Efficient Variable Allocation to dual Memory Banks of DSPs

Viera Sipkova

Christian Doppler Laboratory
Compilation Techniques for Embedded Processors

Institute for Computer Languages
Vienna University of Technology

Outline

- Motivation
- Introduction
- Partitioning Scheme
- Experimental Results
- Conclusion

Motivation

To improve the overall performance -

- DSPs are equipped with *multiple data memory banks*.
- The compiler must *partition program variables* into memory banks.

Dot product (C code)

```
short a[100], b[100];
short dot_product (void )
{ short dot = 0;
   for (int i = 0; i < 100; i++) {
        dot = dot + a[i] * b[i];
   }
   return dot;
}
```


Dot product (assembly code)

```
    movl  a,R0      ||  movl  b,R1
    clr   A0        ||  bkrep 99,LBL
▶   ld    (R0)+,D2  ||  ld    (R1)+,D3
    nop
    mac   D2,D3,A0
    nop
LBL:  ret
```


Variable Partitioning

To solve this problem several approaches at different stages of the compilation flow are possible :

- *High level intermediate representation*
- *Low level intermediate representation*

Previous Work

- *Powell, Lee, Newman, 1992* (assembly code; alternating fashion)
- *Saghir, Chow, Lee, 1996* (post-pass approach; greedy algorithm)
- *Sudarsanam, Malik, 2000* (with register allocation; simulated annealing)
- *Leupers, Kotte, 2001* (before register allocation; ILP)
- *Zhuang, Pande, Greenland, 2002* (post-pass approach; maximum spanning tree)
- *Cho, Paek, Whalley, 2002* (with register allocation; maximum spanning tree and graph coloring)
- *Zhuge, Xiao, Sha, 2002* (model based on DFG; scheduling with variable repartition)

Basic Features

- The partitioner is constructed as a separate optimization phase operating on the *high-level intermediate representation*.
- The *algorithm* of the partitioner is *global*, applied across the basic blocks to all functions of the program.
- *Global* variables and *static local* variables are handled.
- Array variables are treated as monolithic entities.

Partitioning Scheme

The partitioner is based on the concept of the *Interference Graph*.

- Constructing the interference graph.
- Partitioning the interference graph.
- Annotating variables with the partitioning information.

Interference Graph

The edge-weighted undirected graph $G = (V, E)$, where each

ertex $v \in V$ – represents a memory access
dge $e = (v, w) \in E$ – indicates the possible parallel access to v and w (different memory banks)

Interference Edge

Let V be the set of interference vertices of G.

There exists an edge $e = (v, w)$ between $v, w \in V$
if and only if
the contexts enclosing v and w respectively, are
not *control-dependent* && not *data-dependent*.

Edge Weight

To each edge $e = (v, w) \in E$ a weight $A(e)$ is associated :

$$A(e) = EF \times DW(e)$$

EF – *execution frequency*

$DW(e)$ – *distance weight* =

$$\begin{cases} 2 & \text{if } v \text{ and } w \text{ are in the same statement} \\ 1 & \text{if } v \text{ and } w \text{ are in the different statements} \end{cases}$$

Graph Merging

Let $\{v_1, \ldots, v_k\} \subseteq V$ represent different accesses to the same variable. Then :

$\{v_1, \ldots, v_k\}$ *is merged to* v
$(v_i, w) \in E$ *is redirected to* $(v, w),\ i = 1, \ldots, k$
$(w, v_i) \in E$ *is redirected to* $(w, v),\ i = 1, \ldots, k$

For $e = (v, w)$:
$A(e)$ *is redefined to* $Max(A(e_i)) \times k$
$e_i = (v_i, w),\ i = 1, \ldots, k$

Graph Partitioning

Partition the set of vertices of $G = (V, E)$ into two disjoint sets :

$$S \subseteq V \quad \text{and} \quad \bar{S} = V - S$$

that the sum of the weights of all edges $(v, w) \in E$ connecting $v \in S$ and $w \in \bar{S}$, is maximal.

Memory Bank Assigning

Let $(S, \bar{S})$ be the result of the partitioning.
Then each vertex :

$v \in S$ is assigned to memory bank X
$v \in \bar{S}$ is assigned to memory bank Y

Max-Cut

Partitioning the interference graph is solved as the *combinatorial optimization problem Max-Cut* :

$$\textit{maximize} \sum_{e \in Cut(S,\bar{S})} A(e)$$

where $Cut(S, \bar{S})$ is the set of edges with one endpoint in S and the other endpoint in $\bar{S}$.

Max-Cut (NP-Complete Problem)

Sahni and Gonzales, 1976 – the first feasible solution.
Performance guarantee : $0.5 \times optimal\_value$.

Goemans and Williamson, 1994 – the best algorithm.
Performance guarantee : $0.878 \times optimal\_value$.

Burer, Monteiro, and Zhang, 2001 – specialized version of the Goemans-Williamson technique.
Performance guarantee : $0.878 \times optimal\_value$.

Max-Cut (Algebraic Formulation)

Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of vertices of G, and a_{ij} denotes the weight of the edge $(v_i, v_j) \in E$.
When introducing *cut vectors* $x \in \{-1, 1\}^n$ with

$x_i = \ \ 1$ for $v_i \in S$
$x_i = -1$ for $v_i \in \bar{S}$

then the *algebraic formulation* for *Max-Cut* :

$$\textit{maximize} \quad \frac{1}{2} \sum_{1 \le i < j \le n} a_{ij}(1 - x_i x_j)$$
$$\textit{subject to} \quad x_i \in \{-1, 1\},\ i = 1, \ldots, n$$

Max-Cut (Semidefinite Program)

$$\textit{maximize} \quad \frac{1}{2} \sum_{1 \le i < j \le n} a_{ij}(1 - y_{ij})$$

$$\textit{subject to} \quad Y \succeq 0, \;\; y_{ii} = 1, \; i = 1, \ldots, n$$

$$\Downarrow$$

$$\textit{maximize} \quad C \bullet Y$$

$$\textit{subject to} \quad \textit{diag}(Y) = e, \;\; Y \succeq 0$$

Max-Cut
(Goemans' Randomized Solution)

Solution $(y_1, \ldots, y_n)$ represents n points on the surface of the unit sphere corresponding to the set of vertices V.

Construct the hyperplane : $\mathcal{H}(r) = \{y \in \mathcal{R}^n : r^T y = 0\}$ using the random vector r uniformly distributed on the unit sphere.

Partition the vertex set :

$$S = \{v_i \in V \;:\; y_i^T r \ge 0\}$$
$$\bar{S} = \{v_i \in V \;:\; y_i^T r < 0\}$$

Implementation of Partitioning

- Exact method
- Alternate method
- Approximating iterative method (greedy)
- Semidefinite programming relaxation
- Semidefinte rank-2 relaxation

Experimental Results

The partitioning technique was evaluated on the simulator of the xDSPcore. The metrics which the performance is measured – the number of *total cycles*, and the number of *memory conflicts*. We did experiments with :

- DSPstone benchmark suite
- FFT Filters
- GSM Audio Codec

DSPstone Kernels

Kernel	X-Allocating				Partitioning			
	Cycl.	X	Y	Confl.	Cycl.	X	Y	Confl.
dot_product	625	200	0	0	525	100	100	0
convolution	625	200	0	0	525	100	100	0
matrix_mult_1	5368	2100	0	1000	5368	1000	1100	0
matrix_mult_2	5014	2010	0	900	4993	1100	910	0
mat1x3	85	24	0	0	76	9	15	0
lms	219	95	0	0	188	48	47	0
fir2dim	963	304	0	144	963	144	160	0
biquad_n_sects	71	38	0	12	66	21	17	0

DSPstone Kernels

Total Number of Cycles		
Kernel	X-Allocating	Partitioning
dot_product	625	525 (84.0%)
convolution	625	525 (84.0%)
matrix_mult_1	6368	5368 (84.3%)
matrix_mult_2	5914	4993 (84.4%)
mat1x3	85	76 (89.4%)
lms	219	188 (85.8%)
fir2dim	1107	963 (87.0%)
biquad_n_sects	83	66 (79.5%)

FFT Filter (256-point)

e of the Interference Graph: 13							
X-Allocating				**Alternate Allocating**			
cl.	X	Y	Confl.	Cycl.	X	Y	Confl.
341	91046	0	11053	146322	48479	39920	8823 (80%)
Partitioning							
execution frequency (runtime)				without execution frequency			
977	44909	42518	8785 (79%)	152171	29957	61936	7427 (67%)

Total Number of Cycles		
Allocating	**Alternate Allocating**	**Partitioning**
173394	155145 (89.5%)	154762 (89.2%)

GSM Audio Codec

Size of the Interference Graph: 45							
X-Allocating				**Alternate Allocating**			
Cycl.	X	Y	Confl.	Cycl.	X	Y	Confl.
1083592	150484	0	50693	1079696	83756	65648	272 (0.5%)
Partitioning							
execution frequency (runtime)				without execution frequency			
1071731	81818	68642	290 (0.6%)	1073385	90273	60187	389 (0.8%)

Total Number of Cycles		
X-Allocating	**Alternate Allocating**	**Partitioning**
1134285	1079968 (95.2%)	1072021 (94.5%)

Executed Parrallelism

		X-Allocating	**Partitioning**
FFT	total number of cycles	173394	154762 (89.2%)
	total number of RW	91046	87427
	number of RW pairs	11053 (24.3%)	22731 (52.0%)
	number of parallel RW pairs	0	14320 (32.8%)
Codec	total number of cycles	1134285	1072021 (94.5%)
	total number of RW	150484	150460
	number of RW pairs	50693 (67.1%)	58665 (78.0%)
	number of parallel RW pairs	0	58375 (77.6%)

Conclusion

The algorithm :

- attempts to maximize the benefit of dual memory banks.
- is based on the partitioning the graph whose nodes represent variables and edges denote potential parallel accesses.
- operates on the *high-level intermediate representation*; its frame is *global*, not limited to basic blocks.
- finds a quite satisfying memory assignment – *improvement of memory cycles :* $20\% - 50\%$ *improvement of total cycles :* $5\% - 20\%$

Future Plans

- To investigate the impact of the partitioning on the scheduling.
- To explore the memory partitioning for DSP architectures with interleaved memory banks.