SPACE: SYMBOLIC PROCESSING IN ASSOCIATIVE COMPUTING ELEMENTS

Corresponding author:

Krste Asanović
International Computer Science Institute
1947 Center Street
Berkeley
CA 94704
USA

Tel: +1 (510) 642-4274 ext 143
Fax: +1 (510) 643-7684
email: krste@cs.berkeley.edu

Keywords:

VLSI, artificial intelligence, associative processors, content addressable memory

Abstract

The SPACE chip implements 148×36-bit Content Addressable Parallel Processors (CAPPs). In the PADMAVATI prototype system, a hierarchy of packaging technologies cascade multiple SPACE chips to form a 170496 processor array.

Primary applications for SPACE are AI algorithms that require fast searching and processing within large, rapidly changing data structures. The PADMAVATI prototype has a peak performance of 136×10^9 32b comparisons per second. Primitive parallel search and write instructions can be composed into arbitrarily complex arithmetic and logical operations, allowing SPACE to be used as a powerful SIMD processor.

In this paper we describe in detail the architecture and implementation of SPACE.

SPACE: SYMBOLIC PROCESSING IN ASSOCIATIVE COMPUTING ELEMENTS

Denis B. Howe and Krste Asanović

INTRODUCTION

Many AI tasks require extensive searching and processing within large data structures. Two example applications are semantic network processing [Higuchi *et al.*, 1991], and maintaining hypothesis blackboards in a multi-agent knowledge-based system for speech understanding [Asanović and Chapman, 1988]. Associative processors promise significant improvements in cost/performance for these data parallel AI applications [Foster, 1976, Lea, 1977, Kohonen, 1980]. SPACE is an associative processor architecture designed to allow experimentation with such applications.

A large SPACE array has been built as part of the PADMAVATI project [Guichard-Jary, 1990]. The core of PADMAVATI is a MIMD transputer array, where each processor has a small amount of fast on-chip SRAM and a large bank of slower external DRAM. Each transputer acts as controller for a local SPACE array.

In this paper we first present the architecture of SPACE, then describe its implementation within the PADMAVATI prototype. We also present performance figures for a range of primitive operations before concluding.

INSTRUCTION SET ARCHITECTURE

Programming model

Figure 1 presents the programming model for SPACE. Data is stored and processed in an ordered array of 36b associative memory words. Two 36b control registers, the Write Enable Register (`wr`) and the Mask Register (`mr`), modify the effect of `write` and `search` operations as described below. The 36b data width was chosen to match current 32b microprocessors, allowing a full 32 bits of address or data to be stored along with a few tag bits in each word.

Each word w of the associative memory has a single flag bit, $\texttt{f}[w]$. The flag bits are used to select the words that will participate in an instruction and can be conditionally set or reset as a side effect of most instructions. Each word w has connections to the flags of the word before, $\texttt{f}[w-1]$, and the word after, $\texttt{f}[w+1]$, and these flag values can be used in place of $\texttt{f}[w]$ to select the words that will be active during a given instruction. This flag chain connects the words in a linear array. Multi-word operations to be constructed through sequences of primitive instructions. This simple linear communications topology performs well for algorithms of interest and is easily

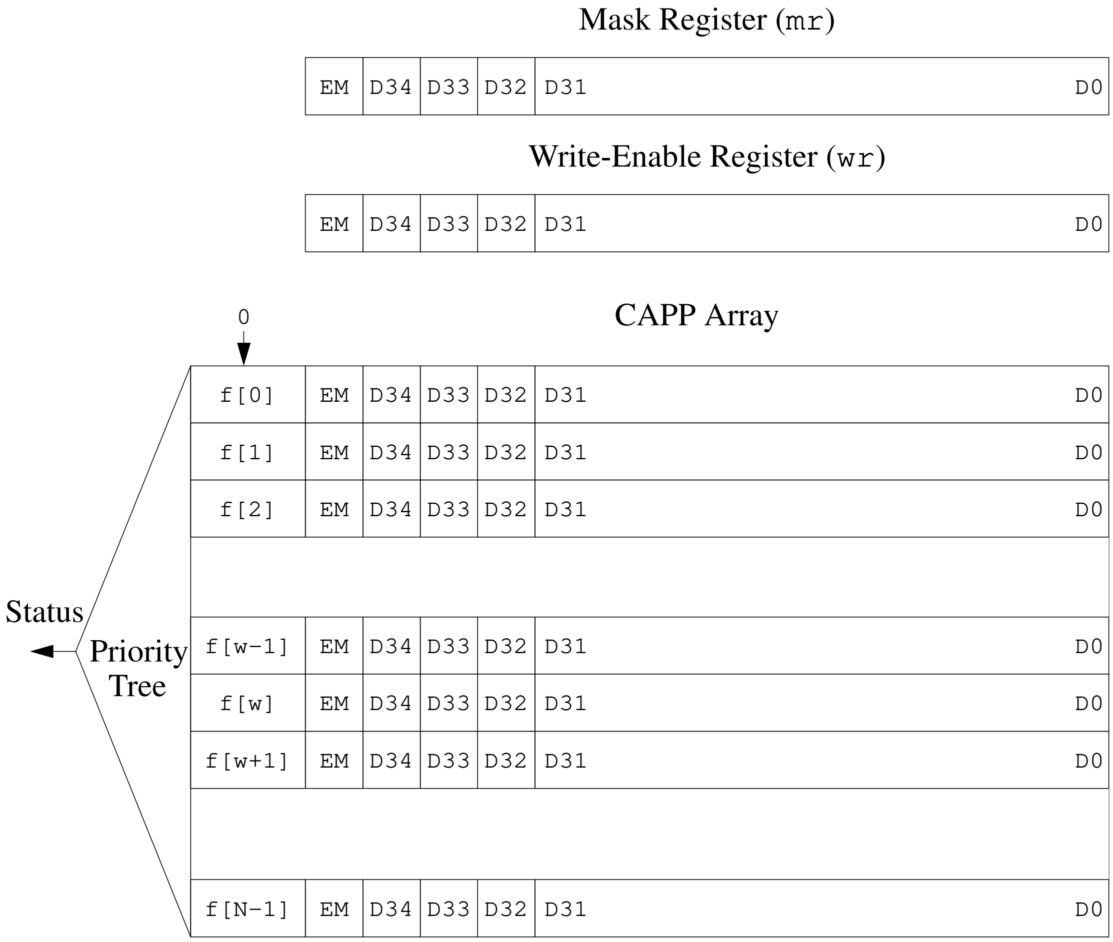

Figure 1: SPACE Programming Model.

scaled to large numbers of processors. The first word in the array (word `0`) has its flag input from the previous word (`f[−1]`) set to `0`.

The architecture includes a priority resolution tree that can activate only the *first* (lowest numbered) selected word for an instruction. Alternatively the resolution tree can be used to update the flags of all words after the first matching word during a search operation. Data words in SPACE are addressed only by their contents, or by the contents of their neighbours.

Instruction format

SPACE instructions are encoded in seven bits as an orthogonal combination of an opcode, a select mode and a flag update mode. The opcode occupies four bits and is encoded as shown in Table 1.

Two further bits in the instruction encode the select mode. These are the Adjacent/Own Flag bit, **AOF** and the Previous/Next Flag bit, **PNF**. Select modes are SPACE's equivalent of the addressing modes found in conventional processors and determine the flag value used to select participating words as shown in Table 2.

New Flag, **NF**, is the final bit in the instruction format and this encodes the flag update mode. The two modes are `set` (**NF** $= 1$) and `clear` (**NF** $= 0$). In the assembler syntax either `s` or `c` is appended to the opcode plus select mode. For read and write

Table 1: Opcode encoding, x indicates don't care.

Assembler Syntax	Operation	**CD**	**RW**	**TS**	**SA**
`wwr`	Write Write-Enable Register	1	0	0	0
`wmr`	Write Mask Register	1	0	0	1
`wbr`	Write both Registers	1	0	1	x
`rwr`	Read Write-Enable Register	1	1	x	0
`rmr`	Read Mask Register	1	1	x	1
`wfi`	Write first selected	0	0	1	1
`wal`	Write all selected	0	0	1	0
`rfi`	Read first	0	1	1	x
`rst`	Read status	0	1	0	x
`smo`	Search for match only	0	0	0	0
`smf`	Search for match & following	0	0	0	1

Table 2: Select Mode Encoding

AOF	**PNF**	words selected	assembler syntax	flag used
0	0	all	`*`	1
0	1	flagged	`@`	`f`[w]
1	0	before flagged	`-`	`f`[$w+1$]
1	1	after flagged	`+`	`f`[$w-1$]

operations, the flags of active words are updated to hold **NF**. For search operations with **NF** $=1$, flags of search *hits* are set and flags of all non-hits are cleared. When **NF** $=0$, a search operation clears the flags of hits and leaves other flags unaffected.

Instructions can be divided by the **CD** bit into those that act on the control registers and those that act on the array. Array instructions can be further divided into three major categories, search, read and write.

Control register instructions

An early decision in the SPACE design was to specify search operand don't cares and write-enabled bit columns through separately loaded control registers rather than by adding a second 36b operand to search and write instructions. In many of the applications we have investigated, the same mask and write-enable values are reused over a number of instructions. By adding programmer visible registers for these values we reduce off-chip operand bandwidth requirements substantially.

The `wwr`, `wmr`, `rwr` and `rmr` instructions allow a 36b value to be written to and read from a control register. The `wbr` instruction allows both registers to be written with the same 36b value in one cycle.

Search instructions

Search instructions take a single 36b operand, the search key, and modify the values of flag bits. Don't cares in the search key are specified with `mr`. Any bit i, for which `mr`[i]$=0$, is a don't care that always matches.

As well as allowing don't care values to be specified in the search key, SPACE also allows don't cares to be stored with the data. One of our applications is Prolog pre-unification, which relies heavily on stored don't cares to represent unification variables in database clause heads [Asanović and Howe, 1989]. To reduce the cost of implementing stored don't cares, we adopted a compromise whereby a data word can be masked in units of a byte rather than bit by bit. Software can be used to simulate stored don't cares, but this simulation is expensive. Simulating the four maskable fields per word in software would take 49 SPACE instructions.

The 36b data word is grouped into four 8b data bytes, **D0–D31**, three tag data bits, **D32–D34** and an Exact/Masked bit, **EM**. The value of the **EM** bit stored in a word affects the way the word responds to searches. If the **EM** bit of a stored data word is set, all 36 bits must match the corresponding masked search key bits. This is an *Exact* word with no stored don't cares; all 35 data bits are available for data storage. If **EM** $= 0$, the word is *Masked* and byte-wise masking is enabled. The most-significant bit of each of the four data bytes controls whether that byte is a don't care. All four bytes and the four tag bits must match the search key to produce a hit. The 8b size of the maskable field provides sufficient granularity for Prolog pre-unification and is convenient for storing and searching 7b ASCII text, while minimising don't care storage overhead.

There are two search instructions: `smo` is search for match only, `smf` is search for match and following. `Smo` compares all selected words with the masked search key and all matching words are considered *hits*. `Smf` compares all selected words with the masked search key and the first matching word and *all* words following are considered *hits*. The `smf` instruction is typically used to flag blocks of words, first flagging all words following a block header word, then clearing all flags following a block trailer word.

If **NF** $= 1$, then the flags of hits are set and the flags of all non-hits are cleared. If **NF** $= 0$, then the flags of hits are cleared and the flags of all non-hits are unchanged. This behaviour differs from that of read and writes, which simply set the flags of active words to **NF**. The scheme used for searches allows the results of successive searches to be AND-ed together, whereas employing the read/write flag update semantics would have OR-ed results instead. We introduced this asymmetry on the basis of our experience in writing various associative algorithms. Especially when coding multi-word operations, it is more common to AND together the results of search operations. If required, search OR-ing can be readily accomplished with a simple series of instructions using a data bit as temporary flag storage.

Write instructions

Write instructions take a single 36b operand and update the value of array data words, modifying flag bits as a side effect. The Write-Enable Register determines which bits of array words are updated in a `write` instruction. Only those bit positions, i, for which `wr[`i`]`$= 1$ are written; others are unaffected. This makes it possible to address specific bit columns as well as specific words. This ability to selectively write individual bit columns, together with the ability to write multiple selected words in parallel, distinguishes a content addressable parallel processor (CAPP) such as SPACE from a less powerful content addressable memory (CAM) [Foster, 1976]. There are two write instructions: `wal` writes to all selected words, `wfi` writes only to the first selected word (if any). The flags of the written words are updated to the value of **NF**.

Read instructions

There is a single read instruction `rfi` which returns the 36b value of the first selected word in the array. The read word's flag bit is assigned the value of **NF**. A value of all `1`s is returned if there are no selected words. In many cases, this allows the readout of the last in a sequence of flagged words to be determined without separate explicit checks of flag status.

The read status instruction `rst` returns a single bit indicating if any flagged words would be selected by a given select mode.

SPACE IMPLEMENTATION

We have constructed a large SPACE system as part of the PADMAVATI project [Guichard-Jary, 1990]. Full custom VLSI SPACE chips are packaged using Tape Automated Bonding onto compact SPACE modules. These modules are attached to conventional circuit boards, which are then connected by a backplane to host processor boards within a MIMD parallel computer system. The following sections describe each level in the packaging hierarchy.

SPACE chip

The SPACE chip implements a small version of the SPACE programming model described in Figure 1. Each chip contains an array of 5328 static CAPP cells arranged as 148 words of 36 bits, a 148-bit flag chain, 36-bit Mask and Write-Enable registers and a 148-input priority resolution tree. The array is never coordinate addressed, so there is no incentive to make the number of words per die a power of two. A conservative 16 transistor static CAPP cell design was adopted to simplify design and ensure robustness. Priority resolution was implemented using a high-radix tree to reduce the latency of selection operations.

The SPACE chip has 56 pins: 7 instruction inputs, a 36b tri-state data bus, 4 pins for cascading chips (pins **PRF**, **NXF**, **REQ** and **PRQ**), 3 timing and control inputs (pins **CS**, **CE** and **PCH**), and 6 power supply connections.

PRF and **NXF** are tri-state pins used to connect the flag chain of a chip to the previous and next chips' flag chains respectively. **REQ** and **PRQ** are used to connect the internal priority resolution tree as a leaf node of an external priority resolution tree. A chip asserts **REQ** if it contains selected words, the external tree asserts **PRQ** if any preceding chip asserted **REQ**. Each SPACE chip in a large system maintains its own copy of the Mask and Write-Enable registers. For write register operations all registers are updated in parallel. For read register operations, the priority resolution tree is used to ensure that only the first chip in the array responds with the values stored in these registers.

The **CS** input is used to partition a large array into independent banks. When low, it disables all chip operations and outputs. Multiple selected chips must be adjacent if flag chain continuity is to be preserved. This feature was added to allow a large SPACE array to be partitioned amongst a number of different applications. For example, on the PADMAVATI machine the run-time system may use one bank to speed global address translation for interprocess communication while the rest of the array is made available for user applications.

SPACE chip timing is controlled by the **CE** input. The **PCH** pin controls the internal prechargers. It can be tied high in which case the chip is fully static but burns

Figure 2: SPACE Chip.

more power, or can be pulsed high between supplying the data and strobing **CE**.

The SPACE chips were fabricated in a 1.2 μm two-level-metal n-well CMOS technology from a fully-custom layout. The die measures $5.8 \times 7.9\,\mathrm{mm}^2$ and contains over 80,000 transistors. The minimum chip cycle time is 125ns.

Figure 2 is a photomicrograph of the chip. Samples were received in June 1989 and passed all functional and speed tests first time, thanks to extensive functional and electrical simulation prior to fabrication. Over 1,200 working parts have since been fabricated for the PADMAVATI project. A more comprehensive data sheet is available [Howe and Asanović, 1990].

SPACE modules

A SPACE module contains twelve SPACE chips, buffering and one stage of external priority resolution, and is functionally equivalent to a 1776-word SPACE chip. There are numerous advantages to mounting the chips on modules rather than directly onto a large circuit board. Fabrication, testing and repair are simplified, and the modules can be reused in different target systems.

Each module measures $100 \times 60\,\mathrm{mm}^2$ and holds twelve 56-pin SPACE chips plus seven 20–28 pin SSI/MSI components. Tape Automated Bonding (TAB) is used to bond the SPACE chips to the module PCB; the remaining components are surface mounted. With TAB, each SPACE chip has a PCB footprint of only $12 \times 10\,\mathrm{mm}^2$. The module PCB has power and ground planes, and 4 signal layers with a trace pitch of 0.5 mm. The pin-out of the SPACE chip was designed to allow devices to be tightly tessellated on the module PCB. Each face of the die has the connections for one data byte. Since all four data bytes within a word are equivalent, the north facing data

Figure 3: SPACE Board.

byte of one die can be directly connected to the south facing data byte of another die, minimising the area needed for routing and vias on the module PCB.

SPACE in PADMAVATI

Each PADMAVATI processor node occupies a standard $280 \times 233\,\mathrm{mm}^2$ 6u board and includes a 25MHz Inmos T800 transputer with 16MB of 200ns DRAM and a 32b asynchronous bus interface. SPACE boards have the same form factor and connect to the processor through the bus interface. Figure 3 is a photograph of a SPACE board. Each board holds six SPACE modules, and each operation can select any subset of the six modules. The selected subset then acts like a single SPACE chip with up to 10656 words.

The transputer acts as the microcode sequencer for the SPACE array, sending instructions and data over the inter-board bus. The SPACE array is memory mapped, with instructions encoded on the bus address lines. SPACE board write and search operations are pipelined, and have a minimum cycle time of 320ns. SPACE read operations require a wait for the returned value, and have a minimum cycle time of 480ns. The transputer limits the rate at which instructions are issued to the board. Measurements on the PADMAVATI prototype give an average instruction cycle time of around 1250ns. A more tightly coupled, dedicated microcontroller could reduce the board cycle time for each SPACE instruction to around 160ns with the existing modules.

The final level of packaging yields a complete PADMAVATI system. Sixteen transputer/SPACE nodes are fully connected through a custom VLSI dynamic routing switch that obeys the transputer link protocol [Rabet *et al.*, 1990]. By partitioning the SPACE array amongst the transputers in this manner, we gain higher array I/O bandwidth and

Table 3: Measured SPACE performance with 170496 words in PADMAVATI.

Operation	Cycles	Operations/s
Scalar-Vector		
36b search	1	136×10^9
1b AND/OR	3	45.5×10^9
1b XOR	8	17.0×10^9
1b half add	3–8	
1b full add	5	27.3×10^9
16b add	83	1.64×10^9
8b$\times$8b multiply	3–539	
32b $=$	8	17.0×10^9
16b $<$	68	2.01×10^9
Vector-Vector		
1b AND/OR	3	45.5×10^9
1b XOR	8	17.0×10^9
1b half add	8	17.0×10^9
1b full add	9	15.2×10^9
16b add	144	947×10^6
8b$\times$8b multiply	539	253×10^6
16b $<$ and $=$	84	1.62×10^9
Vector Reduction		
Find 16b Max/Min	48	2.84×10^9

the flexibility of MIMD control of SIMD subarrays. The total associative storage capacity is 170496$\times$36b words, or around 750KBytes. The complete system is attached as a compute server to a host Sun workstation.

PERFORMANCE

Table 3 lists the number of cycles taken to perform some primitive operations in SPACE, and the resulting measured peak performance for the PADMAVATI prototype. The reader is referred to [Foster, 1976] for a detailed exposition of CAPP algorithms. All these routines are executed conditionally, taking effect only in those words where a user defined tag field in each word matches a user defined tag value.

In the PADMAVATI prototype, execution speed is limited by the transputer used to issue instructions. The performance of the existing modules could be increased by nearly a factor of 8 by using a dedicated microcontroller operating over a synchronous bus. In this case, the peak performance of the array would be over 1×10^{12} comparisons per second.

SUMMARY

SPACE, an associative processor architecture, has been designed to allow experimentation with AI applications. The instruction set is small and highly orthogonal. All data bits in a word support parallel, maskable writes, allowing parallel logical and arithmetic transformations on stored data. Stored data bytes can be individually masked as don't cares. Processors can communicate with their neighbours in a bidirectional linear array to support multi-word data objects, and to allow parallel communication between neighbouring objects.

An implementation of SPACE containing 170496 processors has been completed. This implementation used aggressive packaging techniques to reduce the relative cost of associative storage. Performance has been measured for a range of primitive operations.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the vital contributions to the success of this work made by all those involved, especially Bruce Cameron, John Birch, John Robson and Virendra Patel. This work was partially funded by ESPRIT Project 1219 (967) "PADMAVATI" (Parallel and Associative Development Machine As a Vehicle for ArTificial Intelligence). This project ran from January 1986 to April 1991 and involved GEC Hirst Research Centre in Great Britain, Thomson-CSF in France, CSELT in Italy and FIRST International in Greece [Guichard-Jary, 1990]. Krste Asanović thanks the International Computer Science Institute for its support.

REFERENCES

[Asanović and Chapman, 1988] Asanović, K. and Chapman, J. R. Spoken Natural Language as a Parallel Application. In *Proc. CONPAR 88*, volume B. BCS Parallel Processing Specialist Group, 1988.

[Asanović and Howe, 1989] Asanović, K. and Howe, D. B. Simulation of CAM Pre-unification. PADMAVATI project report, GEC Hirst Research Centre, East Lane, Wembley Middlesex HA9 7PP, Great Britain, January 1989.

[Foster, 1976] Foster, C. C. *Content Addressable Parallel Processors*. Computer Science Series. Van Nostrand Reinhold, 1976.

[Guichard-Jary, 1990] Guichard-Jary, P. PADMAVATI. In Commission of the European Communities, DG XIII: Telecommunications, Information Industries and Innovation, editor, *Proc. Annual ESPRIT Conference '90*, page 227. ESPRIT, Kluwer Academic Publishers, 1990.

[Higuchi *et al.*, 1991] Higuchi, T., Furuya, T., Handa, K., Takahashi, N., Nishiyama, H., and Kokubu, A. IXM2 : A Parallel Associative Processor. In *Proc. 18th Int. Symp. on Computer Architecture*, pages 22–31, 1991.

[Howe and Asanović, 1990] Howe, D. and Asanović, K. PADMAVATI CAM Functional Specification and Data Sheet. PADMAVATI project report, GEC Hirst Research Centre, East Lane, Wembley Middlesex HA9 7PP, Great Britain, April 1990.

[Kohonen, 1980] Kohonen, T. *Content Addressable Memories*. Springer Series in Information Sciences. Springer-Verlag, 1980.

[Lea, 1977] Lea, R. M. Associative processing of non-numerical information. In Reidel, D., editor, *Computer architecture: NATO Advanced Study Institute, St Raphael, France, September 1976*, volume C-32 of *NATO ASI series*, pages 171–215, Holland, 1977. Dordrecht.

[Rabet *et al.*, 1990] Rabet, N. M., Guichard-Jary, P., and Deschatres, M. Padmavati delta network. Technical Report DEL-04, Thomson-CSF DOI, December 1990.