Int. Computer Symposium, Dec. 15-17, 2004, Taipei, Taiwan.

An Efficient Variable Partitioning and Scheduling Algorithm for DSP with Multiple Memory Modules

Yi-Hsuan Lee and Cheng Chen

Department of Computer Science and Information Engineering
1001 Ta Hsueh Road, Hsinchu, Taiwan, 30050, Republic of China
Tel: (8863) 5712121 EXT: 54734, Fax: (8863) 5724176
E-mail: {yslee, cchen}@csie.nctu.edu.tw

Abstract

Multiple on-chip memory modules are attractive to many DSP applications. This architectural feature supports higher memory bandwidth by executing multiple data memory accesses in parallel. However, the performance gain in this architecture strongly depends on the variable partitioning and scheduling method. In this paper, we propose an efficient rotation scheduling with parallelization (RSP), which is extended from our previous studies. RSP includes a simple mechanism to partition variables, and uses rotation scheduling and unimodular transformations to generate effective results. We also design an analytic model to analysis preliminary performances. Based on our analyses, RSP can obtain quite effective results compared with related methods.

1 Introduction

Most scientific and digital signal processing (DSP) applications, such as image processing and weather forecasting, are iterative and usually represented by uniform nested loops [1]. Digital signal processor is a special-purpose microprocessor, which is designed to achieve high performance on DSP applications. Unlike general-purpose CPU, the DSP is designed on Harvard architecture, and often includes independent function units which can operate in parallel [2].

The growing gap of speed between CPU and memory becomes one of the most critical problems for high-performance systems design. Thus, multiple on-chip memory modules are attractive to many DSP applications [3-4]. Since data are partitioned to separate memory banks and accessed simultaneously, this architecture offers higher memory bandwidth and performance potentially. However, its performance gain strongly depends on variable partitioning and scheduling techniques [4].

The variable partitioning and scheduling problem is proven to be NP-complete. *Rotation scheduling with variable repartitioning* (*RSVR*) is an effective heuristic [4], and we propose two algorithms before [5]. In this paper, we design *rotation scheduling with parallelization* (*RSP*) by integrating *unimodular transformations* technique. An analytic model and DSP applications are used to evaluate preliminary performances. From our analyses, RSP is quite effective compared with related methods.

The remainder of this paper is organized as follows. Section 2 describes the problem modeling and related work. Design issues and principles of RSP are introduced in Section 3. Section 4 contains the analytic model and preliminary analyses. Finally, we give some conclusions in Section 5.

2 Fundamental Background

2.1 Modeling the Given Program [4-5]

Multi-dimensional data flow graph (*MDFG*) defined below is widely used to represent uniform nested loop in previous researches. Nodes in the MDFG can be both ALU operations and memory operations. Figure 1(a)(b) shows an nested loop and its MDFG.

```
for i = 1 to m
    for j = 1 to n
        D[i, j] = B[i-1, j] × C[i-1, j-2] ;
        A[i, j] = D[i, j] × 0.5 ;
        B[i, j] = A[i, j] + 1 ;
        C[i, j] = A[i, j-1] + 2 ;
    end
end
```

(a)

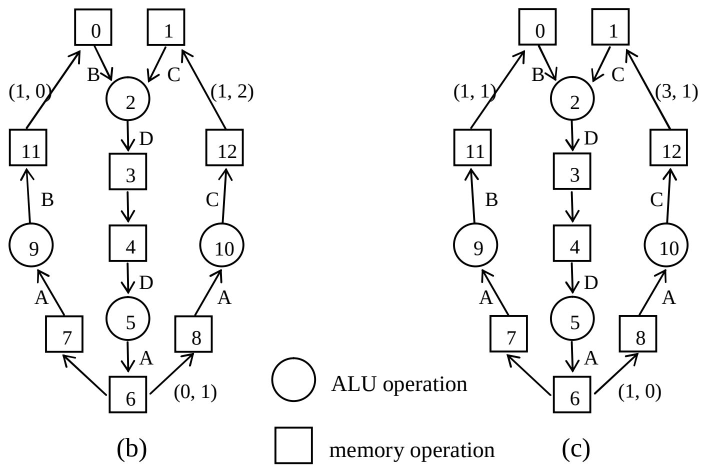


Figure 1. (a) Nested loop, (b) corresponding MDFG, (c) parallelized MDFG.

Definition A multi-dimensional data flow graph (MDFG) $G = (V, E, X, d, t)$ is a node-weighted and edge-weighted direct graph, where V is the set of computation nodes, E is the edge set of precedence relations, $X(e)$ represents the variable accessed by an edge e, $d(e)$ is the delays between nodes, and $t(v)$ is the computation time of node v.

A MDFG is *realizable* if there exists a *schedule vector s* such that $s \cdot d \geq 0$, where d are loop-carried dependencies [6]. An *iteration* is equivalent to execute each node in V exactly once. The period during which all nodes in an iteration are executed without resource constraints is called a *cycle period*. Cycle period is the maximum computational time among paths that have no delay, which will dominate the entire execution time of a nested loop.

2.2 Retiming Technique [7]

Retiming is a technique that redistributes nodes in consecutive iterations to enhance the performance. The *retiming vector* $r(u)$, a function from V ot Z^n, represents the offset between the original iteration and that after retiming. A MDFG $G_r = (V, E, X, d_r, t)$ is created after applying r such that each iteration still has one execution of each node. Delay vectors will be changed accordingly to preserve dependencies.

A *prologue* is the instruction set that must be executed to provide data for the iterative process. An *epilogue* is the complementary set that will be executed to complete the process. Usually, the time required for prologue and epilogue are negligible.

2.3 Unimodular Transformations Technique [8]

Loop transformation is one of basic techniques for parallel compiler design. It changes the execution sequence of iterations to achieve higher degree of parallelism. *Unimodular transformations* technique unifies loop *permutation*, *skewing*, and *reversal*, and models them as elementary matrix transformations. All combinations of these loop transformations can simply be represented as products of the elementary transformation matrices.

2.4 Related Work

Since retiming is useful for generating compact schedules, many scheduling algorithms are designed based on it. In order to solve the variable partitioning and scheduling problem, *Rotation scheduling with variable repartitioning* (*RSVR*) [4], modified from *rotation scheduling* by considering multiple memory modules while constructing a schedule, is proposed. RSVR uses *variable independence graph* (*VIG*) to partition variables initially. When the schedule length cannot be improved in a rotation phase, it will try to repartition variables to shorten the schedule length.

We have proposed *rotation scheduling with unfolding* (*RSF*) and *rotation scheduling with tiling* (*RST*) for the same problem before [5]. RSF and RST use simpler variable partitioning mechanisms, and

```
Input: MDFG $G = (V, E, X, d, t)$
Output: MDFG $G' = (V, E, X, d', t)$
1. $T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$; $G' = G$;
2. while ($\exists$ $(a, 0)$ and $(b, 0)$ in $d'$, for $a, b > 0$)
       $T = T \times \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$; $\forall\, d'(e) \in d, d'(e) = T \times d'(e)$;
3. if ($\exists$ $(a, 0)$ in $d'$, for $a > 0$)
   (a) if ($\exists$ $(b, -c)$ in $d'$, for $b, c > 0$)
           $T = T \times \begin{bmatrix} 1 & 0 \\ \lceil (c+1)/b \rceil & 1 \end{bmatrix}$;
   (b) $T = T \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$; $\forall\, d'(e) \in d, d'(e) = T \times d'(e)$;
4. Return $G' = (V, E, X, d', t)$
```

Figure 2. Loop parallelization algorithm.

```
Input: MDFG $G = (V, E, X, d, t)$, $N$
Output: schedule $S$
1. Allocate variables to $N$ memory modules
2. $G_p$ = parallelize $G$ that the inner loop is parallelizable
3. $G_N$ = unfold $G_P$ with factor $N$
4. Select the schedule vector $s = (1, 0)$
5. $S$ = schedule $G_N$ using list scheduling
6. $S'$ = compact $S$ using rotation scheduling
7. Return $S'$
```

Figure 3. The scheduling steps of RSP.

don't need to repartition variables during rotation phases. From our analyses, RSF and RST are as effective as RSVR, sometimes even outperform it. Besides, they are much efficient than RSVR, because they avoid time consuming steps in RSVR such as VIG construction and variable repartition.

3 Rotation Scheduling with Parallelization

3.1 Motivation

Although RSF and RST are quite effective, they still can be further improved. In order to fit their variable partitioning mechanisms, we apply unfolding and tiling techniques in RSF and RST. That is, their enlarged iterations are composed of original iterations. However, if critical paths of those original iterations are *cascaded* after unfolding or tiling, RSF and RST will obtain very poor results. Therefore, we propose *rotation scheduling with parallelization* (*RSP*), which applies unimodular transformations technique to parallelize the inner loop before unfolding. After loop parallelizing, original iterations within an unfolded iteration are independent. Since this feature leads RSP to avoid the drawback of RSF and RST, we believe it can achieve reasonable scheduling results.

In Section 3.2 and 3.3, we will describe variable partitioning mechanism and loop parallelization algorithm in detail. Scheduling steps of RSP will be listed in Section 3.4. Since nested loop used in DSP applications are usually with depth two, we use the two-dimensional MDFG to design and analysis RSP. However, RSP and the analytic model can be easily extended to cover MDFG with higher dimensions.

3.2 Variable Partitioning Mechanism

Notice that a variable in MDFG indicates an array not a single variable. Similar as RSF and RST, we partition array components based on loop indices in RSP. For DSP with 2~4 memory modules, we design particular mechanisms as follows.

2 memory modules ($k \in \mathbf{N}, k \geq 1$)

Module 1: $[m, 2k - 1]$

Module 2: $[m, 2k]$

3 memory modules ($k \in \mathbf{N}, k \geq 0$)

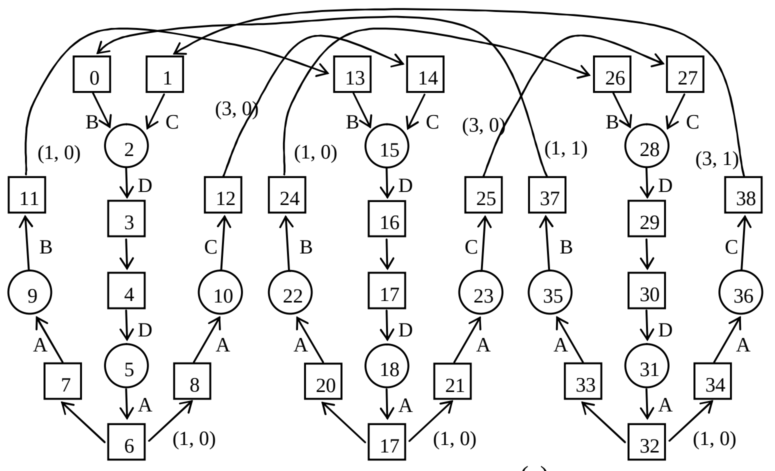
Module 1: $[m, 3k + (m \textbf{ mod } 3)]$

Module 2: $[m, 3k + (m \textbf{ mod } 3) + 1]$

Module 3: $[m, 3k + (m \textbf{ mod } 3) - 1]$

4 memory modules ($k \in \mathbf{N}, k \geq 1$)

Module 1:	$[m, 2k - 1]$	if $(m \textbf{ mod } 4) = 1$
	$[m, 2k]$	if $(m \textbf{ mod } 4) = 3$
Module 2:	$[m, 2k]$	if $(m \textbf{ mod } 4) = 1$
	$[m, 2k - 1]$	if $(m \textbf{ mod } 4) = 3$
Module 3:	$[m, 2k - 1]$	if $(m \textbf{ mod } 4) = 0$
	$[m, 2k]$	if $(m \textbf{ mod } 4) = 2$
Module 4:	$[m, 2k]$	if $(m \textbf{ mod } 4) = 0$
	$[m, 2k - 1]$	if $(m \textbf{ mod } 4) = 2$



	Mul	Add	M1	M2	M3
0	28	10	3	16	1
1		36	4	17	29
2	5		12	25	30
3	18		6	0	38
4	31		7	19	13
5	2	9	14	20	32
6	15	22	11	27	33
7		35	21	24	8
8		23	26	34	37

(b)

Figure 4. (a) Parallelized MDFG, (b) schedule result.

3.3 Loop Parallelization Algorithm

Unimodular transformations technique unifies three loop transformations, but it doesn't explain how to use them. For a nested loop with depth two, we have designed a simple algorithm to parallelize the inner loop as listed in Figure 2 [9]. In RSP we will directly apply this algorithm, and Figure 1(c) contains the parallelized MDFG G_P of Figure 1(a).

3.4 RSP Algorithm

After introducing above two mechanisms, Figure 3 contains scheduling steps of RSP. With the similar reason of RSF and RST, G_P must be unfolded with factor N before applying rotation scheduling, where N is the number of memory modules. Moreover, due to the parallelized inner loop, we can always select (1, 0) as the schedule vector. The unfolded graph G_N and final schedule are shown in Figure 4.

4 Performance Studies

4.1 Preliminary Performance Analysis

In this subsection, we design an analytic model to analysis RSP. At first we define variables used in our analytic model. Given a nested loop with depth two, and its loop bounds of outer and inner loops are m and n respectively. w is the skew factor used to parallelize the inner loop, and two kinds of changed iteration space will be produced after parallelization [9]. A retimed nested loop contains prologue, repetitive patterns, and epilogue phases, where we use variables *prologue*, *length* and *epilogue* to represent their execution lengths. *list* is the execution length of a repetitive pattern produced by *list scheduling*, which is usually greater than *length*. *Retiming depth*, d, is the number of iterations been moved into prologue and epilogue. Besides, if an iteration of RSP contains less than N original iterations, its execution length is defined as $half\,(k, N)$.

Assume temporary variables (A, B, C, D, E, F, G, H) equals to ($mw - w - n$, A **mod** w, n **mod** w, $m - 1$ **mod** N, m **mod** N, $(\lfloor n/w \rfloor - 1)$ **mod** N, $\lfloor n/w \rfloor$ **mod** N, $\lceil n/w \rceil$ **mod** N).

F_1: $list \times dwN\,(d - 1)$

F_2: $(prologue + epilogue) \times (wm + w + n - 2wNd)$

F_3: $(prologue + epilogue) \times (2w\lfloor n/w \rfloor - 2wNd + 2w + B\lceil A/w \rceil + 2C + (w - B)\lfloor A/w \rfloor)$

F_4: $length \times w(\lfloor (m-1)/N \rfloor - d)(m - Nd - N + 1 + D)$

F_5: $length \times (w + n - mw)(\lfloor m/N \rfloor - d)$

F_6: $length \times w(\lfloor (n - w)/wN \rfloor - d)(\lfloor n/w \rfloor - Nd - N + 1 + F)$

F_7: $length \times (2w + B\lceil A/w \rceil)(\lfloor n/wN \rfloor - d)$

F_8: $length \times (2C + (w - B)\lfloor A/w \rfloor)(\lfloor \lceil n/w \rceil / N \rfloor - d)$

F_9: $2w\lfloor (m-1)/N \rfloor \times \sum_{i=1}^{N-1} half\,(i, N)$

Table 1. Experimental results (2 multipliers and 2 adders) (*length*, *d*).

	2 memory modules				3 memory modules				4 memory modules			
	RSVR	RSF	RST	RSP	RSVR	RSF	RST	RSP	RSVR	RSF	RST	RSP
Wave Digital Filter	5,2	9, 1	9, 3	9, 1	5, 3	9, 1	10, 5	9, 1	3, 6	9, 2	10, 8	9, 1
Forward-substitution	6, 3	11, 3	11, 4	11, 1	4, 8	11, 5	11, 8	11, 2	4, 8	11, 9	12, 12	11, 2
IIR 2D	24, 1	40, 1	40, 1	40, 1	14, 1	40, 1	40, 1	40, 1	11, 1	40, 2	40, 3	40, 1
Filter	6, 2	10, 5	10, 3	10, 1	5, 3	11, 9	10, 6	10, 1	4, 3	11, 15	10, 7	10, 1
Discrete Fourier Transform	14, 1	28, 2	28, 0	28, 0	10, 1	28, 4	28, 1	28, 1	8, 3	29, 6	28, 1	28, 1
THCS	4, 2	8, 2	8, 0	8, 0	4, 2	8, 3	8, 1	8, 1	2, 4	8, 5	8, 1	8, 1

F_{10}: $2w \times \sum_{i=1}^{D} half(i, N)$

F_{11}: $(w + n - mw) \times half(E, N)$

F_{12}: $2w\lfloor (n - w)/wN \rfloor \times \sum_{i=1}^{N-1} half(i, N)$

F_{13}: $2w \times \sum_{i=1}^{F} half(i, N)$

F_{14}: $(2w + B\lceil A/w \rceil) \times half(G, N)$

F_{15}: $(2C + (w - B)\lceil A/w \rceil) \times half(H, N)$

The execution time of RSP =

$$F_1 + F_2 + F_4 + F_5 + F_9 + F_{10} + F_{11} \quad \textbf{if } wm + 1 \le w + n$$

$$F_1 + F_3 + F_6 + F_7 + F_8 + F_{12} + F_{13} + F_{14} + F_{15} \quad \textbf{if } wm + 1 > w + n$$

4.2 Experimental Results

Finally, we select some DSP applications to compare RSVR, RSF, RST, and RSP. Suppose both ALU and memory operations take one time unit to execute, Table 1 list our scheduling results.

Notice that in our three algorithms an iteration contains *N* original iterations, so the actual execution time of an iteration equals to the value in Table divided by *N*. From these results, *lengths* obtained by all algorithms are similar, but RSP can obviously get smaller *d*. The reason is that an iteration in RSP is composed of *N* independent original iterations, and its memory operations will be evenly allocated. Hence, the schedule generated by list scheduling will be already quite compact, which can decrease times applying retiming technique and retiming depth.

Figure 5 shows the execution time calculated by above formulas. Except RSVR and RSP, for each application we only sketch the results of RSF or RST depending on which result is better. In this figure, the execution time of RSP is similar to others, sometimes even outperforms them. Thus, like RSVR, RSF, and RST, RSP is also an effective and efficient algorithm.

5 Concluding Remarks

In this paper, we have proposed RSP to schedule nested loops for DSP systems with multiple memory modules. It contains a simple variable partitioning mechanism, and applies unimodular transformation, unfolding, and rotation scheduling techniques to schedule both ALU and memory operations. We also use an analytic module and DSP applications to evaluate RSP. Based on evaluating results, RSP is actually an efficient and effective algorithm compared with RSVR, RSF, and RST.

Reference

[1] V. K. Madisetti, **VLSI Digital Signal Processors: An Introduction to Rapid Prototyping and Design Synthesis**, Butterworth-Heinemnn, 1995.

[2] J. Eyre and J. Bier, “The Evolution of DSP Processors”, *IEEE Signal Processing Magazine*, Vol. 17, Issue 2, pp. 43-51, March 2000.

[3] R. Leupers and D. Kotte, “Variable Partitioning for Dual Memory Bank DSPs”, *Proc. of International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 1121-1124, May, 2001.

[4] Q. Zhuge, B. Xiao, and E. H. -M. Sha, “Variable Partitioning and Scheduling of Multiple Memory

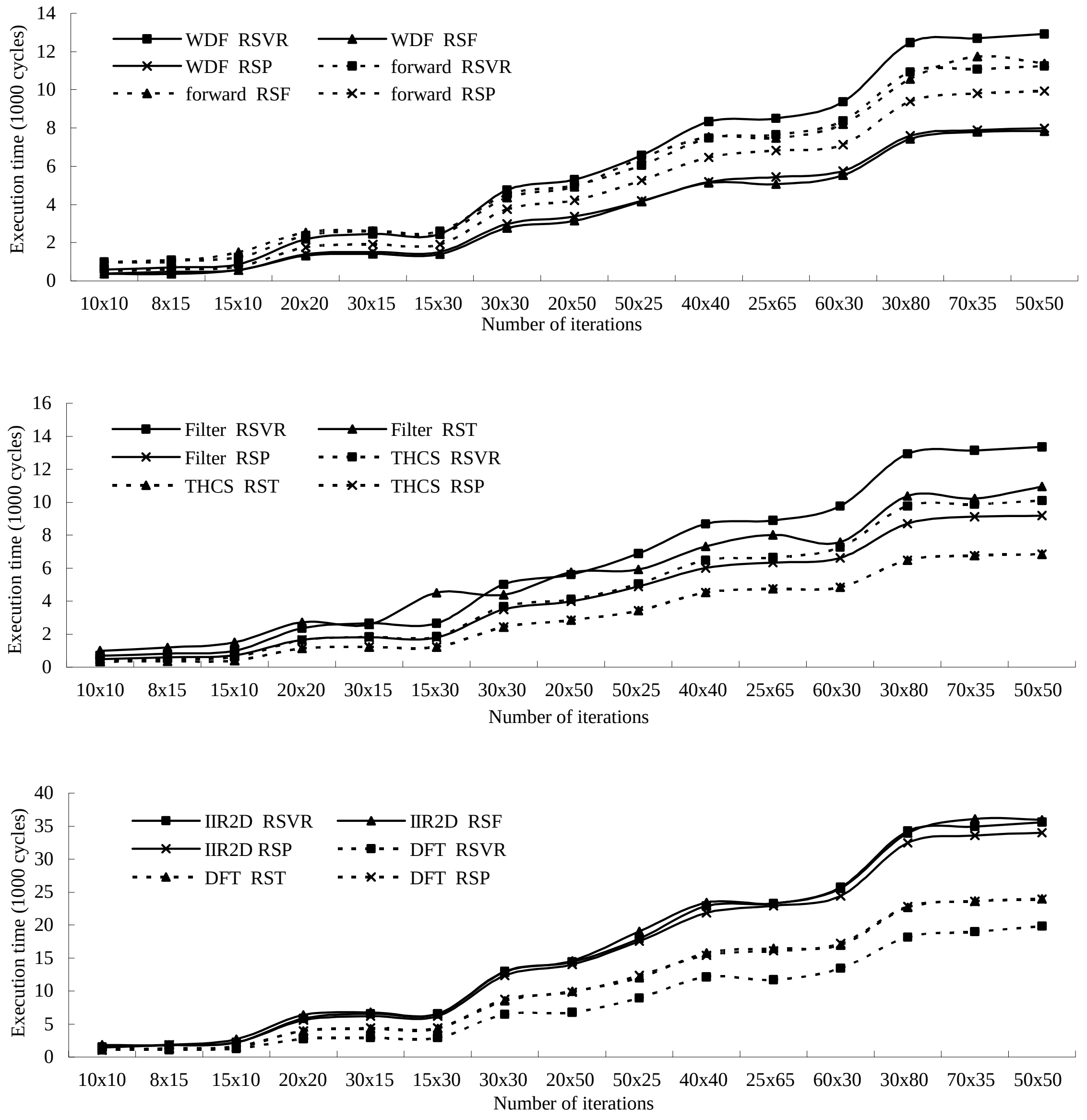


Figure 5. Experimental results (2 multipliers, 2 adders, and 3 memory modules).

Architectures for DSP", *Proc. of International Parallel and Distributed Processing Symposium*, pp. 130-136, April 2002.

[5] Y. -H. Lee and C. Chen, "Efficient Variable Partitioning and Scheduling Methods of Multiple Memory Modules for DSP", *Proc. of 10th Workshop on Compiler Techniques for High-performance Computing*, pp. 80-89, March 2004.

[6] L. Lamport, "The Parallel Execution of DO Loops", *Comm. ACM SIGPLAN*, Vol. 17, No. 2, pp. 82-93, Feb. 1974.

[7] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, Vol. 6, No. 1, pp. 5-35, June 1991.

[8] M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 452-471, Oct. 1991.

[9] Y. -H. Lee and C. Chen, "A Two-level Scheduling Method: An Effective Parallelizing Technique for Uniform Nested Loops on a DSP Multiprocessor", accept and to appear to *Journal of Systems and Software.*