
  (setq dht_array_addr
    (+ dht_array_addr (- floatlen (iabs (% dht_array_addr floatlen)))))
))

########################################
# 4. This is the way how we compute DHT. #
########################################
(setq dht_array_addr (| dht_array_sync dht_array_addr))
(for thread 1 1 threads
  (setq dht_array_sync (+ dht_array_sync
    (dhtpipe1_dht dht_array_addr (-- thread) threads n m inp_array_addr)))
)
(setq inp_array_sync dht_array_sync)
(setq dht_array_addr (| dht_array_sync dht_array_addr))

################################################
# 5. This is the way how we create an IDHT array. #
# 5.1. This is the way how we get address.        #
################################################
(setq idht_array_addr (setq idht_array_addr_
  (asyncheap_create arrays_size)))
```

```
    ######################################################################
    # 5.2. This is the way how we align the address.                     #
    #      (optional: float might be misaligned on some architectures). #
    ######################################################################
    (setq idht_array_sync (& 0
      (setq idht_array_addr
        (+ idht_array_addr (- floatlen (iabs (% idht_array_addr floatlen)))))
    ))

    ##########################################
    # 6. This is the way how we compute IDHT. #
    ##########################################
    (setq idht_array_addr (| idht_array_sync idht_array_addr))
    (for thread 1 1 threads
      (setq idht_array_sync (+ idht_array_sync (dhtpipe1_idht
        idht_array_addr (-- thread) threads n m dht_array_addr)))
    )
    (setq dht_array_sync idht_array_sync)
    (setq idht_array_addr (| idht_array_sync idht_array_addr))

    ########################################################
    # 7. This is the way how we compare input_sequence with #
    #    IDHT(DHT(input_sequence)).                         #
    ########################################################
    (setq inp_array_sync (setq idht_array_sync (& 0
      (setq cmp_res (dhtpipe0_compare inp_array_addr idht_array_addr n m))
    )))
    (if cmp_res
      (outf " Ok.\n" nil)
      (outf " Fail.\n" nil)
    )

    ##########################################
    # 8. This is the way how we release memory. #
    ##########################################
    (asyncheap_delete (| inp_array_sync inp_array_addr_))
    (asyncheap_delete (| dht_array_sync dht_array_addr_))
    (asyncheap_delete (| idht_array_sync idht_array_addr_))

  ))  # "end progn" of main pipeline loop; "end for" of main pipeline loop
  ""

)  # "end progn" of main()
```

This page is intentionally left blank.
This is the last page of the document.