Intel Xeon プロセッサにおける Cache Coherency 時間の性能測定方法

Performance Measurement Method of Cache Coherency Effects on an Intel Xeon Processor System

河辺　峻　田口成美　古谷英祐

KAWABE Shun, TAGUCHI Akiyoshi, FURUYA Eisuke

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は共有せず、L3 キャッシュは同一プロセッサチップの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）を保つための時間を atomic_inc 関数を用いて測定する方法を考案し、実測プログラムを作成した。その結果、Intel Xeon プロセッサ(2.4GHz)システムではキャッシュのコヒーレンシ（一貫性）を保つための時間は、同一チップ内では平均 27.40ns に対して、同一ボード内（異なるチップ間）では平均 113.71ns にもなることが分かった。

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、マルチ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。本研究ではキャッシュのコヒーレンシ（一貫性）を保つための時間を、Linux の C 言語の atomic_inc 関数を用いて測定する方法を考案し、実測プログラムを作成して測定を行い結果を分析する。

２．プロセッサの性能測定方法と結果の考察

２．１　評価に用いたプロセッサ

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(Nehalem 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１つのボード上に２つ搭載されており、QPI(Quick Path Interconnect)でプロセッサチップ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。

今回のプログラムでは affinity 機能を用いて使用するコアを指定した。また OS は Linux の Fedora14(64b)を使用した。

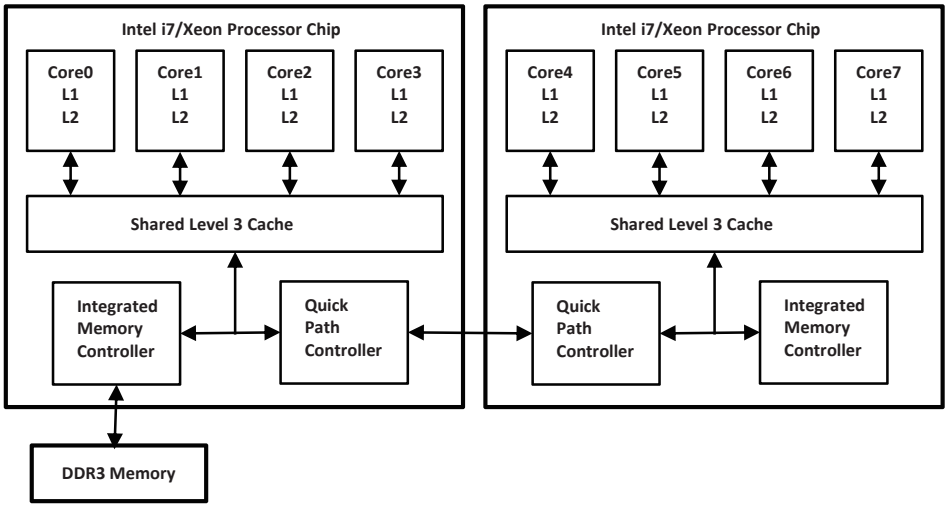

図１　評価に用いたプロセッサ構成図

2.2 性能測定方法

2.2.1 Linux カーネルの atomic_inc 動作

Linux カーネルには、atomic 動作というのがある。マルチスレッドで動作する場合に、x86 アーキテクチャでは共通にアクセスする変数にハード的に lock をかけて更新を行う。atomic_inc 関数は指定した変数に lock をかけて変数の値を＋１する機能である。

```
#define LOCK "lock ; "
typedef struct {volatile int counter; } atomic_t;
atomic_t abc;

static __inline__ void atomic_inc(atomic_t *v) {
  __asm__ __volatile__(
            LOCK "incl %0"
            :"=m" (v->counter)
```

```
                    :"m" (v->counter));
        }
```

としておいて、

```
        atomic_inc(&abc.counter);
```

と書くと変数 abc.counter の値が＋１される。

各々のコアでメモリ上の共通変数に atomic_inc を動作させると、各コアで交互に排他的にメモリ上の共通変数が＋１される。さらにその時、各コアにある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必ず動作する。したがってマルチスレッドプログラミングを用いて、各コアで atomic_inc を交互に動作させると、キャッシュのコヒーレンシ（一貫性）の時間が測定可能になる。

2.2.2　atomic_inc の性能測定方法とプログラム

図１に性能測定構成図を示す。プロセッサチップが２ケあり、１つのチップに４コア（Hyper Thread 機能を使用すると８スレッド並列可）あり、２チップで最大８コア（HT 機能で 16 スレッド並列可）である。

・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)
```

・マルチスレッドの各スレッドの測定部分のプログラム

gettimeofday 関数を持ちいて１０００万回（４コアの場合で約数百ms かかる）のループを測定する。

```
gettimeofday(&st,NULL);
for(a=0;a<tr;a++)
    {
            atomic_inc(&abc.counter);
    }
gettimeofday(&et,NULL);
```

(1) 同一チップ内の atomic_inc の動作

図２において core0(C0)から atomic_inc を行う。まず Memory にあるデータ a を L1 まで持ってくる。この時、L2、L3 の対応エリアも a の値になる。Intel i7/Xeon プロセッサの cache 制御は”writeback”方式であるので、atomic_inc により値が更新されるのはこの場合 L1 のみで、L1 の値が a+1 に更新される。次に core1(C1) から atomic_inc を行う。この場合、真の値は C0 の L1 にあるので、まず C0 の L1 の内容の値 a+1 を C0 の L2 および L3 に書き込む。

この動作の後、C1 は L3 から a+1 の値を L1 まで持って来て値を a+2 に更新する。同じようにして次は core0(C0) から atomic_inc を行う。この場合、真の値は C1 の L1 にあるので、まず C1 の L1 の内容の値 a+1 を C1 の L2 および L3 に書き込む。
この動作の後、C0 は L3 から a+2 の値を L1 まで持って来て値を a+3 に更新する。

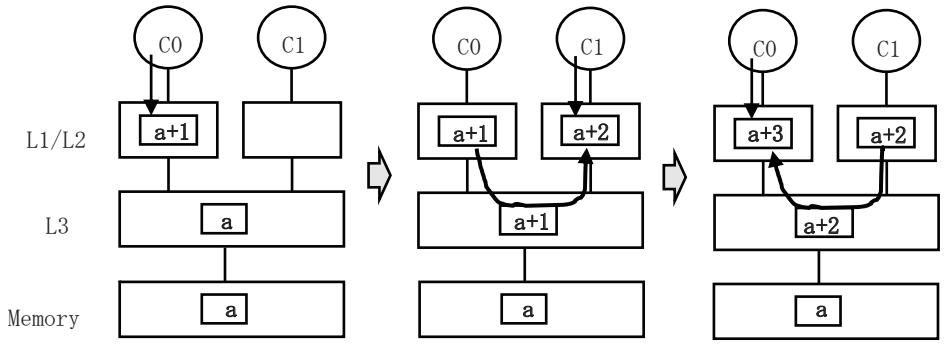

図２　同一チップ内の atomic_inc の動作

(2) 同一ボード内（異なるチップ間）の atomic_inc の動作

図３において core0(C0)から atomic_inc を行う。まず Memory にあるデータ a を L1 まで持ってくる。この時、L2、L3 の対応エリアも a の値になる。Intel i7/Xeon プロセッサの cache 制御は”writeback”方式であるので、atomic_inc により値が更新されるのはこの場合も L1 のみで、L1 の値が a+1 に更新される。次に core4(C4) から atomic_inc を行う。この場合、真の値は C0 の L1 にあるので、まず C0 の L1 の内容の値 a+1 を C0 の L2 および C0 の L3 に書き込む。この動作の後、C4 は C0 の L3 から a+1 の値を QPI 経由で C4 の L3 から L1 まで持って来て値を a+2 に更新する。同じようにして次は core0(C0) から atomic_inc を行う。この場合、真の値は C4 の L1 にあるので、まず C4 の L1 の内容の値 a+1 を C4 の L2 および L3 に書き込む。この動作の後、C0 は C4 の L3 から a+2 の値を QPI 経由で C0 の L3 から L1 まで持って来て値を a+3 に更新する。

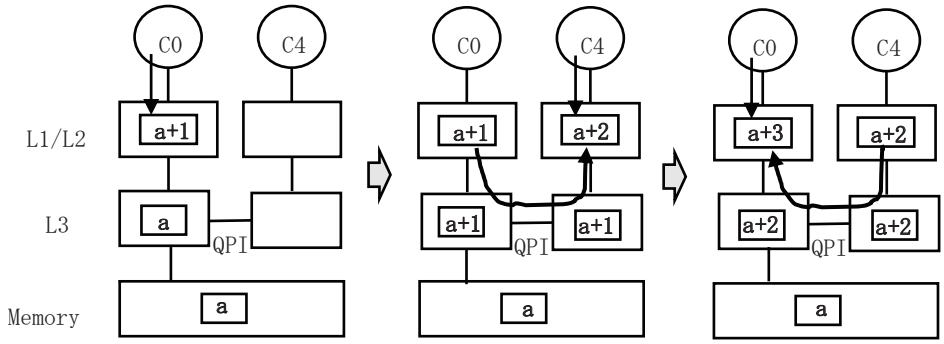

図３　同一ボード内（異なるチップ間）の atomic_inc の動作

２．２　性能実測結果と考察

性能測定は次の４つのケースに分けて行った。

(1) atomic_inc の単体性能　［測定 1］
(2) キャッシュコヒーレンシ動作を伴わない atomic_inc の性能　［測定２］
(3) 同一チップ内の atomic_inc の性能　［測定３］
(4) 同一ボード内（異なるチップ間）の atomic_inc の性能　［測定４］

■測定結果

測定１：core0 にて atomic_inc を実行させて性能を測定する。結果は 10.52ns となった。
ちなみにハード的に lock をかけずに実行すると、結果は 3.97ns であった。

測定２：core0 の同一コア内で HT(Hyper Threading)機能を利用して、２つの atomic_inc を実行させて性能を測定する。この場合キャッシュコヒーレンシ動作は伴わない。結果は平均して 11.13ns となった。

測定３：同一チップ内の atomic_inc の動作として、[C0,C1]、[C0,C2]、[C0,C3]のペアで２つの atomic_inc を実行させて性能を測定する。結果は平均して 38.53ns となった。
測定２で得られた値 11.13ns をこれから引いた値 38.53-11.13=27.40ns が同一チップ内の cache coherency 時間と見なすことができる。

測定４：同一ボード内（異なるチップ間）の atomic_inc の動作として[C0,C4]、[C0,C5]、[C0,C6]、[C0,C7]のペアで２つの atomic_inc を実行させて性能を測定する。結果は平均して 124.84ns となった。
測定２で得られた値 11.13ns をこれから引いた値 124.84-11.13=113.71ns が同一ボード内（異なるチップ間）の cache coherency 時間と見なすことができる。

■考察

CPI(Clock cycle Per Instruction)に与える影響

cache coherency 時間は、同一チップ内では 27.40ns(65.76cyc)、異なるチップ間では 113.71ns(272.90cyc)となる。１命令において基本 CPI を 2.0 としたとき、cache coherency の命令あたりの発生頻度を横軸にとり、縦軸に CPI をとったグラフを図４に示す。
これから分かるように、同一チップ内で発生する cache coherency が CPI に与える影響は比較的軽微である。

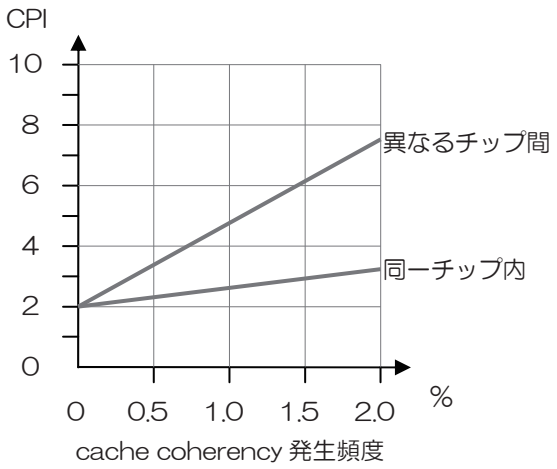

図４　CPI に与える影響

しかし異なるチップ間で発生する cache Coherency が CPI に与える影響は非常に大きい。これは異なるチップ間で発生する cache Coherency 時間が、同一チップ内で発生する cache Coherency 時間の 4.15 倍にもなっていることによる。

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。今回 Linux の C 言語の atomic_inc 関数を用いてこのキャッシュのコヒーレンシ(一貫性)を保つための時間を測定する方法を考案した。Intel Xeon プロセッサ E5620(Nehalem Westmere-EP)の実測結果では、cache coherency 時間は、同一チップ内では 27.40ns(65.76cyc)、異なるチップ間では 113.71ns(272.90cyc)となった。

CPI(Clock cycle Per Instruction)に与える影響では、同一チップ内で発生する cache coherency が CPI に与える影響は比較的軽微であるものの、異なるチップ間で発生する cache Coherency 時間は同一チップ内で発生する cache Coherency 時間の 4.15 倍にもなっているため CPI に与える影響は非常に大きいことが分かった。

このためマルチスレッドを用いたアプリケーションプログラムを作成する場合、このような性能低下があることを十分考慮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

今回は１枚のボードに２つのプロセッサチップ（８コア）が搭載されたケースについて測定した。今後のプロセッサの進化は、複数のプロセッサチップに搭載された複数のコアがメモリを共有する方向で進化していく可能性が高い。例えば１つのプロセッサチップ（８コア）が搭載されたケースで６４コアの大規模な構成であれば、４個のボードにわたってキャッシュの一貫性を保つ処理が行われる。このケースによる cache Coherency 時間はさらに大きくなるのではないかと思われる。今後も引き続きこのようなプロセッサ特性を評価していきたいと思う。

参考文献

[1] Daniel Molka, et.al, ” Memory Performance and Cache Coherency Effects on an Nehalem Multiprocessor System,” 18th ICPACT, pp.261-270, 2009

[2] 河辺峻:８コアプロセッサの性能評価分析,明星大学情報学部紀要,20, pp.15-20(2012)