計算機アーキテクチャ 133－10
（1999. 5. 21）

ハイパフォーマンスコンピューティングに適したメモリ階層の検討

大河原 英喜† 中村 宏†
吉江友照†† 金谷和至††

ハイパフォーマンスコンピューティングを考えていくにあたって、メモリ構成は重要な問題である。特に、大規模な数値計算においては、十分な演算能力が与えられたとしても、下位のメモリ階層へのアクセスによる性能低下が顕著に現れる。本稿では、この問題への解決手法として、チップ上に実装するメモリとして、従来のキャッシュに加えてソフトウェアでアドレス指定可能な主記憶の一部を載せることを考える。具体的な例としてQCDシミュレーションをとりあげ、その有効性の初期評価を行なった。

Preliminary Evaluation of New Memory Hierarchy for High Performance Computing

HIDEKI OKAWARA,† HIROSHI NAKAMURA,† TOMOTERU YOSHIE†† and KAZUSHI KANAYA††

In high performance computing, memory hierarchy affects performance significantly. In large scale scientific application, accesses of lower memory hierarchy occur very frequently because data size is very large. In order to solve this problem, we propose a new memory hierarchy which includes software-controllable on-chip memory. We preliminary evaluate the effectivness of the proposed memory hierarchy on a real application of QCD simulation.

1. はじめに

ハイパフォーマンスコンピューティング(HPC)においては従来のメモリアーキテクチャは有効でなく、下位のメモリ階層へのアクセスが頻発し性能が大きく低下する。下位のメモリ階層へのアクセスによる性能低下の原因は、スループット不足とレーテンシ増大に分類できるが、スループットが不足している場合はレーテンシ隠蔽手法は有効でないため、メモリシステム全体としてのスループットを確保し、その上で適切なレーテンシ隠蔽手法を用いるべきである。今後の半導体実装技術を考えるとオフチップ記憶のスループットはオンチップ記憶のスループットより数倍低いことが予想され、再利用性のあるデータが高い頻度でオンチップ記憶へ載ることが可能なメモリ階層が重要である。

従来、オンチップ記憶としてはキャッシュが用いられ、キャッシュブロッキングなどのキャッシュ上の再利用性を向上させるソフトウェア的な最適化手法も提案されている[1]。しかし、従来のメモリ階層においては、キャッシュのリプレースメントやアロケーションの制御がハードウェアで決められているため、ソフトウェアによる改善には以下の難しさが存在する。

- キャッシュへのマッピングを制御できないので、載せたい配列をうまく載せられない場合がある。一つの配列内での干渉(self interference)を排除する手法も提案されている（文献[2][3]）。
- 複数の配列をキャッシュに載せる場合の配列間の干渉(cross interference)の排除は相当難しい。
- どのデータをキャッシュに載せるかを指定できない（再利用性のないBは載せたくない、等）

そこで、ハードウェアで制御の行なわれるキャッシュメモリのみのメモリ階層ではなく、ソフトウェアで制御可能なメモリ階層を提案する[4]。キャッシュに載せない配列を指定できるようにし、チップ上に実装するメモリとして、キャッシュに加えてアドレス指定可能な主記憶の一部を載せる（以降ではオンチップメモリと呼ぶ）ことを考える。この手法の有効性に関する評価を行なう。

† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo
†† 筑波大学計算物理学研究センター
Center for Computational Physics, University of Tsukuba

2. 提案するメモリ階層

オンチップメモリを用いたメモリ階層として、図1に示す様なメモリ階層を検討する。従来のキャッシュに加え、オンチップメモリ、さらにオフチップメモリからレジスタへの直接転送を考える。

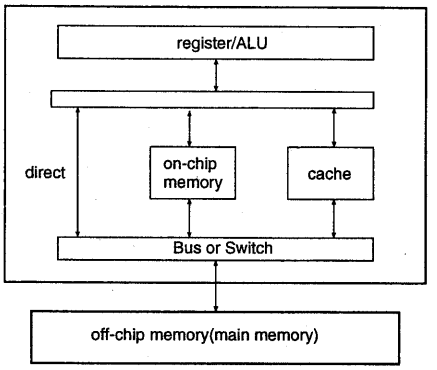

図1　提案するメモリ階層

従来のキャッシュのみのメモリ階層との違いは以下の点である。

- キャッシュに載せる配列と載せない配列をソフトウェアで指定できる。キャッシュに載せない配列は、オンチップメモリに載せるか、毎回オフチップから直接レジスタへ転送を行なう。
- オンチップメモリは、載せる配列データのリプレースメントやアロケーションの制御がソフトウェアで行なえる。キャッシュではリプレースメントやアロケーションの制御がハードウェアで行なわれており、この点がキャッシュとオンチップメモリの大きな違いである。

この違いによる利点として次の様なことが考えられる。キャッシュに載せるか載せないかを指定できるために、例えば、再利用性の低い配列をキャッシュに載せないことで、再利用性の高いデータに関してはその再利用性を他のデータに干渉されることなくキャッシュ上で最大限に活用することができる。キャッシュに載せる配列を減らすことで、複数の配列間の相互干渉の減少も期待される。また、二つ目のリプレースメントやアロケーションの制御がソフトウェアで行なえる性質から、ある配列をオンチップメモリ上に固定してリプレースを行なわないという様なことも可能である。例えば、データサイズが大きい配列の全データを繰り返し順番にアクセスする場合を考える。この場合、例えばフルアソシアティブキャッシュでLRUによりリプレースメント制御を行なった場合は、配列の他のデータをアクセスしていく間にリプレースメントが行なわれてしまい、キャッシュ上では再利用されない。しかし、オンチップメモリにおいては、ある一部の配列をオンチップメモリ上に残しておき再利用性を利用することも可能となる。

以下、この様な利点が性能をいかに向上するかを量子色力学の実アプリケーションを用いて評価する。

3. QCDシミュレーション

今回、大規模数値計算の具体的な例として、現在、超並列計算機CP-PACS（筑波大学）[5]において行なわれているQCDシミュレーション[6]をとりあげた。

QCD（量子色力学）[7]は、素粒子物理学の問題であり、素粒子の強い相互作用を記述する場の理論である。今回とりあげたQCDシミュレーションでは、ハドロン（強い力を感じる粒子）の基本構成粒子であるクォークと、それを結びつけるグルオン場を4次元格子空間上でシミュレーションし、ハドロンの諸性質に関する計算を行う。

各格子点(以後サイトと呼ぶ)にはクォークの自由度を表す3x4の複素行列Gが、隣接する各格子点を結ぶ最小線分(以後リンクと呼ぶ)にはグルオンの自由度を表す3x3の複素ユニタリ行列Uが用意される。

QCDシミュレーションでは、ほとんどの計算時間がグルオン伝搬関数Gを求める計算に費やされる。具体的には、既知の行列D(U)と行列Hを用いて、$H = D(U)G$という線形方程式を解いてグルオン伝搬関数Gを求める。ここでは反復法を用いて線形方程式を解くことを考え、今回は、反復法としてはBiCGStab法を用いたものを取り上げる。反復法の過程で何度も繰り返し行なわれるiteration loopが、計算時間の大半を占め、この部分を高速にすることは重要であるので、このiteration loopに注目して評価を行なった。

反復法を用いて、$H = D(U)G$という線形方程式を解く際には、

$$\mathrm{H = D(U)G} = \sum_{\mu=1}^{8} \mathrm{C_\mu U_\mu G_\mu} \qquad (1)$$

と表される計算（MULT計算と呼ぶ）が何度も行なわれており、処理時間がかかっている。μは4次元の正負方向で8方向それぞれを示すものである。図2に、簡単のため3次元格子空間としたイメージ図を示す。3次元のため(G,U)の組は6組になっている。

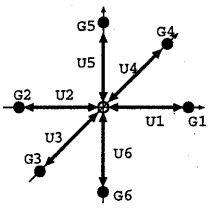

図2　MULT計算の3次元格子空間でのイメージ図

表1　iteration loop全体における load/store数と演算数

	load				store		mul	add
	G,R,B,V,T	U	M	others	G,R,B,V,T	others		
MULT間処理	404352				124416		435456	435456
RBMULT	870912	1492992		497664	124416	497664	3234816	3856896
localmult	124416		746496	373248	124416	248832	1492992	1741824

このMULT計算を4次元格子空間の各サイトにおいて行なう。MULT計算には一回り大きな格子空間上のG,Uの値が必要となる。この余分な空間を、以後、糊代と呼ぶ。

4. 評価対象計算の特徴

評価においては、iteration loopの構造、アクセスする配列のサイズ、再利用性を考慮する必要があるが、データサイズとしては、1プロセッサ辺り6^3*12の4次元格子空間におけるQCDシミュレーションを行なうと想定した。☆。

今回とりあげたシミュレーションでは、iteration loopでは全格子点に対して線形方程式を解くのではなく、一つおきの格子点における計算を行う。全格子空間を、一つおきの格子点による$(6^3*12)/2$サイトの2つのサイト群にわけ、odd,even(G_o,G_e)という様に呼んで区別することにする。iteration loop内の全ての計算は、odd,even単位の配列に対して行われる。iteration loopは、RBMULTルーチン、localmultルーチンと呼ぶルーチンと、その他の処理（ここではMULT間処理と呼ぶ）で構成される。RBMULTルーチンが、処理時間の長いMULT計算を行うルーチンである。実際にiteration loop内で行なわれるload/store数、演算数を表1に示す。

iteration loop内で用いられる配列の、宣言されるサイズと実際にアクセスされるサイズとを、表2に示す。G,R,B,V,Tは、同じサイズの行列が宣言され、この5つの配列サイズの合計をtotalに示している。

表2　iteration loopで使われる配列データ

	宣言	アクセス
G,R,B,V,T	1.36MB	最大1.0MB
total	6.8MB	2.5MB
U	4.1MB	1.5MB
M	3.0MB	3.0MB

☆ CP-PACSプロジェクトでは、現在、並列計算機CP-PACSを用いて24^3*48の4次元格子空間におけるQCDシミュレーションを行っている。CP-PACSの構成は,2048PU(peak 300MFLOPS/PU)となっており、全体で614.8GFLOPSである。将来的にはノードプロセッサ数の増加、ノードプロセッサの高速化から、ノード当り6^3*12の4次元格子空間におけるQCDシミュレーションを想定している。今回は、この想定に基づいた評価を行った。

次にiteration loopの流れを表3示す。ルーチンの横にB_e(0.5MB) ,U(1.5MB)→G_o(0.25MB)の様に示してある場合、配列B_eへの0.5MBアクセスと、配列Uへの1.5MBアクセスをして、0.25MBの配列G_oを計算することを意味している。

表3　iteration loopの構造

処理	計算に使う配列→結果をstoreする配列
1 continue	
MULT間処理1	
call RBMULT	B_e(0.5MB) ,U(1.5MB) →G_o(0.25MB)
call localmult	G_o(0.25MB),M_o(1.5MB)→G_o(0.25MB)
call RBMULT	G_o(0.5MB) ,U(1.5MB) →V_e(0.25MB)
call localmult	V_e(0.25MB),M_e(1.5MB)→V_e(0.25MB)
MULT間処理2	
call RBMULT	R_e(0.5MB) ,U(1.5MB) →G_o(0.25MB)
call localmult	G_o(0.25MB),M_o(1.5MB)→G_o(0.25MB)
call RBMULT	G_o(0.5MB) ,U(1.5MB) →T_e(0.25MB)
call localmult	T_e(0.25MB),M_e(1.5MB)→T_e(0.25MB)
MULT間処理3	
goto 1	

iteration loop内で用いられる配列は、再利用性、アクセスサイズに関して、次のような性質がある。

G,R,B,V,T　iteration loop全体で、連続したルーチン間での再利用性が高い。また、RBMULTルーチン内においては最大8回アクセスされ、ルーチン内での再利用性もある。RBMULTルーチンにおいて、MULT計算に用いられる糊代を含んだサイトデータへのアクセス(0.5MB)と、それとは異なる配列へのアクセス(0.25MB)との合計0.75MBのアクセスが行なわれる。localmultルーチン、MULT間処理においては各ルーチン内でのアクセスサイズは0.75MBより小さい。

U　RBMULTルーチンのみでアクセスされ、ルーチン内では1回しかアクセスされない。RBMULTルーチンにおいて1.5MBがアクセスされる。RBMULTルーチンはiteration loop内で4回呼び出される。

M　localmultルーチンのみでアクセスされ、ルーチン内では1回しかアクセスされない。localmultルーチンにおいて1.5MBがアクセスされる。iteration loop内で、odd部（M_o）を用いたlocalmultルーチンが2回、even部（M_e）を用いたlocalmultルーチンが2回、計4回localmultルーチンが呼

表4　各状態におけるメモリアクセストラフィック

	register (16double/cycle)	direct (16double/cycle)	cache (16double/cycle)	on-chip memory (16double/cycle)	off-chip memory (4double/cycle)
cache hit	1 double	0 double	1 double	0 double	0 double
cache miss	1 double	0 double	2 double	0 double	1 double
on-chip memory	1 double	0 double	0 double	1 double	0 double
direct	1 double	1 double	0 double	0 double	1 double

( )内は想定するスループット

び出される。
U,Mは各ルーチン内での再利用性はなく、iteration loop内のルーチン間における再利用性もわずかである。しかし、iteration loopは反復法の過程で何度も繰り返されるため、iteration loopにまたがった再利用性はある。従って、U,Mに関しても極力再利用を行なうことが望ましい。重要な特徴は、U,Mの再利用性のtime span（一度使われてから次に使われるまでの時間間隔）が、G,R,B,V,Tに比べて大変長いということである。

5. 評価方法

5.1 評価モデル

必要となる演算数とメモリアクセス数、およびハードウェア的に許される演算とメモリアクセスのスループットから性能（の上限値）を見積もる。初期評価として、この段階ではメモリアクセスレーテンシ、演算レーテンシは考えず、スループットのみの評価を行なった。演算処理能力は、(8mul+8add)/cycleの場合と(16mul+16add)/cycleの場合との2つの場合を考える。オンチップメモリ（含、キャッシュ）のスループットを16double/cycle、オフチップメモリのスループットを4double/cycleと想定した演算処理能力とスループットとの比によって性能のボトルネックがどう変わるかを評価していく。

キャッシュについては、1 double/lineのフルアソシアティブキャッシュを仮定した。すなわち、キャッシュとオンチップメモリとの間の得失利害のうち、キャッシュ特有の問題である line conflict （具体的にはself interference と cross interference ）による性能低下は、ここでは考慮されない。また、ラインサイズが大きい通常のキャッシュでは、ストライドアクセス時に不必要なデータ転送を伴い、無駄なトラフィックが生じるが、1double/lineと想定しているため、この問題による性能低下も無い。実際、今回とりあげたシミュレーションでは、一つおきの格子点における計算を行なっているため、連続アクセスしてではなく、ストライドアクセスを行なっている。そのため、ラインサイズが大きい場合には使われない無駄なデータのトラフィックが予想される。1double/lineとした想定においては、その様な無駄なトラフィックは発生しない。これらの仮定はキャッシュにとって非常に有利な仮定であり、オンチップメモリを用いる利点は「どのデータをどの様にオンチップメモリへ載せるか」の指定が可能であるという点のみである。また、storeに関してはキャッシュ上にあるデータに関してはキャッシュまでの書き戻しとして考えた。

1doubleのload/storeを行なう際には、表4に示すようなトラフィックが発生すると仮定している。レジスタにおいては必ず1doubleのトラフィックが生じる。キャッシュについては、キャッシュミス時には、オフチップ↔キャッシュのトラフィックと、キャッシュ↔レジスタのトラフィックとの計2doubleのトラフィックを考慮している。キャッシュに載せないデータは、オンチップメモリに載りきる範囲のデータはオンチップメモリに固定し、それ以外のデータは直接オフチップメモリからデータ転送を行なう(表4)。表4にある様な5種類の各トラフィックのうち最もサイクル数のかかるものをメモリアクセスに必要なサイクルと考えた。

5.2 提案するメモリ階層におけるリプレースメント、アロケーションの制御戦略

全データをキャッシュに載せた場合には、再利用性が高くアクセスするデータサイズが小さいG,R,B,V,Tと、再利用性はあるもののアクセスするデータサイズが大きいU,Mとをアクセスしてくことになる。再利用性のtime span（一度使われてから次に使われるまでの時間間隔）を考えると、G,R,B,V,Tはルーチン内や連続したルーチンで再利用されるためtime spanが短いが、U,Mはiteration loopが何度も繰り返される間での再利用性がほとんどのため、time spanはG,R,B,V,Tと比べて長くなっている。この様な場合、キャッシュ容量が十分でないと、U,Mが再利用される前に、G,R,B,V,Tなどの配列へのアクセスによってリプレースが行なわれ、キャッシュ上にU,Mのデータが残らず再利用性が活用されない。また、G,R,B,V,TとU,Mとの間の相互干渉のためにG,R,B,V,Tの再利用性も十分に活用されなくなる。

そこで、本稿で提案するメモリ階層におけるリプレースメント、アロケーション制御の戦略として次の様に考えて評価を行なう。再利用性の高いG,R,B,V,Tはキャッシュ上での再利用が可能であるため、キャッシュに載せる。iteration loopを通してのデータの再利用性はあるがアクセスされるデータサイズの大きい

U,Mはオンチップメモリに載せる。U,Mに関しては、オンチップメモリに乗りきるだけをオンチップメモリに固定して、乗りきらないデータは、毎回オフチップから直接レジスタ転送を行なう。オンチップメモリ上のデータの入れ替えは行なわない。U,Mを比較した時、Uの方が再利用性の高いため、Uを優先的にオンチップに載せる。

従って、キャッシュ容量が0MBの場合はG,R,B,V,Tは毎回オフチップから転送を行ない、オンチップメモリ容量が0MBの場合はU,Mは毎回オフチップから転送を行なう。

6. 評価結果

iteration loopにかかるサイクル数の評価結果を図3に示す。図の縦軸はiteration loopが反復される際に、iteration loop 一回の実行に必要なサイクル数を示している。横軸でオンチップメモリ容量を変化させ、キャッシュ容量として3つの場合の評価結果をそれぞれ示している。演算能力の違いによって実線と波線と区別している。

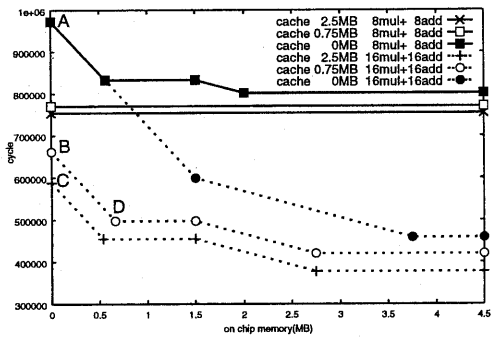

図3 オンチップメモリ、キャッシュの容量による性能変化

7. 考　　察

7.1 演算能力とメモリアクセススループットの比による影響

演算能力が(8mul+8add)/cycleの場合、図3の実線のグラフの様に、オンチップメモリ容量、キャッシュ容量による性能変化が少ない。これは、演算能力が(8mul+8add)/cycleの場合には、演算ボトルネックとなる傾向が強いためである。しかし、演算能力が高い(16mul+16add)/cycleの場合には、図3の点線のグラフの様に、オンチップメモリやキャッシュの容量による性能変化が大きく見られる。演算能力が高い場合には、メモリアクセスがボトルネックとなり、特に、オフチップアクセスがボトルネックとなっているため、オンチップメモリやキャッシュの容量により性能が大きく向上している。

7.2 キャッシュに載せない配列を指定できることによる影響

iteration loopでアクセスされる配列の性質から、再利用性の高いG,R,B,V,Tを有効に再利用することが性能に大きく影響する。ここでは、G,R,B,V,Tを有効に再利用するのに必要なキャッシュ容量を考える。

今回提案したメモリ階層では、キャッシュに載せない配列を指定できる。G,R,B,V,Tだけをキャッシュに載せた場合、図3中のB点に示す様に、0.75MBのキャッシュがあれば、RBMULTルーチン内の再利用性が完全に活かされ、ルーチン間にまたがる再利用性もある程度活かされるため性能が上がる。iteration loop全体でアクセスされるG,R,B,V,Tの総データサイズは表2にあるように、2.5MBであるので、2.5MBのキャッシュがある図3中のC点においては完全にG,R,B,V,Tが再利用されている。B点とC点とを比べても、それほど大きな性能差がないことからも、0.75MBキャッシュがデータの再利用性に極めて有効であることがわかる。

次に、キャッシュに載せるデータをソフトウェアで指定できず、全データをキャッシュに載せる従来のメモリ階層の場合にどの様になるかを考えてみる。各ルーチンでアクセスされるデータサイズや、iteration loopを通しての、各配列に対する再利用性を考えていくと、

- 0.75MBキャッシュ(G,R,B,V,T)
  U,Mは毎回オフチップ
- 2.25MBキャッシュ(G,R,B,V,T,U,M)

の場合において同じ性能が得られることが分かった。これは、U,Mの再利用性のtime spanが長いために、G,R,B,V,Tなどの再利用性のtime spanの短い配列へのアクセスによってリプレースが行なわれてしまい、キャッシュ上にU,Mのデータが残らないためである。この様に、同じ性能を得るのに、全データをキャッシュに載せる場合と、キャッシュに載せないデータを指定できる場合とで、必要となるキャッシュ容量が大きく異なる。キャッシュ容量が十分でないキャッシュ2.25MBの場合に再利用性のtime spanの異なる複数の配列を載せても、time spanの短い配列の再利用性しか十分に活かすことはできない。2.25MBより小さいキャッシュ容量の場合には、キャッシュにG,R,B,V,TとU,Mとを載せても、time spanの長いU,Mが再利用されない上に、time spanの短いG,R,B,V,Tがtime spanの長いU,Mと干渉して、G,R,B,V,Tの再利用性も十分に活かされなくなり、性能が低下する。

$$\begin{pmatrix} G\ cache(0.75MB) \\ U,M\ off-chip \end{pmatrix} = \begin{pmatrix} G,U,M\ cache(2.25MB) \end{pmatrix}$$

また、G,R,B,V,TとU,Mの全データを0.75MBのキャッシュに載せた場合を考えると、データサイズの大きいU,MとG,R,B,V,Tとを同時にアクセスしてい

くために、RBMULTルーチン等のルーチン内部においてもコンフリクトが発生するため、G,R,B,V,Tの再利用性が活かされなくなりオフチップアクセスが増加して性能が悪化することが容易に予想される。

7.3 オンチップメモリの有効性に関する考察

キャッシュに載せないデータを指定できる場合に、オンチップ記憶容量をキャッシュ0.75MBよりも増やすことを考える。キャッシュが0.75Mあれば、G,R,B,V,Tは既にある程度、再利用がされている。そこで、U,Mといった配列の再利用をすることで性能を挙げることを考える。オンチップメモリにUを載せることによって、図3中の点Dの様に、効果的な性能向上が行なわれる。

$$\begin{pmatrix} U\ memory(0.66MB) \\ G\ cache(0.75MB) \\ M\ off-chip \end{pmatrix} > \begin{pmatrix} G\ cache(2.25MB) \\ U, M\ off-chip \end{pmatrix}$$

ここで、Uを載せるためのメモリ階層がオンチップメモリではなくキャッシュだった場合を考える。Uはiteration loopが繰り返される中での再利用性はあるが、データサイズが大きいという性質がある。そのため、フルアソシアティブキャッシュにおいてキャッシュ容量が十分にない場合に、LRUによるリプレースメントが行なわれると、Uは再利用されない。しかし、オンチップメモリを用いることで、図3中の点Dの様に、効果的に性能が向上する。

この様に、ハードウェアで制御されるキャッシュのみのメモリアーキテクチャに比べ、オフチップ↔レジスタ、さらにはオフチップ↔オンチップメモリ↔レジスタという様に、ソフトウェアで制御可能なメモリ階層も用意し、柔軟なメモリアーキテクチャを考えることが、性能向上の有効な方法として考えられる。

8. 今後の展開

今回、QCDシミュレーションをとりあげ、反復回数が大きく、シミュレーション全体の処理時間の多くを占めるiteration loopに注目した評価を行った。オンチップ記憶容量が十分でない場合に様々なデータを全てキャッシュにのせるのでなく、ソフトウェア指定可能なメモリ階層と併用することによって性能が向上される結果が得られた。キャッシュにおいては、再利用性に関わらず古くにアクセスされたデータがキャッシュ上から追い出されてしまうため、time spanの長い再利用性を活用することができない。QCDシミュレーションに限定して考えれば、再利用性の高いG,R,B,V,Tをオンチップ上に載せることが重要であり、然る後に、U,Mを極力オンチップに常駐させることが効果的な性能向上に役立つが、キャッシュのみのアーキテクチャにおいては効率の悪いメモリアクセスが行われることが分かった。

今後、実装を考えるときには、演算能力とレジスタ数、キャッシュ、オンチップメモリなどとのバランスが重要となってくる。あるオンチップのメモリ容量が与えられた時に、キャッシュとオンチップメモリとのバランスや、ソフトウェアでオンチップメモリをどう利用するかといった戦略も性能に影響を与える。今回の初期評価を元に検討を行ない、現実的に考えられる、最適なメモリアーキテクチャを考えていきたい。

また、今後の課題として、具体的なメモリアーキテクチャの実装を考えていくことも挙げられる。オンチップメモリの制御や、メモリ階層全体をソフトウェアからどの様に使い分け可能にするかなどを考えていくことが大きな課題として挙げられる。

参考文献

1) 大河原英喜,"行列積計算におけるメモリアクセスによる性能低下". 東京大学南谷中村研輪講資料, http://www.hal.rcast.u-tokyo.ac.jp/~hideki/work/matrix.ps.gz

2) M. Lam, E. Rothberg and M. Wolf , "The Cache Performance and Optimizations of Blocked Algorithms", Proc. of ASPLOS,pp.63-74,1991

3) P. Panda, H. Nakamura, N. Dutt, and A. Nicolau, "Augmenting Loop Tiling with Data Alignment for Improved Cache Performance", IEEE Trans. on Computers,Vol.48,No.2,pp.142-149,1999

4) 近藤正章, 坂井修一, 朴泰祐, 中村宏,"オンチップメモリを用いたHPCプロセッサの検討", 情報処理学会研究報告,ARC-132,Vol.99,No.21,pp.85-90,1999

5) CP-PACS計画. 筑波大学計算物理学研究センター, http://www.rccp.tsukuba.ac.jp/cppacs/project-j.html

6) 専用並列計算機による場の物理の研究. 筑波大学計算物理学研究センター 平成6年8月 研究進捗状況報告書, http://www.rccp.tsukuba.ac.jp/pub/CPPACS/report94.ps

7) 原康夫, 量子色力学とは何か., 丸善 (1986)