SCIMA アーキテクチャのためのソフトウェア最適化手法の検討

中村実[†] 岩下誠[††]
坂井修一[†] 田中英彦[†]

ハイパフォーマンスコンピューティングでは、メモリアクセスのボトルネックが重要な問題となる。この問題への解決手法として、主記憶の一部をチップ上に実装したメモリ混載型プロセッサのアーキテクチャ SCIMA (Software Controlled Integrated Memory Architecture) が提案されている。本稿では、HPC アプリケーションの一つである宇宙の輻射輸送計算を SCIMA シミュレータ上で評価し、SCIMA のためのソフトウェア最適化手法を検討する。

Software Optimization Methods for SCIMA Architecture

Minoru Nakamura,[†] Makoto Iwashita,[††] Shuichi Sakai[†]
and Hidehiko Tanaka[†]

Memory system has been a performance bottleneck in High Performance Computing World. To solve this problem, SCIMA (Software Controlled Integrated Memory Architecture) which integrates controllable SRAM memory inside the processor chip, has been proposed. In this paper, we examine *Space Radiative Transfer Calculation* problem, evaluate performance of SCIMA architecture, and propose software optimization methods for SCIMA architecture.

1. はじめに

近年のプロセッサは、トランジスタ集積度の進歩、クロック周波数の向上が著しく、スーパスカラや VLIW に代表される命令レベル並列性の活用により演算能力が大幅に向上している。

しかし、メモリにおいては容量面では飛躍的に進歩しているものの、速度面やチップ I/O 数の向上がプロセッサの性能向上に追い付いていないのが現状である。このため、メモリアクセスがボトルネックとなることが従来から指摘され続けてきた。

この問題を削減するためにキャッシュ機構が広く使われてきている。キャッシュは局所性の高いデータを選択して主記憶から高速なメモリ領域に自動的にコピーし、メモリアクセスの高速化を図る。しかし、メモリアクセスパターンによってはデータの再利用性を最大限に使うことが出来ず、かえって性能の低下を招いてしまうという欠点を持っている。

この問題は、ハイパフォーマンスコンピューティング (HPC) を考えていくにあたって、特に重要である。それは、HPC ではメモリへのアクセスがボトルネックがプロセッサの演算性能以上に重要となるからである。

HPC 分野のアプリケーションは一般にデータセットが大きく、データキャッシュのキャッシュミス率が高くなり、性能の低下を招く傾向がある。特にキャッシュ機構に内在するライン干渉 (line conflict) は重要な問題である。Line conflict のうち配列内の要素によって生じる self-interference に関しては、cache blocking 等のソフトウェア最適化手法で削減することが可能である。しかし、異なる配列間で生じる cross-interference は削減・排除することは困難である。Cache のデータ再利用性を上げるためのソフトウェア最適化手法はいくつも考えられているが、どの手法も HPC アプリケーション中に存在するデータ再利用性を活かしきることができない。そこで、cache に代わる on-chip memory を備えたメモリ混載型プロセッサのアーキテクチャである **SCIMA(Software Controlled Integrated Memory Architecture)** が提案されている近藤 99)近藤 2000)大河原 99)。

本稿では、SCIMA アーキテクチャに基づくプロセッサでのソフトウェア最適化手法を検討するために、HPC アプリケーションの一つである宇宙の輻射輸送計算の問題をシミュレーションし評価する。

† 東京大学 大学院 工学系研究科
Graduate school of Engineering, The University of Tokyo
†† 東京大学 大学院 新領域創成科学研究科
Graduate school of Frontier Scientces, The University of Tokyo

2. SCIMA アーキテクチャ

SCIMA は現在研究・開発が進められている HPC 向けのメモリ混載型プロセッサのアーキテクチャである。cache と共に software control 可能な高速な S-RAM を大規模にチップ上に混載する点である。

本節では、その概要を述べる。

2.1 On-chip memory

SCIMA プロセッサのチップ上のメモリ (以下、on-chip memory) は主記憶と同一のアドレス空間を持ち、プログラムから load/store 命令により直接に利用することが可能である (図 1)。On-chip memory はキャッシュヒット時と同様に 1 〜 数サイクル内のレイテンシで load/store が可能となる。

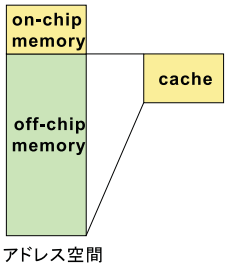

図 1 SCIMA のメモリ空間図

また、on-chip memory と register file 間は高いバンド幅を設けることが可能なので、load/store のスループットを cache よりも高くできる。

On-chip memory と外付けメモリ (以下、off-chip memory) との間のデータ転送は専用命令が設けられている。この専用命令 **page-load/store** は、off-chip memory 間と on-chip memory の間をストライド幅をつけての一括転送が可能となっている。転送されるデータ量は可変であるが、cache の line size よりも大きい単位でのデータ転送を狙い off-chip からのデータ転送を効率化する。また、on-chip memory はページ単位で管理される (1 ページは 128 バイト程度)。異なるページに対する load/store 命令、page-load/store 命令は互いに同時実行することが可能である。

そのため、データ転送命令とその他の演算命令を out-of-order 実行により重畳化し、レイテンシの隠蔽を図ることが可能である。

2.2 Controllable Data Cache

SCIMA の data cache は、プログラムからキャッシュされる領域とされない領域を指定可能な controllable cache である。

off-chip memory を仮想記憶のページ単位で cacheable/uncacheable 属性を指定できる機構を備える。再利用性が低いデータは uncacheable 属性をつけることで、cache を介さない direct access を行うことが出来る。これにより、再利用性がないデータが cache を汚染することを積極的に防ぐことが可能となる。

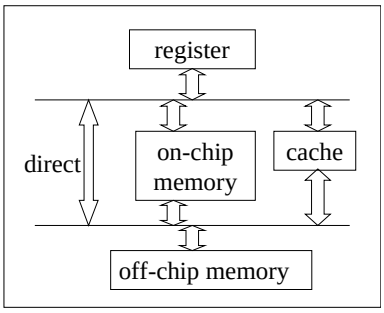

図 2 SCIMA アーキテクチャ

2.3 SCIMA シミュレータ

上記のような特徴を持つ SCIMA プロセッサをクロックレベルでシミュレーションするシミュレータ (**scim**) が開発されている[近藤 2000]。

このシミュレータでは、SCIMA プロセッサの ISA(Instruction Set Architecture) として、MIPS4 R10000 の ISA をベースとして SCIMA 用の拡張命令を加えた命令セット定義している。ただし、MIPS4 では論理レジスタの数は整数、浮動小数ともに 32 本であるが、SCIMA プロセッサではこれよりも多くの数のレジスタを扱える (実際の本数はシミュレータにパラメータで指定する)。

拡張命令やレジスタの拡張は、MIPS4 では定義されているが R10000 では未実装の lwc2、ldc2 などの不正命令を利用して実現する。これらの不正命令に対する動作をシミュレータ側で定義している。

以下、シミュレータの主な特徴を述べる。

- スーパースカラ。reservation station 機構に基づく out-of-order issue、out-of-order execution を行う。
- 100% 分岐予測成功の条件でシミュレーション。
- Instruction cache は 100 % ヒットするものとする。
- メモリ階層は L1 data cache、on-chip memory、主記憶 (off-chip memory) の各ハードウェアについてシミュレーションする。cache は lock-up free キャッシュとなる。L2 data cache は持たない。

3. 宇宙の多次元輻射輸送問題

輻射過程は宇宙の階層の構造の一つであり、電波・マイクロ波・赤外線・可視光・紫外線・X 線・γ 線などの様々な波長域の輻射と、これらを吸収したり放射したりする物質との相互作用である。輻射流体力学の計算は、以下の 3 つのステップに分けられる。

(1) 輻射輸送方程式を解く
(2) 求まった輻射場から輻射のモーメントを計算する

(3) 輻射のモーメントより輻射流体方程式を解く

この中で基本となるのは、1. の輻射輸送方程式を解くことである。そこで本稿では、多次元輻射輸送問題をシミュレーションする。

3.1 計算内容と解法

輻射は一般に、輻射強度と呼ばれる 6 次元位相空間（3 次元配位空間×3 次元運動量空間）上のスカラー関数 $I(\mathbf{x},\mathbf{n},\nu)$（$\mathbf{x}$: 3 次元の位置ベクトル、$\mathbf{n}$: 2 次元の輻射方向ベクトル、$\nu$: 輻射の振動数）で表現される。輻射輸送方程式とは、次のような 6 次元空間内の方程式である。

$$\mathbf{n}\cdot\nabla I(\mathbf{x},\mathbf{n},\nu) = -\chi(\mathbf{x},\mathbf{n},\nu) + \eta(\mathbf{x},\mathbf{n},\nu) \quad (1)$$

ここで $\chi(\mathbf{x},\mathbf{n},\nu)$、$\eta(\mathbf{x},\mathbf{n},\nu)$ はそれぞれ単位体積あたりの吸収係数および放射率であり、数値計算においてはこれらは既知関数とみなす。E (x) などの諸物理量は I より求めることができる。よって、核となる計算は、与えられた η と χ に対して1式を積分して各位置 $\mathbf{x}$ における各方向と $\mathbf{n}$ 振動数 ν に対応する輻射強度 I を求めることとなる。

輻射輸送方程式 (1) において η と χ が既知である場合、方向 $\mathbf{n}$ および振動数 ν が異なるものについては輻射強度 I は互いに独立となる。一方位置 x に関しては, 輻射の伝播する下流側は上流側に依存しているため、必ず上流側から下流側に向かって計算を進めねばならない。

格子法に基づき輻射輸送方程式 (1) を解く方法として Short Characteristics 法を用いる[中本99]。

以下に、X 方向に向かう場合の輻射輸送計算のプログラムの骨格を示す。

```
/* NX, NY, NZ は空間格子数 */

 for (ix = 1; ix < NX; iz++){
   for (iy = 1; iy < NY; iy++){
     for (iz = 1; iz < NX; ix++){
       /*start of iteration loop*/

       読み込まれる配列要素:

       Eta[ix-1][iy][iz],  Eta[ix-1][iy-1][iz]
       Eta[ix-1][iy][iz-1],Eta[ix-1][iy-1][iz-1]
       Chi[ix-1][iy][iz],  Chi[ix-1][iy-1][iz]
       Chi[ix-1][iy][iz-1],Chi[ix-1][iy-1][iz-1]
       I[ix-1][iy][iz],    I[ix-1][iy-1][iz]
       I[ix-1][iy][iz-1],  I[ix-1][iy-1][iz-1]

       Chi[ix][iy][iz],     Eta[ix][iy][iz]
       t[ix-1][iy][iz],     RadJ[ix][iy][iz]

       書き込まれる配列要素:

       I[ix][iy][iz],       t[ix][iy][iz]
       RadJ[ix][iy][iz]

       /*end of iteration loop*/
     }
   }
 }
```

3.2 最適化の戦略

輻射量計算の演算は最内核心部 (以後 iteration loop と呼ぶ) の繰り返しが基本構造となる。Iteration loop は各格子数の積の回数分繰り返され、実行回数が N^6 のオーダーとなる (方向により処理順序が変わることがあるが本質的には同じある)。よって実行時間の大半をこの部分が占めることになり、iteration loop の処理を高速化することが全体の処理の高速化につながる。

Iteration loop 内で扱う変数は, 以下のように分類可能である。

(1) 方向が一定であれば一定の値であり、毎回繰り返し使用される変数
(2) loop 内で計算され参照も loop 内のみで行われる、完全な局所性を持つ変数
(3) 再利用されるまでしばらく時間がかかることのある配列

SCIMA アーキテクチャを考えたとき、上記の 1. 及び 2. はその特性から、cache に載せるデータとすべきである。これらのデータ量は、合計で数 KB にも満たない。3. の配列に関しては、*Chi*・*Eta*・*I*・*t* の 4 つの配列があるが、*Chi*・*Eta*・*I* に関しては再利用のされ方が全く同じであり、*t* に関しても大差は無い。これらの配列は全て振動数と空間位置の 4 次元配列である。しかし輻射の振動数に対する独立性を利用すれば、空間位置の 3 次元配列として扱うことが可能である。また、*Chi*・*Eta* に関しては read-only であるという特徴があり、一定方向・振動数当たりの参照回数が 5 回と決まっている。よって、5 回の参照が行われたらその領域に次なる値を load することが可能であり、効率的に on-chip memory を利用することが可能である。

また、輻射量計算の方向によらず、blocking は大きな効果があると考える。Cache においては、Blocking を行えば cache-miss の発生を軽減することが可能である。On-chip memory においては page-load の際に stride 幅を指定できるため、無駄な line access は生じないが、blocking によりデータの局所性を上げることが可能なため blocking は有効である。そこで、*Chi*・*Eta*・*I*・*t* などの配列部分を block 単位で切り取り on-chip memory に載せるという戦略を取る。

この最適化方針では、再利用性に注目してデータを cacheable/uncacheable に分けるという SCIMA の controllable cache は使用しない。

3.3 ソフトウェア最適化

3.2 節の戦略に基づき、輻射量計算の配列全体を適当な大きさに blocking する。block 内で必要になるデータを page-load 命令で on-chip memory に一括して転送する処理、on-chip memory 内の情報のみで演算を行う処理、演算が終わったデータを page-store 命令で

off-chip memory に書き戻す処理の 3 種類をパイプライン化する (図 3)。

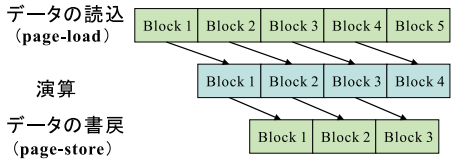

図 3　演算とデータ転送の重畳

block の格子サイズを N^3 とした場合、配列 $Chi \cdot Eta \cdot I$ はこれよりも 1 格子大きい $(N+1)^3$ のデータ、t は $(N+1)*N^2$ のデータ、$RadJ$ は N^3 のデータを読み取る必要がある。page-load/store 命令はストライドアクセスによる転送が指示できるので、それぞれの配列が $N+1$ または N 回の page-load 命令の実行によりデータ転送が可能である。ただし SCIMA プロセッサは、page-load/store 命令とその他の演算命令を out-of-order に実行可能ではあるが、大量の page-load/store 命令を連続して発効し過ぎると Reservation Station のエントリがあふれてストールしてしまう。そこで、page-load 命令は一度に連続して発効するのではなく、ブロック内の一番外側のループに合わせて逐次に発効していくものとする。データの書き戻しを行う page-store 命令も同様である。

4. 評　　価

プログラムは方向及び振動数を固定して、1 PU により配位空間全体の輻射量計算を行うと仮定してシミュレーションした。現在、輻射量計算の問題の規模は空間格子数 256 以上である。しかし、所要メモリ量及びシミュレーション実行時間の制約により、空間格子数は 32 に制限している。

SCIMA プロセッサとの比較のために、on-chip memory を持たない cache のみのプロセッサを用意し、それぞれに cache blocking・on-chip memory blocking 最適化を行ったプログラムを実行する動作をシミュレーションした。

4.1　内蔵メモリサイズの検討

輻射量計算は空間格子数 N=256 以上の計算が必要とされている。一方、チップ上に混載可能な SRAM は数十 MB となる見通しである。シミュレーションは N=32 で行うため、このときにチップ上のメモリ量をどれくらいにして評価を行うべきかを検討する必要がある。

cache の line conflict として self interference と cross interference があるが、self interference に関しては blocking で軽減が可能なため、cross interference のみに関して考える。

ここで 3 次元配列 A[N][N][N]、B[N][N][N](N は line size より十分大きいとする) を用いた、輻射のプログラムの配列へのアクセスを簡略化しモデル化したプログラムにおける cross interference を考える (他の変数の影響は無視する)。

```
for (x = 0; x < N-1; x++)
  for (y = 0; y < N-1; y++)
    for (z = 0; z < N-1; z++){
      /* A[ix][iy][iz],A[ix+1][iy][iz],
         B[ix][iy][iz],B[ix+1][iy][iz]
         を用いた計算 */
  }
```

格子数が N とすると、3 次元配列 A[N][N][N] を考えると、まず、(x,y,z)=(0,0,0) において A[1][0][0] の line が cache に載る。この line 上のデータが次に参照されるのは、(x,y,z)=(1,0,0) のときであり、このとき配列 B が self interference を起こしていないとすると cache に載る配列 B の line 数は $2*N*N/linesize$ となる。これより、cache の miss 率はこの条件においては N^2 のオーダーに従うと考えられる。以上より、N=256 での計算に対して、N=32 でシミュレーションを行うときは 128KB ～ 256KB 程度が妥当である。

そこで、cache のみを備えたプロセッサでは 128KB の L1 data cache を、On-chip memory を備えたプロセッサではスカラー変数をキャッシュするのに最小限必要な 8KB の data cache と 128KB の on-chip memory を持つものとする。

また、cache のみを備えたプロセッサは 4-way set associative cache、on-chip memory を備えたプロセッサは 2-way set associtative cache と差を付ける。

4.2　プログラムの作成方法

現在、SCIMA プロセッサ用のコンパイラはまだない。そのためシミュレータに入力するプログラムは、以下の手順で作成する。

(1)　シミュレーション対象プログラムを C で作成する (page-load などは疑似関数として記述する)。
(2)　SGI MIPS pro C コンパイラによりアセンブラを作成する。
(3)　トランスレータによって疑似関数をシミュレータで定義された拡張命令に変換する。同時に、浮動小数点レジスタ数の拡張を行い spill code を除去する。
(4)　手作業による命令のスケジューリングを行う。
(5)　アセンブラによってバイナリを作成する。

4.3　評価プログラム

Cache Only

On-chip memory を持たない Cache のみを用いた最適化。cache blocking を行っている。

Scima Opt.1 (No Prefetch)

先行ロードを行わない on-chip memory を用いた最適化。一辺の格子が 8 のサイズでブロック化する。

ブロック内で必要になるデータは、必要になった地点で page-load している。**Cache Only** と異なり、1 回で 500 〜 650 バイトのデータが転送される。
On-chip memory は 128KB のうち 26KB を使っている。

Scima Opt.2 (Prefetch)

先行ロードを行った on-chip memory を用いた最適化。一辺の格子が 8 のサイズでブロック化する。ブロックの演算を行っている間に、次のブロックで必要になるデータを先行ロードする。
On-chip memory は 128KB のうち 77KB を使っている。

Ideal

Scima Opt.2 のプログラムを、off-chip memory のレイテンシが 0、バスバンド幅が実効無限大、cache size 無限大、full-associative cache という理想状態で実行した結果を参考のために加える。

シミュレーションの際に、cache size、on-chip memory size、cache associativity をパラメータとして可変にして比較を行う。また、浮動小数点演算ユニットの数を 2 〜 8 の間で変化させている。
その他の各条件は以下のように設定した。

プロセッサ動作周波数	内部 2GHz
	外部 1GHz
バスバンド幅	16 bytes
整数レジスタ数	32
浮動小数レジスタ数	64
Cache line size	32 bytes
On-chip memory page size	128 bytes
整数演算 unit	1
Off-chip memory L/S Unit	4
Cache に対する L/S Unit	2

5. 結果と考察

Cache Only、Scima Opt.1、Scima Opt.2 の各プログラムをシミュレーションした結果を 図 4 にあらわす。また、Scima Opt.2 の実行結果は図 5 に拡大してあらわす。

Cache Only、Scima Opt.1、Scima Opt.2 とも off-chip memory のレンテンシが 0 の条件ではほぼ同じ実行サイクルで実行されているが、レイテンシが増加に対するグラフの傾きは Cache Only、Scima Opt.1、Scima Opt.2 の順で大きくなってくる。

グラフの傾きから判断すると、Scima Opt.2 は off-chip memory のレイテンシの影響をほとんど受けていない。これは、page-load による先行ロードがほぼ完全に行われいてデータ転送レイテンシの隠蔽に成功しているためである。例えば、FPU 2、off-chip memory レイテンシ 0 の条件では、全実行サイクルから判断すると、先行ロードは約 65,000 サイクル前に開始されていることになる。

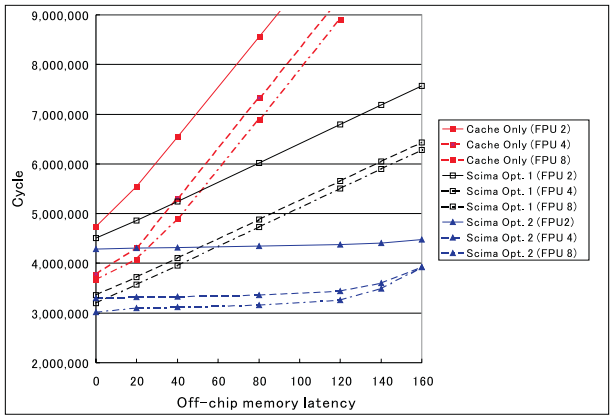

図 4　アルゴリズム別の実行サイクルの評価 (格子数 32)

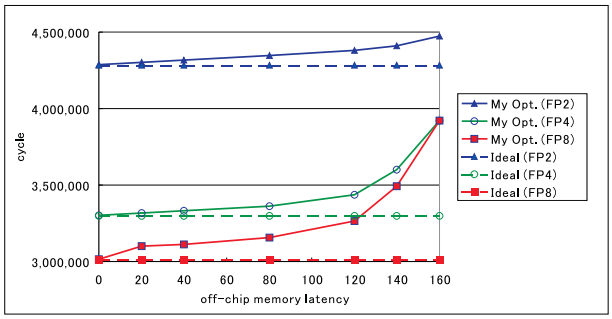

図 5　Scima Opt. 2 の実行サイクルの評価 (格子数 3 2)

Cache Only、Scima Opt.2 は共に先行ロードを明示的に行わないプログラムである。両プログラムとも off-chip memory のレイテンシに比例して実行サイクルを伸ばしているが、その傾きには差がある。これは、Cache Only が cache line size 単位 (32 バイト) でのデータ転送を行っているのに対して、Scima Opt.2 が page-load 命令により 500 〜 650 バイトを一括して転送するため、プリフェッチの効果が出ているためである。

浮動小数点ユニット (FPU) の数を 2 個、4 個、8 個と変化させた場合の性能向上についてみると、FPU を 2 → 4 では 20% 程度の性能向上が得られるのに対して、4 → 8 では 3% 程度の性能向上しか得られない。

性能向上が鈍磨する原因は、SCIMA 用の専用コンパイラが現在なく MIPSpro C Compiler の出力を利用しているため、Instruction scheduling・Register allocation が SCIMA に最適化されていないことが挙げられる。

Scima Opt.2 のFPU 8 の場合に、Reservation Station のエントリ数を調整して、load/store のエントリ数を 8 から 32 に調整するとさらに 14 % の性能改善が見られた (表 6)。コードに対しては、基本ブロック内での Instruction scheduling を行っているが、これだけでは十分でないことが分る。

条件	実行サイクル
L/S の RS エントリ 8	3,014,058
L/S の RS エントリ 32	2,594,630

図 6 Scima Opt.2、FPU 8 の場合の実行サイクル

実際のコードは、iteration loop 1 回の中に、条件分岐 1 つと 2 または 3 回の exp 関数の呼び出しを含んでいる。性能向上を目指すには、大域的、関数間での Instruction scheduling を行う必要がある。

6. まとめと今後の課題

本稿では、HPC 分野をターゲットにして研究・開発されている SCIMA アーキテクチャに対して輻射輸送問題をソフトウェア最適化し、SCIMA シミュレータ上でその評価を行った。

そして、page-load/store 命令を用いた on-chip memory blocking の手法が、従来の cache 機構と cache blocking 手法より優れた性能を発揮することを確認した。特に page-load/store を用いた先行ロードが成功するとデータ転送レイテンシがほぼ完全に隠蔽することを示せた。

しかし、現在は SCIMA プロセッサ用のコンパイラが開発されていないため、命令レベルの Scheduling まで含めた SCIMA 最適化を行うことができない。SCIMA 用 C コンパイラを開発することが急務である。

謝　　辞

本研究を進めるにあたり、東京大学 先端科学技術センターの中村 宏助教授、筑波大学 計算物理学研究センターの朴泰祐助教授、中本 泰史氏、梅村 雅之氏から適切な御助言を頂きました。また、東大先端研 近藤 正章さんからは SCIMA シミュレータを、大河原英喜さんからはバイナリトランスレータを使わせていただき、またご指導いただきました。深く感謝致します。

参　考　文　献

[近藤 2000] 近藤正章**オンチップメモリを用いた HPC 向けプロセッサの提案** 筑波大学大学院 工学系研究科 修士論文,2000 年 2 月

[大河原 99] 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に適したメモリ階層の検討** 情報処理学会研究報告,1999 年 5 月

[中本 99] 中本泰史**多次元輻射輸送計算コードの超並列計算機への実装** 情報処理学会研究報告,1999 年 12 月

[近藤 99] 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** 情報処理学会研究会報告,1999 年 3 月