N94-18379

A Special Purpose Silicon Compiler For Designing Supercomputing VLSI Systems

* N.Venkateswaran,P.Murugavel,V.Kamakoti,M.J.ShankarRaman,S.Rangarajan, M.Mallikarjun,B.Karthikeyan,T.S.Prabhakar,V.Satish,P.R.Venkatasubramaniam, R.Sivakumar,R.Srinivasan,S.Chandrasekhar,G.Suresh,M.B.Karthikeyan, S.Ramachandran,S.Sankar,P.V.Balaji,P.Kishore

** F.Lawrence,S.Pattabiraman,G.Suresh,V.Arun Shankar,A.Ashraf,V.Balaji, M.Balaji,Sunil.K,R.Devanathan,Y.Eliyas,K.Krishnan,B.Krishna Kumar, B.Kumaran,N.Rajesh & G.Vijay Venkatesh

**Department of Computer Science and Engineering.
Sri Venkateswara College of Engineering
University of Madras
Nazarethpet, Madras 602 103, India.

Abstract- **Design of general/special purpose Supercomputing VLSI systems for numeric algorithm execution involves tackling two important aspects namely their computational and communication complexities. Development of software tools for designing such systems itself becomes complex. Hence a novel design methodology has to be developed. For designing such complex systems a special purpose silicon compiler is needed in which**

1. **The computational and communicational structures of different numeric algorithms should be taken into account to simplify the silicon compiler design.**

2. **The approach is macrocell based.**

3. **The software tools at different levels,algorithm down to the VLSI circuit layout, should get integrated.**

In this paper a special purpose silicon (SPS) compiler based on PACUBE macrocell VLSI ARRAYS [1] for designing supercomputing VLSI systems is presented. It is shown that turn-around-time and silicon real estate get reduced over the silicon compilers based on PLAs,SLAs and gate arrays.

Characteristics 1 and 2 above enable the SPS compiler to perform systolic mapping (at the macrocell level) of algorithms whose computational structures are of GIPOP (Generalized Inner Product Outer Product) form [2]. Direct systolic mapping on PLAs, SLAs and gate arrays is very difficult as they are micro-cell based. A novel GIPOP processor is under development using this special purpose silicon compiler.

---

* pursuing their higher studies/employed in India or abroad. Contact for communication regarding this paper N.Venkateswaran, Additional Professor, Dept of Computer Science and Engineering, Sri Venkateswara College of Engineering,University of Madras,India. Authors' names listed randomly.

1 Introduction

No silicon compiler has yet been developed exclusively for tackling the complexity in designing supercomputing VLSI systems. In developing such a compiler besides achieving reduced turn-around-time and area, computational performance of mapped functional units and architectural characteristics like systolic mapping should also be considered. The latter two factors are not taken care of in micro-cell based silicon compilers.

In conventional compilers the software tools are not integrated from algorithm down to the VLSI circuit level. The integration of software tools at different levels can be achieved by adopting novel methodologies in designing supercomputing VLSI systems (processors and arrays) and developing novel algorithm mapping techniques. This integration reduces the software complexity to a great extent.

The turn-around-time is high if gate arrays, PLAs and SLAs are employed for supercomputing system synthesis. Further the silicon compilation should take place at a much higher level than at the gate or the micro-cell level for supercomputing system design.

The PACUBE macro-cell structure is a combination of PLAs, SLAs,gate arrays and standard cells.Besides storage and logic elements an important computing unit,the DRAA (Dynamically Reconfigurable Array Adder [1]) is also present (Fig 1). The presence of the DRAA reduces the turn-around-time and silicon area drastically for designing special purpose VLSI systems. The DRAA helps in achieving high performance due to its functional and architectural characteristics. If the DRAA were to be mapped on to PLAs, SLAs and gate arrays the performance will get degraded.The special purpose silicon compiler built for mapping supercomputing systems on the PACUBE arrays achieves all the above factors.

2 PACUBE Macrocell Array And Systolic Mapping Tools.

2.1 Unified GIPOP Operations On Macrocell Arrays

Execution of number of numeric algorithms involve inner-product operations. Several VLSI systems have been proposed for executing numeric algorithms whose computational structures are of inner-product form. For this purpose VLSI arrays of inner-product step processors have been employed conventionally.

In general the computational structure of numeric algorithms are complex. However on a closer study it is noted that these structures can be brought under a generalized form called Generalized Inner-Product Outer-Product (GIPOP) functions. These are

$$I = \sum_{i=1}^{n} \frac{(A_i * B_i)}{C_i} \cdots \quad (1)$$

$$O = \prod_{i=1}^{n} \frac{(A_i + B_i)}{C_i} \cdots \quad (2)$$

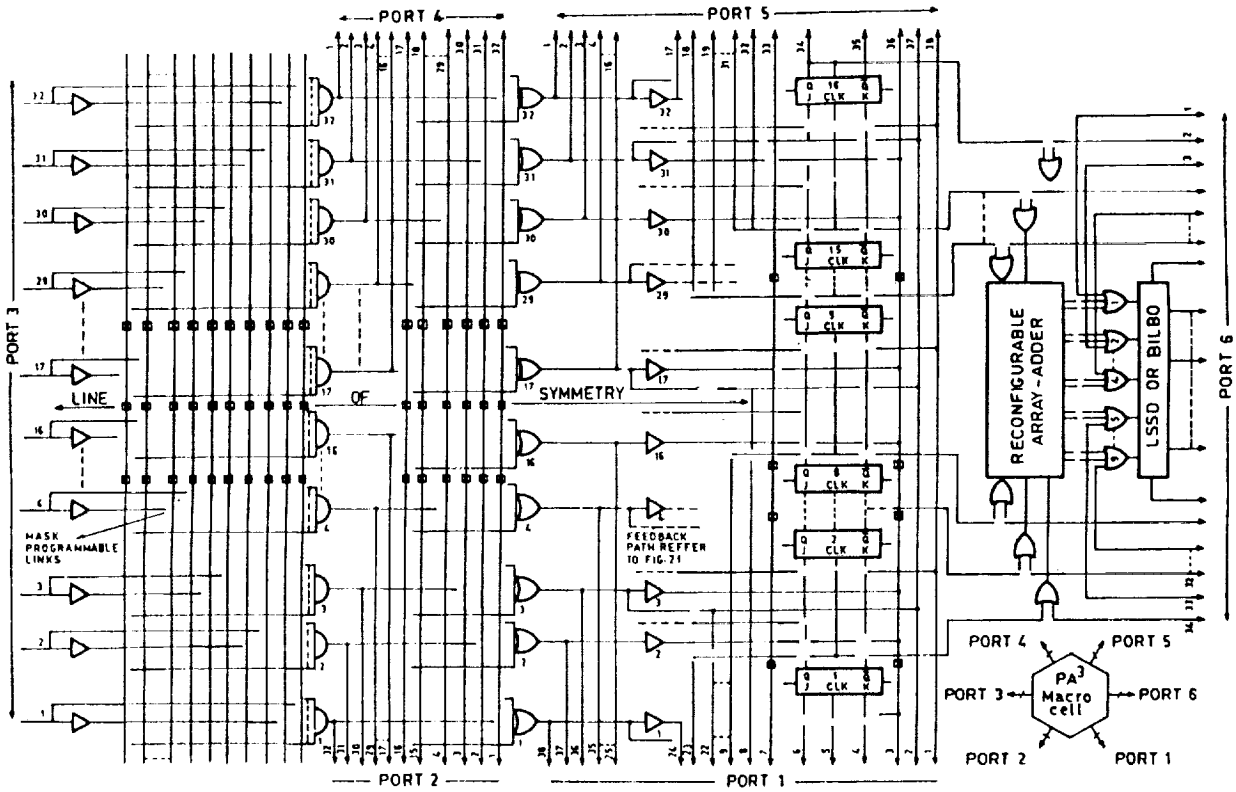


Figure 1: PAi^3 (ASLA) Logic Model

The computational structures of numeric algorithms may involve a mathematical combination of the above two equations. Evaluation of GIPOP functions based computations involve inner-product operations, chain multiplications, outer-product operations and reciprocal operations.

It is shown in this paper that by using PACUBE VLSI arrays [1] these GIPOP functions can be evaluated as a sum of equivalent weighted inner-product functions only. Also massive parallelism can be employed in GIPOP operations. This unification of the execution processes of GIPOP functions only in terms of equivalent inner-product operations is achieved by establishing an identical inter and intra macrocell connections (data flow mapping) on PACUBE arrays for chain multiplication and inner-product operations. Both the outer-product and reciprocal operations can be expressed in terms of sum of chain multiplication operations [3].

Let the inner-product,chain multiplication, outer-product and reciprocal operations be defined as follows.

$$\sum_{i=1}^{T_t}\{w(S_i)_p * w(K_i)_p\} = I\{w(S_i)_p, w(K_i)_p, T_t\} \quad (3)$$

$$\prod_{i=1}^{T_t}\{w(S_i)_p + w(K_i)_p\} = O\{w(S_i)_p, w(K_i)_p, T_t\} \quad (4)$$

$$\prod_{i=1}^{T_t}\{{}_w(S_i)_p\} = C\{{}_w(S_i)_p, T_t\}$$

$$= C\{{}_w(S_1)_p, {}_w(S_2)_p, \cdots {}_w(S_{T_t})_p\} \quad (5)$$

$$\frac{1}{{}_w(S_i)_p} = R\{{}_w(S_i)_p\} \quad (6)$$

$S1, S2, \ldots, Si$ and $K1, K2, \ldots, Ki$ are the operands of word length p bits.

w - The weight of the MSB of the word length.

T_t - Number of inner-product terms (operand pairs) or
Number of chain multiplication terms (operands).

$${}_w(S_i)_p \longrightarrow \{{}_{w_n}(S_i)_{p_n}, {}_{w_{n-1}}(S_i)_{p_{n-1}}, \cdots {}_{w1}(S_i)_{p_1}\} \quad (7)$$

$${}_w(K_i)_p \longrightarrow \{{}_{w_n}(K_i)_{p_n}, {}_{w_{n-1}}(S_i)_{p_{n-1}}, \cdots {}_{w1}(K_i)_{p_1}\} \quad (8)$$

p_i - Word length of the partition i, $i = 1$ to n

w_i - Weight of the MSB of the partition $(p_1 + p_2 + \cdots p_i)$

Using the relationships (7) and (8) we get

$$p = p_1 + p_2 + \cdots + p_n = \sum_{i=1}^{T_t} p_i$$

$$w = w_1 + w_2 + \cdots + w_n = \sum_{i=1}^{T_t} w_i \quad (9)$$

('*' Defines binary multiplication)

$${}_w(S_i)_p * {}_w(K_i)_p = {}_{w+w}(Q_i)_{p+p} \quad (10)$$

2.2 Inner-product operations on PACUBE Arrays

Execution of inner product operations on PACUBE arrays has been dealt in [1]. To achieve massive parallelism in evaluating inner-product functions partial product arrays (PPAs) of the different product terms

$$(A_1 * B_1, A_2 * B_2, \cdots, A_n * B_n)$$

are obtained in parallel (forming a massive array) and added simultaneously [1]. Refer to Fig. 2a. There are three different ways of massive array formation and reduction [2]. They are called MAR1, MAR2 and MAR3 processes. The figure 2a corresponds to

$$I({}_3(A_i)_4, {}_3(B_i)_4, 4)$$

function.The reduction processes corresponding to MAR2 and MAR3 are similar to MAR1 reduction process presented in [1].

The MAR Process should be chosen such that the following important criteria are taken care of

1. The operands [4,4] sub matrices of the massive array injection into the PACUBE array should be simpler.
2. The partial sum bits output corresponding to the sum output of the massive array should occur in consecutive cycles of the array reduction process.
3. The partial sum output should be of same length in all cycles .
4. The partial sum output should occur only in the peripheral macrocells.
5. The intra macrocell data flow should be regular i.e. there is no multiplexing of data between macrocells.
6. Number of array reduction cycles and number of macrocells should be minimum.

2.3 Chain Multiplication On PACUBE Arrays

The application of PACUBE array can be extended for chain multiplication operation. Consider the massive array formation for

$$C\{_3(A_i)_4,\ 3\}$$

Similar to inner product operation three different types of massive array can be formed for executing chain multiplication. Refer to Fig. 2b. Only MAR2 massive array formation is shown.

2.4 Unification Of Chain Multiplication And Inner Product Operations On PACUBE Arrays

There is a striking similarity between array formation corresponding to the inner product and chain multiplication operations. The only difference is in the array sizes. The massive array of $C\{_3(A_i)_4,\ 3\}$ is larger than that of $I\{_3(A_i)_4,\ _3(B_i)_4,\ 4\}$. In this example the difference in array sizes is not much. In general the massive arrays corresponding to the inner product and chain multiplication operations can be made of comparable sizes by a proper choice of the word length and number of terms. Hence identical array reduction process having same inter and intra macrocell data flow can be established on the PACUBE macrocell arrays. The operand injection points differ for these operations. Hence structurally these two operations are equivalent. Such equivalent pairs may not exist for certain values of word length and number of terms. In some cases even if the equivalent pairs exist the word length of the pairs may be of odd values. The word length of the equivalent pairs should have values in powers of 2. It is preferable to have equal word length for such equivalent pairs. The values of number of terms (problem size) can be adjusted to achieve this.

For example in the equivalent pair $I\{_8(S_i)_8,\ _8(K_i)_8,\ 8\}$ and $C\{_8(S_i)_8,\ 3\}$ the word lengths are same but the number of terms are different. The equivalent pair

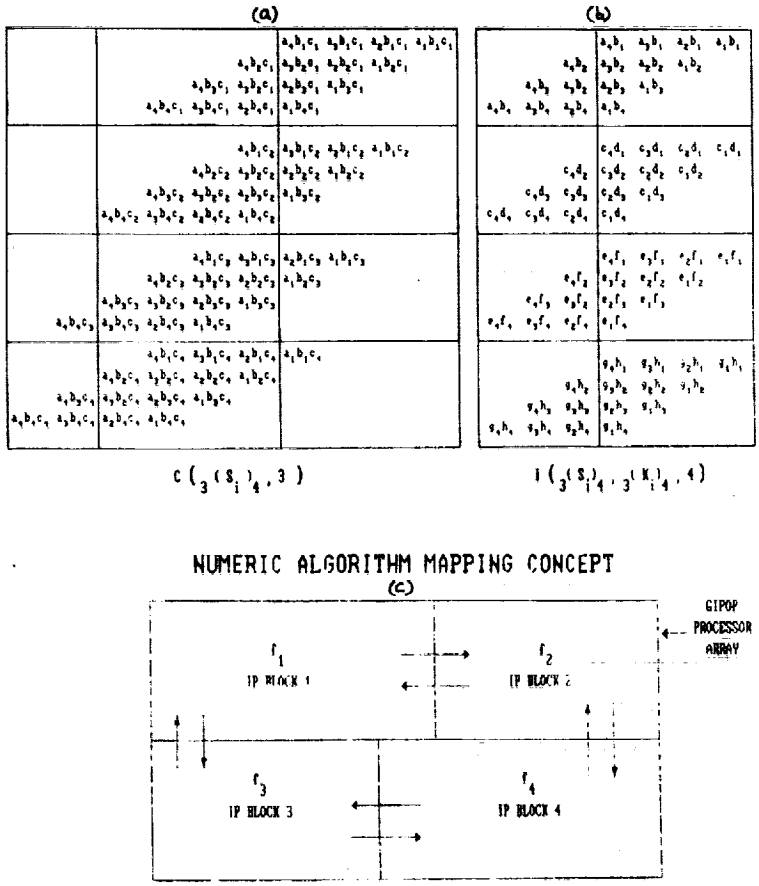


Figure 2: MAR process-2 array formation

$I\{_{16}(S_i)_{16}, {}_{16}(K_i)_{16}, 4\}$, $C\{_{16}(S_1)_{16}, {}_{8}(S_2)_{8}, {}_{8}(S_3)_{8}, 3\}$ has different word length and problem size. But $C\{_{16}(S_1)_{16}, {}_{16}(S_2)_{16}, {}_{16}(S_3)_{16}, 3\}$ can be easily decomposed (by proper word length and term/problem size partitioning) in terms of $C\{_{16}(S_1)_{16}, {}_{8}(S_2)_{8}, {}_{8}(S_3)_{8}, 3\}$. That is $C\{_{16}(S_i)_{16}, T_i\}$ can be decomposed to $I\{_{16}(S_i)_{16}, {}_{16}(K_i)_{16}, 4\}$. Further details on this is dealt in section 3.3.

This leads to a unified PACUBE VLSI array for executing Inner product and Chain multiplication operations. Outer product operation and high speed multiplicative division algorithm [3] are based on chain multiplication operation. Hence the execution processes of GIPOP functions can be unified on the PACUBE macrocell arrays.

2.5 Systolic Mapping Of GIPOP Functions

An algorithm has been developed for automatic systolic mapping of GIPOP function execution on the P-arrays and implemented under DOS. (Fig 3). The unification of the execution processes of the GIPOP functions has greatly simplified the development of systolic mapping tools.

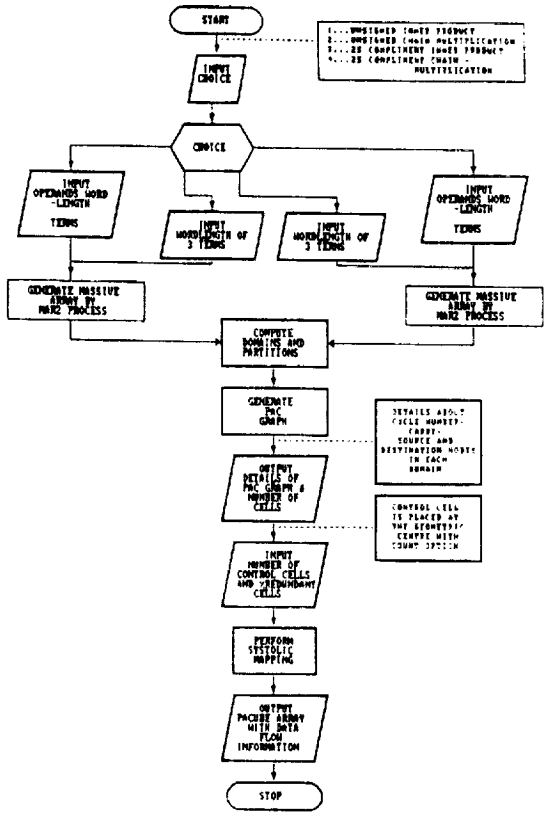


Figure 3: Systolic mapping of GIPOP functions on Pacube arrays

3 GIPOP Processor Array And Software Tools For Algorithm Mapping

3.1 GIPOP Processor

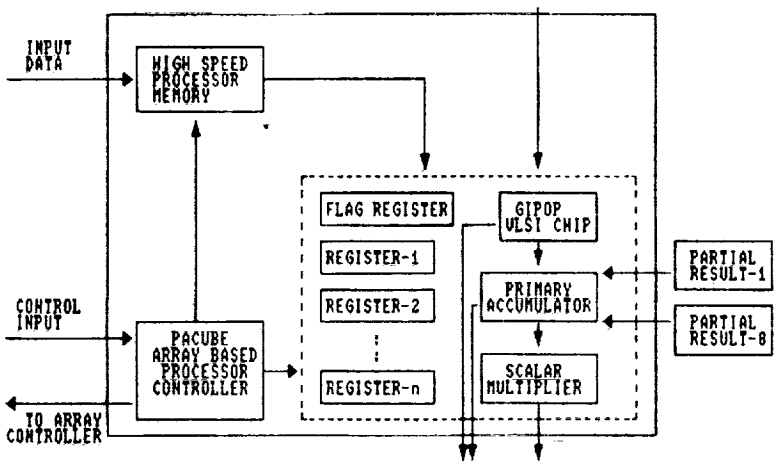
Existing processor arrays meant for supercomputing are classified into special purpose and programmable general purpose arrays. An attempt to combine the advantages of these approaches has culminated in a novel processor design approach, namely GIPOP processor arrays. This novel approach is expected to give very high performance/cost ratio for supercomputing systems [2].

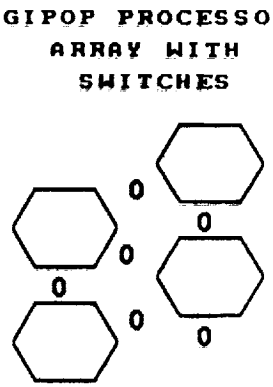
The internal architecture of the GIPOP processor and its instruction set is shown in Figure 4. The simulation of the instruction set has been completed. The GIPOP processor can execute the equivalent pair $I\{_3(S_i)_4, {}_3(K_i)_4, 4\}$ and $C\{_3(S_i)_4, 3\}$.

The equivalent pairs are chosen based on the PACUBE macrocell, GIPOP processor and the array complexities.

3.2 GIPOP Processor Array

Two levels of pipelining take place in algorithm execution on the GIPOP processor array partly shown in Figure 4 ,one at the macro-cell level within the GIPOP chip and the other at the processor level. The Processor level pipelining is controlled by the Array Control





GIPOP INSTRUCTION SET

INSTRUCTION	FORMAT
MOV	DESTINATION , SOURCE , VECTOR CYCLES

INSTRUCTION	OPERANDS		COMMENTS
MOV	GPR , M	;	Pipelining of data from memory
		;	or input port through
MOV	GPR , I	;	general purpose registers.
MOV	CM , M	;	Loads three four-bit operands
		;	for chain multiplication operation
MOV	CM , I	;	either from memory or input port.
MOV	IP , M	;	Loads eight four-bit inner-product
		;	operands from memory
MOV	IP , I	;	or input port.
MOV	PAC , M	;	Loads eight four-bit
		;	massive array operands
MOV	PAC , I	;	in to the primary accumulator.
MOV	PAC , GPOP	;	Accumulate the output of GIPOP
		;	in the primary accumulator.
MOV	MLT , M	;	Loads four-bit multiplier from
		;	memory to scalar multiplier.
MOV	MLT , PAC	;	Loads the multiplicand from the
		;	accumulator to scalar multiplier.
MOV	0 , GPR	;	Transfer the ouput
		;	of general purpose registers,
MOV	0 , GPOP	;	GIPOP , primary accumulator ,
		;	and the scalar multiplier
MOV	0 , PAC	;	to the output port.(these
		;	loadings can be done in
MOV	0 , MLT	;	parallel).

$$[I_{3}(Si)_{4,3}(\overline{Ki})_{4}, C_{3}(S1)_{4,3}(S2)_{4,3}(\overline{S3})_{4}, 3)]$$

Figure 4: GIPOP processor archuitecture for equivalent pair

Unit (ACU) and the macrocell level pipelining is controlled by the chip control unit (CCU). The hardware complexity of the switch lattice depends on the processor complexity, data communication complexity in an algorithm and the word length. The design of the ACU is under progress.

3.3 Mapping Of Numeric Algorithms On GIPOP Processor Array

Mapping of numeric algorithms on the GIPOP processor array involves tackling the computational (levels 1 & 2) and communicational (level 3) complexities.

Level 1. Decomposing the computational structure of the algorithm in terms of GIPOP equations which are further decomposed in terms of the inner product functions only [1].

Level 2. The architectural capabilities of a GIPOP processor is bounded by problem size and wordlength. Suitable algorithms for problem size and word length partitioning of GIPOP functions and the corresponding software tools have been developed. The algorithm is based on definitions (3) - (6) []. Refer to Figure 5.

Level 3. a. Proper loading of input operand frames into the high speed processor memory (Block level memory loading) from the system memory.
b. Programming the processor and array control units.

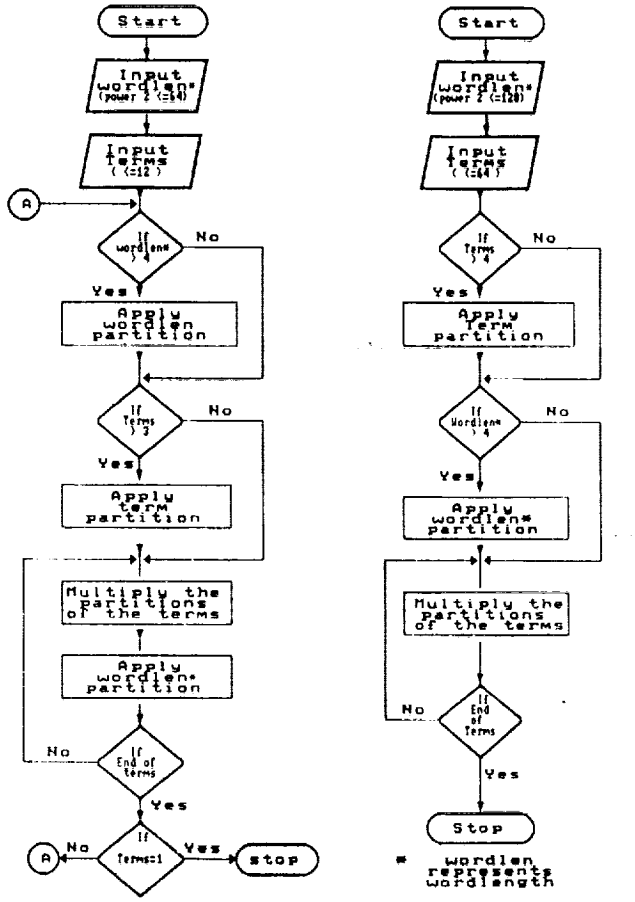
Numeric algorithm mapping on the GIPOP arrays basically involve mapping of different inner product blocks (IP Blocks) of variable complexities (see Fig 2c). Functionally each of these IP Blocks may correspond to different GIPOP operations.The data flow between the corresponding group of GIPOP processors (making an IP block) can be syntactically described including both the functional and behavioral aspects of the GIPOP processor [2].Mapping of an algorithm on the arrays is to get this syntactical description of the data flow.

Software tools for levels 1 and 3 are being developed. The novel concept of mapping numeric algorithms on the GIPOP processor array shown in Figure 2c greatly simplifies the development of software tools for levels 1 and 3 above.

4 PACUBE Logic Level

4.1 Inter And Intra Cell Routing.

An efficient algorithm for inter and intra macrocell routing has been developed taking into account the shortest path considerations. The software tools are shown in Figure 6 as flowcharts.



Chain Multiplication Inner Product

Figure 5: Wordlength & term partitioning

4.2 Subprogram (Functional Units) Library Generation And Linking.

Several subprograms, sequential and combinatorial,have been mapped on the PACUBE macrocell. An efficient PACUBE Hardware Description Language (PHDL) has been developed. The subprogram linking is done using this PHDL and the related software tool has been developed. (Fig. 7)

5 PACUBE Circuit Level Discussions

5.1 Interactive Layout Generation And Checking.

A software tool for the generation of device level layouts in an interactive fashion has been developed (Fig. 8). It provides four layers viz Diffusion, Polysilicon and two levels of metal. The package supports both n-well and p-well CMOS processes. Lambda based design rules have been adopted. Special facilities such as mirroring, translation, rotation,

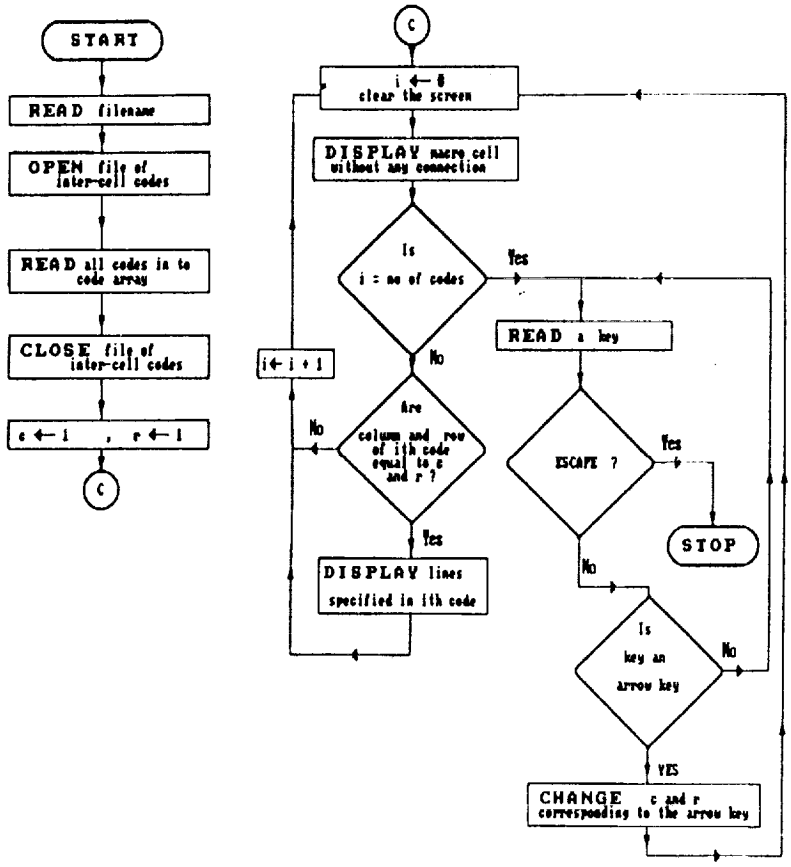


Figure 6: Inter-cell logic level tool

step & repeat , cut & paste and scaling are available to aid in faster design. The layout of an entire macrocell has been developed using this tool. Layout geometries are expressed using PACUBE device level codes. The design rule checker developed is edge based and performs both inter and intra layer checking.

The different functional units of the macrocell are treated as standard cells and depending on the application the required standard cells can be placed within the macrocell. This option gives rise to macrocell arrays with different sizes of macrocells.

5.2 Intracell And Intercell Mapping.

A software translator for the PACUBE logic level code to PACUBE device level code conversion and a device level to Caltech Intermediate Format (CIF) translator has also been developed.

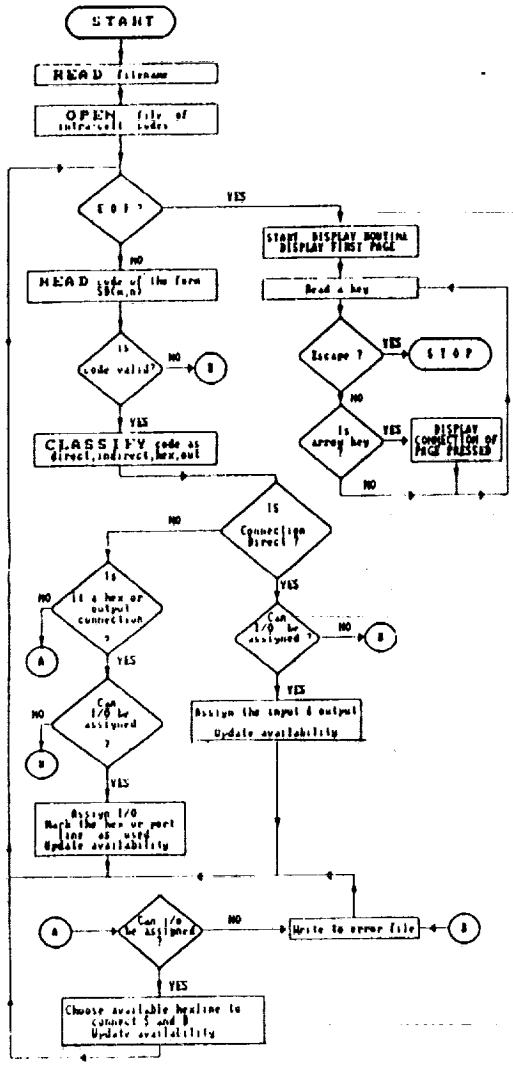


Figure 7: Intra-cell logic level tool

5.3 Simulation

Circuit simulation of different functional units of the macrocell for 1.2 micron technology using PSPICE is nearing completion. Logic level simulation of PACUBE macrocell for GIPOP operations has been carried out using the PACUBE logic simulator.

6 Conclusion

In this paper novel methodologies have been proposed for designing silicon compilers for synthesising supercomputing VLSI systems. The important criteria for such a silicon compiler are integration of its software tools and the architectural considerations.

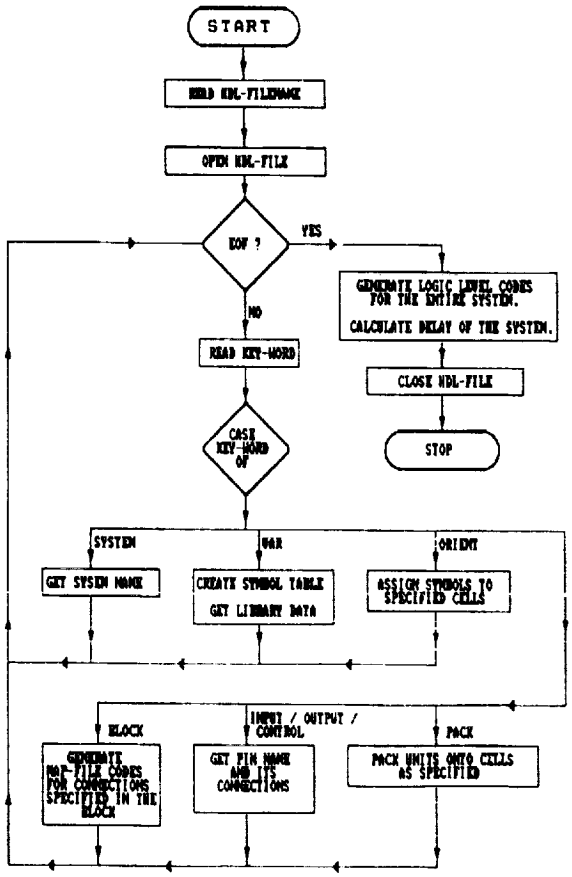


Figure 8: Sub-program linker

7 Future Work

An algorithm for extracting the computational complexities of numeric algorithms in terms of GIPOP functions is to be taken up. A methodology is to be developed for mapping the communication graph of numeric algorithms on to the GIPOP processor array as shown in Figure 2c. Reconfigurable fault-tolerant PACUBE arrays [2] has been developed and the corresponding software tools to incorporate this into the silicon compiler has to be taken up.

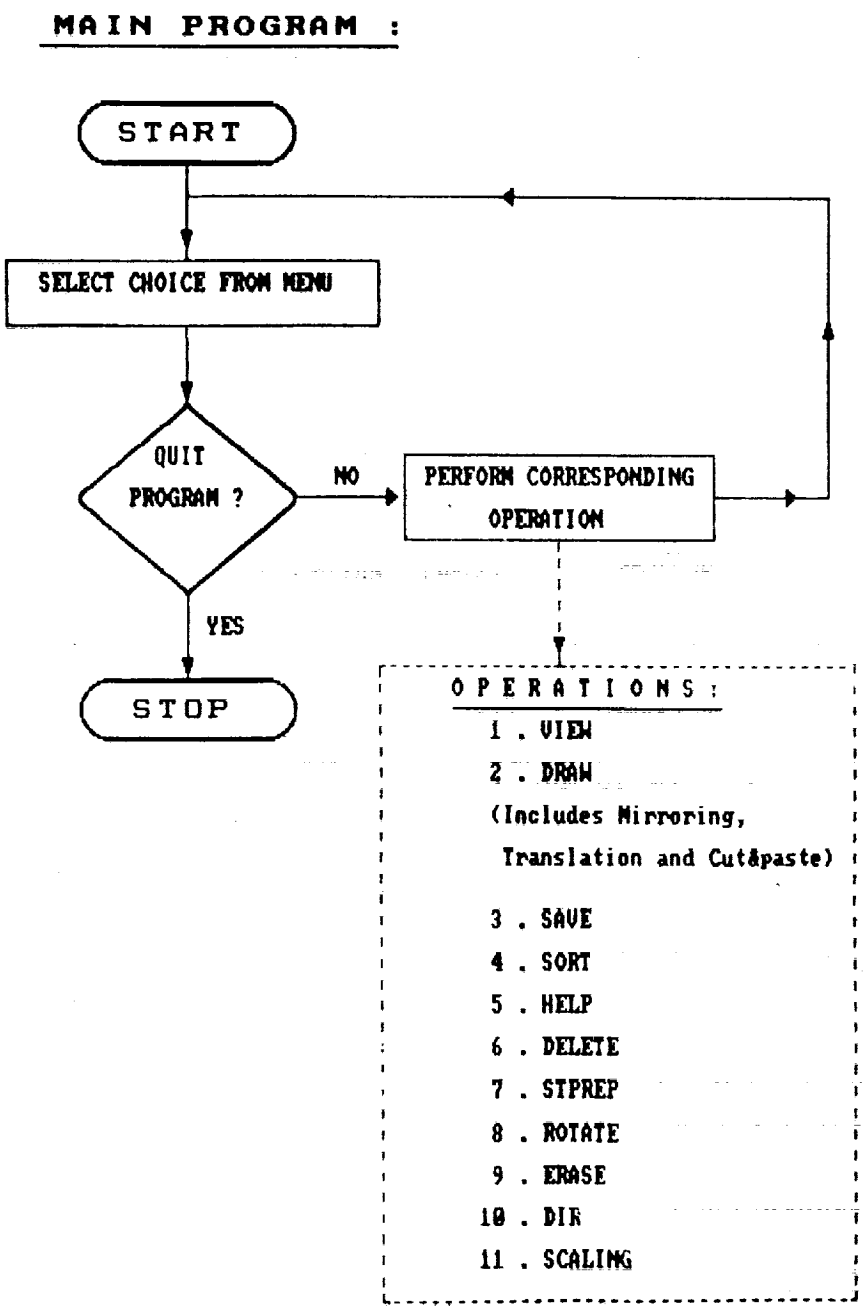


Figure 9: Pacube macro cell based device level layout editor

References

[1] N.Venkateswaran, PACUBE Arrays For Supercomputers, *Proc. of the First International Conference on Supercomputing Systems*, conducted by the IEEE Computer Society and Pentagon, held at Florida, U.S.A., Dec.6-10,1985.

[2] N.Venkateswaran, et al., Supercomputing Systems on PACUBE VLSI Arrays, Research Report. Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, University of Madras, India.

[3] Joseph J.F.Cavanagh, Digital Computer Arithmetic Design and Implementation, McGraw Hill publishing house.