The DAMSON Loader Notes (Version 1.3)

Last Edited: Paul Richmond (09/02/15)

Table of Contents

Introduction 1
Requirements 2
Loader Usage 2
Mapping 3
Routing 4
Interrupt Vectors 5
Logging 5
  Runtime Log Descriptions 6
  Loader Logging Descriptions 6
Loading and the SpiNNaker Memory Layout 6
Simulation 9
References 10

Introduction

The DAMSON loader is responsible for taking the outputs of the DAMSON code generator and executing them on the SpiNNaker hardware (see Figure 1). This requires a process or mapping DAMSON nodes to appropriate cores, initialising cores with the correct data and code and coordinating the execution of a DAMSON program on the SpiNNaker hardware device.

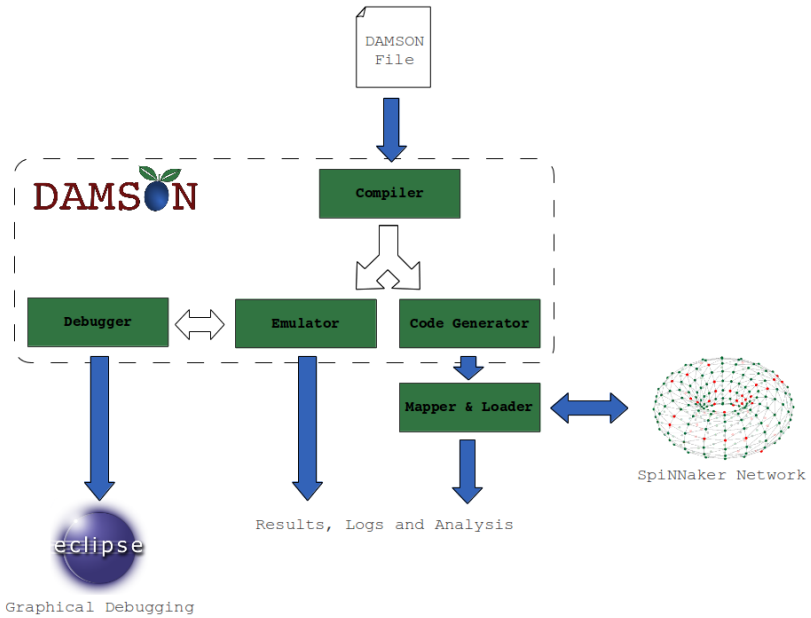

Figure 1: Overview of the DAMSON toolchain

Requirements

The DAMSON loader executable expects a single argument, a loader file (*.ldr) output by the DAMSON code generator. This loader file will have been generated by the code generator alongside a number of prototype module objects and a makefile (dmake) which links them with the DAMSON runtime. It is important that dmake has been executed and that the linked prototype SpiNNaker modules (*.aplx) are located within the same working directory as the loader file. The format of the loader file is described in the code generator documentation but briefly consists of a DAMSON alias data made up of; initial global data values, SDRAM (external) data values, a set of DAMSON interrupts and a set of logging descriptions.

Two additional files are required by the loader which must be in the working directory;

- spinnaker.ini - An additional configuration file (spinnaker.ini) is also required in the working directory of the loader executable. This file is expected to contain the IP address of the SpiNNaker hardware and the x and y dimension of the chip layout (space separated on the first line). Future versions will allow a subsequent line separated set of dead cores which should be ignored during the mapping process. For the 4 chip SpiNNaker boards this file will contain the plain text 192. 168. 0. 52 2 2. This specifies the address of the SpiNNaker board and 2D chip layout (e.g. 2x2).
- boot.bin - The latest version of the SpiNNaker SCAMP boot image.

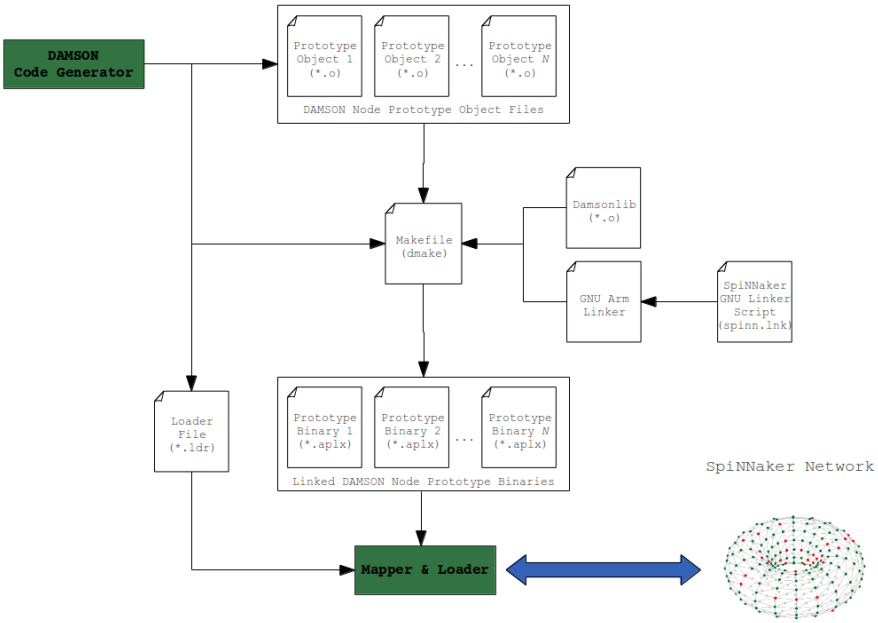

Figure 2 - The DAMSON loader (and mapper) inputs and relationship with the DAMSON code generator

Loader Usage

The DAMSON code generator outputs a loader file (*.ldr) and makefile (dmake) and a number of DAMSON prototype nodes binary objects (*.o). These files should be placed in the damson_loader

directory and linked with damsonlib using the dmake file as follows;

```
make -f dmake
```

The result will be that each DAMSON prototype object will be linked and rearranged ready for loading to a file with the *.aplx extension. The damson loader can be built from source using the provided makefile. i.e.

```
make -f loader.make
```

This will produce an executable file 'loader'. The loader has only one input, a loader file from the DAMSON code generator. Using this file the loader will: transfer alias data and the compiled and linked *.aplx files to the hardware, start program execution, relay any debug output to the console and upon completion save any log data to local files.

Using the modified (two node with different loop sizes) test22.d example, the loader can be executed as follows;

```
loader test22.ldr
```

The following output should be provided in the console;

```
Connected to SpiNNaker version 0.96
200   f(1)=1
100   f(1)=1
200   f(2)=2
100   f(2)=2
Node (200) exit 99
100   f(3)=6
100   f(4)=24
100   f(5)=120
100   f(6)=720
100   f(7)=5040
100   f(8)=40320
100   f(9)=362880
100   f(10)=3628800
Node (100) exit 99
SpiNNaker ticks: 0
SpiNNaker time: 2749 ms
Loading time: 5507 ms
```

Mapping

SpiNNaker cores are addressed in the following ‘x, y, z’ format where ‘x’ and ‘y’ are the x and y position of the SpiNNaker chip respectively and z is a CPU core number in the range 0 to 17. SpiNNaker CPU cores have both a physical and virtual address. Physical addresses are entirely abstracted from all aspects of the loader and are only translated from their virtual counterparts

within DAMSONlib where required. Each CPU core at virtual address 0 is always reserved by the SCAMP system as a monitor core and is responsible for handling inter chip communication and debug output. Each subsequent core is assigned a virtual core number sequentially during the boards start-up phase. As it is expected that SpiNNaker contains a number of dead or inactive cores, the DAMSON loader and runtime system assume that there are 16 cores available per chip for DAMSON program execution (with virtual addresses 1-16).

An optimum processes for mapping DAMSON cores to SpiNNaker addresses would globally balance communication across active cores to minimise the number of routing entries and reduce packet transmission distances. This is difficult optimisation processes which draws parallels with the quadratic assignment problem and requires heuristic algorithms to find a near (or locally) optimal solution. As information about the global communication of all cores is required for this type of mapping this is gathered from the loader file (in a separate first pass) and passed to the mapping function in a minimal format (i.e. node numbers with a list of associated node interrupts).

The current mapping process does not perform any optimisation and uses simple linear assignment. The linear assignment method simply starts handing out addresses (starting from 0,0,1) by incrementing the core number until a chip is full and then moving to a new chip (first in the y direction and then in the x direction). By assigning virtual core addresses in this way, it is guaranteed that any chip with active DAMSON processing cores will always utilise the CPU at address 1 (i.e. the root DAMSON core) and that the root chip (0, 0) will similarly be utilised. This is important as this CPU will perform chip level initialisation required by all cores (such as setting routing tables). Any future mapping algorithm must also ensure these constrains are met.

During the mapping processes a ‘core map’ is recorded for each chip to indicate which CPUs are active. This core map is required by the root DAMSON core to allow it co-ordinate system synchronisation, both locally between CPUs on the same chip and globally with the root chip (0,0,1). The core map format is an array (with a size determined by the width and height of the chip layout size) of 32bit map values. Within each map value each bit location corresponds to the virtual core address number. E.g. The single map value **0x16** has the following bit field;

0000 0000 00000 0000 0000 0000 000**1** 0**11**0

This indicates that CPU cores **1**,**2**, and **4** are active. Bit 0 represents the monitor core and its value is ignored as are any bits locations beyond 17th most significant bit.

Routing

Routing tables are generated after the system has been completely mapped as system wide knowledge of the DAMSON to SpiNNaker address mappings is required to generating routing keys. Routing tables are generated per chip rather than per core due to the multicast nature of routing. A single routing entry can be used to either route a packet off chip in multiple directions or interrupt multiple CPU cores within the same chip. The DAMSON loader routing table format for a single chip consists of an integer indicating the number of routing entries followed by a set of routing entry value pairs. A single value pair contains an interrupt number (the originating DAMSON node number) and a 32 bit routing key. Routing keys utilise bits 0-5 to act as binary flags for off chip routing and bits 6-22 to act as binary interrupt flags for virtual CPU core addresses (see Table 1). These are translated to routing entries with physical addresses (but using the same routing key format) by the runtime system during initialisation.

Routing Key Bit	Route
0	East
1	North East
2	North
3	West
4	South-West
5	South
6	CPU Core 0 (monitor)
7	CPU Core 1
…	…
23	CPU Core 17

Table 1: Routing key bit fields

Interrupt Vectors

The purpose of the Interrupt vector is to provide a translation between an interrupt from a given DAMSON core number to a specific code position. In order to provide an efficient lookup mechanism a hash table structure is loaded to local SpiNNaker memory (DTCM). For simplicity this uses simple linear probing of the DAMSON node number to resolve hash collisions. As SpiNNaker has no native divide support the total hash table size is limited to a power of 2 number to allow an efficient modulus operator to be implemented on the ARM architecture. The hash table is populated by the loader during the loading stage from the list of interrupt, code offset pairs in the loader file. An additional interrupt value is located at the the beginning of the interrupt vector (before the hash table) to hold an interrupt from the timer (which has a special interrupt node number of 0). If no timer interrupt is specified this has a code offset of 0.

Logging

Logging within DAMSON takes the form of either a 'log' or 'snapshot' which both have the same directive format. Functionally, logs are recorded (as a log entry item to SDRAM) on clock interrupts (either every interrupt or according to the frequency specified in the log definition) and snapshots are recorded on calls to the sendpkt system function (the frequency is ignored).

The loader has two responsibilities with respect to logging. Firstly, the logger must create and load a set of log/snapshot descriptions for use within the runtime system. The runtime log descriptions are checked on timer interrupts and sendpkt calls to see which values should be logged (if any). Secondly the loader must store descriptions of the logging formats and output locations so that upon simulation completion, log values can be retrieved from the hardware and output to files using the desired format.

The loader file format contains the information required to generate both loader and runtime log descriptions. Both log descriptions have a handle value so that recorded log entries from the runtime system can be tied back to an appropriate loader log description.

Runtime Log Descriptions

Runtime log descriptions use the following C structure format to hold the logging window and interval times as well as the global variable items which should be logged. Global variable items are saved in the log_globals array as runtime SpiNNaker DTCM addresses (see loading and memory layout section). The loader builds a separate list of RuntimeLogItems for logs and snapshots as each list is used within different parts of the runtime system.

```
typedef struct
{
        uint    handle;
        uint    start_time;     //us
        uint    end_time;       //us
        uint    interval; //us
        int     interval_count; //used at runtime
        uint    log_items;
        int     log_globals[MAX_LOG_ITEMS];
} RuntimeLogItem;
```

During runtime execution, individual log entries are saved to SDRAM with a handle a number of log items and the value of the log globals. SDRAM is filled with log entries sequentially (from the space allocated to the CPU core renaming after any user externals) by any RuntimeLogItem. The handle value is used to associate any particular log entry to a given log format. The exact process of logging individual log entry items is described further within the runtime system documentation.

Loader Logging Descriptions

The loader log descriptions do not need any information regarding the logs time window or intervals, only the log format and output location. Log entries are read back from SDRAM after execution completion and the handle is used to ensure they are output to the correct location using the desired output format. A custom *fprintf* function provided to ensure that fixed point values are output in decimal notation (as in the emulator) rather than interpreted as integers.

Loading and the SpiNNaker Memory Layout

During the loading stage each SpiNNaker core mapped to a DAMSON node is initialised with zero data and then set according to data from the loader file. This includes any global or external data, the interrupt vector hash structure, runtime log item descriptions and the routing tables.

SpiNNaker has a number of unique memory spaces including; Instruction Tightly Coupled Memory (ITCM), Data Tightly Coupled Memory (DTCM) and Synchronous Dynamic Random Access Memory (SDRAM). The first atsk of the loader is to zero initialise any areas of DTCM and SDRAM space used by DAMSON. This is achieved by constructing and executing an APLX command table (see the APLX specification document) in an area of shared system memory (at 0xf500000). This table has a series of memory set commands which cause the core to set 0 values when the APLX table is processed. Following zero initialisation, various copy and execute commands are issued to physically load each core. SpiNNaker commands use SpiNNakers native SpiNNaker Command Packet (SCP) format (see SCP specification document) which are sent as SpiNNaker Datagram Packets (SDP) packets (see SDP specification document). To simplify this process a C library (the SpiNNaker Runtime API) is utilised by the loader which wraps a number of common commands including;

- Loading of Memory – Loads a number of bytes at a given memory to a given SpiNNaker memory location by chucking data into SDP packets.
- Read Memory – Reads a number of bytes to local (host) memory address from a given SpiNNaker memory location by retrieving a set of cunked SDP packets.
- Intelligent Load Memory – Loads only none zero bytes of memory at a given memory to a given SpiNNaker memory location by chucking data into SDP packets.
- Starting Program Execution – This will cause the hardware to decode an APLX command table at a given address. An APLX command table consists of a number of memory copy, memory set and program register set commands allowing the core to effectively begin program execution.

At the start of DTCM is a small amount of reserved runtime space for the SpiNNaker C environment (for holding runtime C globals). Following this, the DAMSON global vector consists of 50 words of reserved space (for important values and addresses) followed by the user (or program) globals from the loader file, the interrupt hash structure and finally the runtime log item descriptions. The remainder of DTCM is used by the loader for holding the aplx code module (at the address 0x403800, see figure 1) and upon execution (which does not start until the entire system is loaded) is recycled by the runtime system as runtime stack space.

The 50 reserved DAMSON words are summarised below in table **. A number of these are set by the loader.

Global Index	Description
0	Total global vector size in words (including 50 reserved words)
1	Address of sdiv function
2	Address of fpmult function
3	Address of fpdiv function
4	Address of vcopy function
5	Interrupt Vector hash table size (power of 2 plus 1)
6	
7	
8	Number of logs
9	Number of snapshots
10	
11	Address of sendpkt system procedure
12	Address of delay system procedure
13	Address of printf system procedure
14	Address of exit system procedure
15	Address of signal system procedure
16	Address of wait system procedure
17	Address of tickrate system procedure

18	Address of putbyte system procedure
19	Address of putword system procedure
20	Address of readsdram system procedure
21	Address of writesdram system procedure
22	Address of syncnodes system procedure
23	
24	Debug mode for the node
25	End address of log entries in SDRAM (set by runtime on exit)
26	
27	
28	
29	
30	
31	Address of getclk system function
32	
33	
34	Address of createthread system function
35	Address of deletethread system function
36	Address of getbyte system function
37	Address of getword system function
38	Address of bitset system function
39	
40	Interrupt Vector hash table start address
41	
42	
43	Log list start address
44	Snapshot list start address
45	
46	
47	
48	Number of active SpiNNaker chips
49	DAMSON Node number

Table 2: DAMSON Reserved Table (Red indicates values set by the loader, otherwise values are set during runtime)

SpiNNaker SDRAM is partitioned between 16 usable CPU cores each with 7 megabytes of addressable space each. The first entry within the SDRAM space for each core is the external vector

size required by the cores program. The loader sets this value and copies all external data. All remaining space is used for storing log entry values. The remaining 16 megabytes of SDRAM is allocated as 'shared' system space and is loaded with system data including core maps and routing tables (required by the root DAMSON core).

Simulation

Following complete loading of the SpiNNaker system according to the loader file the loader is responsible for starting execution and providing debug output. In order to achieve this any loaded cores are first started by issuing an APLX start command on the loaded aplx modules stored within DTCM. This will cause the runtime system and damson code to be copied to ITCM, and C globals to be copied to the reserved C runtime space in DTCM and for the runtime system to begin executing on each core. The exact start-up timing each each core is not important as first job of the runtime system is to synchronise all cores before starting the damson main function.

The loader handles debug output by creating a polling thread which waits for any printf output from the program (or diagnostic output from the runtime system) on port 17892. Any printf packets are output to the console in the same format as the emulator (i.e. prefixed with the DAMSON node number). A specially formatted printf packet can also be detected which indicates that all SpiNNaker cores have completed execution and hence the DAMSON program has completed. Upon receiving this command the loader will processes any log data and then exit.

References

SDP Specification - http://solem.cs.man.ac.uk/documentation/spinn-app-4.pdf
SCP Speficiation - http://solem.cs.man.ac.uk/documentation/spinn-app-5.pdf
APLX File Format - http://solem.cs.man.ac.uk/documentation/spinn-app-3.pdf