第15章 システム全体での最適化

本章では，システムを組み込んだときに注意すべき点や，システム全体での最適化などについて解説します．

15-1 キャッシュ・コヒーレンスの問題

C6000 DSPコアでSDRAM上の領域に書き込んだ後，HPI経由でホストCPUが読みにいくと，書き込む前の値が読めてしまったことがありませんか？　データ・キャッシュを使用していると，このような現象が生じることがあります．これを「コヒーレンスが保たれていない」といいます．問題となる場合は，次の二つが考えられます．

外部デバイスからペリフェラルを経由して，

1. DSPに接続している外部メモリ上にあるデータを読み込む場合
2. DSPに接続している外部メモリ上に書き込む場合

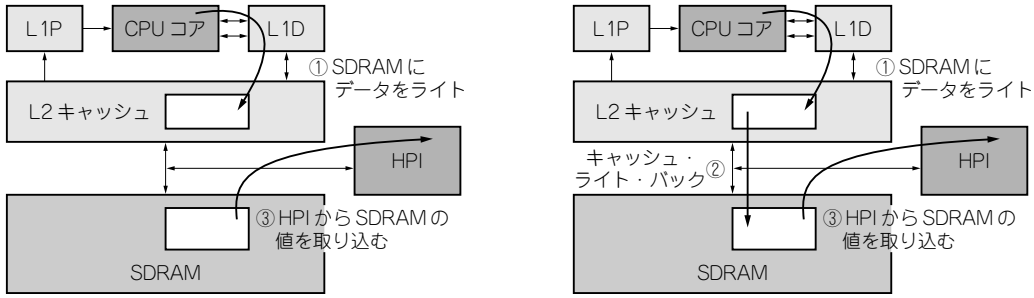
それぞれ発生する状況とその回避策を解説します．

1. DSPに接続している外部メモリ上にあるデータを読み込む場合（図15-1）

今，何らかの演算をDSPで行って結果を外部SDRAMに格納し，外部ホストがHPI経由でその結果を読み込むシステムを考えます．

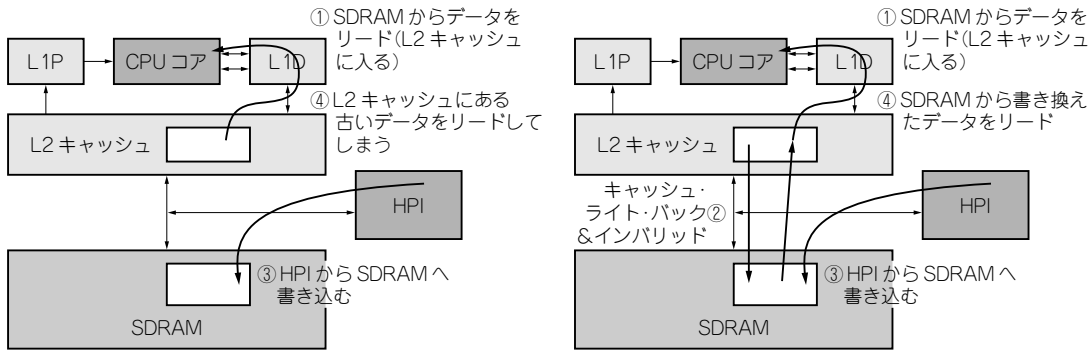
DSPコアは最初に演算を行い，その結果をSDRAMに書き込みます．すると，最新の演算結果は必ず，①L1D/L2キャッシュに格納され，SDRAMにはすぐに反映されません．この状態で，③外部ホストがこのSDRAMの内容を読み込んでも，最新の結果を読み出すことができません．

この状況を回避するために，演算結果を書き込んだ後，②L2キャッシュに入っている最新結果をSDRAMに出力する処理を行う必要があります．この処理を行うために，キャッシュ・ライト・バック関数`CACHE_wbL2()`が用意されています．この関数でSDRAMに出力すれば，外部ホストは最新の結果を読むことができます．



（**a**）HPI 経由で正しくデータが読めない場合　　（**b**）HPI 経由で正しくデータが読める場合

図15-1　DSPに接続している外部メモリ上にあるデータを読み込む場合



（**a**）HPI 経由で正しくデータが読めない場合　　（**b**）HPI 経由で正しくデータが読める場合

図15-2　DSPに接続している外部メモリ上に書き込む場合

2. DSPに接続している外部メモリ上に書き込む場合（図15-2）

次に，外部ホストがHPI経由でデータをSDRAMに書き込み，そのデータを使ってDSPコアが演算を開始するシステムを考えます．

この場合，最初はL2/L1Dキャッシュに何も入っていません．HPI経由でデータをSDRAMに書き込めば，DSPコアは書き込んだ新しい値を読み込み，演算できます．そのとき，SDRAMの値は①L2キャッシュに入ります．2回目，同じように外部ホストは③HPI経由でSDRAMにデータを書き込みます．すると，もちろん，SDRAMの値は今書き込んだ値になりますが，L2キャッシュに入っている値は1回目の値になっています．④このまま，DSPコアが演算すると，1回目に読み込んだ値を使ってしまいます．

この状況を回避するため，SDRAMにデータを書き込む前に，③L2キャッシュから1回目の値をなくす必要があります．このためにキャッシュ・ライトバック・インバリッド関数`CACHE_wbInvL2()`が用意されています．この関数で，書き込む領域の内容をL2キャッシュから削除した後にSDRAMに書き込めば，DSPコアは新しいデータを読むことができます．

C645x DSPのメモリ・ウィンドウでは，**図15-3**のように，各データがどこに格納されているか色

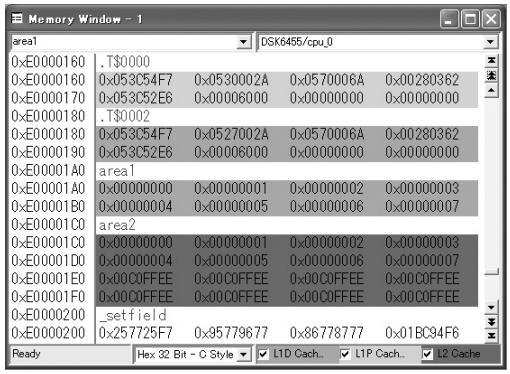


図15-3　CCS3.2でのメモリ・ウィンドウ画面

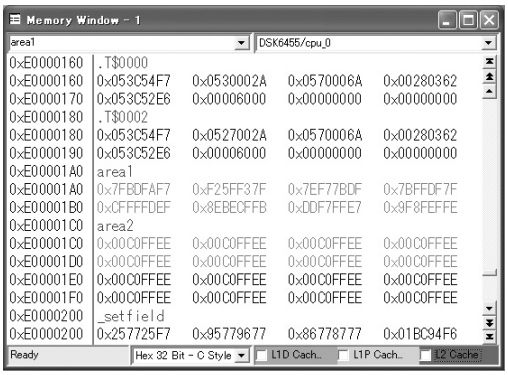


図15-4　外部DDR2に格納されている値を表示

別で表示できるようになっています．色の設定は，右クリック->Propertiesで表示する画面で選択できます．また，メモリ・ウィンドウの右下にあるL1D/L1P/L2のチェックによって，L1D/L1P/L2メモリの内容を選択して見ることもできます．もし，このチェックをすべて外した場合は，現在の外部メモリの内容を表示します(**図15-4**)．ここから，実際の外部メモリの値とコアから見た値が違うことが分かります．もしこの機能で外部ホストが外部メモリにあるarea2データを読み込んでも，キャッシュに入っている新しいデータを読むことができません．

ちなみに，DSPの内部メモリに対してデータを読み込むか書き込むかを行う場合，C64x+コアのL1P以外は必ずキャッシュ・コヒーレンシが保たれます．ユーザが何かする必要はありません．たとえば，もし，外部ホストがL2RAMにデータを書き込んだ場合，C64x/C67xコアではL1Dキャッシュ内の対応する領域を無効にします．C64x+コアでは，L2RAMとL1Dキャッシュ内の値を両方同時に書き込みます．

15-2　プログラム/データ配置で性能が変わる

キャッシュがうまく動作しない場合は，プログラムやデータの配置を変えてみるように第7章で記しました．実際，キャッシュがどのように動作するかを調べるために，キャッシュ・ヒット/ミスをグラフィカルに表示するキャッシュ・チューンと，タグの情報を含むキャッシュの状態を表示するキャッシュ・タグ・ビューアーという開発ツールが用意されています．

● キャッシュ・チューン

シミュレータでのみ動作できるツールで，時間を横軸にしたグラフでキャッシュのヒット/ミスとその番地を表示します．この表示をするためには，まずデータを計測します．これはプロファイラを使用します．Profile->SetupでProfile設定の画面を開き，Activity画面で「Collect Cache Information over time」を選択します．あとは，  をクリックしてプロファイル機能をONにし，計測してくだ

さい．計測後，Profile->Tuning->CacheTuneでキャッシュ・チューンを開くと，**図15-5**のように結果を表示します．

右側のグラフは縦軸がアドレス，横軸が時間(サイクル数)を示しています．キャッシュ・ヒット/ミスやキャッシュ・リード/ライトを色分けして表示しています．グラフ上にカーソルをもっていくと，カーソル位置のセクションを左上に表示します．そのセクションが.textのようなプログラムの場合，ダブルクリックすると，その部分のプログラムが表示されます．

このグラフ上でキャッシュ・ミスが連続して生じている部分では，プログラムやデータの配置を変更することによって高速化できる可能性があります．しかし，どのようなパターンのときに配置を修正したらよいか分かりにくいと思います．そのようなときには，キャッシュ・チューン画面内のData Cache/Program Cacheのタブをクリックすると，アドバイス画面内にヘルプが表示されます．その中のPatternをクリックすると，キャッシュ・ミスが生じているパターンの例が表示されます．この例とよく似ている場合はExampleをクリックして，表示される修正例を参考に変更してください．

実際にメモリの配置を変更する方法として，以下のような方法が挙げられます．

▶例1：変数を指定したメモリ領域に配置し，アラインおよびメモリ・バンクの設定を行う

リスト15-1のように，C6000のC言語上で，変数をユーザが定義したセクションに配置できます．その指定は，`#pragma DATA_SECTION`を使用することで指示できます．セクション名は自由に付けられます．このセクションをリンカのコマンド・ファイルに記載することによって，指定したメモリ領域に配置できます．また，変数をアラインした境界から配置したり，指定したメモリ・バンクから変数を配置したりすることもできます．

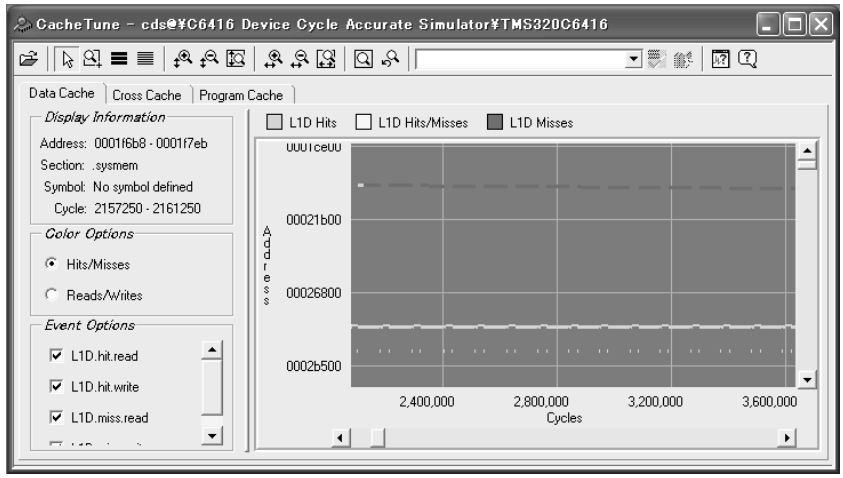


図15-5　キャッシュ・チューンの画面

▶例2：配列/変数の配置位置をずらす

リスト15-2では，指定したセクション内でダミーの配列を入れて，L1Dキャッシュの1ライン分ずらしています．

▶例3：プログラムを関数ごとに配置変更する

#pragma CODE_SECTIONを使用することにより，指定した関数をユーザが定義したセクションに配置できます．このセクションをリンカのコマンド・ファイルに記述すれば指定したメモリ領域に配置できます．

```
#pragma CODE_SECTION(f1,".funct1")  /* f1関数を.funct1セクションに配置 */
```

また，ある領域をアクセスする前にL1Dキャッシュに入れる関数として，touch関数(C64x用)を用意しています．プログラムによっては，この関数を先に実行して必要なデータをL1Dに入れてから実際のプログラムを実行すると，高速に実行できる場合があります．

リスト15-1　C6000のC言語上で変数をユーザが定義したセクションに配置する

```
#pragma DATA_SECTION(in1, ".mydata")  /* .mydataというセクションにin1を配置 */
#pragma DATA_SECTION(in2, ".mydata")  /* .mydataというセクションにin2を配置 */
#pragma DATA_ALIGN(in1, 32)           /* in1を32バイトでアラインした境界から配置 */
#pragma DATA_MEM_BANK(in1, 0)         /* in1をバンク0から配置 */
#pragma DATA_MEM_BANK(in2, 2)         /* in2をバンク2から配置 */
short in1[N];
short in2[N];
```

リスト15-2　L1Dキャッシュの1ライン分ずらす

```
#pragma DATA_SECTION(w, ".mydata")
#pragma DATA_SECTION(x, ".mydata")
#pragma DATA_SECTION(pad, ".mydata")
#pragma DATA_SECTION(h, ".mydata")
#pragma DATA_ALIGN  (w, CACHE_L1D_LINESIZE)
short w [N];
short x [N];
char pad [CACHE_L1D_LINESIZE];
short h [N];
```

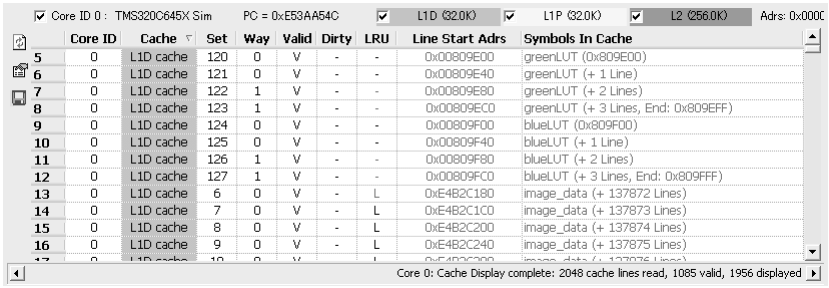


図15-6　キャッシュ・タグ・ビューアーの画面