NASA Contractor Report 181845
ICASE Report No. 89-30

ICASE

EVALUATING LOCAL INDIRECT ADDRESSING IN SIMD PROCESSORS

David Middleton
Sherryl Tomboulian

Contract No. NAS1-18605
May 1989

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665-5225

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Evaluating Local Indirect Addressing in SIMD Processors*

David Middleton
Sherryl Tomboulian

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center, Hampton VA 23665

Abstract

In the design of parallel computers, there exists a tradeoff between the number and power of individual processors. The single instruction stream, multiple data stream (SIMD) model of parallel computers lies at one extreme of the resulting spectrum. The available hardware resources are devoted to creating the largest possible number of processors, and consequently each individual processor must use the fewest possible resources. Disagreement exists as to whether SIMD processors should be able to generate addresses individually into their local data memory, or all processors should access the same address. We examine the tradeoff between the increased capability and the reduced number of processors that occurs in this single instruction stream, multiple, locally addressed, data (SIMLAD) model. We assemble the factors that affect this design choice, and compare the SIMLAD model with the bare SIMD and the MIMD models.

1 Introduction

There is a tradeoff in the design of parallel computers between the number and power of the individual processors. That is, the available budget of hardware resources can be allocated to many simple processors or fewer more powerful ones. The single instruction

*This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18605 while the authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

stream, multiple data stream (SIMD) model of parallel computers lies at one extreme of the resulting spectrum[1]. Hardware resources are devoted to creating the largest possible number of processors to the extent that individual processors lack independent control units, instead performing identical operations on data in their local memories according to instructions issued by a centralized controller.

Disagreement exists as to whether SIMD processors should be able to generate the addresses for these accesses individually, or must use identical addresses to access their memories. Several machines have been built using each model; for example, the MPP and Thinking Machine's CM1 use common addresses, while the ILLIAC IV and the CM2 allow for individual addressing [17,10,24,1].

We examine the consequences of including local addressing in the design of an SIMD machine. We use the term SIMLAD (for "locally addressed data") to describe SIMD machines in which the processors use local addresses and the term SIMCAD (for "commonly addressed data") for those in which the same address is used by all processors. The term MAMD has been used [2], but we prefer SIMLAD as better distinguishing the facility from MIMD. We compare the abilities and costs of SIMLAD with both SIMCAD and fine-grained MIMD designs. The goal is to establish a basis for deliberately choosing whether to include local addressing in SIMD designs, since it appears that recent SIMD designs have provided local addressing as a by-product of other design decisions.

This study is necessarily somewhat preliminary since fair comparison requires extensive study in two directions. First, the hardware costs of local addressing depend heavily on the design effort invested in optimizing VLSI circuits, and second, the effectiveness of local addressing depends heavily on the effort invested in designing algorithms to exploit this specific ability. Consequently, local addressing will likely

[1]The ILLIAC IV design, built under the supercomputer philosophy that hardware cost is negligible, does not seem concerned with a hardware budget. However, in not having been constructed to its full size, it provides support for this view.

appear significantly less attractive initially than its potential and so it is important that local addressing be studied and exploited while there exist machine designs that provide it.

It has been pointed out that the SIMCAD model is equivalent to a sequential machine having word lengths equivalent to the number of SIMD processors and including some strange bit-permutation instructions corresponding to the SIMD machine's communication operations [16,4]. Our concern is with efficiency, so that for example we wish not to ignore multiplicative factors; thus we disregard such equivalence arguments as failing to reflect the importance of the conceptual model of computation on algorithm design and programming.

As a first step then, this paper attempts to assemble all the advantages and disadvantages of local addressing. Where possible these factors are quantified, but in general they are used to indicate where the strengths of the different choices lie in application or hardware design space. We separate the factors into the disadvantages due to the fact that the processors, or PEs, consume more resources and so must be less numerous, and the advantages due to the fact that the PEs become more powerful.

Word size is an important factor in the value of local addressing, since the local addresses constitute an additional data type that would profit from wider than bit-serial data paths. In SIMD designs generally, single bit word lengths are pervasive, with other data types being implemented bit-serially, for two reasons. First, single bit words match the original highly specific applications for which the machines were intended. Second, bit serial arithmetic exchanges simplicity in the ALU for complexity in the control unit [5], and since SIMD machines replicate the data paths and share the control paths, this is an effective tradeoff. Wider data words provide two advantages to the SIMLAD model. First, local address calculations, such as adding an index to a base, use fewer operations than in bit-serial machines. Second, relatively complex data structures, for which the advantages of SIMLAD over SIMCAD are more pronounced,

can be referenced with fewer addresses and correspondingly less address arithmetic. We expect, however, that the one-bit word size will remain heavily favored and so do not consider different word sizes at present.

2 Other work

Local addressing is neither new, having appeared in the ILLIAC IV [24], nor forgotten [12]. The following designs provide local addressing: the ILLIAC IV, the SPHINX, the CM2, the Blitzen, the GF11, the BSP. The following designs require common addressing: the CLIP4, the GAPP, the MPP, the CM1, the HCL (a pyramid machine), the DAP, the BVM.

Several useful surveys of SIMD machines have appeared recently and local addressing is prominent as a distinguishing characteristic. Tuck studies SIMD machines in order to create portable languages suited to describing SIMD algorithms [30]; local addressing of memory and of I/O ports are among the distinguishing characteristics he uses. That study appropriately ignores constant multiplicative factors and so provides a broader, less detailed view of SIMD designs.

Snyder presents a refinement of Flynn's taxonomy that mainly applies to the SIMD division [7,25]. His taxonomy basically distinguishes between multiple streams of data values and multiple streams of data addresses, which directly differentiates SIMLAD and SIMCAD designs.

Sanz and Cypher study architectures and algorithms for image processing on massively parallel computers, mainly SIMD ones [22]. Local addressing, both for memory and I/O ports, is included among the characteristics important to algorithm design. It may be noted that of the nine SIMD machines they mention, only two, the SPHINX and the CM2 provide local addressing. It is also noteworthy that they consider less than 1 Kbit of memory per PE to be inadequate for algorithm design.

3 Costs

We consider the disadvantages of local addressing in terms of the increased consumption by each PE of hardware resources, principally VLSI chip area. Larger PEs are slower, but more importantly, the number of PEs drops.

In SIMCAD designs, the memories of many processors can share a single address-decoder tree fed with an address from the controller. Adding local addressing to SIMD processors involves providing each processor with a separate address-decoder tree and a path to it from a location in the processor, that holds the address. We assume this location is the ALU since that is general and involves minimal further hardware additions.

Asymptotically, a decoder tree uses $O(k)$ nodes to convert $lg(k)$ address bits[2] into k enable lines controlling the storage area. Organizing the storage cells as a rectangle with half the address bits enabling certain storage cells and the other address bits selecting from among those, takes $O(\sqrt{n})$ nodes (and thus chip area) to enable a specific storage cell according to the $lg(n)$ address bits. Such analysis is inadequate due to the wide variability that aspects of design and fabrication may produce in the areas of nodes or the density of the layout.

The following examples demonstrate a likely range for the reduction in processor numbers due to providing the PEs with individual address decoders. The PixelPlanes project has built several SIMD graphics display computers [20]. In the PixelPlanes3 system, the address decoder on each chip occupied the same area as the storage (32 bits per processor) and ALU for about six processors, so adding local addressing to PixelPlanes3 would reduce the number of processors by a factor approaching seven. In PixelPlanes4, the address decoder occupied about three times the area of a single processor, now containing 72 bits of storage, so that local addressing would reduce the

[2] $lg \equiv log_2$

number of processors by up to four. At the other end of the spectrum, consider a hypothetical SIMD machine constructed around commercial memory designs. The address decoding (and other control) logic surrounding the actual storage area in several recent generations of such chips has remained at about 30% of the overall chip area, despite an order of magnitude increase in memory capacity [6]. Thus, in SIMD designs where memory capacity dominates the ALU in resource consumption, local addressing might be expected to reduce the number of processors by at least 1.4.

These examples suggest that SIMLAD designs will have fewer processors than corresponding SIMCAD ones by a factor of between 1.4 and 7, and that the small penalties occur in designs having large amounts of memory per processor.

This is important since the trend with SIMD machines, particularly offspring of successful designs, is towards large increases in the memory associated with each PE. PixelPlanes5 will have 256 bits per PE with a further 4 Kbits of backing store [8]. The Blitzen machine, an extension to the MPP which had 1 Kbits, is to have 64 Kbits [17,21]. Thinking Machine's CM1 had 2 Kbits per processor; the CM2 has 64 Kbits [11,1]. DADO, which grew from the Non-Von SIMD data-base machine, is expected to use 2 Kbytes per processor, although that is predominantly instruction store [23,26].

However, the chip area consumed by gates performing useful work is often much less than that consumed by the wiring connections between them, especially if the large driver transistors that are required to send signals through the wires with acceptable speed are included[3]. Therefore, even for SIMCAD designs, the principal costs most likely lie in the paths joining the address source, the decode tree, the storage cells and the ALU. Minimizing the wiring costs requires placing each PE and its storage close to the address decoder and the latter close to the address source. This suggests replicating the address decoder several times in each chip, especially as the density

[3] Large empty areas on a chip often arise from the inability of various modules to fit together. Thus, the storage of PixelPlanes PEs in long strips rather than rectangles is likely much more important for its reduction in fragmentation than it is for any affect it has on the size of the address decoder.

of integration continues to rise. From this perspective, the cost of SIMLAD becomes the extent to which further replication of the decoder to provide one for each PE falls below such an optimum.

In comparing SIMLAD and SIMCAD, we distinguish one-time costs, such as design time, from the inherent, repetitive, costs such as cycle time or chip area (with the attendant yield). We consider the latter factors to be important. Design time is a significant cost however, and attempts are usually made to reduce it, which is unfortunate since subtle implementation decisions made during the design phase have significant impact on the repetitive costs of the resulting design. Examples of these design decisions include the layout of the decode tree, the length and width of the nodes in the decode tree (increasing one to reduce the other may worsen the node area yet improve the overall fit of the nodes with each other), the length and width of the storage area (which relates to the number and width of the words in each PE memory), the available power consumption, the intended clock speed, the number and types of the layers provided by the targeted fabrication process, and the attendant design rule restrictions.

Each factor individually may alter the area cost considerably. This can be seen by considering various memory cells available in the MCNC VLSI library [6]. For example, dynamic and static memory cells using metal data paths differ in area by a factor of 1.8. Static cells using polysilicon paths differ in area from those using metal by a factor of 1.2. (Furthermore, such publically available cells are significantly less area-efficient than commercial memory designs [31].) That there are many inter-related factors, each of which may alter the area by a factor of almost two suggests that considerable design effort is required to search the design space for near or adequately optimal implementations.

Thus, design costs pose a significant hurdle for SIMLAD designs. Until they are considered worthwhile, the design investment required to reduce their cost reasonably

close to their potential (whatever it may be) will not be invested, possibly causing their cost to remain prohibitive.

It may well be that local addressing has been provided in some recent SIMD machines as a by-product of other design decisions rather than as a deliberate increase in the capability of PEs. Two factors support this interpretation. Algorithmic techniques and language enhancements associated with the feature are lacking. In fact, there is a significant speed penalty associated with the use of local addressing over common addressing in the CM2. Furthermore, the use of external commercial memory chips suggests that local addressing is a by-product of the need to increase the memory of PEs while mitigating additional design time costs. (In one instance however, local addressing was added in response to customer pressure). Implementing local addressing with external memories suffers significant time delays and area penalties associated with the chip boundaries separating PEs from their memories.

4 Uses

The advantages of local addressing reduce to the increased utilization of the PEs. In conventional SIMD processing, variations in the operation of the PEs require separating the PEs into groups, and sending each group its instructions while other groups are disabled. Using local addresses as parameters to the instruction stream may allow the PEs to be divided into fewer groups, reducing the time that PEs spend disabled.

As a first consequence, local addressing may improve a space-time product for a fixed number of processors. The amount of resources for a given number of PEs increases by the cost of the local addressing while the time for program execution decreases by the extent to which code sequences merge through the use of parameterized addressing. As a second consequence, local addressing may improve the space-time product for a fixed amount of hardware, that is, for fewer PEs. If the utilization of

PEs increases by a factor of k by merging code sequences, then each actual SIMLAD PE might simulate $\lfloor k \rfloor$ of the original PEs, taking no longer than the original set of code sequences would require[4]. Local addressing is not required for such simulation, as is demonstrated by the concept of virtual processors as expounded by Thinking Machines, but it does extend the use of virtual processors to more cases, as shown by the load balancing example below.

It is recognized that evaluating different parallel computers requires the use of appropriate algorithms for each. The effectiveness of SIMLAD designs relies on developing new parallel algorithms that do not require the independent operation of multiple instruction streams but can exploit local addressing. This suggests that the power of local addressing manifests in enabling PEs to manipulate individual data structures with identical operations.

4.1 Data Structures

Local addressing allows individual PEs to use data structures that vary from PE to PE depending on particular data or the previous actions performed by the individual PE. Such structures include arrays and linked lists, accessed by local indices and pointers, from which dequeues, trees and tables can be constructed.

One method for programming SIMCAD machines consists of enumerating all possible cases and issuing the appropriate instructions for each, one after another, while the appropriate PEs for each case are enabled. Thus, for example, since a stack with space for ten elements can be in one of eleven states (ignoring specific entry values), a SIMCAD algorithm can use such a structure with only an elevenfold reduction in speed (ignoring the overhead of enabling different PE groups [14]). This suggests that local

[4] For a SIMLAD PE to simulate $\lfloor k \rfloor$ SIMCAD ones, it should properly have $\lfloor k \rfloor$ times the memory. SIMCAD then becomes less the replication of address decoders to match PEs than the reduction of PEs to match address decoders. Such a trend opposes the fine-grained "logic-in-memory" aspect of the SIMD philosophy.

addressing is more useful for algorithm design in SIMD machines with relatively large memories, since those can exploit data structures that are too large to be simulated.

We examine various areas where individual data structures might enhance PE operation.

4.2 Communication Operations

Communication among PEs provides an excellent arena for evaluating SIMLAD. An essential and expensive part of parallel processing, communication in various forms has been studied exhaustively for both SIMD and MIMD processors. Consequently, the best known approaches for these other machine models represent fair targets against which SIMLAD may be compared. The challenge is to invest comparable effort in devising effective communication strategies for the SIMLAD model. The following is a preliminary study; further research is in progress.

Communication operations can be characterized according to something analogous to the binding times of the application's communication paths to the hardware connections between PEs. We divide communication operations into three broad categories. In the first, the hardware connectivity corresponds to the pattern of communication required by the target application: data movement in the problem occurs between immediate neighbors in the hardware. An example would be the 2D mesh communication used in the image processing problems which initially motivated designs such as the MPP. In these cases, local addressing is of no benefit.

The second category contains static communication patterns. These patterns do not depend on the particular values being moved, yet are unrelated to the physical structure of the PE connections. These patterns can be determined prior to execution in a "compile-time" phase. In this situation, a method is needed to control the independent movement of messages over multiple arcs in the hardware network.

A table driven approach has been developed that supports such communication in

SIMCAD machines [28]. Given a mapping of the problem's graph nodes into the PEs, each arc in the problem graph can be supported by a sequence of physical connections between the corresponding PEs. This description of each such path is distributed through the intermediate PEs so that each knows only the next PE in line. The different arcs in the problem graph are laid out and initiated so as to avoid collisions. The method involves a global clock, each tick of which corresponds to the progress of each message by one step along its path. Each PE, which may lie on several paths, contains a table in which each entry indicates, at the given clock tick, the neighbor that should receive the message currently held in the PE. Local addressing of memory is unnecessary for table access since all PEs sequentially access their tables in time with the clock; nevertheless, local addressing of I/O ports allows for denser representations of the tables which is important since table size is a limitation on this method.

Local addressing on the I/O ports noticeably improves this method. Since neighbor addresses are part of the SIMCAD instruction stream, each clock tick contains a minor cycle that iterates through each possible communication direction. Local addressing of neighbors, that is I/O ports, improves this method by the degree of connectivity among the PEs: this is a factor of 4 in a two dimensional mesh and a factor of 12 in a hypercube of 16K nodes. The hardware costs of decoding a local address of two or three bits would be negligible in comparison with the interface hardware.

Sorting operations represent a common benchmark for communication systems, although Potter suggests that sorting is less appropriate than associative processing techniques for SIMD algorithms [18,19]. Many sorting algorithms can be viewed as a fixed permutation of message holders with PEs conditionally exchanging the contents of two holders as they pass through [3,15]. Such algorithms fit in the second category since at a low level view, the message holders repeatedly undergo a fixed regular movement independent of their contents; only at a higher level are the messages seen as following irregular paths to arbitrary destinations.

The third category contains dynamic communication patterns in which the pathways are determined as the messages travel. This category includes operations where PEs calculate messages' destinations individually (perhaps to implement fault tolerant features). Various supplemental factors may have an impact on these kinds of operations: whether PEs may be sources or destinations for multiple messages, whether all PEs must participate in the operation and the kind of indexing used to label the PEs for calculating destinations. The third category differs from the second in that the actual communications between PEs are not predictable before the operation begins.

With dynamic, run-time, routing arises the possibility of collisions: two messages may, at the same time, need either to enter one PE or to leave a PE by the same path. MIMD computers handle such collisions by using buffering within the nodes. Local addressing enables SIMD machines to perform similar buffering through the use of locally addressed queues. Buffer overflow can be handled with similar techniques in each case.

As in the second category, local addressing of I/O connections simplifies the dynamic routing involved by removing the need for the controller to specify instructions separately for communication occurring in each separate direction.

4.3 Computation models

SIMD PEs having individual data structures can implement new mechanisms on which effective algorithms might be based. (This study is concerned with efficient operation rather than questions like Turing equivalence).

Local addressing enables PEs to perform (independent) function evaluations by table lookup. For example, consider two dimensional hexagonal cellular automata such as are used to study fluid dynamics [27]. Each node in such a model determines at each step whether particles leave that node along any of the edges depending on the particles that just arrived along those edges. This involves computing six boolean

functions, each of (the same) six boolean inputs. A 64 element table in each SIMLAD PE would consume fewer than 400 bits and would operate about 64 times faster than a SIMCAD simulation, assuming the different overheads for each approach are similar. Since local addressing probably reduces the total number of PEs by less than eight, mapping eight problem nodes into each SIMLAD PE will most likely yield a faster solution for a given problem size and hardware budget than a SIMCAD machine would yield.

Given that the PEs' operations are table-driven, it is not necessary that the tables be identical across PEs, although differences might require some translation of the values that pass between the PEs.

Local tables can also be used to implement finite state machines and, in conjunction with other local data structures, push down and linearly bounded automata[5]. For such a model, all PEs simultaneously transform an <input, oldstate> pair into an address within the table, dereference that location to yield an output and a new state (and a stack or head operation if necessary), and then use those values appropriately.

One technique used in SIMD machines involves creating several "virtual" PEs in each physical PE [11]. The physical PEs identically support each virtual PE in turn, and some advantages may accrue through pipelining effects. Local addressing allows some variability in the operation of the different virtual PEs which can be used to implement restricted load balancing. For example, in the calculation of Mandelbrot diagrams, the identical iterative computation is repeated for all gridpoints in the complex plane. Since there are large variations in the number of iterations in different regions, load-balancing can be accomplished by mapping widely separated points and the associated virtual PEs into a single physical PE [29]. A PE finishing one point early can immediately proceed with the next one it holds.

[5] Obviously, this ignores the fact that the resources are finite. Such an assumption is often made to allow the separation of algorithm development from memory size considerations.

Data flow models of computation can be implemented by using local arrays to hold queues of input tokens; this is in addition to any use of buffering occurring during the communication of tokens between PEs. It is worth noting however, that the token queues in the Manchester tagged dataflow computer can hold at least 64K tokens, so that data flow models of computation may well be impractical for SIMD PEs because of the space cost [9].

In the abstract, local addressing is a step towards independent instruction streams. Each PE can implement a local instruction address register pointing to a list of local instructions. The host broadcasts what is effectively microcode for each instruction after enabling those PEs whose register points to the given instruction. Specifics of the instruction set (for example, whether enabled processors are executing a common program module, or a common instruction such as a floating point addition, that is RISC or CISC) is a subsidiary issue. In practice however, such simulation is likely to be ineffective. The memory required for holding local programs probably dwarfs the hardware required to decode instructions locally to the extent that a straightforward MIMD design would better exploit the hardware resources.

In conclusion, SIMCAD machines are unable to make effective use of a rich assortment of algorithmic techniques which have been studied extensively for conventional computers, ones that exploit data structures. Local addressing extends the standard SIMD programming techniques of associative or data-parallel processing by making these techniques also available.

5 Conclusion

Adding local addressing to SIMD processors increases the hardware consumed by each PE and consequently reduces the total number of processors (for a given hardware budget). This reduction is not large however, most likely being less than eight and

probably less than three or four if local memories exceed 1 Kbit.

As a somewhat separate issue, local addressing of neighbors in input/output instructions noticeably improves some communication operations for negligible hardware cost in comparison to that required to drive signal wires between PEs.

SIMLAD designs improve with respect to SIMCAD ones as memory size increases, for two reasons. First, the fraction of the total PE devoted to the additional decoding hardware decreases, so that the reduction in number of PEs tends to drop to around two. Second, increasing the memory that can be independently accessed increases the size of the data structures that local addressing provides, increasing the advantage provided with respect to the limitations caused by common addressing in the SIMCAD design.

Increasing memory size at the same time reduces the marginal cost of local instruction storage and decoding and so at some point, an MIMD approach becomes preferable to either SIMD approach. The requirements of the FFP machine and the DADO machine, two fine grained MIMD computers, suggest that instruction storage may require at least several kilobytes [13,23,26]. Further hardware is required for local instruction decoding, so a useful niche may exist for SIMLAD designs among parallel processors, when they are classified by the amount of memory provided with each PE.

This study is preliminary in that more detailed research is required in two separate directions. Efforts in VLSI design and layout are needed to reduce the hardware consumed by providing PEs with their own address decoding mechanism. Algorithms need to be developed that exploit the opportunities afforded by local addressing within the data-parallel realm of single instruction stream execution. Communication operations provide a reasonable target application within which local addressing may be studied.

Acknowledgements

The authors would like to thank several people for useful discussions, including John Poulton and Kye Hedlund at UNC Chapel Hill, Jothy Rosenberg at Duke Uni-

versity, Wayne Detloff MCNC, Guy Steele, Bradley Kuzmal and others at Thinking Machines, and Dennis Martin of Halcyon Corporation (chip packaging).

References

[1] CM2 Manual, Thinking Machines Corporation, 1988.

[2] *[Use Of MAMD For Local Addressing]*, Second Symposium on the Frontiers of Massively Parallel Computation, in preparation, Virginia, October 1988.

[3] K. E. Batcher, *Sorting Networks And Their Applications*, The Proceedings of AFIPS 1968, pp. 307-314.

[4] K. Batcher, Keynote Address, First Symposium on the Frontiers of Massively Parallel Scientific Computation, Maryland, September 1986.

[5] G. A. Blaauw, F. P. Brooks, *Computer Architecture*, two-volume work by Addison-Wesley in 1990, (in preparation).

[6] W. Detloff, VLSI designer, Microelectronics Center of Nth. Carolina, personal communication, May 1987.

[7] M. J. Flynn, "Some Computer Organizations And Their Effectiveness", IEEE Transactions Computers, Volume C-21, No. 9, September 1972, pp. 948-960.

[8] H. Fuchs et. al, "Curved Surfaces and Enhanced Parallelism In Pixel-Planes", Computer Science Technical Report, UNC Chapel Hill, 1988.

[9] J. R. Gurd, C. C. Kirkham, I. Watson, "The Manchester Prototype Dataflow Computer", CACM, Volume 28, No. 1, January 1985, pp. 34-52.

[10] W. Hillis, "The Connection Machine", MIT Press, Cambridge, Mass, 1985.

[11] W. Hillis, G. Steele, "Data Parallel Algorithms", CACM, Volume 29, No. 12, December 1986, pp. 1170-1183.

[12] B. Kuzmal, "Simulating Applicative Architectures On The Connection Machines", Masters Thesis, MIT Department of Electrical Engineering and Computer Science, June 1986.

[13] G. Magó, D. Middleton, "The FFP Machine - A Progress Report", 1984 International Workshop on High-Level Computer Architecture, May 1984, pp. 5.13-5.25.

[14] D. Middleton, "Implementing Nested Conditional Statements In SIMD Machines", ICASE Report No. 89-27, NASA CR-181832, April 1989.

[15] D. Nassimi, S. Sahni, "Benes Network And Parallel Permutation Algorithms", IEEE Transactions on Computers, Volume C-30, No. 5, May 1981.

[16] D. Parkinson quoting C. A. R. Hoare in a tutorial, Second Symposium on the Frontiers of Massively Parallel Computation, Virginia, October 1988.

[17] J. L. Potter, "The Massively Parallel Processor", The MIT Press, 1985.

[18] J. L. Potter, "Programming the MPP", in "The Massively Parallel Processor" edited by J. L. Potter, 1985, The MIT Press, pp. 218-229.

[19] J. L. Potter, "Associate Data Structures For Massively Parallel Computers", Second Symposium on the Frontiers of Massively Prallel Computation, Virginia, October 1988.

[20] J. Poulton, H. Fuchs, J. Austin, J. Eyles, J. Heinecke, C. Hsieh, J. Goldfeather, J. Hultquist, S. Spach, "PIXEL-PLANES: Building a VLSI-Based Graphic System", Computer Science Press. 1985 Chapel Hill Conference on VLSI, May 1985, pp. 35-60.

[21] J. Reif, Duke University, personal communication, Sept. 1987.

[22] J. L. C. Sanz, R. E. Cypher, "The Prospects For Building And Programming Massively Parallel Image Processing Architectures", IBM Research Report RJ 6231, April 1988.

[23] D. Shaw, "The NON-VON Supercomputer", Computer Science Technical Report. Columbia University, August 1982.

[24] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, R. A. Stokes, "The ILLIAC IV Computer", IEEE Transactions on Computers, Volume C-17, 1968, pp. 746-757.

[25] L. Snyder, "A Taxonomy of Synchronous Parallel Machines", Proceedings 1988 International Conference on Parallel Processing, August 1988, pp. 281-285.

[26] S. Stolfo, D. Miranker, "DADO: A Parallel Processor for Expert Systems", Proc. 1984 International Conference on Parallel Processing, August 1984, pp. 74-82.

[27] T. Toffoli, N. Margolus, "Cellular Automata Machines: A New Environment for Modeling" the MIT press, 1986.

[28] S. Tomboulian, "A System for Routing Directed Graphs on SIMD Architectures", ICASE Report No. 87-14, NASA CR-178265, March 1987.

[29] S. Tomboulian, "Indirect Addressing and load Balancing for faster solution to the Mandelbrot set on SIMD Architectures", Fifth Annual Massively Parallel Symposium , Columbia, April 1989, and ICASE Report 89-33, NASA CR-181847, May 1989.

[30] R. Tuck, "An Optimally Portable SIMD Programming Language", Second Symposium on the Frontiers of Massively Parallel Computation, Virginia, October 1988, and Computer Science TR88-048, UNC Chapel Hill.

[31] R. A. Wagner, personal communication.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-181845 ICASE Report No. 89-30		

4. Title and Subtitle	5. Report Date
EVALUATING LOCAL INDIRECT ADDRESSING IN SIMD PROCESSORS	May 1989
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
David Middleton Sherryl Tomboulian	89-30
	10. Work Unit No.
	505-90-21-01

9. Performing Organization Name and Address	11. Contract or Grant No.
Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225	NAS1-18605
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report
National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
Richard W. Barnwell

Final Report

16. Abstract

In the design of parallel computers, there exists a tradeoff between the number and power of individual processors. The single instruction stream, mulitple data stream (SIMD) model of parallel computers lies at one extreme of the resulting spectrum. The available hardware resources are devoted to creating the largest possible number of processors, and consequently each individual processor must use the fewest possible resources. Disagreement exists as to whether SIMD processors should be able to generate addresses individually into their local data memory, or all processors should access the same address. We examine the tradeoff between the increased capability and the reduced number of processors that occurs in this single instruction stream, multiple, locally addressed, data (SIMLAD) model. We assemble the factors that affect this design choice, and compare the SIMLAD model with the bare SIMD and the MIMD models.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
SIMD computers, local addressing, indirect addressing	60 - Comp. Oper. & Hardware 61 - Comp. Prog. & Software Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	19	A03

NASA FORM 1626 OCT 86