［大規模科学計算システム］高速化推進研究活動報告第5号より転載

並列コンピュータExpress5800の高速化

スーパーコンピューティング研究部　江川隆輔　岡部公起
情報部情報基盤課　伊藤英一　小野敏　山下毅
日本電気株式会社　撫佐昭裕　神山典　小久保達信
吉村健二　坂本英顕　金野浩伸
NECシステムテクノロジー株式会社　曽我隆

5.1 Express5800の特徴

Express5800/A1080a-D（以下，Express5800）は，ノードあたりIntel Xeon X7560プロセッサ（以下，X7560）を4台搭載し，512GBの主記憶装置を有するスカラ型コンピュータである．東北大学サイバーサイエンスセンターは，Express5800を6ノード導入している．表5.1-1にExpress5800の主要諸元を示す．

表 5.1-1 Express5800 主要諸元

項目		諸元
最大演算性能		289.92GFLOPS
CPU数		4
コア数		32
CPU	名称	Intel Xeon X7560
	動作周波数	2.26GHz
	最大演算性能	9.06GFLOPS（コアあたり） 72.48GFLOPS（CPUあたり）
	QPI	25.6GB/s
	キャッシュ	L1：コアあたり32KB(命令)+32KB(データ)
		L2：コアあたり256KB(命令/データ共用)
		L3：CPUあたり24MB
主記憶装置	容量	512GB
	最大データ転送能力	34.1GB/s（CPUあたり）

X7560は8コアを有するIntel社製の64bitプロセッサであり，メモリコントローラをCPUに内蔵し，CPU－主記憶装置間を直結している．また，QPI（quick path interconnect）アーキテクチャを用い，CPU間の転送性能は，25.6GB/sである．図5.1-1にExpress5800のプロセッサ構成を示す．

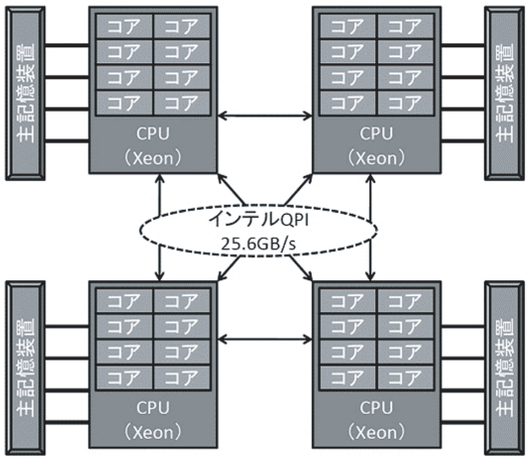

図 5.1-1 Express5800 のプロセッサ構成図

5.2 高速化技法

Express5800 で高い性能を得るためには，キャッシュの適切な利用とベクトル命令の活用が重要になる．ここでは Express5800 が搭載する X7560 の特性を活かした最適化手法を説明する．

5.2.1 キャッシュ・ブロッキングによる高速化

X7560 と主記憶装置間のメモリバンド幅は，34.1GB/s である．プロセッサあたりの理論ピーク性能が 72.48GFLOPS であることから，演算性能あたりのメモリバンド幅（Byte/FLOP）は 0.46 となり，SX-9 の 2.5 と比較してデータ供給能力が低いことがわかる．そのため，Express5800 ではキャッシュを有効に用いて，データ供給能力を補う工夫が必要になる．本節では，キャッシュ・ブロッキングによりキャッシュ内のデータを利用し，メモリのアクセス回数を削減する最適化手法について説明する．

(1) X7560 のキャッシュ構造

X7560 は，図 5.2-1 に示すように主記憶装置との間に L1 キャッシュ，L2 キャッシュ，L3 キャッシュの 3 階層のキャッシュを有している．L1 キャッシュと L2 キャッシュは各コアが個別に有しているが，L3 キャッシュは 8 つのコアで共有している．

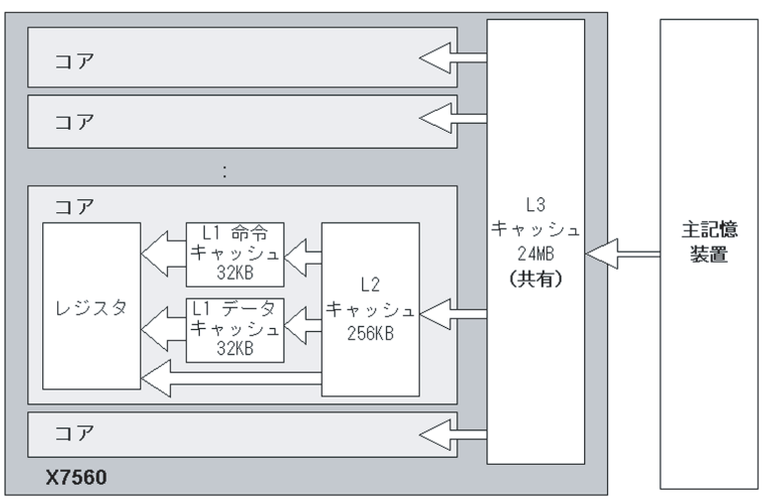

図 5.2-1 X7560 のキャッシュ構成図

キャッシュの特性として，L1，L2，L3 キャッシュの順にレジスタへ高速なデータ転送が可能である．したがって，プログラムの実効性能を向上させるためには，L1 キャッシュおよび L2 キャッシュを効果的に用いることが必要である．図 5.2-2 にキャッシュとプログラムの実効性能の関係を示す．この図は c=c+a(i,j)*b(i,j)の計算を行うループに対して，配列 a と配列 b の合計のメモリサイズを横軸に，実効性能を縦軸に取ったグラフである．配列 a と配列 b のデータが L1 キャッシュに乗っている間は高い性能を示しているが，データのサイズが L2 キャッシュの大きさになると性能は低下し，L3 キャッシュの大きさになるとさらに低下する．この結果から，高い実効性能を得るためには配列 a と配列 b のデータを L1，L2 キャッシュに乗せることが重要であることがわかる．

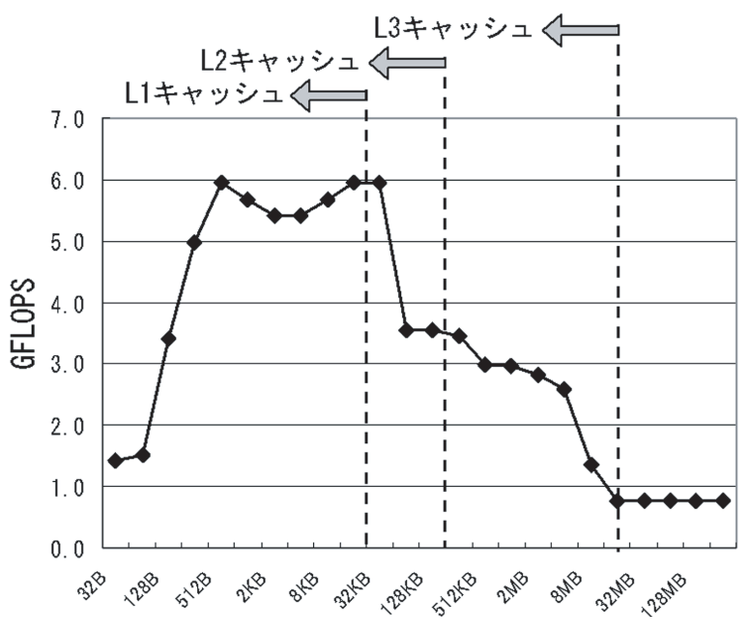

図 5.2-2 c=c+a(i,j)*b(i,j)の実効性能

(2) キャッシュ・ブロッキング

キャッシュ・ブロッキングとは，キャッシュ上にあるデータの再利用性を高めるため，DO ループをキャッシュサイズに合わせて分割する手法である．

図 5.2-3 に行列積を行うループのコードを示す．この DO ループは配列 a と配列 b の積を配列 c に集計する 3 重ループ構造である．最内の i の DO ループで，配列 a と配列 c の要素に連続アクセスしている．配列 b は i の DO ループに依存していないので，この最内側ループが実行されている間，配列 b は同じ要素が利用される．以下にキャッシュ・ブロッキングにおけるループ長の決め方と手順を示す．

配列 a は j の DO ループに依存していないので，j の DO ループを実行している間，一次元目の要素数(n=1,000 の場合，倍精度実数で 8,000 バイト)だけアクセスする．このサイズであれば配列 a は L2 キャッシュ上に乗せることができる．しかし，配列 c の全データがアクセスされるため，配列 a のデータは L2 キャッシュから追い出される．

```
parameter(n=1000)
   :
do k=1,n
  do j=1,n
    do i=1,n
     c(i,j) = c(i,j) + a(i,k)*b(k,j)
    enddo
  enddo
enddo
```

図 5.2-3 オリジナルの行列積ループ

ここで k の DO ループを実行している間, 配列 a のデータを L2 キャッシュに留め置くために, j の DO ループを分割し, 配列 c がアクセスする範囲を制限する. L2 キャッシュのサイズは 256KB であり, 32,768 個の倍精度実数データを乗せられる. 最内の i のループ長が 1,000 の場合, j のループの長さを 25 にすれば, 配列 a と配列 c の両方でアクセスする要素数は 26,000 個となり L2 キャッシュに乗せることができる. そこで図 5.2-4 に示すように, j と k の DO ループの外に増分 25 のループを設け, j の DO ループをループ長 25 でブロッキングする.

```
parameter(n=1000)
   :
do j2=1,n,25
  do k=1,n                 } j の DO ループでキャッシュ・ブロッキング
    do j=j2,j2+24
      do i=1,n
       c(i,j) = c(i,j) + a(i,k)*b(k,j)
      enddo
    enddo
  enddo
enddo
```

図 5.2-4 j の DO ループでブロッキングする行列積ループ

(3) 性能評価

表 5.2-1 にキャッシュ・ブロッキング前後の性能結果を示す. キャッシュ・ブロッキングの結果, メモリアクセス回数が削減され, 1.22 倍の性能向上が得られている.

表 5.2-1 行列積ループにおけるキャッシュ・ブロッキングの効果

	実効性能(GFLOPS)
キャッシュ・ブロッキング前の行列積ループ	2.24
j の DO ループでキャッシュ・ブロッキング	2.74

5.2.2 ベクトル命令の利用

X7560 のコアは，1 クロックサイクルあたり最大 4 演算を実行可能なアーキテクチャを採用している．このアーキテクチャを最大限に活用するため，コンパイラによるベクトル化が重要となる．

(1) ベクトル化の利用方法

コンパイラがベクトル化を行うのは，最内の DO ループに対し，以下の条件を満たす場合である．

① データの定義・引用に依存関係がないこと

② ループ中に複数のデータ型の演算(倍精度実数と単精度実数など)が混在しないこと

③ 間接参照によるメモリアクセスがないこと

Express5800 用のコンパイラf95は，コンパイルオプションを指定しなくても，ベクトル化可能なループをベクトル化する．プログラム中の DO ループがベクトル化されることは，コンパイル時にオプション「-vec-report3」を指定することで確認できる．図5.2-5にベクトル化される例と，図5.2-6にベクトル化されない例を示す．

図5.2-5では，DOループがベクトル化の必要要件を満たしているので，コンパイラはベクトル化を行う．しかしながら図 5.2-6 では，ループ中の配列 c が配列 itbl を用いた間接参照となるので，コンパイラがデータの依存関係を解析できないため，ベクトル化が行われない．

```
      subroutine sub(n,a,b,c)
      real*8 a(n,n),b(n,n),c(n,n)
       do j=1,n
         do i=1,n
           c(i,j) = c(i,j) + a(i,j)*b(i,j)
         enddo
       enddo
       return
       end

test.f(7): (col. 7) remark: loop was not vectorized: not inner loop.
test.f(6): (col. 9) remark: LOOP WAS VECTORIZED.
```

ベクトル化された

図 5.2-5 ベクトル化が行われる例

```
      subroutine sub(n,a,b,c,d)
      real*8 a(n,n),b(n,n),c(n,n)
      integer*4 itbl(n)
      do j=1,n
        do i=1,n
          c(itbl(i),j) = c(itbl(i),j) + a(i,j)*b(i,j)
        enddo
      enddo
      return
      end

test.f(8): (col. 7) remark: loop was not vectorized: not inner loop.
test.f(7): (col. 9) remark: loop was not vectorized: existence of vector dependence.
test.f(6): (col. 11) remark: vector dependence: assumed ANTI dependence between c line 6 and c line 6.
test.f(6): (col. 11) remark: vector dependence: assumed FLOW dependence between c line 6 and c line 6.
```

配列 c の依存関係が解析できないため
ベクトル化できない

図 5.2-6 ベクトル化が行われない例

(2) ベクトル化による性能

図 5.2-3 のコードを用いて，ベクトル化される場合と，ベクトル化を抑止する場合の性能差を示す．なおベクトル化を抑止するため，コンパイルオプション「-no-vec」を利用している．表 5.2-2 に示しすように，ベクトル化を行うことで 1.66 倍の性能差になることがわかる．

表 5.2-2 行列積がベクトル化された場合とベクトル化されない場合の実効性能

	実効性能(GFLOPS)
ベクトル化を抑止する場合	1.35
ベクトル化される場合	2.24