A new Partitioning Scheme for Improvement of Image Computation

Christoph Meinel
FB IV - Informatik
University of Trier
meinel@uni-trier.de

Christian Stangier
FB IV - Informatik
University of Trier
stangier@uni-trier.de

Abstract— Image computation is the core operation for optimization and formal verification of sequential systems like controllers or protocols. State exploration techniques based on OBDDs use a partitioned representation of the transition relation to keep the OBDD-sizes manageable. This paper presents a new approach that significantly increases the quality of the partitioning of the transition relation of finite state machines. The heuristic has been successfully applied to reachability analysis and symbolic model checking of real life designs, resulting in a significant reduction in CPU time as well as in memory consumption.

I. Introduction

The computation of the reachable states (RS) of a finite state machine (FSM) is an important task for synthesis, logic optimization and formal verification. The increasing complexity of sequential systems like controllers or protocols requires efficient RS computation methods. If the RS are computed by using Ordered Binary Decision Diagrams (OBDDs) [2], the system under consideration is represented in terms of a transition relation (TR). Since the monolithic representation of the circuit's TR usually leads to unmanageable large OBDD-sizes, the TR has to be partitioned [3, 6]. The quality of the partitioning is crucial for the efficiency of the RS computation. The computation of transitions will be unnecessarily time consuming, if the TR is divided into too many parts. On the other hand a number of partitions that is too small will lead to a blow-up of OBDD-size and hence, memory consumption. Partitioning the TR is usually done without utilizing any external information. The standard method is to to sort the latches according to a benefit heuristic [7, 10] and then, apply a clustering algorithm. This clustering algorithm follows a greedy scheme [5] that is guided only by OBDD-size, i.e if the OBDD-size of a partition is exceeding a certain threshold a new partition has to be created.

Recently a new approach [9] has been proposed to improve partitioning of the transition relation. This approach utilizes RTL (register transfer level) information provided by the hardware description of the design. The RTL information is used to find a grouping of the transition functions. Experiments have shown that this approach performs significantly better for modularized designs than the standard method.

In this paper we propose a heuristic for partitioning of TRs that adopts the grouping paradigm of the RTL heuristic without actually using RTL information. The result is a broader applicability.

Our heuristic has been successfully applied to real life benchmark examples given in Verilog. The application of our heuristic to model checking [4] reduced the computation time by 72% and memory consumption by 70% (overall) compared to the standard method.

II. Preliminaries

A. Hardware description languages

Since modern complex designs require a structured hierarchical description to be feasible they are currently written in a hardware description language (HDL) at register transfer level (RTL). The term RTL is used for an HDL description style that utilizes a combination of *data flow* and *behavioral constructs*. Logic synthesis tools take the RTL HDL description to produce an optimized gate level netlist and high level synthesis tools at the behavioral level output RTL HDL descriptions. Verilog [12] and VHDL [8] are the most popular HDLs used for describing the functionality at RTL. Within the design cycle of optimization and verification the RTL level is an important and frequently used part.

The design methodology in Verilog is a top down hierarchical modeling concept based on modules, which are the basic building block. Our experimental work is based on designs written in this language.

B. Partitioned Transition Relations

The computation of the RS is a core task for optimization and verification of sequential systems. The essential part of OBDD-based traversal techniques is the transition relation TR:

$$\mathrm{TR}(x, y, e) = \prod_i (\delta_i(x_i, e) \equiv y_i),$$

which is the conjunction of the transition relations of all latches (δ_i denotes the transition function of the ith latch). This *monolithic* TR is represented as a single OBDD and usually is much too large to allow an efficient computation of the RS. Sometimes a monolithic TR is too large to be represented with OBDDs. Therefore, more sophisticated RS computation methods make use of a *partitioned* TR [3], i.e. a cluster of OBDDs each

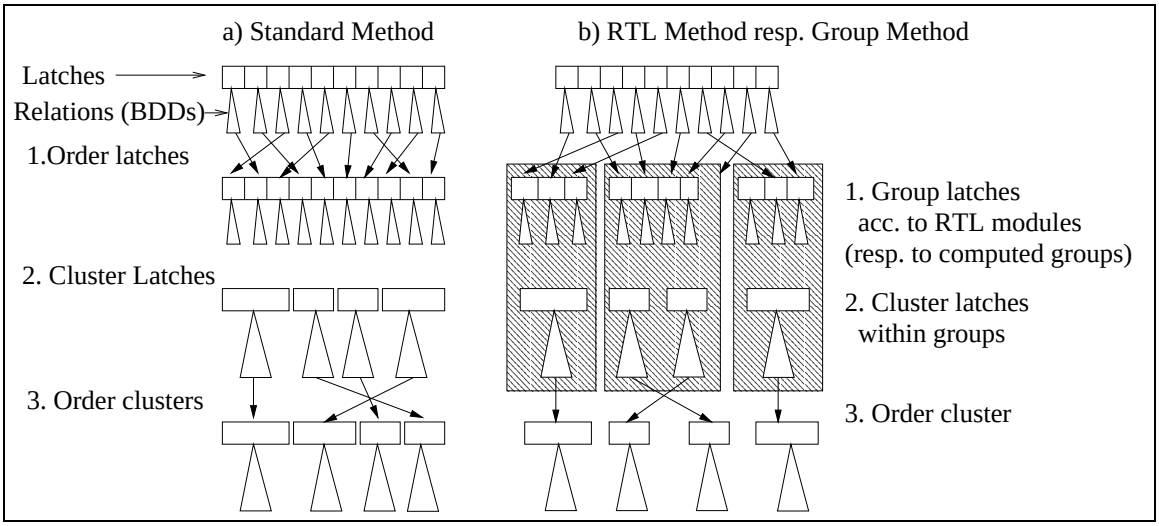

Fig. 1. Schematic of Partitioning Strategies.

of them representing the TR of a group of latches. A transition relation partitioned over sets of latches $P_1, \ldots, P_j$ can be described as follows:

$$\mathrm{TR}(x, y, e) = \prod_j \prod_{i \in P_j} (\delta_i(x_i, e) \equiv y_i).$$

The RS computation consists of repeated image computations $Img(\mathrm{TR}, R)$ of a set of already reached states R:

$$Img(\mathrm{TR}, R) = \exists_{x,e}(\mathrm{TR}(x, y, e) \cdot R)$$

With the use of a partitioned TR the image computation can be iterated over P_j and the $\exists$ operation can be applied during the product computation *(early quantification)*. The so called *AndExist* [3] or *AndAbstract* operation performs the AND operation on two functions (here partitions) while simultaneously applying existential quantification ($\exists_x f = f_{x=1} \vee f_{x=0}$) on a given set of variables, i.e the variables that are not in the support of the remaining partitions. Unlike the conventional AND operation the AndExist operation does not have a polynomial upper bound for the size of the resulting OBDD, but for many practical applications it prevents a blow-up of OBDD-size during the image computation.

Since the number of quantified variables depends on the order in which the partitions are processed, finding an optimal order of the partitions for the AndExist operation is an important problem. Geist and Beer [7] presented a heuristic for the ordering of partitions each representing a single state variable. A more sophisticated heuristic for partitions with several variables is given by [10].

III. Partitioning of Transition Relations

The quality of the partitioning is crucial for the efficiency of the RS computation. The image computation is iterated over the partitions and includes costly product (i.e. AND) computations. Therefore, maintaining a large number of partitions is time consuming. A small number of partitions may lead to unmanageable large OBDDs. One extremum of this trade-off is the partitioning where each latch forms a partition, which is usually small but requires many iterations. The other extremum is a monolithic TR, that can be computed in one iteration but has large OBDD-size. Furthermore, the ordering of latches and clusters is crucial for an efficient AndExist operation.

In the following we will describe the standard partitioning strategy, the *RTL partitioning heuristic*, which is the basis of our work and our new approach.

A. Common partitioning strategy

A common strategy for partitioning of the TR as it is used e.g. by VIS [5, 10] proceeds in three steps:

1. **Order latches**. First, the latches are ordered by using a benefit heuristic [7] that performs a structural analysis of the latches' transition function to address an effective AndExist operation. The main target of the heuristic is to keep the number of variables during the AndExist small, therefore it uses a greedy scheme to minimize the balance of introduced next-state variables and quantified present-state variables. Additionally, influences like highest index of a variable to be quantified out are considered.
2. **Cluster latches**. The single latch relations are clustered by again following a greedy strategy. Latches are added to a OBDD (i.e. by performing AND) until the size of the OBDD exceeds a given threshold.
3. **Order clusters**. In the last step the clusters are ordered similarly to the latches by using a benefit heuristic (VIS uses the same heuristic as in Step 1).

Figure 1 a) gives a schematic overview of this process.

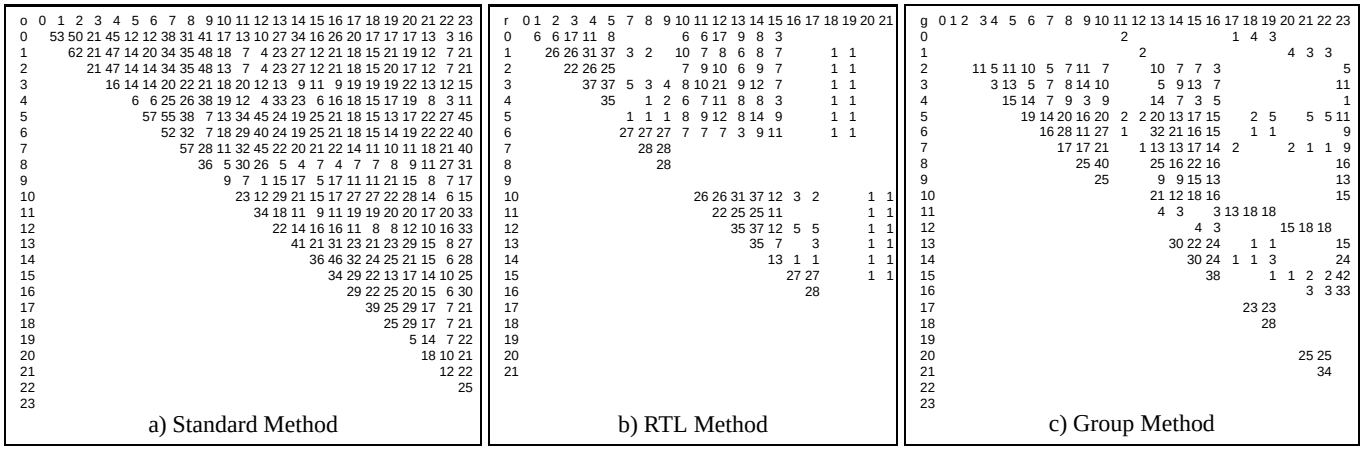

Fig. 2. Cluster dependencies for different partitioning schemes.

B. RTL based Partitioning Heuristic

Recently a different partitioning strategy has been proposed. The main idea of the RTL partitioning heuristic [9] is to utilize the hierarchical RTL information of the given circuit to obtain a good partitioning.

Although the standard method optimizes the partitioning twice, its main disadvantage is that it only uses structural information to optimize the partitioning for an efficient schedule for the AndExist operation during the image computation.

The RTL heuristic improves this optimization by including additional semantical information about the represented functions. The heuristic that is based on Verilog-RTL proceeds in three steps:

1. **Group latches**. The latches are grouped according to the modules given in the top module of the RTL description in Verilog. Within the groups the latches are ordered by bit numbers.

2. **Cluster groups**. The groups represent borders for the clusters. There is no cluster containing latches from different groups. To control the OBDD size of the clusters, the greedy partitioning strategy is applied within the groups. The clustering given by the groups lowers the influence of the arbitrary clustering produced by the OBDD-size threshold. Thus, resulting in a more *natural* partitioning.

3. **Order clusters**. As a possible last step the clusters may be ordered by using the benefit heuristic from the standard method.

Figure 1 b) gives an overview of this strategy.

C. Group based Partitioning Heuristic

The RTL heuristic performs significantly better for modularized circuits than the standard method. But what, if the RTL information is not accessible during RS computation or the information is lost due to e.g. optimization?

In the following we develop a heuristic that adopts the benefits of the RTL heuristic without using RTL information.

Cluster dependencies

The quality of the partitioning may be described by a cluster dependency matrix CDM. Entry (i, j) of CDM contains the number of support variables that cluster i and cluster j have in common. As the number of common variables gets higher the dependency increases.

Figure 2a gives the CDM of a typical benchmark example (p62_L_L_V02) when the standard method is used (the CDM is symmetric, so the lower part has been omitted). The TR resulting from the standard method has 23 clusters. It can be seen that dependencies between all clusters exist and many of them are quite high. The maximum is 62 common variables.

Figure 2b shows the resulting CDM for the RTL method. It is easy to see that:

- some clusters are not connected (shown by empty fields),
- the dependency is small on module borders (cluster 0-1, 9-10, 17-18) and
- the overall dependency is smaller (maximum at 37 common variables).

All the points mentioned above indicate a better *early quantification*, and thus a more efficient AndExist operation.

Based on these observations we develop our heuristic.

Group Heuristic Algorithm

The basic idea of the heuristic is to find groups of strongly connected latches and to merge these groups until a reasonable number of mostly independent groups remains..

Therefore, the heuristic proceeds in two phases: During the first phase a matrix LDM for latch dependencies is created. Entry (i, j) of the matrix LDM contains the number of OBDD variables that both latch i and latch j are depending on. Please

mention that the matrix contains numbers of variables and not the variables itself. Computing the dependencies based on the variables itself would be to hard to compute. The runtime of this phase of the heuristic is $O(\#latches^2 \cdot \#vars)$.

During the second phase the groups of latches are determined with the help of the latch dependencies matrix LDM. The idea is to put latches that have a high number of variables in common into one group. By decreasing the dependency threshold latches are added to existing groups or form new groups.

The problem using this very basic approach is that there always exists a certain dependency between all latches (e.g. clock signals). There are also latches that are only very loosely coupled to other latches. Thus, in this form the heuristic would result in a single large group before all latches have been grouped.

To avoid this effect we introduced an additional criterion for the separation of groups. The separation is realized by avoiding merges of groups, whose dependencies differ too much. Groups are indexed by the order in which they have been created. The difference of two group indexes gives a criterion for the difference in the amount of dependency. If the indexes of the group differ too much, a merge is forbidden. In our case it turned out the a difference of 3 is a good choice to obtain a reasonable number of separated groups. The runtime of this phase of the heuristic is $O(\#latches^3 \cdot \#vars)$.

For a sketch of the grouping heuristic see Figure 3.

After computing the groups of latches the partitioning is computed by applying the clustering strategy of the RTL heuristic outlined in Figure 1b.

Figure 2c shows the application of the group heuristic to our benchmark example (for experimental results see Table I). The following can be seen from the CDM:

- the overall dependency compared to the standard method is much smaller (maximum 42),
- many clusters are not connected,
- the group structure is comparable to the RTL method.

The CDM of the group method is not as structured as the CDM of the RTL. But, this result is not not surprising: The RTL method is a high-level method, while the group method simulates high-level effects with low-level information i.e. it works much more heuristically than the RTL method. Nevertheless, the CDM looks promising and the grouping has a strong influence on performance of the image computation as experiments show.

```
initialize group to 0;
for (dep = maxdep downto 1){
    for (i = 1 to #latches){
        for (j = 1 to #latches){
            if(depmatrix[i][j] == dep){
                igroup = group[i]; jgroup = group[j];
                if(!igroup && jgroup) group[i] = jgroup;
                if(igroup && !jgroup) group[j] = igroup;
                if(!igroup && !jgroup){
                    #groups++;
                    group[j] = group[i] = #groups;
                }
                if(igroup && jgroup &&
                    abs(igroup-jgroup)< 3){
                  rename groups to min(igroup,jgroup)
                }
            }
        }
    }
}
```

Fig. 3. Algorithm in pseudocode for group heuristic.

IV. Experiments

A. Implementation

We implemented our strategy in the VIS-package [5] (version 1.3) using the underlying CUDD-package [11] (version 2.3.0). VIS is a popular verification and synthesis package in academic research. It inherits state of the art techniques for OBDD manipulation, image and reachable states computation as well as formal verification techniques. Together with the vl2mv translator VIS provides a Verilog front-end.

B. Benchmarks

For our experiments we used Verilog designs from the Texas97 benchmark suite [1]. This publicly available benchmark suite contains real life designs from industry and academics including:

- MSI Cache Coherence Protocol
- PCI Local BUS
- PI BUS Protocol
- MESI Cache Coherence Protocol
- MPEG System Decoder
- DLX
- PowerPC 60x Bus Interface

The benchmark suite also contains properties given in CTL formulas for verification.

We chose those designs that represent RTL (i.e. including more than one module) rather than gate level descriptions. Considered were those designs that could be read in and whose transition relation could be build respecting our system limitations. Too small examples (CPU time $<$ 20s) were not considered.

C. Experimental Setup

We left all parameters of VIS and CUDD unchanged. The most important default values are:

- Partition cluster size = 5000
- Partition method for MDDs = inout

- OBDD variable reordering method = sifting
- First reordering threshold = 4004 nodes

The model checking was preceeded by a forced variable reordering. The CPU time was limited to 2 CPU hours and memory usage was limited to 200MB. All experiments were performed on Linux PentiumIII 500Mhz workstations.

D. Results of Model Checking Experiments

In our experiment series we performed model checking on the basis of the Texas97 benchmarks.

For results see Table I. Img.comp. is the sum of forward and backward image computation performed during the analysis. Part gives the number of partitions of the transition relation. The OBDD-size of the transition relation cluster and the peak number of live nodes is given by TRn resp. Peakn. The CPU time is measured in seconds and given as Time. The columns denoted with % describe the improvement in percent[1].

At the bottom of Table I you can find the sum of all numbers of partitions, BDD-sizes and CPU-times. Also, the *average of the relative improvement* is given as well as the *total improvement*.

The experiments show significant improvements in time and space: The overall CPU time decreased by 72% overall and 48% on average. The method outperforms the standard method in 46 of the 48 benchmarks. The decrease in computation time ranges up to 94%. The OBDD peak sizes could be lowered by 70% overall and 40% on average.

V. Conclusion

We presented a heuristic for optimizing the partitioning of the transition relation for reachable states computation and model checking of sequential systems. The heuristic adopts the paradigm of a recently introduced RTL-based heuristic for partitioning, but without using RTL information. Thus, resulting in a broader applicability of the heuristic.

The heuristic significantly decreases computation time and memory consumption during reachable states computation and model checking and thus, allows more efficient optimization and verification.

References

[1] A. Aziz et. al., *Texas-97 benchmarks*, http://www-cad.EECS.Berkeley.EDU/Respep/Research/Vis/texas-97.

[2] R. E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers, C-35, pp. 677-691, 1986.

[3] J. R. Burch, E. M. Clarke and D. E. Long, *Symbolic Model Checking with partitioned transition relations*, Proc. of Int. Conf. on VLSI, 1991.

[4] J. R. Burch, E. M. Clarke, D. L. Dill, L. J. Hwang and K. L. McMillan, *Symbolic model checking:* 10^{20} *states and beyond*, Proc. of Logic in Computer Science (LICS'90), pp. 428-439, 1990.

[5] R. K. Brayton, G. D. Hachtel, A. L. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. A. Edwards, S. P. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy and T. Villa, *VIS: A System for Verification and Synthesis*, Proc. of Computer Aided Verification (CAV'96), pp. 428-432, 1996.

[6] O. Coudert, C. Berthet and J. C. Madre, *Verification of Synchronous Machines using Symbolic Execution*, Proc. of Workshop on Automatic Verification Methods for Finite State Machines, LNCS 407, Springer, pp. 365-373, 1989.

[7] D. Geist and I. Beer, *Efficient Model Checking by Automated Ordering of Transition Relation Partitions*, Proc. of Computer Aided Verification CAV'94, pp. 294-310, 1994.

[8] R. D. M. Hunter and T. T. Johnson, *Introduction to VHDL*, Chapman & Hall, 1996.

[9] Ch. Meinel and C. Stangier, *Speeding Up Image Computation by using RTL Information*, Proc. of Formal Methods in CAD (FMCAD'00), LNCS 1954, pp. 443-454, 2000.

[10] R. K. Ranjan, A. Aziz, R. K. Brayton, C. Pixley and B. Plessier, *Efficient BDD Algorithms for Synthesizing and Verifying Finite State Machines*, Proc.of Int. Workshop on Logic Synthesis (IWLS'95), 1995.

[11] F. Somenzi, *CUDD: CU Decision Diagram Package*, ftp://vlsi.colorado.edu/pub/ .

[12] D. E. Thomas and P. Moorby, *The Verilog Hardware Description Language*, Kluwer, 1991.

[1] $0 <$ improvement < 100; $-100 <$ impairment < 0.

		Standard VIS				Group Method						
	Img.comp	Peakn	Parts	TRn	Time	Peakn	%	Parts	TRn	%	Time	%
PCIabnorm.PCI	304	176276	14	28613	253	131123	26	8	18920	34	143	44
PCInorm.PCI	206	81123	15	35124	56	69291	15	9	25027	29	47	16
TWO.contention	37	97623	7	11865	47	276428	-64	9	11251	5	219	-77
TWO.pixley_cpu	162	1357100	8	13118	1539	1010344	26	10	10137	23	1712	-9
multi_main.multim	45	38694	5	14700	33	33423	14	5	4837	67	24	28
p62_LS_LS_V01.ccp	64	166074	23	49952	200	119048	28	23	44808	10	123	38
p62_LS_LS_V01.p6liveness	99	452267	23	49952	827	129015	71	23	44808	10	135	84
p62_LS_LS_V02.ccp	53	146494	22	59487	107	115090	21	23	45749	23	69	36
p62_LS_LS_V02.p6liveness	96	167454	22	59487	181	115090	31	23	45749	23	75	59
p62_LS_L_V01.ccp	64	176540	23	49684	216	117820	33	23	43248	13	118	46
p62_LS_L_V01.p6liveness	99	1614200	23	49684	3833	197370	88	23	43248	13	229	94
p62_LS_L_V02.ccp	54	148560	23	62140	106	101891	31	23	46766	25	65	38
p62_LS_L_V02.p6liveness	89	183811	23	62140	193	101891	45	23	46766	25	68	65
p62_LS_S_V01.ccp	64	176540	23	49684	211	117820	33	23	43248	13	117	44
p62_LS_S_V01.p6liveness	99	1614200	23	49684	3564	197370	88	23	43248	13	223	94
p62_LS_S_V02.ccp	54	148560	23	62140	109	101891	31	23	46766	25	68	38
p62_LS_S_V02.p6liveness	89	183811	23	62140	199	101891	45	23	46766	25	68	66
p62_L_L_V01.ccp	52	164244	23	48961	194	120133	27	23	44419	9	70	64
p62_L_L_V01.p6liveness	87	477543	23	48961	935	124875	74	23	44419	9	113	88
p62_L_L_V02.ccp	53	144504	23	48971	180	116850	19	23	46006	6	72	60
p62_L_L_V02.p6liveness	96	242452	23	48971	402	121655	50	23	46006	6	118	71
p62_L_S_V01.ccp	75	168782	22	62479	121	109282	35	23	45404	27	72	41
p62_L_S_V01.p6liveness	118	192410	22	62479	231	115293	40	23	45404	27	80	65
p62_L_S_V02.ccp	55	140767	22	57365	108	96261	32	24	44445	22	63	41
p62_L_S_V02.p6liveness	96	140767	22	57365	112	96261	32	24	44445	22	64	42
p62_ND_LS_V01.ccp	83	396642	24	63506	831	266366	33	25	53233	16	556	33
p62_ND_LS_V01.p6liveness	93	4648753	24	63506	>2h	431664	91	25	53233	16	1012	86
p62_ND_LS_V02.ccp	103	191386	22	63321	356	149972	22	24	46085	27	253	29
p62_ND_LS_V02.p6liveness	192	1564426	22	63321	3922	1433962	8	24	46085	27	2149	45
p62_ND_L_V01.ccp	75	356794	25	65964	824	307993	14	24	51799	21	609	26
p62_ND_L_V02.ccp	133	5860454	23	60383	>2h	1514059	74	24	49366	18	3414	53
p62_ND_L_V02.p6liveness	168	5573568	23	60383	>2h	464599	92	24	49366	18	760	90
p62_ND_S_V02.ccp	84	150630	23	46744	187	126833	16	24	47162	0	122	35
p62_ND_S_V02.p6liveness	177	645918	23	46744	1221	297501	54	24	47162	0	366	70
p62_S_S_V01.ccp	43	147063	23	62209	102	98031	33	23	44030	29	63	38
p62_S_S_V01.p6liveness	80	153012	23	62209	107	98031	36	23	44030	29	64	40
p62_S_S_V02.ccp	37	129492	23	54800	96	94675	27	22	46649	15	58	39
p62_S_S_V02.p6liveness	74	129492	23	54800	96	94675	27	22	46649	15	59	39
p62_V_LS_V01.ccp	108	283494	24	58415	575	237586	16	24	48828	16	357	38
p62_V_LS_V01.p6liveness	114	4483034	24	58415	>2h	266327	94	24	48828	16	403	94
p62_V_LS_V02.ccp	90	165200	23	52073	234	116772	29	25	43552	16	136	42
p62_V_LS_V02.p6liveness	178	1059895	23	52073	2093	334884	68	25	43552	16	432	79
p62_V_S_V01.ccp	82	213245	23	61795	258	127609	40	24	46415	25	141	45
p62_V_S_V01.p6liveness	127	964988	23	61795	2421	176536	82	24	46415	25	231	90
p62_V_S_V02.ccp	84	163439	22	54807	200	121734	25	24	47839	13	140	30
p62_V_S_V02.p6liveness	177	351553	22	54807	515	150424	57	24	47839	13	230	55
two_processor.prop2	264	903917	4	12311	756	187331	79	3	5700	54	470	38
two_processor_bin.prop2_bin	140	252974	4	11610	154	197393	22	3	5842	50	135	12
Sum		37390165	981	2451121	58105	11232363	1909	996	1981540	977	16311	2320
Average of the relative improvement							40%			20%		48%
Total improvement						**70%**		**-1%**	**19%**		**72%**	

TABLE I COMPARISON OF ORIGINAL VIS PARTITIONING AND GROUP HEURISTIC