

DTIC FILE COPY

4

Unclass
SECURITY CLASS

AD-A204 256

1. REPORT NO.		READ INSTRUCTIONS BEFORE COMPLETING FORM	
4. TITLE (and Subtitle)		3. RECIPIENT'S CATALOG NUMBER	
A Taxonomy of Synchronous Parallel Machines		5. TYPE OF REPORT & PERIOD COVERED Technical Report	
7. AUTHOR(s)		6. PERFORMING ORG. REPORT NUMBER	
Lawrence Snyder		8. CONTRACT OR GRANT NUMBER(s) N00014-86-K-0264	
9. PERFORMING ORGANIZATION NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
University of Washington Department of Computer Science Seattle, Washington 98195		12. REPORT DATE August 1988	
11. CONTROLLING OFFICE NAME AND ADDRESS		13. NUMBER OF PAGES 5	
Office of Naval Research Information Systems Program Arlington, VA 22217		14. SECURITY CLASS. (of this report) Unclassified	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Distribution of this report is unlimited.			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)			
18. SUPPLEMENTARY NOTES			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) reference stream, Flynn's taxonomy, synchronicity, computer architecture, synchronous parallel computers			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A new classificational scheme is presented which is consistent with Flynn's taxonomy but is more expressive. The crucial idea is to recognize that a reference stream is composed of both values and addresses; their treatment exposes critical features of an architecture. This insight, together with the accompanying formal mechanism built on top of it, enables a large variety of recently developed (since Flynn's work) machines to be distinguished, including VLIW, multigauge, systolic arrays, and the Connection Machines. Though the resulting taxonomic structure is illuminating, the most important result			

DTIC
JAN 23 1989
S H D

DD FORM 1473
1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (continued)

of the classification is the discovery that synchronous execution is NOT a defining property of computer architectures, but is a derived property, a consequence of other architectural features. The evidence for this result and the consequences for machine classification are presented.

A Taxonomy of Synchronous Parallel Machines

Lawrence Snyder

Department of Computer Science
University of Washington
Seattle, Washington 98195¹

Abstract: A new classificational scheme is presented which is consistent with Flynn's taxonomy but is more expressive. The crucial idea is to recognize that a reference stream is composed of both values and addresses; their treatment exposes critical features of an architecture. This insight, together with the accompanying formal mechanism built on top of it, enables a large variety of recently developed (since Flynn's work) machines to be distinguished, including VLIW, multigauge, systolic arrays, and the Connection Machines. Though the resulting taxonomic structure is illuminating, the most important result of the classification is the discovery that synchronous execution is NOT a defining property of computer architectures, but is a derived property, a consequence of other architectural features. The evidence for this result and the consequences for machine classification are presented.

1 Introduction

In 1966 Flynn [1] introduced his classification of computers. This taxonomy proved to be very useful, giving us terminology like SIMD and MIMD that endures to this day. The taxonomy, however, has long been described as too coarse, unable to distinguish between computers that seem to computer architects to be quite different. Though other classifications have been offered [2-5], the fact that Flynn's classification has lasted for so long without being replaced and enhanced is a testament to the difficulty of discovering something better.

In this paper a new taxonomy is presented for synchronous parallel computers. It has no pretensions of being complete nor of capturing all features of synchronous parallel computers. The taxonomy does clarify important distinctions among recently developed parallel computers, such as the VLIW machines, multigauge machines and certain SIMD machines such as the Connection Machines.

The key idea of the taxonomy is to quantify the components of the fetch/execute cycle that process I-streams and D-streams. To make fine distinctions among machines, one must separate these reference streams into their address and value components, because addressing and value processing are crucial features by which machines differ.

Using this kind of analysis, a taxonomy is constructed. Many of the machines that are placed into different classes here would have been classified by Flynn's scheme as SIMD, so this approach permits finer distinctions to be made. Only a small number of classes have been described, and only one or two machines per class have been identified. Thus, there remain substantial opportunities for further research.

¹This research funded in part by the Office of Naval Research Contract N00014-86-K-0264, National Science Foundation Grant CCR-8416878 and Air Force Office of Scientific Research Contract 88-0023.

Perhaps the most important result derived from the taxonomy concerns the property of "synchronicity". The author and apparently many other researchers have treated synchronicity as a primary classificational property: we have spoken of "the synchronous vs. the asynchronous" machines as if this should be an important way to distinguish between machines. It is not. The criterion used for classifying machines in this taxonomy tells when a machine must have all of its instructions start at the same time, and when it is not necessary. This determination is based on how the machine addresses and processes instructions and data. Machines which must begin all instructions at the same time will automatically be synchronous; for those machines that need not begin their instructions at the same time it is an "engineering decision" whether to make them synchronous or asynchronous. Thus the quality of being synchronous is a derived property: A machine must have it because of other features, or it is a noncritical implementation feature.

2 Preliminaries

A *reference stream*, S , of a computer is a finite set of infinite sequences of pairs,

$$S = \{ (a_1 < t_1 >)(a_2 < t_2 >) \dots, \\ (b_1 < u_1 >)(b_2 < u_2 >) \dots, \dots, \\ (c_1 < v_1 >)(c_2 < v_2 >) \dots \}$$

the first component of each pair being a nonnegative integer, called an *address*, and the second component being an n -tuple of nonnegative integers, called *values*, such that n is the same for all tuples of all sequences. An element of a reference stream is called a *reference sequence*. An *I-stream* is a reference stream whose values are interpreted as *instructions*; a *D-stream* is a reference stream whose values are interpreted as *data*.

The interpretation of these definitions is simple. The elements of reference sequences are address, value pairs, the values simply being the contents fetched from (or stored to) memory at the address. A sequence of elements can be thought of as the history of the addresses and values moving between a processor and its memory space. An I-stream is made up of a finite set of these sequences, the number depending on how many instruction sequences the machine can process at one time; and a D-stream is made of a finite set of data sequences, the number depending on how many distinct operations the machine can perform at one time.

Although the I-streams and D-streams have been defined in an intuitive manner, their form is not convenient

89 1 19 034

for analysis. Accordingly, the following reassociation must be performed. Let

$$S = \{ (a_1 < t_1 >)(a_1 < t_2 >) \dots, \\ (b_1 < u_1 >)(b_2 < u_2 >) \dots, \\ (c_1 < v_1 >)(c_2 < v_2 >) \dots \}$$

be a reference stream. Define two sequences: The address sequence of S , denoted S_a , is a sequence whose i^{th} element is a tuple formed from the addresses from the i^{th} elements of each sequence of S ,

$$S_a = \langle a_1 b_1 c_1 \rangle, \langle a_2 b_2 c_2 \rangle, \dots$$

and the value sequence of S , denoted S_v , is the sequence whose i^{th} element is a tuple formed by concatenating the value tuples from the i^{th} elements of each reference sequence of S ,

$$S_v = \langle t_1 u_1 v_1 \rangle, \langle t_2 u_2 v_2 \rangle, \dots$$

Notice that although a reference stream is a set of sequences, address and value sequences are just sequences of tuples.

It is possible to interpret these definitions as grouping the corresponding addresses and corresponding value tuples of a reference stream S into S_a and S_v , respectively.

Let S_x be a sequence of n -tuples; the width of the sequence, $w(S_x) = n$.

Proposition 1: Let S be a reference stream with n -tuple values, then

$$w(S_a) = |S| \text{ and } w(S_v) = n |S|$$

where $|X|$ denotes the cardinality of the set X .

A computation is a pair (I, D) , where I is an I-Stream and D is a D-stream. Computers are classified by the computations they execute. A computer executes the computation (I, D) provided it presents $w(I_a)$ instruction addresses to memory to be fetched simultaneously, it decodes and interprets $w(I_v)$ instructions simultaneously, it presents $w(D_a)$ operand addresses to memory simultaneously, and it performs $w(D_v)$ operations on distinct data values simultaneously. The computer is described by the notation

$$I_{w(I_a)w(I_v)} D_{w(D_a)w(D_v)}$$

Notice that we speak of the computation executed by a computer. This is a definitional simplification, and is sufficient since any desired sequence of instructions or data is a subsequence of the infinite streams of the computation. Observe the relationship between this point and the Enumeration Theorem of recursive function theory.

Let d_1, d_2, d_3 and d_4 be predicates called class designators; then a machine is said to be member of the class denoted by

$$I_{d_1 d_2} D_{d_3 d_4}$$

if and only if $d_1(w(I_a))$, $d_2(w(I_v))$, $d_3(w(D_a))$, and $d_4(w(D_v))$. (Commas may occasionally be inserted between the subscripts for clarity.)

Example 2: By appropriating for our class designators Flynn's "s" and "m" to denote the predicates "is-one" and "is-many", it is possible to classify some familiar machines using the mechanisms developed so far.

Let a von Neumann machine, which Flynn classified as SISD, execute the computation (I, D) . From his classification we have

$$|I| = |D| = 1.$$

By Proposition 1, then, we have

$$w(I_a) = w(D_a) = 1.$$

Moreover, since instructions are decoded serially, $w(I_v) = 1$ and since they are executed serially, $w(D_v) = 1$. Therefore the von Neumann machine is described as

$$I_{1,1} D_{1,1}$$

It is classified with the present notation as

$$I_{ss} D_{ss}$$

since the predicate "s" is true for all four widths.

Now consider two machines that Flynn's taxonomy lumped in the SIMD category, the MPP and the Illiac IV. (Ignore for the moment the fact that these have bit serial and word parallel PEs, respectively.) The single instruction stream means $|I| = 1$ for both machines. By the same reasoning just used for the von Neumann machine, the instruction streams for both machines are described as I_{ss} .

For data, consider the MPP first. Recall that the MPP controller broadcasts the same data memory address to all PEs [6], and so the machine has a single D-stream in our terminology; $|D| = 1$. However, a value is fetched from each PE memory, so the values of this stream are 16384-tuples. Thus,

$$w(D_a) = 1 \text{ and } w(D_v) = 16384,$$

which certainly satisfies the "multiple" class designator. So, the MPP is described as

$$I_{1,1} D_{1,16384}$$

and is classified as

$$I_{ss} D_{sm}$$

The MPP has a "multiple data stream" but the multiplicity applies only to the data values, not to the data value addressing.

For the Illiac IV on the other hand, the controller broadcasts a base address to all PEs, each of which may produce its own address by adding in the contents of a local index register [7]. This means that $|D| = 64$; there are 64 operand address streams simultaneously produced by the machine and each of them references a single value, i.e. each data address is associated with a 1-tuple. Accordingly,

$$w(D_a) = w(D_v) = 64$$

***I*_{1.1} *D*_{64.64}**

 $I_{33}D_{mm}$

3 Discussion

Ian Davis

*I*_{1.1}*D*_{1.16384}

4 Properties of Address and Value Sequences

 $I_{d_1 d_2} D_{d_3 d_4}$

DTIC
COPY
INSPECTED

$$a \leq v \text{ and } a' \leq v'$$

$$v < v'$$

5 More Machine Classes

ulist : special

A-1

- s is the predicate "equals 1",
- c is the predicate "from 1 to some (small) constant", and
- m is the predicate "from 1 to an arbitrarily large finite number".

Though the c and m designators have no upper limit in principle, they are intended to convey two different meanings. When the c designation is used the range has a hard upper limit usually due to internal constraints in the architecture and cannot be easily increased by a substantial amount. An example might be the number of instructions that can be packed in the instruction word of a VLIW machine[8]; for any given word size it is fixed, and even though the word size can be increased this is probably not the intended nor the rational way to generalize the given machine. The m designation, however, is used when the quantity can be easily generalized or scaled. An example is when additional PEs can be added as with the MPP.

These distinctions are not always clear, of course, and judgement must be applied. An example is the question of how to classify a machine with processors connected to a bus [9]. In principle, there is no limit to how many processors can be attached to a bus, but with the addition of each processor the congestion increases, and this is an internal constraint reducing the performance. Is this a " c " or " m " case? Arguments can be made on both sides; we leave the question open for the moment.

It is now possible, using the class designators, Proposition 3 and the convention to define a number of machine classes. Notice that there is no attempt to be complete in either defining classes or categorizing machines:

- $I_{ss}D_{ss}$ von Neumann machines.
- $I_{ss}D_{sc}$ "packed" von Neumann[10]; the machine can fetch several distinct data values from fixed positions from one address and simultaneously apply the same operation to them. Many machines have some instructions of this form, e.g. performing 2 half word adds on the word at a given address; all (ALU) instructions for a machine in this class would have this capability.
- $I_{ss}D_{sm}$ SIMD Parallel Machines with no addressability, such as the MPP, the Connection Machine 1 [11] and systolic arrays [12].
- $I_{ss}D_{cc}$ SIMD Multigauge machines [13]; these are von Neumann machines which can (optionally) split their datapath to process multiple, independent operand streams at once.
- $I_{ss}D_{mm}$ Addressable SIMD Parallel Machines, such as Illiac IV and the CM2 [14].
- $I_{sc}D_{cc}$ VLIW Machines [8]; the machine fetches and executes several instructions stored in one instruction address.
- $I_{cc}D_{cc}$ MIMD Multigauge machines [13]; these are von Neumann machines which can (optionally) split their fetch/execute cycle to pro-

cess multiple, independent instructions concurrently.

$I_{mm}D_{mm}$ MIMD Parallel Machines, including machines such as the Ultracomputer[15] and the Cosmic Cube[16].

Clearly, the list is not complete in terms of either the classes listed or the machines recognized as members of any given class. Much work remains.

6 Discussion of the Taxonomy and The Origins of Synchronous Computation

One is struck by at least two aspects of the foregoing classification: A large and diverse set of machines are lumped into the last classification, $m m m m$, and nowhere in the taxonomy has the synchronous requirement been mentioned, except in the paper's title. These two observations are related.

In effect the taxonomy uses as its "criterion for classification" the number of repeated instances of the principle functional activities of the fetch/execute cycle. So, machines are distinguished by how many instructions they can decode at once or how many operations on separate data they can perform simultaneously. But these are not the characteristics we think of as distinguishing the different MIMD parallel computers. Rather, we think of them as being different depending on whether or not they have global shared memory or what their interconnection topology is. These are features unrelated to the fetch/execute cycle. So, lumping MIMD parallel computers in the $m m m m$ class says only that by the criterion applied, they are all equivalent.

This is unsurprising and is not evidence of weakness in the classification. Indeed, it might point to why efforts to find criteria suitable for classifying all parallel machines have so far been unsuccessful: Qualities that are important for some computers are for other computers, unimportant, irrelevant or even misleading as a guide to classification. To the extent that the taxonomy provides insight by its classifications, the "criterion" amounts to being a useful way of looking at some computers. (For cases where topology matters for synchronous machines, e.g. between the MPP and the CM1, see the next section.)

Interestingly, the "criterion" apparently mandates the synchronous property. By using "the number of repeated instances of the principle functional activities of the fetch/execute cycle" as the basis for classification, we are thinking of machines as either a single f/e cycle that has certain components replicated, or multiple copies of a f/e cycle. In the former case (all legal machines of the form $sy, y \in \{s, c, m\}^3$) synchronous execution is mandated because there is only one cycle running. With the current structure of the taxonomy this leaves only the cy classes and $m m m m$. Though there is no requirement in the model that these be synchronous, the class designators provide some basis for deciding: The c designation carries with it the implication