Application Report
SPRA517

Loop Partitioning on the TMS320C6x

Richard Scales *Digital Signal Processing Solutions*

Abstract

This document descibes the process used to develop loop partitioning for the Texas Instruments (TI™) TMS320C6x digital signal processor (DSP). The ‘C6x code generation tools go through five basic phases when scheduling a loop. At each of these stages there is a set of heuristics which try to make intelligent decisions where multiple options exist. This document focuses on the third stage, partitioning, which can have great impact on the performance achieved with the tools. The document contains a discussion of the problem and of a suggested solution. Also included are listings of several examples and an illustration of the state diagram of the example.

Design Problem

The ‘C6x code generation tools go through five basic phases when scheduling a loop:

1) Front-end C optimizations (C compiler only).
2) Instruction selection (C compiler only).
3) Partitioning (C compiler and assembly optimizer).
4) Instruction scheduling (C compiler and assembly optimizer).
5) Register allocation (C compiler and assembly optimizer).

The first three phases can have a direct impact on the fourth and most important stage, instruction scheduling. At each of the first three stages, a set of heuristics tries to make intelligent decisions where multiple options exist. This document focuses on the third stage, partitioning, as this can have great impact on the performance achieved with the tools.

Sometimes non-optimal partitioning can be the limiting factor in attaining the highest possible performance. Once the instructions are chosen in phase 2, the compiler and/or assembly optimizer must decide which instructions to execute on the A side and which to execute on the B side in phase 3. This can have a direct affect on res MII because only one cross path from A to B and one cross path from B to A are available on any given cycle . If the partitioning is poor, either the number of cross paths or the number of functional units on a particular side can become a limiting factor in the partitioned resource bound shown in Example 1.

Solution

The following code development flow is recommended to achieve the highest performance on loops:

1) Compile native C code
2) Add const declarations and loop count information.
3) Optimize C code using intrinsics and other methods.
4) Write linear assembly.
5) Add partitioning information to the linear assembly.

The fifth stage, partitioning, is necessary when optimal partitioning is not achieved with the compiler or assembly optimizer.

Example 1 shows example feedback obtained from the compiler and assembly optimizer when using the -mw option. This information is valuable for pointing out potential problems with partitioning. Notice that the unpartitioned resource bound on the loop iteration interval is 3 but after partitioning it is 4. We can see below that this is due to 4 X cross paths on the A side and that there are 10 non-M unit instructions on the A side. Each of these forces the minimum iteration interval to be at least 4.

Example 1. Feedback Example

```
;*----------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop label : LOOP
;*      Loop Carried Dependency Bound : 3
;*      Unpartitioned Resource Bound  : 3
;*      Partitioned Resource Bound(*) : 4
;*      Resource Partition:
;*                                 A-side   B-side
;*      .L units                      0        0
;*      .S units                      2        2
;*      .D units                      2        2
;*      .M units                      2        2
;*      .X cross paths                4*       3
;*      .T address paths              2        2
;*      Long read  paths              1        0
;*      Long write paths              0        0
;*      Logical  ops (.LS)            4        1  (.L or .S)
;*      Addition ops (.LSD)           2        1  (.L or .S or .D)
;*      Bound(.L .S .LS)              3        2
;*      Bound(.L .S .D .LS .LSD)      4*       2
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 4  Schedule found with 4 iterations in parallel
;*      Done
;*----------------------------------------------------------------*
```

By passing partitioning information to the assembly optimizer, it is possible to improve the loop minimum iteration interval to 3 even after partitioning. Example 2 shows the linear assembly for the feedback in Example 1. Notice the functional units specified in boldface. These pass enough information to the tools to improve the partitioning between the A and B sides of the loop.

Example 2. Linear Assembly for IIR Filter

```
_iir  .cproc    cptr0,sptr0
      .reg cptr1, s01, s10, s23, c10, c32, s10_s, s10_t
      .reg p0, p1, p2, p3, s23_s, s1, t, x, mask, sptr1
      .reg s10p, ctr
      MV      cptr0,cptr1
      MV      sptr0,sptr1
      MVK        50,ctr      ; setup loop counter
LOOP:   .trip 50
      LDW .D1T1  *cptr0,c32  ; CoefAddr[3] & CoefAddr[2]
      LDW .D2T2  *cptr1,c10  ; CoefAddr[1] & CoefAddr[0]
      LDW .D1T2  *sptr0,s10  ; StateAddr[1] & StateAddr[0]
      MV         s10,s10p    ; save StateAddr[1] & StateAddr[0]
      MPY .M1    c32,s10,p2  ; CoefAddr[2] * StateAddr[0]
      MPYH       c32,s10,p3  ; CoefAddr[3] * StateAddr[1]
      ADD        p2,p3,s23   ; CA[2] * SA[0] + CA[3] * SA[1]
      SHR        s23,15,s23_s; (CA[2]*SA[0] + CA[3]* SA[1])>>15
      ADD .2     s23_s,x,t   ; t=x+((CA[2]*SA[0]+CA[3]*SA[1])>>15)
      AND        t,mask,t    ; clear upper 16 bits
      MPY        c10,s10,p0  ; CoefAddr[0] * StateAddr[0]
      MPYH       c10,s10,p1  ; CoefAddr[1] * StateAddr[1]
      ADD        p0,p1,s10_t    ; CA[0] * SA[0] + CA[1] * SA[1]
      SHR        s10_t,15,s10_s ; (CA[0]*SA[0] + CA[1]*SA[1])>>15
      ADD        s10_s,t,x ; x = t+((CA[0]*SA[0]+CA[1]*SA[1])>>15)
      SHL        s10p,16,s1     ; StateAddr[1] = StateAddr[0]
      OR         t,s1,s01       ; StateAddr[0] = t
      STW .D1    s01,*sptr1     ; store StateAddr[1]& StateAddr[0]
[ctr] ADD        -1,ctr,ctr     ; dec outer lp cntr
[ctr] B          LOOP           ; Branch outer loop
            .endproc
```

Example 3 shows the improved result. Now the minimum iteration interval is 3 even after partitioning and a schedule with ii=3 is found.

Example 3. Feedback Example After Partitioning

```
;*--------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop label : LOOP
;*      Loop Carried Dependency Bound : 3
;*      Unpartitioned Resource Bound  : 3
;*      Partitioned Resource Bound(*) : 3
;*      Resource Partition:
;*                                  A-side   B-side
;*      .L units                       0        0
;*      .S units                       2        2
;*      .D units                       3*       1
;*      .M units                       2        2
;*      .X cross paths                 2        1
```

```
;*      .T address paths             1        3*
;*      Long read  paths             0        1
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        3  (.L or .S)
;*      Addition ops (.LSD)          2        3  (.L or .S or .D)
;*      Bound(.L .S .LS)             1        3*
;*      Bound(.L .S .D .LS .LSD)     3*       3*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 3  Schedule found with 5 iterations in parallel
;*      Done
;*----------------------------------------------------------------*
```

The goal in defining functional units and/or sides, is to split the loop into two halves (A and B) with minimal cross paths and minimal effect on the scheduling. When partitioning a loop in linear assembly, try the following:

1) Minimize the number of cross paths. This usually involves looking at the dependency graph of the loop and splitting it such that there are the fewest number of paths crossing to the opposite side. Figure 1 shows a split where only one cross path is required. Keep in mind that dependencies due to conditional registers do not require a cross path (i.e., an instruction on the A side which is conditional on a B register does not use the cross path).

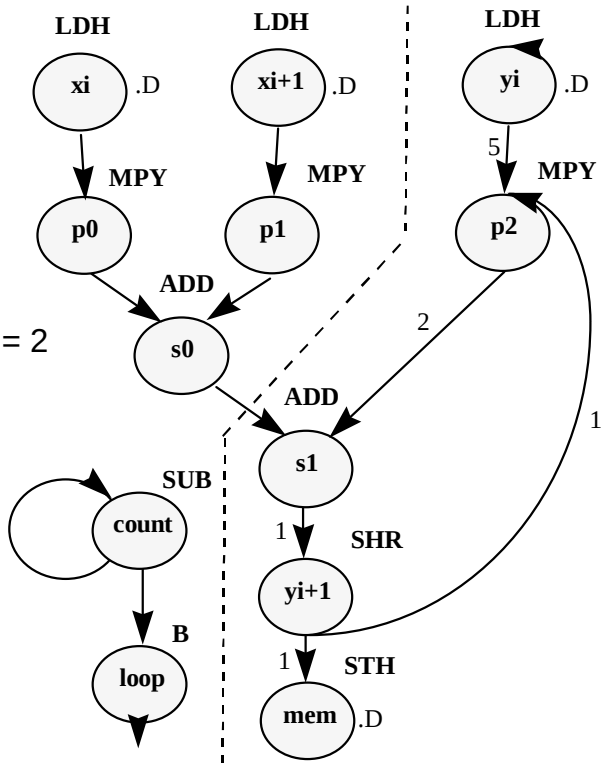
2) Choose a fairly even number of instructions for each side.

3) Force even numbers of certain functional units on each side. Figure 1 shows that even though there are four instructions that require a .D unit, they can be split on opposite sides to allow for an iteration interval of 2. The same is true of the three multiplies. Rather than putting all three on the same side, one is put on the opposite side.

4) Force even numbers of instructions that write to a conditional value on each side. Since there are a more limited number of conditional registers (there are 5 as opposed to the full 32 available for other source operands), it is easier to register allocate multiple conditional registers if they are split evenly between the two sides. If you have an uneven number, put more on the B side since there are three conditional registers on this side and only two on the A side.

5) Use the T1 and T2 path directives to force the result of Loads and the source of Stores to a particular side (it can be different than the side the D unit is on). Refer to the Memory Banks section in the “Assembly Optimizations” chapter of the TI *TMS320C6000 Programmer's Guide* (literature number SPRU198C) for more detailed information and examples.

Figure 1. Splitting a Dependency Graph



- Unpartitioned Resource Bound = 2
  - 4 .D Units
  - 3 .M Units
  - 2 .S Units
  - 3 gen purpose ADDS

TI Contact Numbers

INTERNET

TI Semiconductor Home Page
www.ti.com/sc

TI Distributors
www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas
Phone +1(972) 644-5580
Fax +1(972) 480-7800
Email sc-infomaster@ti.com

Europe, Middle East, and Africa
Phone
Deutsch +49-(0) 8161 80 3311
English +44-(0) 1604 66 3399
Español +34-(0) 90 23 54 0 28
Francais +33-(0) 1-30 70 11 64
Italiano +33-(0) 1-30 70 11 67
Fax +44-(0) 1604 66 33 34
Email epic@ti.com

Japan
Phone
International +81-3-3457-0972
Domestic 0120-81-0026
Fax
International +81-3-3457-1259
Domestic 0120-81-0036
Email pic-japan@ti.com

Asia
Phone
International +886-2-23786800
Domestic
Australia 1-800-881-011
TI Number -800-800-1450
China 10810
TI Number -800-800-1450
Hong Kong 800-96-1111
TI Number -800-800-1450
India 000-117
TI Number -800-800-1450
Indonesia 001-801-10
TI Number -800-800-1450
Korea 080-551-2804
Malaysia 1-800-800-011
TI Number -800-800-1450
New Zealand 000-911
TI Number -800-800-1450
Philippines 105-11
TI Number -800-800-1450
Singapore 800-0111-111
TI Number -800-800-1450
Taiwan 080-006800
Thailand 0019-991-1111
TI Number -800-800-1450
Fax 886-2-2378-6808
Email tiasia@ti.com

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE (“CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty, or endorsement thereof.