８コアプロセッサの性能評価分析

Performance Analysis of 8 Cores Processors

河辺　峻

KAWABE, Shun

要旨

プロセッサは現在マルチコア化による高速化が進んでいる。並列処理が可能なプログラムやスループットを主とする多重プログラムにとっては高速化が期待できる。しかしキャッシュ構成は複雑化しており、それぞれのコアが所有するキャッシュは、L1,L2,L3 に階層化されている。最新の Intel プロセッサチップでは、L1/L2 キャッシュはコア間で共有せず、L3 キャッシュはコア間で共有している。また複数のプロセッサチップを搭載するサーバでは、L3 キャッシュ間で情報の交信を行っている。このような構成において、キャッシュのコヒーレンシ（一貫性）を保ちながら性能の特性がどのようになるかについて評価プログラムを作成して８コアプロセッサ性能の特性を評価した。

１．はじめに

現在、プロセッサのシングルコアの周波数が消費電力や発熱問題により頭打ちになり、デュアルコアやクアッドコアが普及しつつある。そしてこれからはさらにプロセッサのコア数が増える方向へと進んでいくと思われる。本研究では８コアプロセッサを用いてキャッシュの整合性を取るためにどれくらい性能が低下するかを Linux の C 言語を用いて分析プログラムを作成して評価する。８コアとは演算処理を行うコアが８つ存在するプロセッサの事を指す。今回の構成は外部的には２つのプロセッサチップで構成され、１つのプロセッサには４つのコアがある。

２．８コアプロセッサの性能評価

２．１　評価に用いるプロセッサ

図１に今回の評価で用いたプロセッサの構成図を示す。

Intel E5620(Westmere-EP)プロセッサは図１に示すように、１つのプロセッサに４つのコアがあり各コアごとに 32KB のデータおよび命令キャッシュと 256KB の L2 キャッシュを持ち、各コアが共有する 12MB の L3 キャッシュを持っている。周波数は 2.40GHz で TPD は 80W である。このプロセッサチップがボード上に２つ搭載されており、L3 コントローラでプロセッサ間の情報の交信を行っている。また、それぞれのコアが HT(Hyper Threading)機能を持って

いる。このためプログラムからは論理的には 16 のプロセッサがあるように見える。

今回の評価では affinity 機能を用いて使用するコアを固定し、HT(Hyper Threading)機能は使用しないようにした。また OS は Fedora14(64b)を使用した。

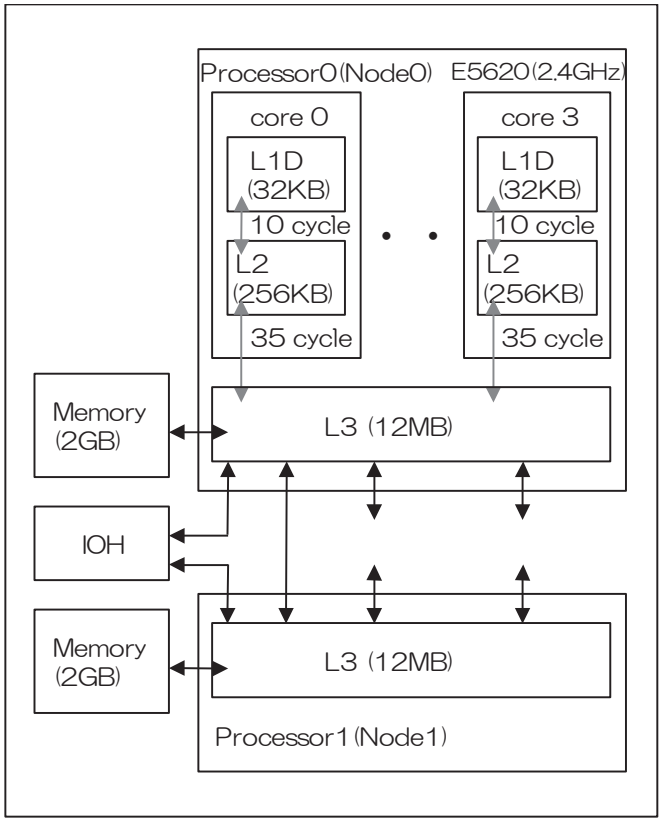


図１　評価に用いたプロセッサ構成図

２．２　８コアの性能評価手法と結果

(1) 測定プログラム

メモリ上の連続したエリア(8B)にコア対応に 1B 毎に更新(0xFF との排他的論理和)を行い、それぞれのコアでの 1 ループの性能(clock cycle)を測定する。

Core0	Core1	Core2	Core3	Core4	Core5	Core6	Core7
gbuf[0]	gbuf[1]	gbuf[2]	gbuf[3]	gbuf[4]	gbuf[5]	gbuf[6]	gbuf[7]

それぞれのコアが所有するキャッシュは、ブロックあるいはラインと呼ばれる 64B 単位にメモリからデータを持ってくる。したがって 8B をコア対応に 1B 毎に更新させると、各コアで同じ 64B のデータを共有するため、キャッシュのコヒーレンシ（一貫性）を保つ必要が出てくる。

・affinity 機能を使用したコアの指定（コア 0 を指定した例）

```
cpu_set_t mask0;
CPU_ZERO(&mask0);
CPU_SET(0,&mask0);
rv=sched_setaffinity(0,sizeof(mask0),&mask0)
```

・測定部分のプログラム（コア 0 の例）

gettimeofday 関数を持ちいて 10 億回（約数秒かかる）のループを測定する。

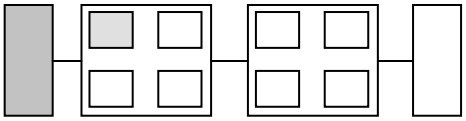
```
gettimeofday(&st,NULL);
for(a=0;a<tr;a++)
    {
        gbuf[0] = gbuf[0]^0XFF;
    }
gettimeofday(&et,NULL);
```

ここの部分を gcc でコンパイルしたアセンブラコードを次に示す。

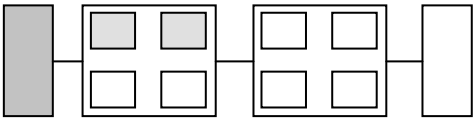
```
        call    gettimeofday
        movl    $0, -20(%rbp)
        jmp     .L4
.L5:
        movq    gbuf(%rip), %rax
        movq    gbuf(%rip), %rdx
        movzbl  (%rdx), %edx        /* gbuf[0]の内容を edx に格納 */
        notl    %edx                /* edx の内容を反転 */
        movb    %dl, (%rax)         /* edx の下位 1B を gbuf[0]に格納 */
        addl    $1, -20(%rbp)       /* a++ */
.L4:
        movl    tr(%rip), %eax      /* tr の内容を eax に格納 */
        cmpl    %eax, -20(%rbp)     /* tr と a を比較 */
        jl      .L5                 /* L5 へループ */
        leaq    -64(%rbp), %rax
        movl    $0, %esi
        movq    %rax, %rdi
        call    gettimeofday
```

(2)同一プロセッサチップ(同一ノード)による測定(affinity で core を固定)

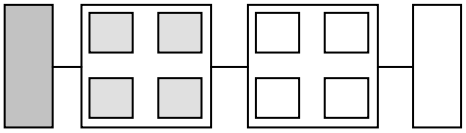
・1core



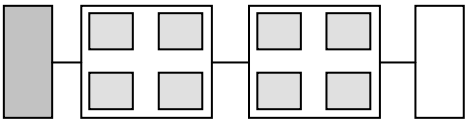
・2core



・4core



・8core



・結果

clock cycle

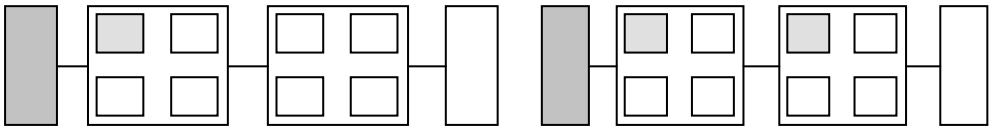
	1core	2core	4core	8core
	5.42	8.84	16.02	41.25
		8.46	18.83	42.76
			18.81	43.01
			18.80	42.73
				42.32
				42.74
				43.01
				42.67
平均	5.42	8.65	18.12	42.56
	1	1.60	3.34	7.85

1core の場合は 5.42 clock cycle かかっている。アセンブラコードから見ると 9 命令のループとなっている。すべての命令とデータが L1 キャッシュにあり、4 命令程度のスーパースカラ機能を持っているのでこの値は妥当な値と考える。2core, 4core, 8core になるに従い、cache coherence のオーバーヘッドが大きくなり、それぞれ 8.65, 18.12, 42.56 clock cycle となる。1core の場合を 1 とすると clock cycle の比は、それぞれ 1.60, 3.34, 7.85 となる。

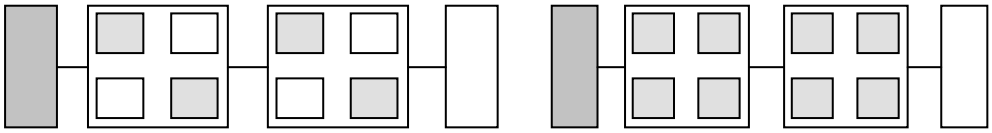
ちなみにメモリ上 64B ごと離れたエリアにコア対応に 1B 毎に更新(0xFF との排他的論理和)を行い、それぞれのコアでの 1 ループの性能(clock cycle)を測定すると、各コアでの 1 ループの性能は、それぞれ 5.70, 6.04, 6.10, 6.22, 6.15, 6.20, 6.07, 6.03 となり、ほとんど変わらない。

(3)異なるプロセッサチップ(異なるノード)による測定(affinity で core を固定)

・1core　・2core



・4core　・8core



・結果

clock cycle

	1core	2core	4core	8core
	5.42	13.18	19.80	41.19
		13.18	19.35	42.30
			18.44	43.01
			16.97	43.03
				42.71
				42.71
				42.69
				42.67
平均	5.42	13.18	18.64	42.54
	1	2.43	3.44	7.85

1core の場合は同じく 5.42 clock cycle かかっている。2core, 4core, 8core になるに従い、cache coherence のオーバーヘッドが大きくなり、それぞれ 13.18, 18.64, 42.54 clock cycle となる。1core の場合を 1 とすると clock cycle の比は、それぞれ 2.43, 3.44, 7.85 となる。これらの値は、同一プロセッサチップ(同一ノード)の値と比較して悪くなっている。

(4)考察

図２に 1core の性能を１としたときのクロックサイクル数の比を示す。これからも分かるように同一ブロックの内容を各コアでキャッシュに取り込んでしまうと、更新エリアは各コアで異なっていても、キャッシュのコヒーレンシ（一貫性）を保つための論理回路が動作して性能が大幅に低下する。この低下の割合は、コア数が増えるほど大きく、また異なるプロセッサチップ(異なるノード)の方が大きい。

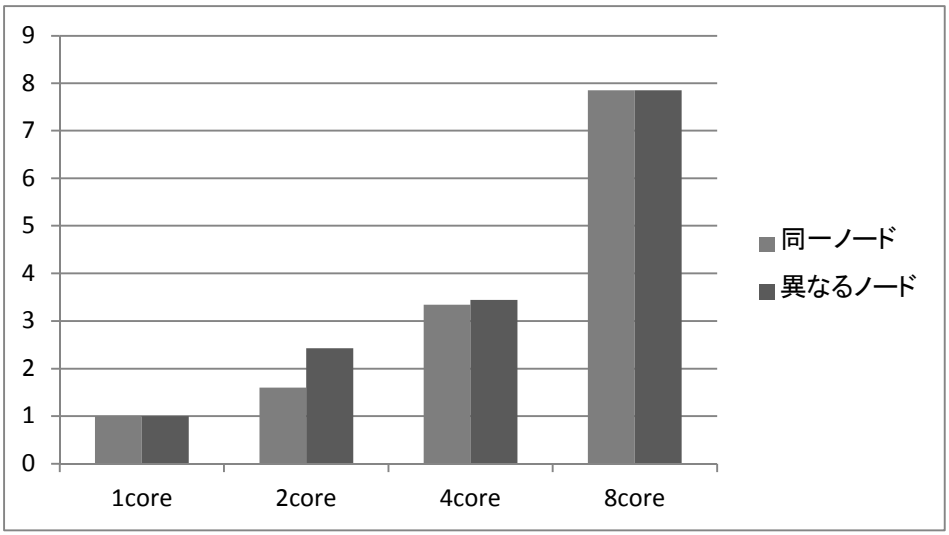


図２　1core の性能を１としたときのクロックサイクル数の比

3. 結論

マルチコアプロセッサのキャッシュ構造においては、キャッシュの一貫性を保つ処理の為、性能が大幅に低下する場合がある。この問題は既に文献[1] [2] [3]で指摘してきた。これに対してプロセッサが進化するにつれて、このキャッシュのコヒーレンシ（一貫性）を保つためのオーバーヘッドは改善されているようにも見える。しかしながら今回の報告のように各コアでエリアは異なっていてもブロック(64B)を共有する場合の性能低下は著しい。
このためマルチコアを用いたアプリケーションプログラムを作成する場合、書き込みブロック(64B)をコア間で共有しないように､十分注意してプログラミングを行わなければならない。

4. 今後の課題

今後のプロセッサの進化は、１つのプロセッサに搭載するコアを増やしていく方向で進化していく可能性が高い。コアがさらに増えた場合、キャッシュの一貫性を保つ処理による性能低下はさらに大きくなるのではないかと思われる。今後も引き続きプロセッサ特性を評価していきたいと思う。

参考文献

[1] 橋本壮広，河辺峻：Intel 系デュアルコアプロセッサの性能解析手法，明星大学情報学部紀要，15， pp. 71-92（2007）
[2] 河辺峻、佐藤祐治：マルチコアプロセッサの性能評価分析，明星大学情報学部紀要，17・18， pp. 41-46（2010）
[3] 河辺峻、森下真次：４コアプロセッサの性能評価分析，明星大学情報学部紀要，19， pp. 15-20（2011）