Interface between PetaSim V0.3 and Emulators

Yuhong Wen

Northeast Parallel Architecture Center (NPAC)
Syracuse University
111 College Place
Syracuse, NY, 13244
{wen}@npac.syr.edu

Interfacing PetaSim and Emulators requires the following:

- a common target machine description format (nodeset, linkset),
- a common application description format (dataset, distribution, execution script).

a. Target Machine Description

In PetaSim, nodeset and linkset objects are used to define the computer architecture and memory hierarchies within each architecture. PetaSim reads architecture description from two files, which correspond to nodeset and linkset descriptions. In order to interface with Emulators, PetaSim needs to enhance the target machine description to include format to identify members of nodesets and linksets.

The PetaSim V0.3 will take the following formats of input data as nodeset and linkset description: (The inner structure of nodeset and linkset classes will be enhanced to support the member identification)

nodeset.ini: (Nodeset description file)

nodeset_name type number grainsize portcount (floatspeed) // CPU nodeset optional
node0_name portname0 link00_name portname1 link01_name ....
(total "portcount" of nodeset-linkset pairs)
node1_name portname0 link10_name portname1 link11_name ....
....
(total "number" of nodeset definitions)

(another nodeset define)
....

==

linkset.ini: (Linkset description file)

linkset_name type duplex number latency bandwidth
link0_name nodeset_00 nodeset_01
link1_name nodeset_!0 nodeset_11
....
(total "number" of nodeset-link-nodeset pairs)

(another linkset define)
....

==

For Example:

In Fig1, a 4-processors distributed-shared memory architecture, we can define this architecture in the following nodeset file and linkset file:

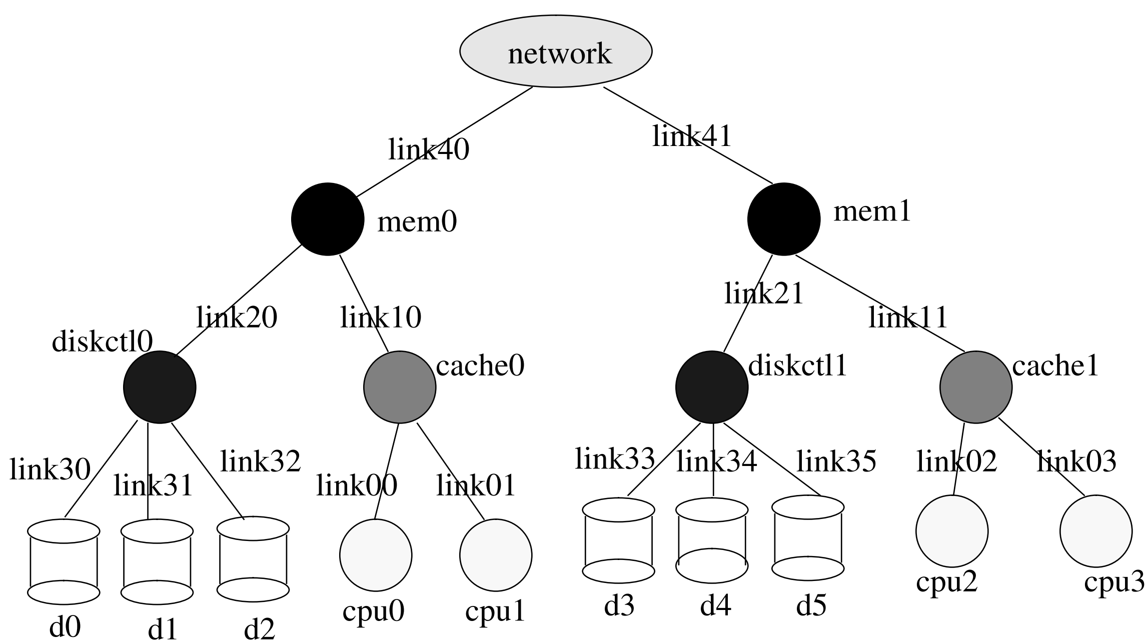

Fig1: A 4-processors Distributed - Shared Memory System

nodeset.ini: (Nodeset description)

cpu CPU 4 64 1 1.0e-8
cpu0 cache0 link00
cpu1 cache0 link01
cpu2 cache1 link02
cpu3 cache1 link03

cache Cache 2 1048576 3
cache0 cpu0 link00 cpu1 link01 mem0 link10
cache1 cpu2 link02 cpu3 link03 mem1 link11

mem Memory 2 134217628 3
mem0 cache0 link10 diskctl0 link20 network link40
mem1 cache1 link11 diskctl1 link21 network link41

diskctl Disk 2 13421762800 4
diskctl0 mem0 link20 d0 link30 d1 link31 d2 link32
diskctl1 mem1 link21 d3 link33 d4 link34 d5 link35

disks Disk 6 13421762800 1
d0 diskctl0 link30
d1 diskctl0 link31
d2 diskctl0 link32
d3 diskctl1 link33
d4 diskctl1 link34
d5 diskctl1 link35

network Switch 1 0 2
network0 mem0 link40 mem1 link41

linkset.ini: (Linkset description)

link0 Updown Full 4 0.001 104857600
link00 cpu0 cache0
link01 cpu1 cache0
link02 cpu2 cache1
link03 cpu3 cache1

link1 Updown Full 2 0.001 104857600
link10 cache0 mem0
link11 cache1 mem1

link2 Updown Full 2 0.05 10485760
link20 mem0 diskctl0
link21 mem1 diskctl1

link3 Updown Full 6 0.05 10485760
link30 diskctl0 d0

link31 diskctl0 d1
link32 diskctl0 d2
link33 diskctl1 d3
link34 diskctl1 d4
link35 diskctl1 d5

link4 Updown Full 2 0.1 512000
link40 mem0 network
link41 mem1 network

b. Application Description

Currently, PetaSim reads application description from three files which correspond to dataset, distribution and execution script. Emulators will provide the dataset descriptions and execution script. Same as the above, we also need to enhance the execution script statements to include the formats to identify the members of nodesets and linksets in the architecture.

dataset.ini: (Dataset description file)

name type totalsize bytesperunit floatperunit operationsperunit
....

==

dist.ini: (Distribution description file)

name type
....

exec.script: (Execution Script)

Current version of PetaSim has the following format of statement:

- o **send** DATAFAMILY **from** MEM-LEVEL-L **to** MEM-LEVEL-K
- o **move** DATAFAMILY **from** MEM-LEVEL-L **to** MEM-LEVEL-K

In order to identify the members in nodesets and linksets, we will enhance the statement formats in the following:

- **send** {DATAFAMILY_SET} **from** {MEM-LEVEL-L_SET} **to** {MEM-LEVEL-K_SET}
- **move** {DATAFAMILY_SET} **from** {MEM-LEVEL-L_SET} **to** {MEM-LEVEL-K_SET}

where DATAFAMILY_SET is a set of dataset identification names, and MEM-LEVEL-L_SET is a set nodeset name identifications in MEM-LEVEL-L. The *i*th dataset in DATAFAMILY_SET will be *move* / *send* from *i*th nodeset in MEM-LEVEL-L_SET to the *i*th nodeset in MEM-LEVEL-K_SET. DATAFAMILY_SET and MEM-LEVEL_SET can contain only one element, which means the datamovement of a specific dataset in the memory hierarchy, or contain all the members, which means the whole level dataset movement.

In this case of statement format, we not only can identify the specific member of nodesets and linksets, but also we can keep the statements of execution script in small number. We don't need to provide a separate statement for every dataset movement in the architecture's hierarchy. We can involve a group of dataset movements in a single execution script statement.

Meanwhile, the *compute* statement will be modified to:

- **compute** {DATAFAMILY_SET} on {MEM-LEVEL-L_SET}

PS: This version of interface design hasn't considered the Dataset Hierarchy, Execution script library, and statement dependency yet. But I think first of all, we can link the PetaSim and Emulators together through this interface.