非構造格子 CFD ソルバにおける CPU キャッシュの有効利用法

坂下雅秀（大興電子通信株式会社），橋本敦，松尾裕一（宇宙航空研究開発機構）

The efficient use of Last-Level-Cache(LLC) on CPU with unstructured CFD solver

Masahide Sakashita（DAIKO DENSHI TSUSHIN, LTD.），
Atsushi Hashimoto, Yuichi Matsuo（Japan Aerospace Exploration Agency）

ABSTRACT

10 peta-FLOPS supercomputer "K computer" will begin cooperation in this autumn, and investigation of exa-scale computing already has begun. The other hand, memory throughput will not be enough to supercomputing. So, we consider to using Last-Level-Cache(LLC) on the CPU chip efficiently. Generally, the unstructured CFD solver is parallelized by region decomposition with "Halo". Then, small region which can be computed on LLC is made by decomposing recursively to region. And, with transposing data enough at first, data of another region composed are not required later.

１．はじめに

864筐体，82,944CPU（663,552コア）という巨大なシステムで論理ピーク演算性能11.3ペタフロップスを実現した京速コンピュータ「京」（「京」）の共同利用が本年（2012年）秋に開始されようとしている[*1*]．さらにその先を見据えて，エクサフロップス時代を実現するための様々な検討も，既にはじまっている[*2*][*3*].

そのような現状において，CFDの分野においても10,000並列を超えるような大規模並列計算への対応が求められている．

一方で，スーパーコンピュータ（スパコン）を構成する要素に目を向けてみると，CPUの演算性能向上にメモリのアクセススピード（メモリスループット）が追いつかない状況になりつつある．そこで，メモリより少ない容量ではあるが，十分高速にアクセス可能なキャッシュメモリ（キャッシュ）と呼ばれる機構がCPU内に設けられている．演算に必要なデータのメモリからの供給が十分でなくなりつつある現状では，いかにこのキャッシュを有効に利用するかということも，大規模並列計算への対応と並んで，将来に向けてのスーパーコンピューティングにおける重要な課題のひとつとなっている．

そこで，アプリケーションとして，非構造格子CFDソルバを念頭に，後者のキャッシュ有効利用の問題について考察する．キャッシュは，階層構造を持つことが一般的であるが，ここでは，そのうちのもっともメモリに近いキャッシュ(LLC：Last Level Cache）についてのみ考えることとする．

２．演算性能とメモリスループット

まずはじめに，「京」を中心として演算性能とメモリスループットの現状について概観する．

「京」は，SPARC64 VIIIfxと呼ばれるCPUを搭載している．そこで，同じSPARC64系のCPUを搭載したハイエンドテクニカルコンピューティングサーバ FX1（FX1）とPRIMEHPC FX10（FX10）を比較対照として傾向を見ていくこととする．ここに，FX1およびFX10は富士通株式会社（富士通）のHPC（High Performance Computing）用途向け製品である．FX1のCPUは，SPARC64 VIIIfxの前世代にあたるSPARC64 VIIであり，FX10は次の世代となるSPARC64 IXfxである．いずれのCPUも，open SPARCアーキテクチャに基づいて富士通が開発したCPUであるが，SPARC64 IXfxだけはLSI Corporationが共同開発者として参画している．表1に，各CPUの主な性能諸元を示す．表中のBF比（Byte per Flop）とは，メモリスループットを論理演算性能で除した値のことであり，演算性能に対してメモリスループットが十分であるか否かを判断するひとつの指標である．

表1 主な性能諸元[*4*][*5*][*6*]

	FX1	「京」	FX10
CPU	SPARC64 VII	SPARC64 VIIIfx	SPARC64 IXfx
コア数	4	8	16
ピーク性能	40GFLOPS	128GFLOPS	236.5GFLOPS
メモリスループット	40GB/s	64GB/s	85GB/s
BF比	1	0.5	0.36
キャッシュ	6MB	6MB	12MB
出荷開始時期	2008年	2010年	2012年

表1よりBF比が年々減少していることがわかる．図1に1980年の性能を1とした場合のCPUとメモリの性能の推移を示す．この図から，CPUの性能は年率60%程度の性能向上率であるのに対して，メモリの性能向上率は7%程度にしかなっておらず，メモリスループットの問題が，決して今に始まったことではないことがわかる．

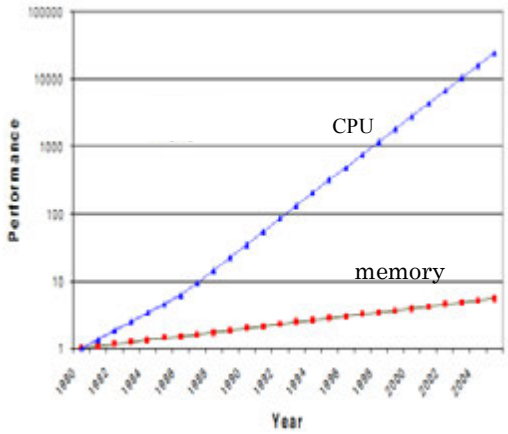

図1 CPUとメモリの性能の推移[7]

そこで，以下では圧縮性Navier-Stokes方程式を解く非構造格子CFDソルバ[8]にとって，現状のシステムのBF比がどの程度のものであるか評価することを試みる．

非圧縮性Navier-Stokes方程式を解く場合，計算に必要なデータ量に比べて計算量がもっとも多いのは，数値流束を計算する部分である．したがって，この部分を計算するのに必要なBF比について考える．表2に，FX1において実測した数値流束を計算する際に必要な浮動小数点演算量（FLOP）を示す．表中，HLLEおよびRoeは，数値流束を計算する場合の代表的なスキームである．実測は，1ノード（1cpu,4process）で，格子点数2,416,496のデータを用い，時間積分200ステップで行い，計算に要した時間（CPU time）および実効性能（MFLOPS）を実測し，これより演算量（GFLOP）を求めた．

表2 数値流束の計算に必要な演算量

	CPU time[s]	MFLOPS	GFLOP
HLLE	30.50	4198.59	128.07
Roe	38.70	4852.63	187.80

この結果より，HLLEスキームによる数値流束の計算に必要な1格子点あたりの演算量は，

$$128\times10^{9}[\mathrm{FLOP}]\div2.4\times10^{6}[\mathrm{grid}]\div200[\mathrm{step}]\cong300[\mathrm{FLOP}]$$

となる．一方で，この計算には，最低限，入力として5次元の保存量ベクトル2本と，出力として，やはり5次元の数値流束ベクトル1本が必要となるので，これをバイト数に換算すれば，

$$(5\times2+5)[\mathrm{word}]\times8[\mathrm{byte/word}]=120[\mathrm{byte}]$$

となる．ただし，非構造格子CFDソルバにおける数値流速の計算で通常必要となる間接参照用のインデックステーブル等付加的なデータの入出力は無視し，数値流速の計算に最低限必要なデータのみを考慮した．また、計算は倍精度実数型で行うものとした．この結果，HLLEスキームによる数値流束の計算に必要なBF比は，

$$120[\mathrm{byte}]\div300[\mathrm{FLOP}]=0.4$$

と計算できる．この結果からは，FX1のBF比1は数値流束の計算に十分であり，FX10のBF比では不十分ということが言える．

同様に，数値流束をRoeスキームにより計算することにした場合に必要なBF比は，およそ0.27程度であると計算できる．

ここで注意しなければならないのは，入力データにメトリックを含めていないことである．メトリックは，計算格子をどのような座標系で定義するかによって変わるので，ここでは評価の対象から除外した．実際には，メトリックも入力データとなることが一般的であり，その場合には，より大きなBF比が必要となる．また，CFDソルバにおける数値流束の計算以外の部分は，演算量が相対的に少なくなるため，やはり大きなBF比を要求する．このことを考え合わせると，FX1はともかく，「京」のBF比0.5でもCFDソルバの計算には不十分である．このBF比の不足を補うための機構がキャッシュであり，これを有効に利用することが必要な時期に来ていることがわかる．

３．キャッシュの有効利用法

CFDソルバの開発に係る人の多くは，CFDに関する専門知識を有しているものの，スパコンのハードウエアやコンパイラによる最適化手法に関する知識を持っているとは限らない．ここでは，そのような人でも，なるべく追加の知識を必要とせずに，比較的簡単にキャッシュを有効に利用できる方法について考える．

キャッシュを有効利用するための基本的な考え方は，キャッシュ上にあるデータを使ってなるべく多くの計算を行うようにすることである．一方で，CFD計算を行うためのプラットホームとしては，スパコンかPCクラスタが一般的であり，その結果MPIによりプロセス並列化されていることが多い．並列化の概略は以下の通りである（図2）．

(1) 解析領域をいくつかのリージョンに分割する．
(2) 隣接格子点の物理量を参照可能にするため「袖」と呼ばれる領域を確保する．
(3) 適宜，「袖」にデータを転送しながら計算を進める．

そこで，プロセス数を順次増加させて，領域をより多くのリージョンに細分化して行ったときにキャッシュミス率がどうなるかを実測した例を図3に示す．これよ

り，プロセス数の増加とともにキャッシュミス率は単調に減少していることがわかる．これは，リージョンに含まれる格子点数が減少することにより，一旦キャッシュにロードされたデータが他のデータをロードする必要からメモリに書き戻される可能性が減少したことによるものであり，キャッシュがより有効に利用されていることになる．

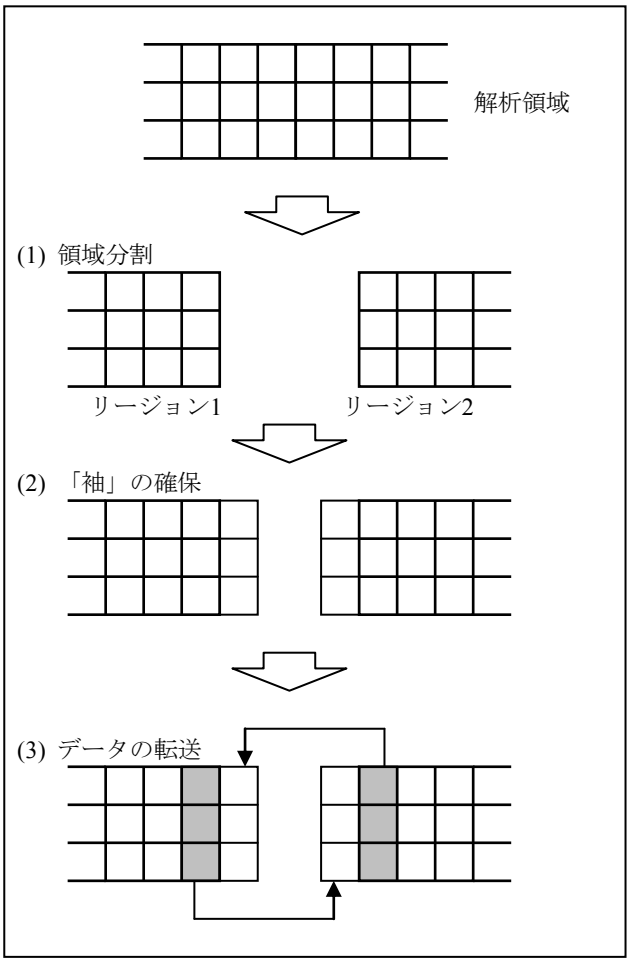

図2 MPI並列化の概要

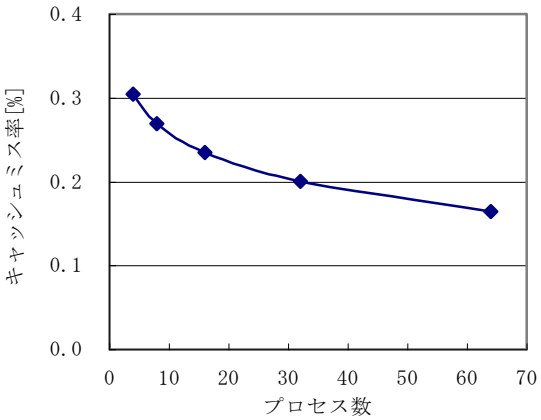

図3 プロセス数とキャッシュミス率の関係

しかし，領域の細分化には問題もある．図3の実測を行ったのと同じ条件のもと，プロセス数の増加とともに計算時間がどのように変化するかを測定し並列化効率を求めた結果を図4に示す．この結果からは，領域の細分化には性能的に限界があることがわかる．これは，プロセス間の負荷分散のアンバランスやプロセス間通信のオーバーヘッドなどの要因による性能低下が，キャッシュの有効利用による性能向上を上回ってしまうことが原因である．このことから，プロセス数を増大させずに処理を細分化する方法を考えなければならないことがわかる．

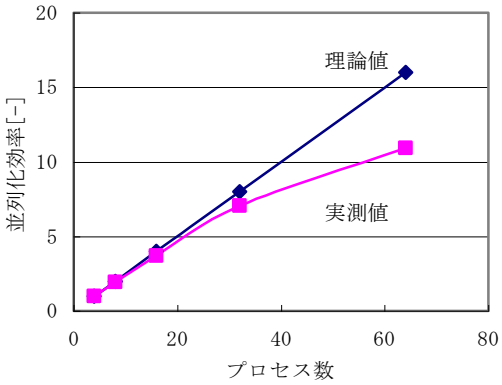

図4 プロセス数と並列化効率の関係

プロセス数を増大させずに処理を細分化するひとつの方法として，領域分割により生成された各リージョンに対して，再帰的に領域分割を適用し，サブリージョンを生成する方法が考えられる．すなわち，図2において解析領域をリージョン，リージョンをサブリージョンと読み替えればよい．適切なサブリージョンの大きさは，ハードウエアの構成に依存するが，1コア当たり1MBのキャッシュがある場合には，1,000格子点程度が目安であると考えられる．

ただし，データの転送（プロセス間通信）に関しては工夫を要する．今，「勾配の計算」—「勾配制限関数の計算」—「数値流束の計算」という非構造格子CFDソルバにおける典型的な計算の流れを例に考える．図5に一連の計算の流れの概略を示す．通常の領域分割による並列化では，まず計算に必要となる「袖」の部分のデータを転送した後（step1），「勾配の計算」を行う（step2）．この計算結果が次の「勾配制限関数の計算」で使用されるため，全てのリージョンで「勾配の計算」が行われた後，計算結果を「袖」の部分に転送する（step3）．転送が終わったのちに「勾配制限関数の計算」を始めることができる（step4）．「数値流束の計算」（step5）では，「勾配の計算」の計算結果とともに「勾配制限関数の計算」の計算結果も必要となるので，やはり計算を始める前に全てのリージョンで「勾配制限関数の計算」が終了

し「袖」の部分へのデータの転送（step6）が終わっていなければならない．

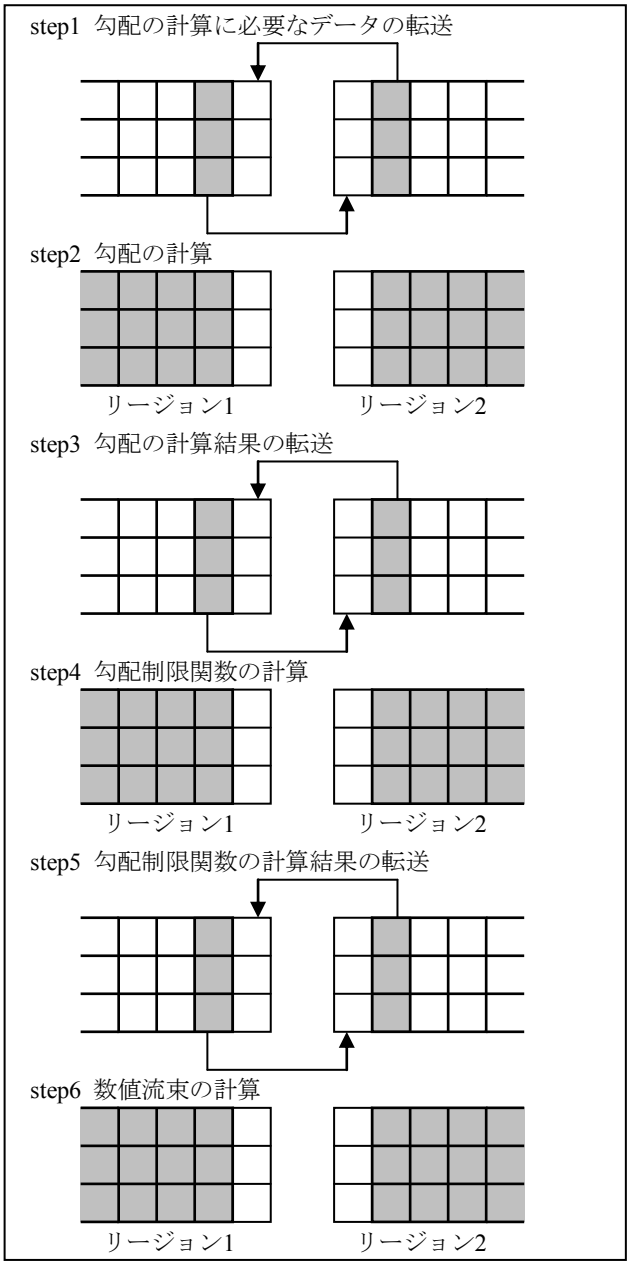

図5 非構造格子CFDソルバの計算の流れの概略

しかし，この方法をそのままサブリージョンにおける計算に応用すると，あるサブリージョンで「勾配の計算」に使用するためにキャッシュに乗せられたデータが，別のサブリージョンの計算をするためにキャッシュ上から消去されてしまい，効率の良いキャッシュ利用法とは言えない．そこで，計算と「袖」へのデータ転送を交互に繰り返すのではなく，「袖」を余分に確保して，最初にそこへ必要十分なデータを転送しておくことにすれば，そのサブリージョンに関する計算だけを連続して行うことが可能となる．今の場合，「袖」を1格子分ではなく3格子分用意し，「勾配の計算」では「袖」の部分を2格子分，「勾配制限関数の計算」では「袖」の部分を1格子分，余分に計算することにすれば良い（図6）．もし，n段階の計算を行いたい場合には，n格子分の「袖」を用意すればよい．

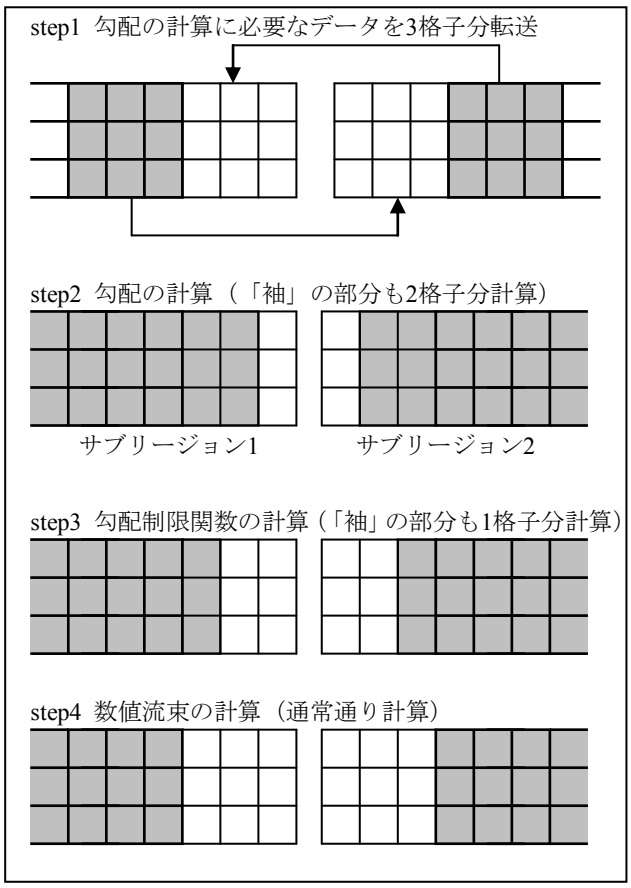

図6 サブリージョンにおける計算の流れの概略

この（「袖」の部分まで計算に含める）方法は，プロセス間通信を最適化する場合によく用いられる方法である．計算量は増えるが通信回数は減っており，通信量が同じであれば通信のオーバーヘッドが削減される．このことから，計算時間の増加に比べて，通信のオーバーヘッドの削減効果が大きければ全体の計算時間は減少する．非構造格子CFDソルバの場合には，勾配の計算結果を「袖」に転送するデータ量に比べ，勾配の計算に必要となる「袖」のデータ量の方が少ないので，転送データ量の削減にもなっている．そして，メモリスループットが十分でないハードウエアの場合には，キャッシュを有効に利用することにより計算時間が大幅に削減されることが期待できる．

実装に際しては，サブリージョンiに含まれる格子点の番号を$n_{s,i}$

$$n_{s,i} = \sum_{j=0}^{i-1} m_j + 1 \qquad i = 1, \cdots n$$

$$m_0 = 0$$

から始め，サブリージョン内で連番になるように付け替えるとよい．ただし，m_jはサブリージョンjに含まれる格子点の総数であり，nはサブリージョンの数である．この番号付け替えにより，サブリージョンを生成したことによる既存プログラムのデータ構造の変更が不要になる．そして，サブリージョン内のデータにアクセスするためには，各サブリージョンにおける格子点番号の最大値$n_{e,i}=n_{s,i+1}-1$に加えて$n_{e,0}=0$を保存したテーブルを用意しておけばよい．このとき，サブリージョンiに含まれる格子点の番号は，$n_{e,i-1}+1$から$n_{e,i}$である．

以上の方法は，MPIによるプロセス並列化の手法を再帰的に適用するものであり，実際にCFDソルバの並列化を行ったことのある人であれば，新たな知識を必要とせずに実装可能であるところに最大の特徴がある．

４．まとめ

「京」を中心とした現状のスパコンに対して，非構造格子CFDソルバの計算で必要となるBF比が確保されているか概算した結果，十分とは言えない状況になりつつあることが確認された．

そのため，BF比の不足を補うための機構であるキャッシュの有効利用法について考察した．CFDソルバがMPIにより領域分割の手法で並列化されていることを前提に，同様の方法を分割されたあとの領域に再帰的に適用することで，キャッシュに乗る程度の細かい領域を作ることが可能となる．このとき，「袖」を十分な量確保し，必要に応じて袖の部分も計算することにより，他の領域の計算結果を参照することなく計算を進めることが可能となる．これにより，キャッシュ上のデータが他の領域の計算のためにメモリに書き戻されるのを避けることができる．

この方法には，MPIによる領域分割の手法で並列化することができるならば，他の知識を必要とせずにキャッシュの有効利用が計れるという利点もある．

参考文献

[*1*] JHPCI
https://www.hpci-office.jp/

[*2*] 文部科学省
『今後のハイパフォーマンス・コンピューティング技術の研究開発について」の報告書のとりまとめ』
平成23年7月15日
http://www.mext.go.jp/b_menu/houdou/23/07/1308508.htm

[*3*] 東北大学サイバーサイエンスセンター他主催
「戦略的高性能計算システム開発」
http://www.open-supercomputer.org/workshop/

[*4*] ハイエンドテクニカルコンピューティングサーバFX1
http://jp.fujitsu.com/solutions/hpc/products/fx1.htm

[*5*] 世界最速スーパーコンピュータ「京」
http://www.nsc.riken.jp/shirutsudoi/8-houkoku/p_01.pdf

[*6*] PRIMEHPC FX10
http://jp.fujitsu.com/solutions/hpc/products/primehpc/

[*7*] Mucci, P., "Memory Bandwidth and the Performance of Scientific Applications: A Study of the AMD Opteron Processor" AMD Technical Whitepaper, June 2004.
http://web.eecs.utk.edu/~mucci/latest/mucci_pubs.php

[*8*] Hashimoto, A., et al., "Toward the Fastest Unstructured CFD Code 'FaSTAR'" AIAA paper 2012-1075