Chapter 2

Accelerator Chip Memory Mapping

In order to understand how the accelerator hardware executes instructions, you must understand two things:

- How the accelerator hardware is mapped into the physical address space of the SPARC
- How the physical address space are mapped into the virtual address space that is seen by PN processes.

2.1 Memory Architecture

Each PN has two accelerator chips associated with it; each chip houses two Vector Units (VU) and two banks of physical memory, each 8MB in size.

For the Curious: Currently, each bank of memory is 8 bytes (or 1 doubleword) wide, plus error-correction bits, and 1024K doublewords deep, giving a total of 8 MB per bank, or 32 MB per PN. Future hardware enhancements will permit memory sizes of up to 128MB per bank, or 512MB per PN. The remainder of this discussion assumes the current 8MB per node memory size.

The accelerator chips have two functions:

- They provide memory access for the SPARC IU. In this role, which we will call “remote access,” the accelerator chips function exactly like memory controllers, and provide no arithmetic capability.

- They provide vector processing capabilities through the two VUs on each chip. Each VU accesses one bank of physical memory. We refer to this as "local access."

Each VU also has its own register "file" (internal register bank). The register file contains 128 single-precision registers, which can be used in aligned pairs as double-precision registers. The VU's perform vectored or scalar arithmetic operations between register and/or memory data, yielding results back into the register file. The results can then be stored to memory.

Both memory and register vector data can be accessed in a strided fashion.

2.2 Physical Addressing

The four memory banks are positioned at physical addresses *N*`00000000` hex (where *N* is the bank number, 0 through 3). The SPARC is free to map virtual pages to any page within the physical space, as long as the offset within the bank does not exceed the size of the memories (8 Mbytes at present).

VU arithmetic operations are initiated by sending instructions to one or more VU's. Each instruction includes the following information:

- The instruction itself, a word (SHORT format) or doubleword (LONG formats) indicating the arithmetic operation (e.g. faddv for a vectored floating point add) and the registers to be used as sources and destination.
- A Memory Address; specifically, the offset into the bank at which the memory data is located (additional memory data may be located at subsequent memory locations, separated by a stride).
- An indication of which VU or VUs are to execute the instruction.

To initiate an instruction, the SPARC writes (stores) the instruction word (or doubleword) to the physical address *N*`8`*mmmmmmm* hex where *N* is the VU which is to perform the instruction and *mmmmmmm* is the Memory Address (the offset into that VU's bank). The setting of bit 32 in the physical address (the `8` in the above format) signals that this is not a remote write access but rather a request for the execution of a vector unit instruction.

The value of *N* determines which VU or VUs execute the instruction:

- The values 0 through 3 for *N* cause the specified single VU to execute the instruction.
- The value 8 causes all VUs to execute the instruction in a SIMD fashion.
- The values 9 (for VUs 0 and 1) and 11 (for VUs 2 and 3) cause the pair of VUs on a single accelerator chip to execute the instruction.

Usage Note: Simultaneous execution on multiple VUs provides the best floating point performance on the CM-5; this is the normal method of operation in the data parallel programming model.

The Mode Registers of the accelerator chip are mapped to physical location **`FF`***N***`80`***xxxx* hex where *xxxx* is the offset to the desired register and *N* is the VU number. When writing Control Registers (only), the multiple-VU values of *N* (8, 9, and 11) can be used to write the control registers of multiple VUs at once.

Note: Most Mode registers (such as the **`dp_vector_length`** register) are shared by the two VUs on the accelerator chip; writing such registers for either VU, therefore, effectively writes them for both VUs. The Vector Mask and Vector Mask Buffer registers are the exception to this, since each VU requires its own vector mask data.

Similarly, the data registers of the VUs are mapped at physical address *N***`4000`***xxxx* hex, where *xxxx* is the byte offset into the register file of the desired register (this value must be word-aligned). Doubleword reads and writes are supported to doubleword-aligned addresses within this space (for accessing double-precision register data). As before, *N* indicates the VU from or to which data is read or written, and again, when writing Control Registers (only), the use of the multiple-VU values of *N* (8, 9, and 11) allows you to write the control registers of multiple VUs at once.

The following table summarizes the physical address space used to map and control the VUs from the SPARC processor:

PHYSICAL ADDRESS	USAGE FROM SPARC
Nmmmmmmmm	Remote access — This address reads or writes words or doublewords from the memory bank of VU *N*, at offset *mmmmmmmm*.
N`8`*mmmmmmm*	Local access — Writing a value D to this address causes VU *N* to execute D as an instruction using the memory bank offset specified by *mmmmmmm*.
`FF`*N*`80`*xxxx*	Local access — Reads or writes the accelerator Control Register *xxxx* for VU *N*.
N`4000`*xxxx*	Local access — Reads or writes the accelerator data register file for VU *N* at byte offset *xxxx* (word-aligned).

2.3 Virtual Mapping

The physical memory layout described above is mapped into the PN virtual memory space for each process running under CMOST, the CM5 operating system. For normal pages (the text, data, and bss segments and the SPARC stack), pages can be allocated at random from the pool of physical pages in the normal fashion.

The control registers and data registers of the accelerator chips must also be mapped.

Finally, and most importantly, memory must be allocated for the storage of user data with which VU operations will be performed. This is where all parallel data (CM fields) will be stored.

The physical memory allocated for this purpose must:

1. span all VU memory banks so parallel VU operations can be performed

2. be contiguous, because vectored VU operations which use memory vectors stride the physical address (after translation), rather than the virtual address.

As a result, “bands” of memory spanning across the banks and consisting of multiple contiguous pages of each bank must be allocated. For the purpose of dynamic data usage, the OS must be prepared to grow and shrink these bands as the result of changing memory usage by multiple time-sharing processes.

Two parallel bands are allocated, one for a parallel stack and one for a parallel heap, in much the same fashion as on the CM-2. Let us say that these bands are allocated at physical addresses *Nssssssss* hex and *Nhhhhhhhh* hex respectively (for all *N*, across all banks). This memory must then be mapped twice:

First, the SPARC will want to read and write data from the parallel stack space. For this purpose, we map 4 virtual regions to the 4 physical regions (one for each bank) that make up the parallel stack.

Since we will want to write data to all banks at once, we also need a virtual region that maps to **8***ssssssss*, i.e. the stack addresses with *N* set to **8** to indicate all VUs. Similarly we want virtual regions that map to **9***ssssssss* and **B***ssssssss* to broadcast write data to pairs of VUs. For the heap parallel data space, we need 7 regions that map to the physical memory allocated for the parallel heap.

These 14 regions define the so-called Parallel Data Space.

Second, we need a way to execute VU instructions that use addresses within those same regions. As a result, we need a duplicate of all 14 regions above, but with mapping set to physical addresses *N***8***sssssss* and *N***8***hhhhhhh*,i.e. the addresses with bit 32 set, used for issuing instructions as described above.

These 14 regions define the so-called Parallel Instruction Space.

Along with 7 regions for mapping the Control Registers (with all values of *N*) and 7 more for the Data Register (again for all values of *N*), we have defined a total of 42 virtual regions. The symbolic names of the regions and where they map are shown in the table below, and diagramed in the subsequent figure. The symbols shown are defined as constants in the header file **dp.h**.

Region name	Address	Meaning
DPV_STACK_DATA_0	**0***ssssssss*	Read/write parallel stack for VU 0
DPV_STACK_DATA_1	**1***ssssssss*	” for VU 1 (R/W)
DPV_STACK_DATA_2	**2***ssssssss*	” for VU 2 (R/W)
DPV_STACK_DATA_3	**3***ssssssss*	” for VU 3 (R/W)
DPV_STACK_DATA_ALL	**8***ssssssss*	Write parallel stack for all VUs
DPV_STACK_DATA_0_AND_1	**9***ssssssss*	” for VU 0 and 1
DPV_STACK_DATA_2_AND_3	**B***ssssssss*	” for VU 2 and 3

Region name	Address	Meaning
DPV_HEAP_DATA_0	**0***hhhhhhhh*	Read/write parallel heap for VU 0
DPV_HEAP_DATA_1	**1***hhhhhhhh*	” for VU 1
DPV_HEAP_DATA_2	**2***hhhhhhhh*	” for VU 2
DPV_HEAP_DATA_3	**3***hhhhhhhh*	” for VU 3
DPV_HEAP_DATA_ALL	**8***hhhhhhhh*	Write parallel heap for all VUs
DPV_HEAP_DATA_0_AND_1	**9***hhhhhhhh*	” for VU 0 and 1
DPV_HEAP_DATA_2_AND_3	**B***hhhhhhhh*	” for VU 2 and 3
DPV_STACK_INST_PORT_0	**08***sssssss*	Operation on stack data for VU 0
DPV_STACK_INST_PORT_1	**18***sssssss*	” for VU 1
DPV_STACK_INST_PORT_2	**28***sssssss*	” for VU 2
DPV_STACK_INST_PORT_3	**38***sssssss*	” for VU 3
DPV_STACK_INST_PORT_ALL	**88***sssssss*	” for all VUs
DPV_STACK_INST_PORT_0_AND_1	**98***sssssss*	” for VUs 0 and 1
DPV_STACK_INST_PORT_2_AND_3	**B8***sssssss*	” for VUs 0 and 2
DPV_HEAP_INST_PORT_0	**08***hhhhhhh*	Operation on heap data for VU 0
DPV_HEAP_INST_PORT_1	**18***hhhhhhh*	” for VU 1
DPV_HEAP_INST_PORT_2	**28***hhhhhhh*	” for VU 2
DPV_HEAP_INST_PORT_3	**38***hhhhhhh*	” for VU 3
DPV_HEAP_INST_PORT_ALL	**88***hhhhhhh*	” for all VUs
DPV_HEAP_INST_PORT_0_AND_1	**98***hhhhhhh*	” for VUs 0 and 1
DPV_HEAP_INST_PORT_2_AND_3	**B8***hhhhhhh*	” for VUs 0 and 2
DPV_CTL_REGS_0	**FF080***xxxx*	R/W Control Register for VU 0
DPV_CTL_REGS_1	**FF180***xxxx*	” for VU 1
DPV_CTL_REGS_2	**FF280***xxxx*	” for VU 2
DPV_CTL_REGS_3	**FF380***xxxx*	” for VU 3
DPV_CTL_REGS_8	**FF880***xxxx*	Write Control Register, all VUs
DPV_CTL_REGS_9	**FF980***xxxx*	” for VUs 0 and 1
DPV_CTL_REGS_B	**FFB80***xxxx*	” for VUs 2 and 3
DPV_DATA_REGS_0	**04000***xxxx*	Read/write Register file for VU 0
DPV_DATA_REGS_1	**14000***xxxx*	” for VU 1
DPV_DATA_REGS_2	**24000***xxxx*	” for VU 2
DPV_DATA_REGS_3	**34000***xxxx*	” for VU 3
DPV_DATA_REGS_ALL	**84000***xxxx*	Write Register file for all VUs
DPV_DATA_REGS_0_AND_1	**94000***xxxx*	” for VUs 0 and 1
DPV_DATA_REGS_2_AND_3	**B4000***xxxx*	” for VUs 2 and 3

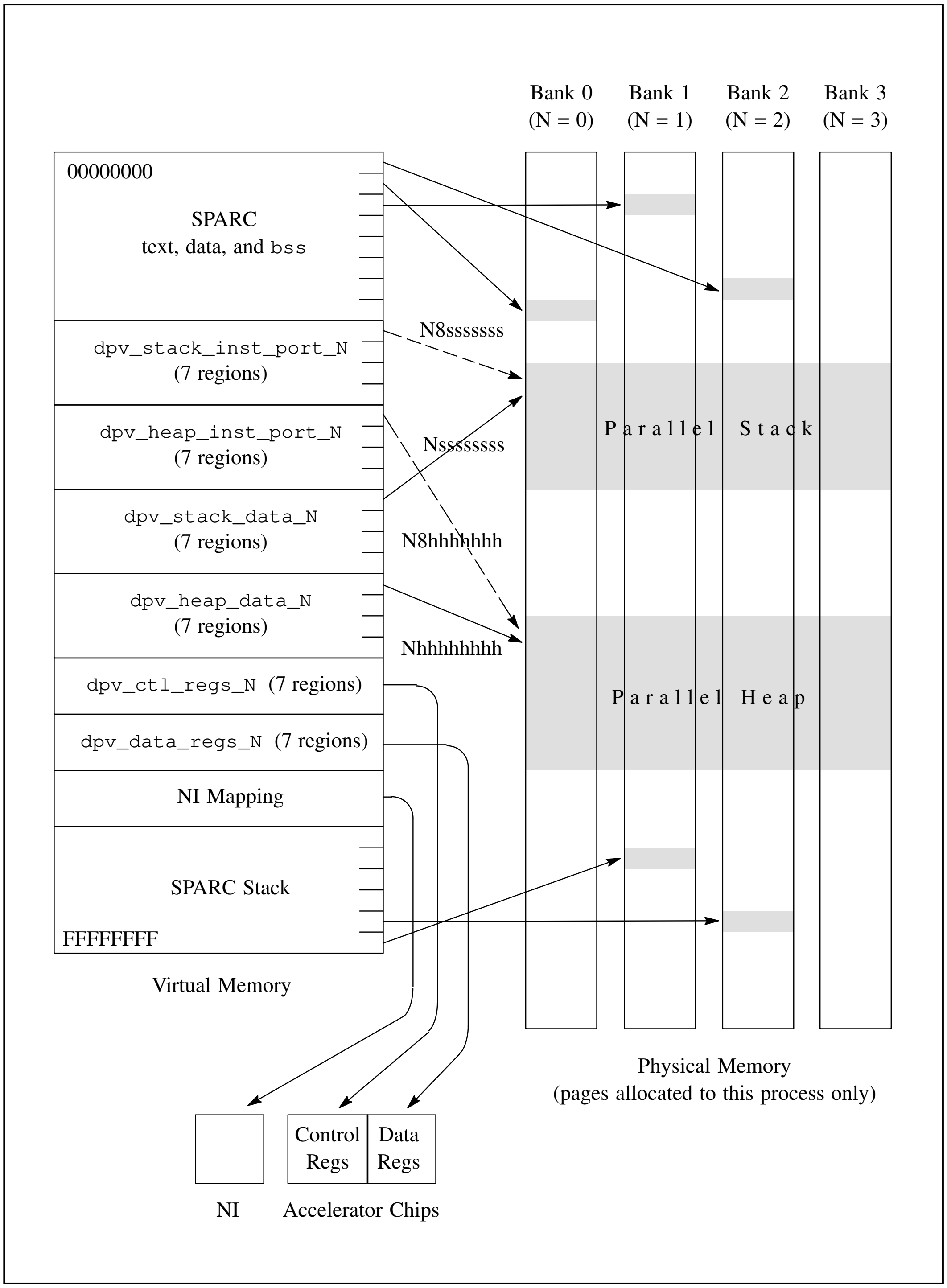

Figure 3. Virtual address mapping.