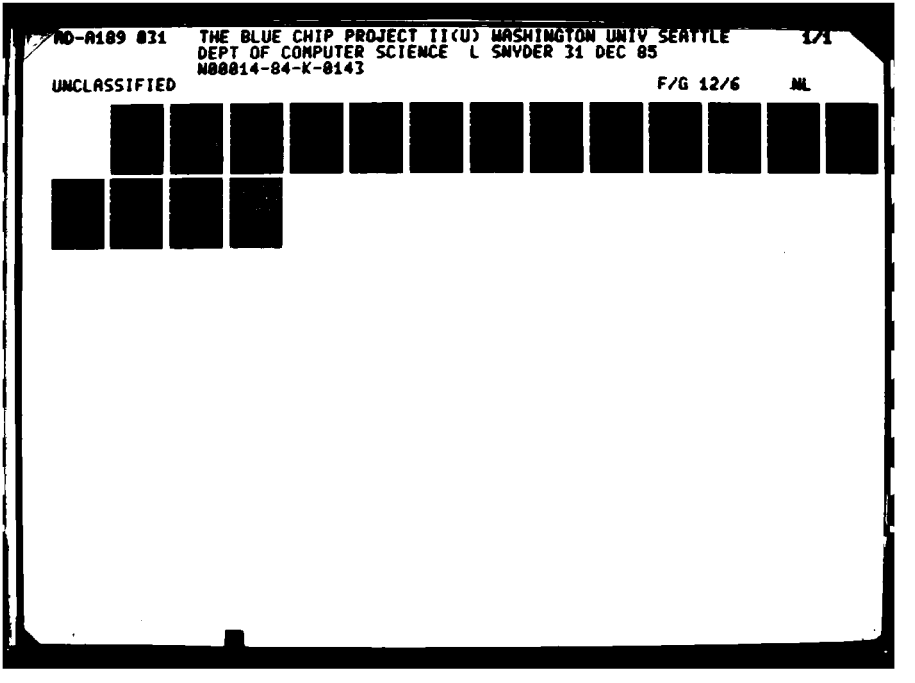
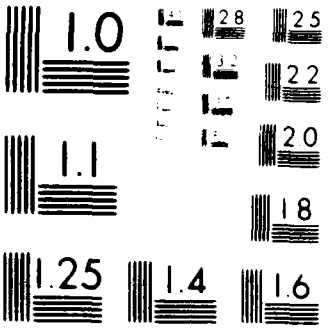

AD-A189 031 THE BLUE CHIP PROJECT II(U) WASHINGTON UNIV SEATTLE 1/1
DEPT OF COMPUTER SCIENCE L SNYDER 31 DEC 85
N00014-84-K-0143
UNCLASSIFIED F/G 12/6 NL


1.0
1.1
1.25
1.4
1.6
2.8
2.5
2.2
2.0
1.8

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: none	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Final Report N00014-84-K-0143		5. TYPE OF REPORT & PERIOD COVERED: Final 9/15/83 to 2/28/85
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Lawrence Snyder		8. CONTRACT OR GRANT NUMBER(s): N00014-84-K-0143
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Washington, Department of Computer Science, Seattle, Washington 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Information Systems Program, Arlington, VA 22217		12. REPORT DATE: December 31, 1985
		13. NUMBER OF PAGES: 14
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report): Unclassified
		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)

parallel computation, VLSI, CHiP Computer, Pringle Computer, coordination, contractions, Configurable Highly Parallel Computer, wafer scale integration, graph embeddings, Poker, parallel language

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The major accomplishments of the Blue CHiP Project are described in the areas of theory, algorithms, architecture, software, and VLSI.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Final Report for N00014-84-K-0143

Lawrence Snyder

The following text describes technical achievements for the above mentioned contract, entitled "The Blue CHiP Project II". It must be noted that the work sponsored by this contract was the final portion of a large project known as the Blue CHiP Project, Contract N00014-81-K-0360, SRO-100. The Blue CHiP Project was funded in two parts because the Principal Investigator moved to the University of Washington from Purdue. This report will detail significant progress of both contract periods.

Since the project is organized into five research topic areas – theory, algorithms, architecture, software and VLSI – it is convenient to organize the report into those five subheadings. See Figure 1 for a diagram relating major components of the project.

1 Theory

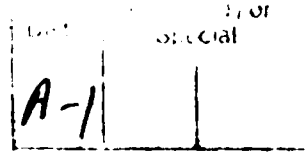
Theoretical analysis has been of great use in other areas of the project. Here we describe those results that are of particular interest in their own right.

Minimax edge length. The embedding of a graph into the plane is a useful abstraction for the layout of a circuit into a VLSI technology, or the specification of the communication structure of a CHiP program in the CHiP computer's lattice. Since signals take time to propagate along a wire, the length of a line in an embedding will indicate the amount of time for a particular communication and the longest edge length of an embedding will indicate the minimum clock cycle for a circuit. It is for these reasons that we have studied the minimax edge length of graph embeddings, i.e. graph layouts that have the horstest maximum length edge over-all embeddings. Since most circuit graphs contain a tree as a subgraph, we have focused on the minimax edge length in binary trees.

The first set of results concerns the rather surprising fact that it is not possible to achieve simultaneously the shortest possible edges and the smallest possible area of a layout[1]. For example, trees with their n leaves on the perimeter of a convex region (the typical case in VLSI) may have area Ω (n log n) or have minimum-maximum edge length Ω(n/log n), but not both. When the area is minimum, the edge length is O(n/log log n) and when the edge length is minimum, the area is $O(n^{1+e})$, $e > 0$.

The second set of results concerns the simultaneous achievement of minimum area, minimum edge length and planarity of layouts for binary trees[2]. This result builds on work of Valiant and is surprising when one considers the number of constraints simultaneously achieved.

Tile Salvage. If one considers a VLSI wafer as a region covered with tiles, then after the wafer has been "probed", the dysfunctional chips are given red dots, the wafer is "diced-up" and the functional tiles are saved. Suppose now that the wafer is a checker board of



black and white tiles and the same process is repeated, except that now we wish to save adjacent, functional black-white pairs. (The purpose, of course, is to get larger chips by patterning half the circuit on the black tiles and the other half on the white tiles).

We have shown that there is an efficient ($O(n^{2.5})$) algorithm for maximizing the connected, functional, black-white pairs [3]. The basic technique is to use a variant of the "marriage" problem solution. Unfortunately, from the point of view of the motivation, the problem of maximizing the adjacent, functional red-white-blue-green quads on a wafer of alternating rows of alternating red-white and alternating blue-green tiles is NP-complete. That is, the problem of making bigger chips by grouping four good adjacent tiles together is computationallly intractable. This negative result is not as negative as it first appears however. First, the wafer is not "arbitrarily large" as required by the theory of intractability. Second, one need not *always* maximize the salvaged chips - it is possible to waste a few. Third, our work has stimulated the work of Brenda Baker at Bell Labs to develop very good heuristic solutions that supersede the ones proposed. Best of all, however, this work has lead to a much more basic understanding of the computational difficulty of general planar layout problems, and with Berman, Leighton, and Shor, we have solved a number of open problems such as planar graph partitioning. These results, as yet unpublished, largely explain the sources of complexity for planar layouts. Indeed, David Johnson has recently extended these results so that the theory of planar layouts is even more complete.

Coordination. The old theory of cellular automata, due to von Neumann, assumed that all cells read and write simultaneously in every direction on every step. H. T. Kung's systolic arrays do this simultaneous I/O too. But the more general parallel algorithms hosted by the CHiP Computer cannot assume that the I/O is so regular. Processors must be assumed to read and write when necessary, oblivious to the actions of the senders and receivers. It is for this reason that the CHiP machine assumes a "data driven" model of computation: reads, proceed only after data has arrived, and writes send unless doing so will cause buffer overflow. Such a scheme is safe but it is not very efficient because of the overhead of the handshaking protocol that must take place to support the asynchronous I/O timing.

We have developed [4] a model of parallel computation in which the above mentioned phenomena can be expressed: synchronous I/O, data driven I/O, asynchronous I/O, etc. Moreover, the model is capable of comparing the various strategies fairly, because each communication paradigm is a parameter to a common model. This is the first time that different communication mechanisms have been so unified. Moreover, we have done more; we have made these theoretical results practical.

A system of parallel processors is said to be *coordinated* if each write in the system is immediately (i.e. on the next step) answered with a read. It has been shown that the problem of recognizing whether a system is coordinated is P-SPACE hard, i.e. compu-

tationally intractable [5]; this result is proved only for completeness. More positively, it has also been shown that there are polynomial time algorithms for determining whether certain restricted classes of parallel processes are coordinated. Furthermore, it has been shown [6] that there are polynomial time algorithms for constructing coordinated parallel systems from certain families that are not coordinated. This result is significant because it implies that a compiler's code generator could accept programs that were written using an "expensive" (i.e. high in overhead) data driven semantics and convert them into efficient coordinated programs. This implication has motivated a large amount of software work to develop such code generators. They are now complete and experiments are being run to determine the amount of improvement from coordination.

Among the experimental results that can be reported, we know (from Cuny's work) that the Kung-Leiserson systolic band-matrix multiplication algorithm requires 1.16 times longer to execute in date-driven mode than in coordinated (i.e. synchronous) mode. Additionally, the "duty" cycle of this systolic array is only 1/3 as originally defined, i.e. each processor is executing on only every third step. But using techniques developed by Cuny for this project, the processors can be fully utilized.

Using the Poker software from this project (see below) Cuny had done much more extensive experiments that indicate that the savings from coordination can be very large. For example, speed ups of 50% have been achieved.

2 Algorithms

The greater part of the application algorithms work has been done in collaboration with Dennis Gannon and has been encouraged, but not directly supported by this contract. Of special note is Gannon's work with Panetta ("Restructuring SIMPLE for the CHiP Architecture," Purdue Technical Report, 1984) in which they report on how this classic Livermore benchmark can be run on the CHiP machine.

Linear Recurrences. Because the CHiP architecture is so novel, there was little guidance, initially, on how to formulate effective algorithms. Our work on solving linear recurrences [7] took up this challenge in a way that also suggested how known algorithms might exploit the CHiP machine. Building on a technique developed by Chen, Kuck and Sameh, we developed several algorithms to solve linear recurrences depending on various assumptions on where the input is located. To see the basic idea, visualize a matrix in which rows are selected for "elimination", first in adjacent pairs, then in pairs separated by a row of zeros, then by three zero rows, etc. This strategy naturally induces a binary tree structure on the rows of the matrix which manifests itself as a binary tree structure in the CHiP machine's switch lattice. The form of the tree varies depending on assumptions of where the data resides. The resulting algorithms are all optimal, based on both the

"unit" cost and "proportional to distance" cost measures.

Processor Interconnection Structures. The graph that is embedded into the lattice when one programs a processor interconnection structure reflects the communication structure of the algorithm. As a means of exploring the general question of what types of communication can be supported efficiently in the lattice, we studied a variety of processor interconnection structures [8]. These include optimal embeddings for the ubiquitous binary tree and the torus. Additionally we invented an ingenious technique for routing data down the corridors of a lattice. (Recall that the corridor is the row of switches separating PEs). One would expect that a k corridor lattice could route$k$ data paths between two processors, but it is possible to achieve $3k$ data paths, all distinct, by a technique called "lacing" which exploits the cross over and diagonal edges that are provided in the normal eight degree lattices.

Databases. Although supercomputers are typically associated with purely numerical computation, we have investigated non-numeric applications related to databases [9, 21]. The main results include the implementation of various known sorting algorithms on the CHiP machine, and a unified processing paradigm for database operations that is applicable to the construction of a special purpose machine. Specifically, it is possible to formulate the operations of union, intersection, difference, remove duplicates and sorting as variants on one processing algorithm. The key observation is that sorting algorithms augmented with some tag bits can with additional hardware, do each of the five operations equally efficiently. If a special purpose device is built with this approach, it may be prudent to choose a lattice that has an aspect ration that is$(n/\log n) \times (\log n)$ rather than square in order to provide greater access to external data storage.

As a part of our work on the Poker environment, we have programmed a large number of basic algorithms, typically for numerical linear algebra. In addition, we have written certain nonnumerical algorithms such as sorting. This software was, at project's end, being assembled into a library for use by other scientists in the spirit of classic subroutine libraries.

3 Architecture

Perhaps the most visible contribution of the project is that collection of ideas known as the CHiP Computer. Of these, the one attracting the greatest interest is the externally imposed configurability based on circuit switched regional communication. But there are many other interesting aspects of the architecture that have attracted our attention.

Quantifying the CHiP machine. In the early papers describing the CHiP architecture [10, 11], it was observed that certain characteristics of the design were variable and that the CHiP "machine" was really a family of computers. The following question was left

open: What are the ranges of optimality for these characteristics? Although the question has concerned us throughout the project, no really definitive answer has ever been given in print. It is appropriate that we offer our best judgement as of the moment:

- n, the number of processors, will likely be a perfect square since there is no apparent advantage with a non-square lattice, except for database applications. (Note that this statement depends on using the CHiP for general computation; there could be an advantage for special purpose devices.) Furthermore, it is quite convenient, e.g. for trees, that n be (an even) power of 2, although it is hardly a necessity. It seems appropriate to build computers with 64 to 4K processors; more processors are possible but further architectural analysis is required.

- d, the degree of the PEs and switches, should be 8. There has been no need for larger degrees and no consistent pattern to the cases where smaller degrees are acceptable.

- c, crossover level, must be 2 but 3 is frequently useful, especially for narrow corridor cases; it was recently thought that because there is a significant cost 3 would be a reasonable compromise from 4, the maximum, but results within the past six months imply that 4 is probably mandatory.

- w, internal corridor width, must be at least 2 which is completely adequate for a 64 PE lattice; in the $64 < n < 4096$ range, 4 would probably suffice. Because additional width is so expensive, especially in terms of pins, a w=2 choice could be a fair tradeoff for large lattices.

- u, external corridor width, must equal internal width w, because anything less represents a poor place for savings.

- p, phases, can be as low as 4 but 8 or 16 might be better choices provided the PE memory is comparably larger than the 2K currently provided on the Pringle.

- m, local memory for PEs, should be as large as possible consistent with keeping the memory on board; of course this trades off with functionality and features like floating point take precedence over memory[12]. The current 12K limit is being upgraded to a more realistic 8K.

The remaining parameters such as data path width are so sensitive to issues such as pin availability, clock skew, etc. that no general statement appears to be safe.

Pringle. Detailed emulation of a sequential computer by a sequential computer is often estimated to lose a factor of 1000 in performance; evaluating a parallel computer will lose an additional factor at least proportional to the number of processors, but probably much

more if the communication is great. So, we built a hardware emulator in order to make realistic sized runs feasible. But since it was premature to do a VLSI implementation, it was impossible to exploit the benefits of VLSI for the lattice. Consequently, our emulator is not a true CHiP computer, since it replaces the lattice with a polled bus [13,22,23].

The Pringle has 64 processing elements each of which is composed of an Intel 8031 microprocessor, an Intel 8231 floating point processor, a 2K x 8 RAM and a 4K x 8 EPROM*2. [1] The RAMs of the PEs are memory mapped into the address space of the controlling 8086 minicomputer for convenient down-loading and debugging. The PEs communicate with each other by writing to an 8 bit latch or reading from a queue that is 16 bytes deep. The source and destination of the I/O is managed by a polling device that cyclically consults each processor to see if it wishes to write from any of its 8 logical ports; if so, the destination PE and port number are found in an internal table (corresponding to the configuration setting), the data is entered into the queue of the appropriate PE, and it is tagged with the correct port number. This bus achieves a transfer rate of 64 Mbits/sec. using the designed 32MHz clock rate. The processors achieve a rated speed of 64M (8-bit) instructions per second. The Pringle was powered up and ran its first (diagnostic) program in March 1983 and the switch was checked out and running by August 1983. By December 1983 a second copy was completed (using parts funded by the National Science Foundation). By April 1984 the Pringle was fully interfaced to its software environment and the first program, written and run from the Poker System, was run on the Pringle. (One copy of the machine resides at Purdue University in D. Gannon's lab; the Blue CHiP Project copy is at the University of Washington in the Blue CHiP lab).

The External Input/Output (XIO) system has been designed, built and checked out. The system uses four Winchester disks each of which has its controller connected to a host 8086 minicomputer. Each of these host devices, in turn, is connected to eight one chip microcomputers (Intel 8031s) which are, in turn, interfaced into the Pringle's switch.

Data for and from the external environment is viewed as moving between the XIO and the processor lattice as a collection of "streams", i.e. data value sequences. To achieve this illusion using normal files, an (as yet unimplemented) operating system views each file as a set of k streams where the first (k field) record of the file is the first element of each stream. The hosts read files from their disks and "break" then into streams which they route to the "stream transfer" microcomputers. These machines serve as buffers from the streams, delivering values to the PEs as required.

Notice that our 64 processor Pringle requires 36 computers to support the XIO. It would be accurate to conclude that in parallel computation, supporting the external input/output is more than half of the problem.

[1] It should be mentioned that Intel Corporation donated a substantial amount of hardware (all of the processors and memories) thus stretching contract funds.

4 Software Support

We have produced an enormous amount of software to support the project's work including:

- LAPSE, a VLSI layout programming language,
- CONFIG, an offline language for switch lattice programming,
- SIM, an event-based CHiP machine simulator,

plus numerous smaller incidental systems. Each of these has been of considerable use to the project but not of sufficient external interest to be documented beyond project notes. There is, however, one other somewhat more notable system.

Poker. Implemented in the C programming language to run on a VAX 11/780 under UNIX, Poker is a parallel programming environment designed to support CHiP programming [14-16, 24, 25]. Poker provides facilities to assist the programmer with nearly all phases of parallel programming: programming processor elements, programming processor communication structures, compiling, assembling, coordinating, loading, running, specifying input/output files, tracing and debugging. Poker programs can be emulated on a full Pringle software emulator, they can run on the Pringle hardware, and one day, they will be able to run on a CHiP computer.

The Poker environment uses two displays in its work station: a conventional display and a 1024 x 768 pixel bit mapped display for graphics support. It was written during the summer of 1982 by 10 very committed gentlemen: [2] the "Poker Players". Over the course of the Blue CHiP Project and the Blue CHiP Project II, a dozen other programmers contributed full or part-time to the Poker implementation. Among its novel features is the ability to program the communication structure of the algorithm (i.e. the lattice) using graphics. The programmer "draws a picture" of the data paths to be used by the algorithm; a compiler then converts the graphical form into a symbolic form suitable for down loading into the Pringle. This is the first example of graphic programming of symbolic text.

Another novel feature is the way that Poker system handles processor to processor communication. The I/O behavior of a processor is based on a data driven semantics, i.e. the writers write immediately (unless the buffer would overflow) and the readers wait until data has arrived before proceeding. This scheme engenders considerable overhead due to "handshaking". However, an experimental version of Poker has the ability to coordinate the programs using the synthesis algorithms described above.

The Poker system allows the programmer to "watch" the execution of his program on the graphics display. Specifically, while the emulator is running, values that have

[2] Many of the project personnel, including most of these fellows were receiving graduate fellowships from other sources; this is another way in which contract funds have been stretched.

been designated as "trace variables" are continuously displayed on the screen. Thus, the programmer watches the dynamic behavior of his program. At any time the programmer wishes he can stop emulation and change (i.e. *poke*) any of the displayed variables; he can then resume execution. This facility is completely integrated into the environment so that the entire context of the source program is available to the programmer while he is tracing and debugging.

5 VLSI

The CHiP Computer has been designed to be easily and efficiently implemented in VLSI, but it is premature to actually demonstrate such an implementation. This fact has not prevented us from spending considerable effort exploring VLSI-related matters.

Switch Design. Over the course of the project, we have designed, perhaps, half a dozen different versions of the basic lattice switch. These have been used chiefly to evaluate different architectural choices, so it was unnecessary to fabricate them. However, to test performance, it is necessary to implement a design; so early in 1983 we fabricated our first switches. [3]

We did not simply construct a switch. Rather, we built an experimental apparatus in silicon which enabled us to investigate a variety of operating conditions. We instrumented the system for the following set of conditions in any or all combinations: switch-to-switch on-chip communication, switch-to-switch off-chip communication, switch-to-switch communication via a 3500 lambda delay line. The test setting was organized so that variables line delay through signal capture pad drivers could be factored out of the measurement.

The fabricated chips were returned in April 1983 and found to be functionally correct on the first time through fabrication! The detailed measurements indicate that the signal transit time of the switch is on the order of 25 nanoseconds for nMOS technology.

Processor Displacement Methodology. It is possible to be too good at designing VLSI chips! Specifically, as silicon technology advances ahead of packaging technology and as better parallel algorithms are developed, it is possible to place so much parallel processing circuitry on a chip that it cannot be provided with data fast enough to keep the processors busy. The problem has been over parallelized.

Clearly, unless there is greater bandwidth across the chip's perimeter, there is no way to get more work done per unit time. But it is still possible to utilize the silicon. The strategy is to multiplex the processor elements so that a larger problem can be solved in the given area ***at a rate that matches the I/O bandwidth.*** Such a scheme is called the Processor Displacement Methodology[17] and it has been shown to be effective for problems such as dynamic programming.

[3] **The fabrication was funded by the DARPA MOSIS facility and thus incurred no contract costs.**

The CHiP Design Methodology. Managing the complexity of very large VLSI designs has long been regarded as a problem that can be solved with hierarchical design methodologies. The intent is to begin with small designs that can be composed with repetition to form larger elements. The difficulty is that the methodology emphasizes the *composition* rather than the *repetition.*

In the CHiP Design Methodology the repetition is emphasized over the hierarchy [18,26]. Specifically, the CHiP machine is used as an abstraction for programmable silicon. The designer specifies his chip functionally at the highest, i.e. least detailed, level by programming the CHiP machine to do the algorithm. This program will be composed of a small collection of individual PE programs operating in parallel and repeated throughout the lattice. Next these constituent PE programs are themselves coded as CHiP machine programs but using a simpler set of primitives. Through a series of refinements, the algorithms of the constituent PEs of level k are recoded as CHiP programs (using the whole lattice) at level k + 1 until the primitive set of operations used for the program is so simple that they can either be directly implemented as VLSI layouts or are stored in a cell library. At this point, the entire design can be produced through substitution of the k + 1st layout into the cells of the kth layout. There is a hierarchy, but it will likely be shallow (two or three levels) and the repetition of cells will be emphasized.

Wafer Scale Integration. As it becomes more and more difficult to achieve higher VLSI device densities through better fabrication techniques, it is natural to seek alternatives. One such approach is to consider making the chip bigger, which in the limit means using the whole silicon wafer for a single circuit.

Our approach has been to solve what might be called the "logical" problems of wafer scale integration by exploiting the CHiP architecture's configurability [19, 20]. Specifically, we pattern onto the wafer a CHiP lattice which has been augmented with extra, redundant switches. Because of these extra switches and because switches are so small that the chance of failure is unlikely, we can expect a high yield for the switches. This permits the switches to be used to interconnect the functional processing elements. Since these are much larger, it is more likely for them to be dysfunctional; they will be sparse over the lattice. Our strategy of connecting them together restricts the distance that the communication signals must travel by linking PEs that are in the same neighborhood. The result of the interconnection of the PEs by the switches is a dense, logical CHiP lattice built from a sparse physical lattice.

The wafer scale approach has the desirable property that the algorithms for constructing the lattice and for finding the functional PEs are very efficient. The price we pay for this efficiency is that not all processor elements will necessarily be used. Still, the approach has proved effective. A systolic array processor can be built on a 5" wafer produced by a fabrication line that achieves 20% yield on chips, and this processor will be a dense array

of 16 x 16 elements at least 99% of the time. By additional tricks, it is possible to build a 28 x 28 = 784 systolic processors on a wafer using the ground rules described above.

6 Conclusion

The foregoing discussion concentrates on an enumeration of the specific, published results of the project. There have also been numerous other results which have not made it into print. These manifest themselves indirectly by influencing the way problems are chosen and the way the directions of future research are selected.

The objectives outlined in the proposal for the Blue CHiP Project have been achieved. In addition, the Poker System is being distributed; it is the first programming environment for nonshared memory parallel computers, a major achievement of the Project. It is filled with new ideas and will hopefully lead to a new generation of parallel programming systems.

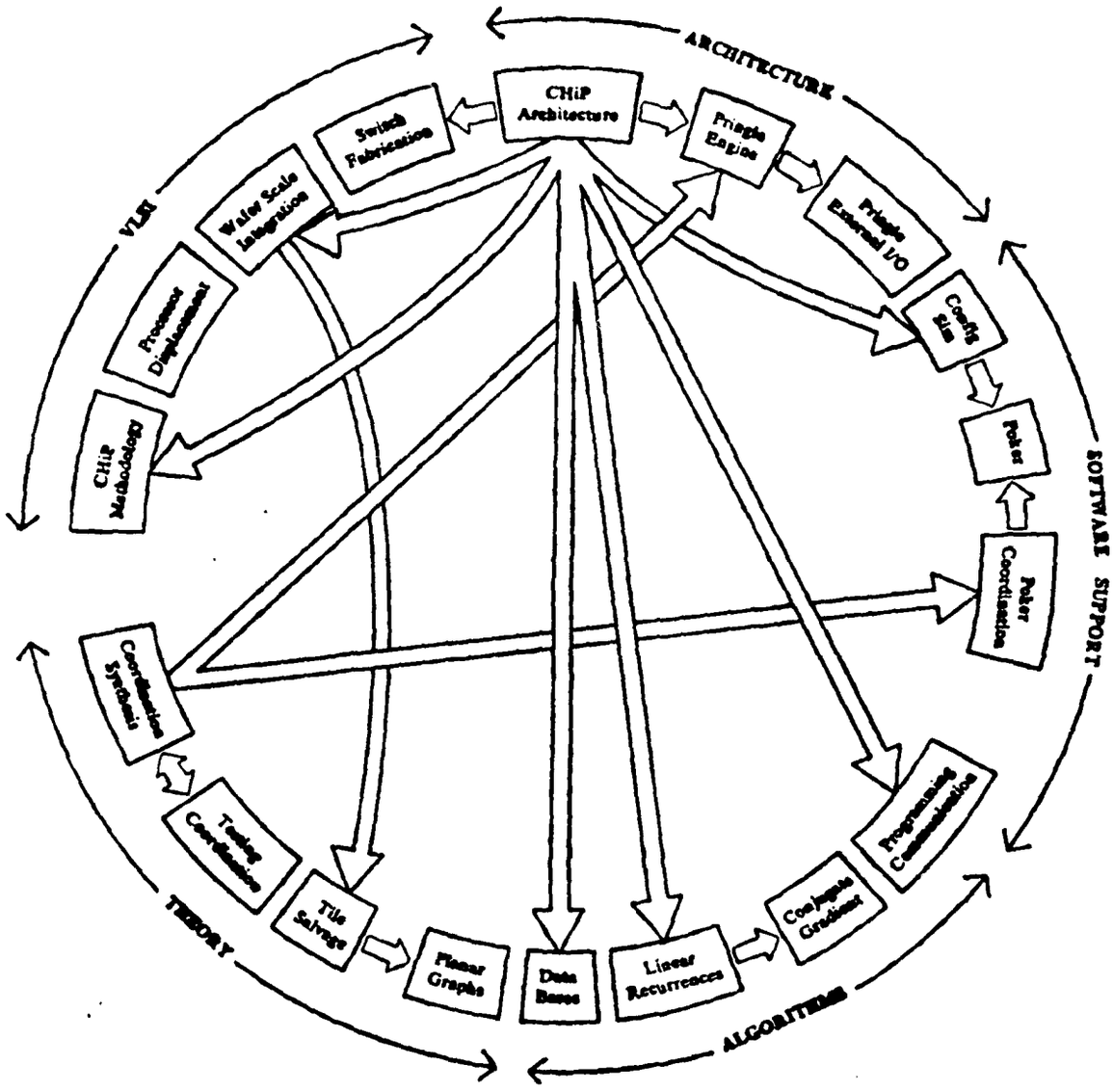


Figure 1: Major project accomplishments and their interrelationships: An arrow means that the source topic motivated or solved a problem for the target topic.

7 References

1. M.S. Paterson, W. L. Ruzzo and L. Snyder, "Bounds on Minimax Edge Length for Complete Binary Trees," *Proceedings of the 13th Annual Symposium on the Theory of Computing*, ACM, 1981, pp. 293-299.

2. Walter L. Ruzzo and Lawrence Snyder, "Minimum Edge Length Planar Embeddings of Trees," *Proceedings of the Carnegie-Mellon University Conference on VLSI Systems and Sciences*, H.T. Kung, Bob Sproull and Guy Steele, eds., Computer Sciences Press, 1981, pp. 119-123.

3. Francine Berman, Frank Thomson Leighton, Lawrence Snyder, "Optimal Tile Salvage," Technical Report CSD-TR-396, Purdue University, 1982.

4. Janice E. Cuny and Lawrence Snyder, "A Model for Analyzing Generalized Interprocessor Communication Systems," Technical Report CSD-TR-406, Purdue University, 1982.

5. Janice E. Cuny and Lawrence Snyder, "Testing Coordination for Homogeneous Parallel Algorithms," *Proceedings of International Conference on Parallel Processing*,, IEEE, 1982, pp. 265-267.

6. Janice E. Cuny and Lawrence Snyder, "Conversion from Data-flow to Synchronous Execution in Loop Programs," Technical Report CSD-TR-392, Purdue University, 1982.

7. Dennis Gannon and Lawrence Snyder, "Linear Recurrence Systems for VLSI: The Configurable Highly Parallel Approach," *Proceedings of International Conference on Parallel Processing*, IEEE, 1981, pp. 259-260.

8. Lawrence Snyder, "Programming Processor Interconnection Structures," Technical Report CSD-TR-381, Purdue University, 1981.

9. Ching-Chih Hsiao, "Highly Parallel Processing of Relational Databases," Technical Report CSD-TR-460, Purdue University, 1983.

10. Lawrence Snyder, "Introduction to the Configurable, Highly Parallel Computer," *Computer*, Vol. 15(1), January 1982, pp. 47-56.

11. Lawrence Snyder, "Overview of the CHiP Computer," in VLSI 81, John P. Gray, ed., Academic Press, 1981, pp. 237-246.

12. Lawrence Snyder, "Supercomputers and VLSI: The Effect of Large-Scale Integration on Computer Architecture," *Advances in Computers*, Marshall C. Yovits, eds., 1984, pp. 1-33.

13. J. Timothy Field, Alejandro A. Kapauan and Lawrence Snyder, "Pringle: A Parallel Processor to Emulate CHiP Computers," Technical Report CSD-TR-433, Purdue University, 1983.

14. Lawrence Snyder, "The Design of the Poker Parallel Programming Environment," Technical Report CSD-TR-409, Purdue University, 1982.

15. Lawrence Snyder, Steven S. Albert, Carl W. Amport, Brian G. Beuning, Alan J. Chester, John P. Guaragno, Christopher A. Kent, John Thomas Love, Eugene J. Shekita, Carleton A. Smith, "The Poker Programming Environment and its Impelementation," Technical Report CSD-TR-410, Purdue University, 1982.

16. Lawrence Snyder, "The Poker (1.1) Programmers Guide," Technical Report CSD-TR-434, Purdue University, 1983.

17. David M. DeRuyck, Lawrence Snyder, John D. Unruh, "Processor Displacement: An Area-Time Trade-Off Method for VLSI," *Proceedings of the MIT Conference on Advanced Research in VLSI*, Paul Penfield, Jr., ed., Artech House Books, Inc., 1982, pp. 182-187.

18. Lawrence Snyder, "Configurable, Highly Parallel (CHiP) Approach to Signal Processing Applications," *Proceedings of Technical Syposium East '82*, Society of Photo-Optical and Instrumentation Engineers, 1982, pp. 8-16.

19. Kye S. Hedlund and Lawrence Snyder, "Wafer Scale Integration of Configurable, Highly Parallel Processors," *Proceedings of International Conference on Parallel Processing*, IEEE, 1982, pp. 262-264.

20. Kye Sherrik Hedlund, "Wafer Scale Integration of Parallel Processors," Technical Report CSD-TR-422, Purdue University, 1982.

21. C. C. Hsiao and Lawrence Snyder, "Omni-Sort: A Versatile Data Processing Operation for VLSI," *Proceedings of the International Conference on Parallel Processing*, IEEE, 1983, pp. 222-225.

22. Alejandro Kapauan, J. Timothy Field, Dennis B. Gannon and Lawrence Snyder, "The Pringle Parallel Computer," *Proceedings of the 11th International Symposium on Computer Architecture*, IEEE, 1984, pp. 12-20.

23. Alejandro Kapauan, Ko-Youg Wang, Dennis Gannon, Janice Cuny and Lawrence Snyder, "The Pringle: An experimental system for parallel algorithm and software testing," *Proceedings of the International Conference on Parallel Processing*, IEEE, 1984.

24. Lawrence Snyder, "Introduction to the Poker Parallel Programming Environment," Technical Report CSD-TR-443, Purdue University, 1983.

25. Lawrence Snyder, "Parallel Programming and the Poker Programming Environment," *Computer*, Vol. 17[7], July 1984, pp. 27-36.

26. Lawrence Snyder, "The Role of the CHiP Computer in Signal Processing," *VLSI and Modern Signal Processing*, S.Y. Kung, ed., Prentice-Hall, 1984 [Invited].

27. Francine Berman, "Parallel Computations with Limited Resources," *Proceedings of the Conference on Information Sciences and Systems*, pp. 675-679, The Johns Hopkins University, 1983.

28. Ching-Chih Hsiao, *Highly Parallel Processing of Relational Databases*, Technical Report CSD-TR-460, Purdue University, 1983.

29. Dennis Gannon, Lawrence Snyder and John Van Rosendale, "Programming Substructure Computations for Elliptic Problems on the CHiP System," in Ahmed K. Noor (editor), *Impact of New Computing Systems on Computational Mechanics*, pp. 65-80, The American Society of Mechanical Engineers, 1983.

30. Francine Berman and Lawrence Snyder, "On Mapping Parallel Algorithms into Parallel Architectures," in Robert M. Keller (editor), *Proceedings of the International Conference on Parallel Processing*, pp 307-309, IEEE, 1984.

31. Lawrence Snyder and K. S. Hedlund, "Systolic Architectures – A Wafer Scale Approach," in *Proceedings of the International Conference on Computer Design: VLSI in Computers*, 1984.

END

DATE

3-88

DTIC