1981

Overview of the Blue CHiP Computer

Lawrence Snyder

Report Number:
81-377

OVERVIEW OF THE CHiP COMPUTER

Lawrence Snyder

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907 USA

1. INTRODUCTION

There has been a rush to exploit VLSI technology by building special purpose devices tailored to a particular complex algorithm: tree machines for searching, sorting and expression evaluation, systolic array processors for numerical calculations, graph and combinatorial algorithms. The leverage comes from identifying locality (for high integration) and uniformity (for mass production) in the algorithm. But the problem remains:

> How does one *compose* these algorithmically specialized processors into a larger system?

We must put them together to solve more complex problems.

One solution to the composition problem is to attach a variety of these specialized processors to a bus, but the benefits of the devices are lost in wasteful interprocessor data movement. Alternatively, the algorithmically specialized processors could be emulated by microprocessors connected in a perfect shuffle or other general interconnection network, but this wastes enormous area, forgoes locality and introduces routing delays. The devices could be wired together, but this achieves only one composition and the best order for the outputs of one processor is not always the best order for the inputs to the next. The Configurable, Highly Parallel (CHiP) computer permits the composition of algorithms in a way that retains the locality and uniformity of the special purpose devices while providing flexibility.

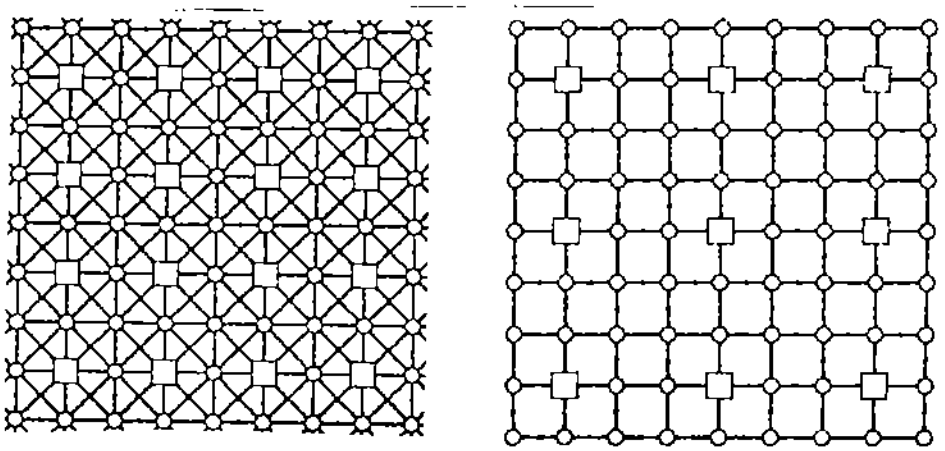
2. THE CHIP ARCHITECTURE FAMILY

First we present an overview of the main components and function of CHiP computers. In subsequent sections the

capabilities and limitations of the components are discussed in detail.

Informal Overview

CHiP architectures are characterized by a switch lattice connecting a set of homogeneous microprocessors (PEs) that is a slave to a controlling sequential computer (the controller). The *switch lattice* is a regular structure formed from programmable switches connected by data paths. The PEs are connected to the switch lattice at regular intervals rather than being directly connected to each other. External storage devices connect to the lattice at the perimeter switches.



(a) *Figure 1* (b)

Figure 1 shows two examples of switch lattices. The switches are shown as circles, the PEs as squares and the data paths as lines. Although the PEs and switches are drawn in roughly the same scale in the figure, the PEs are substantially greater in both area and capability. The examples represent a portion of the lattice that may contain 2^8 to 2^{16} PEs. Current technology permits only a portion of the lattice to be placed on a single chip, but because of the characteristics of the architecture discussed below, "wafer level" fabrication is possible.

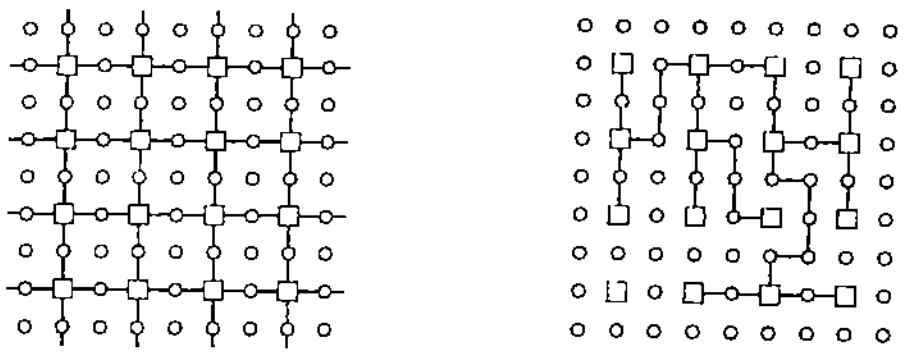
The switches are circuit rather than packet switches and each contains sufficient local memory to store several configuration settings. A configuration setting enables the switch to establish a direct, static connection between two or more of its incident data paths. For example, to achieve a mesh interconnection pattern of PEs for the lattice in

Figure 1(a), we assign North-South configuration settings to the switch rows and East-West settings to the switch columns. This pattern is illustrated in Figure 2(a). The same lattice has been configured into a binary tree pattern in Figure 2(b).

As mentioned in the Introduction one motivation for the CHiP architecture is to provide a flexible means of composing algorithms to solve large problems. Accordingly, we can visualize an algorithm as being divided into a sequence of phases, each with its own interconnection pattern. The PEs will each be performing different operations in the various phases as the emulated devices change.

To prepare for a sequence of phases, the controller loads the switch memories with the proper configuration settings to achieve the different interconnection patterns. This is performed by means of a separate interconnection "skeleton" that is transparent to the lattice. Typically, the loading of switch settings takes place in parallel with the loading of program segments for the phase into the PE memories. The configuration settings for the same phase are loaded into the same memory location in all of the switches. For example, the settings for a tree could be stored in location 1, the settings for a mesh in location 2, etc.

On a broadcast command from the controller, all switches implement the configuration setting in the same location. With the entire lattice configured, the PEs begin synchronously executing the instructions stored in their local memory in response to this same broadcast command. PEs need not know to whom they are connected; they simply execute instructions such as READ EAST and WRITE NORTHWEST. The configuration remains static until the controller broadcasts another command causing a different configuration setting to be implemented. The new interconnection pattern for the next



(a) (b)

Figure 2

phase is established in a single logical step and PE instruction execution resumes. (Detailed examples of this use of configurability are given in Gannon and Snyder 1981.)

Clearly, members of the family of CHiP architectures can differ in many ways: complexity of the lattice, functional complexity and memory capacity of the PEs, number of PEs, interconnection capability and memory capacity of the switches, width of the data paths, geometry and controller capacity. We next discuss some of these possibilities.

Processing elements

The computational capacity of the PEs largely determines the degree to which a CHiP machine is a general purpose computer and is thus influenced by the intended applications. For example, if the CHiP computer is to be used to simulate the action of other VLSI circuits for design verification purposes, PEs with a few dozen gates suffice to emulate a node of the circuit. Numerical algorithms provide an enormous class of applications requiring a floating-point arithmetic capability and substantial (100-200 instruction) programs. Since complex functions can be implemented in software, our intuition says that memory capacity is more important than functional capability. Early experience with CHiP algorithm design corroborates this view. We have been able to make effective use of a technique Schwartz (1980) calls "summarizing", and therefore we recommend that PEs used in an $n \times n$ lattice have sufficient memory to store at least n data values.

Switches

The operation of the switches is very simple and they might be implemented entirely with "steering" or "pass" transistors were it not necessary to intersperse drivers. Even so, they occupy area that is only a small factor larger than the minimum, m^2, where m is the number of wires of the data path. The number c of memory locations for storing configuration settings will be small (≤ 16) and the degree d, which is the number of incident data paths, will be either four or eight.

The crossover capability of a switch refers to the number of distinct data path groups that can be independently connected by a switch. We refer to "groups" rather than path pairs since fan-out is possible, i.e. more than two directions can be connected simultaneously. The crossover number g varies from 1 (no crossover) to $d/2$, and will

generally be two since it appears to be difficult to make very effective use of more crossovers.

The total number of bits required at a switch is thus, dgc, one for each direction for each crossover group for each configuration setting. Though this number is modest, it can be reduced by permitting settings to be assigned by the controller while the PEs are executing. This "asynchronous loading" capability exploits the fact that configurations often differ in only a few positions.

Lattice

The lattice structure determines the efficiency of PE utilization and the convenience of embedding interconnection patterns. The crucial variable is the *corridor width*, w, the number of data paths separating two adjacent PEs. (Recall that a single data path is formed from m wires.) Figure 1(a) shows a $w=1$ lattice while Figure 1(b) shows a $w=2$ lattice. Both lattices are uniform in the sense that all PEs are separated by the same size corridors, but it is possible to have a lattice with a variety of corridor widths. For example, the lattice of Figure 1(b) could be enhanced by enlarging the width of every fourth row and column of corridor pairs to a width of 4. Such an approach permits the lattice to be interpreted at several "levels" of detail: as an $n/4 \times n/4$ PE lattice with corridors of width 4 composed of logical processors that are a 4×4 lattice with $w=2$, or as an $n\times n$ lattice with $w=2$, by ignoring two of the four added corridors.

To see the impact of a particular choice of corridor width, we must study how the lattice *hosts* an interconnection pattern graph. There are two considerations when hosting a pattern graph: PE degree and edge density of complex interconnection paths. The matter of the PE degree, the number of incident edges, can be dismissed easily. If the pattern graph vertices, which correspond to PEs, have a degree in excess of the four or eight degree PEs typically provided, then we simply couple PEs together to give a larger "logical PE". For example, two adjacent degree four PEs can be logically coupled together to give a degree six PE; one of them could simply act as a buffer. Although this reduces the number of available PEs, the problem arises infrequently, since few processes require such large numbers of operands simultaneously. When few operands must be received from many possible sources, the large degree problem can be solved with the fan-in provided by the switches.

The second problem is that the graph's edge density may

require many different data paths to pass through a region of the CHiP lattice. In theory, even a one corridor lattice can host such a pattern, but to do so may require PEs to be unused in the region in order to provide sufficiently many paths. For example, Figure 3 shows an embedding of $K_{4,4}$ into a degree eight version of the lattice of Figure 1(b). In order to provide paths for the sixteen edges, the center four PEs must be unused. Increasing the corridor width obviously raises PE utilization.

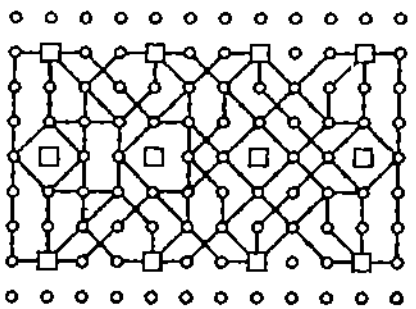


Figure 3

It also lowers the PE density. The fact that the number of PEs is linear in the area of this lattice means that PE utilization is inversely related to the area required to embed the pattern graph in the plane. Graphs requiring a nonlinear area will underutilize PEs just as circuits described by these graphs are composed mostly of wire, (Thompson 1980).

The decision on how wide corridors must be is influenced by the intended interconnection patterns and how economically necessary it is to maximize PE utilization. Fortunately, many algorithmically specialized processors developed for VLSI implementation have linear area interconnection graphs and can be hosted with optimal or near optimal PE utilization when the corridor width is only two. But to host any planar graph in an $n \times n$ PE lattice, an (average) width proportional to at least $(\log n)^{\frac{1}{2}}$ will be necessary, (Leighton 1981), to achieve optimal PE utilization. (Valiant, 1981 shows that $\log n$ width suffices.) For optimal utilization in more complex pattern embeddings such as the shuffle-exchange graph, a much larger width is required (e.g. proportional to at least $n/\log n$, Thompson 1980). These lattices with nonconstant corridor width have sublinear PE density per unit area.

The impact of these results is as follows.

> Up to constant multiplicative factors, the CHiP lattice with constant corridor width uses the silicon area as efficiently as direct VLSI implementations for all pattern graphs; CHiP lattices with constant corridor width as well as such universal interconnection structures as the shuffle-exchange graph cannot use the silicon area any more efficiently than direct VLSI implementation or constant corridor CHiP lattices and they are less efficient for linear area pattern graphs.

Evidently, a constant width corridor is indicated.

The estimates of the previous paragraphs are based on asymptotic results involving large constant factors and refer to a purely planar model. They can serve as guidelines (especially for "wafer level" fabrication), but more practical considerations are likely to influence the implemented lattice structure. For example, if only a small portion of the lattice fits on a single chip, the chips must be wired together which gives an opportunity to implement a complex nonlocal interconnection structure in the "third dimension". "Pin" limitations will also influence the decision. It may be more efficient to use the pins to increase the parallelism of data transmission using wide data paths through a narrow corridor than to use them for a wide corridor of narrow data paths. The benefits would accrue to the linear area patterns.

Efficient Embedding of Interconnection Patterns

Even though an interconnection pattern graph may have a linear area embedding in the plane, a direct translation of that embedding into the CHiP lattice may not be perfectly efficient. For example, the well known "hyper-H" planar embedding of a binary tree (Mead and Conway, 1980), when literally translated into a lattice of corridor width one leaves nearly half of the PEs unused. The reason is that unlike plain silicon, the CHiP lattice provides predetermined sites for the PEs which must be respected.

It is possible (Snyder 1981) to have a perfectly efficient embedding for the complete binary trees in a lattice for which $w=1$. That is, a $2^k \times 2^k$ PE lattice can host a complete binary tree with $2^{2k}-1$ nodes. Figure 4 illustrates a portion of this intricate, but straightforward, embedding.

Although the embedding of Figure 4 is efficient in terms

of PE utilization, there are other considerations to be weighed. In particular, propagation delay is an important problem and this embedding contains paths of length proportional to n for an $n \times n$ PE lattice. There are planar embeddings for complete binary trees with paths of length proportional to $n/\log n$ (Paterson *et al.*), which is the best possible. These embeddings can be made just as efficient in terms of PE utilization and would probably be preferred.

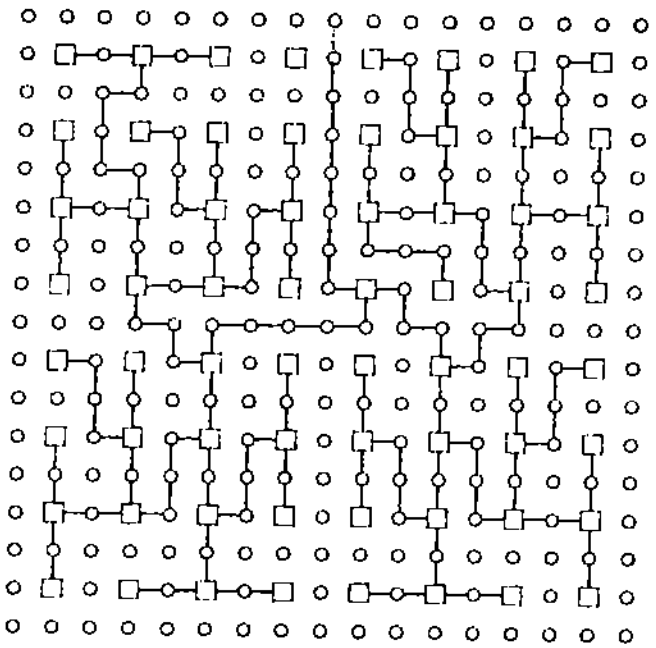


Figure 4

3. CONCLUSIONS

We have introduced the CHiP architectures and discussed some of the design decisions influencing their structure. The multiphased processing paradigm was discussed as a means of using configurability to compose algorithms. But there is another way to use configurability to compose algorithms.

Pipelined algorithms are best composed by coupling the PEs of one algorithm with those of the next in such a way that the outputs of the first become inputs to the second. The CHiP processor supports that composition method easily. Each of the algorithms is embedded in a region of the lattice so that the input side(s) of the first region is adjacent to the perimeter and the output side of this and subsequent processor arrays are adjacent to their

successor algorithm's PEs. This approach uses "intraphase" configurability (as opposed to the "interphase" configurability of the previously mentioned paradigm) to arrange for the embedding and connectivity of the processor arrays and to "scale" them since their size often depends on a parameter of the input. The controller can make these modifications at loading time. A complete example of composing the Kung and Leiserson systolic arrays (Mead and Conway, 1980) is given in Snyder, 1981.

Obviously, the CHiP architecture is very fault tolerant. Once a faulty PE, switch or data path is discovered, it is a simple matter to configure around the offending element(s). Perhaps the most effective way to use this facility is for wafer level fabrication. With this approach the wafer is viewed as a enormous chip where the dicing corridors are used for data path corridors. A wafer is accepted if, after testing, a regular $k \times k$ sublattice was found to be functional. Onboard mapping circuitry could map the addresses of the logical PEs and switches onto the functional elements of the wafer.

In summary, the CHiP architecture provides flexibility for algorithm composition. Accordingly, the many design parameters left unspecified in this discussion will be set based on our ongoing research into algorithm design for the CHiP architectures.

ACKNOWLEDGEMENTS

It is a great pleasure to thank my colleagues, Dennis Gannon, Janice Cuny, George Holober, Ching Hsiao, Paul McNabb and Kye Hedlund, who have contributed in innumerable ways to these ideas. The work described herein is part of the Blue CHiP project which is supported in part by Office of Naval Research Contracts N00014-80-K-0816 and N00014-81-K-0360. The latter is Task SRO-100.

REFERENCES

Gannon, D.B. and Snyder, L. (1981), "Linear Recurrence Systems For VLSI: The Configurable, Highly Parallel Approach," Int'l Conference on Parallel Processing, (to appear)

Leighton, F.T. (1981), "New Lower Bound Techniques For VLSI" Twenty second Ann. Symposium on the Foundations of Computer Science, IEEE (to appear)

Mead, C. and Conway, L. (1980) *Introduction to VLSI Systems* Addison-Wesley

Paterson, M.S., Ruzzo, W.L. and Snyder L. (1981) "Bounds on the minimax edge length for complete binary trees," Thirteenth Annual Symposium on the Theory of Computing, ACM

Schwartz, J.T. (1980)"Ultracomputers", ACM Transactions Programming Languages and Systems

Snyder, L. (1981) "Introduction to the Configurable, Highly Parallel Computer," Purdue University Tech. Report 351

Thompson, C.D. (1980), "A Complexity Theory for VLSI," Ph.D. Thesis, Carnegie-Mellon University

Valiant, L.G. (1981)"Universality Considerations for VLSI Circuits", IEEE Transactions on Computers