メモリプロファイリングツール hardmeter 設計と実装

よしおかひろたか (hyoshiok@miraclelinux.com)

ミラクル・リナックス株式会社
http://www.miraclelinux.com/

YLUG (Yokohama Linux Users Group)
http://www.ylug.jp/

目次

- hardmeterに関するFAQ
- パフォーマンスチューニングはなぜ必要か？
- なぜメモリアクセスが重要なのか？
- メモリプロファイリングツール(hardmeter)とは何か？何でないか？
- 動作原理
- 従来方法とその問題点
- hardmeterのインストール
- hardmeter利用した性能分析
- カーネルプロファイリングツール

hardmeterに関するFAQ

- hardmeterとは何か？
- 何がハックなのか？
- VTuneとどう違うのか？

パフォーマンスチューニングはなぜ必要か？

- ハードウェアは安くなっている
- ソフトウェアのチューニングに時間をかけるのは無駄？

- ソフトウェアのチューニングによってＸ％速くなったとする。
  一方Ｘ％速いハードウェアはいくら高いか？

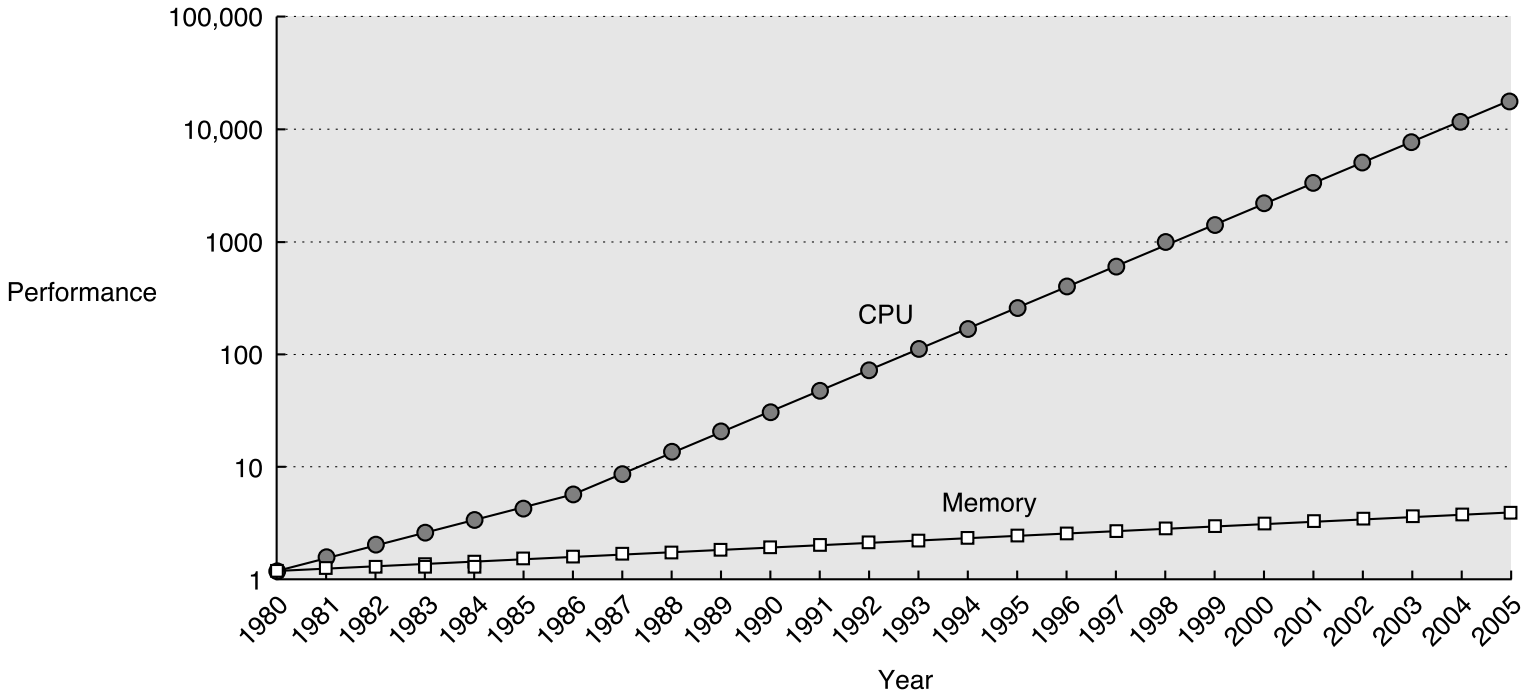
人々は常に速いハードウェアを求めている。同じハードウェアならより速い
ソフトウェアを求めている。

コンピュータは速くなっているのか？

- 主記憶4MB、ハードディスク40MBバイト、swapoutのコスト？
- 主記憶4GB、ハードディスク400GBバイト、swapoutのコスト？

OSは速くなっているか？

なぜ、メモリアクセスが重要なのか



Intel Xeon 2GHz/Main Memory 1GBでの実測値

メモリ階層	アクセスコスト
L1	1 nsec
L2	9 nsec
Main memory	196 nsec

メインメモリへのアクセスは一次キャッシュのアクセスに比べ２桁遅い。
数百クロックCPUは待つことになる。

キャッシュミスをプログラマに知らせるツールが必要。

メモリプロファイリングツール(hardmeter)とは何か？

- キャッシュミスをプログラマに知らせるツールである。
- キャッシュミスの場所、原因を簡単に特定でき、性能向上のヒントを提供する。(キャッシュミスの回数を数えるだけではない)
- Pentium 4/Intel Xeon用メモリイベントを精密に測定する
- オープンソース(GPL)

hardmeter とは何でないか？

- コンパイラではない
- デバッガではない
- イベントトレーサではない
- 商用製品ではない

従来のプロファイラ

イベントが発生した時に、割り込みハンドラーが CPU の状態を保存する。
(通常は PC)

問題点

- 割り込みハンドラを起動するレイテンシ (数 1000 クロック) がおおきい
- 命令キャッシュを汚染する
- 割り込みが精密ではない

最近のプロセッサの特徴：

- 長いパイプライン。(パイプラインのいろいろなステージでイベントが発生する)
- スーパースカラ。(同時に複数の命令を実行する)
- 投機的な実行。(後にキャンセルされる命令もメモリアクセスをする)
- out of order実行。(実行の順番がソースコードの並び順でない)

従来のプロセッサでは同時に複数の命令が実行しているので、どの命令がイベントを発生させたか (例えばキャッシュミスを発生させたか) を特定できない。

取得したPCの値は実際のイベントを発生させたPCより先を行く。(Pentium 4では65命令以上隔たって分布)

動作原理

Pentium 4のhardware monitoring facilitiesを利用している。PMC (Performance Monitoring Counters) は各種ハードウェアイベントを計測する。

- PMC – Performance Monitoring Counters
- PEBS – Precise Event Based Sampling
  イベントが発生した時点でのハードウェア状態をマイクロコードが自動的にカーネル空間に退避する。(ソフトウェアを利用しない)
  プログラムのどの場所(論理アドレス)で、どのメモリにアクセスしているかが精密にわかる。

PEBSを利用したことがhardmeterのハックである。:-)

hardmeterの実装

- Linux Kernel Patch
- ユティリティ(ebsコマンド)
- ユーザー用API (Application Programming Interface)

hardmeterのインストール

http://sourceforge.jp/projects/hardmeter/

- ソースの入手
  - hardmeterソースコード
  - perfctr ソースコード
  - Linux Kernel ソースコード
- perfctrにhardmeterのパッチをあてる
- Linux Kernelにperfctrのパッチをあてる
- kernelをビルドする

hardmeterを利用した性能分析

- hardmeterを利用してプロファイリング (データ収集)
- プロファイリングデータの分析
  - トップ 10 分析
  - ソースコード 分析
  - メモリアクセス分析

イベントサンプリング(データ収集)

- ebsコマンドを利用する
- APIを利用してプログラムから呼びだす

hardmeter利用例 (ebsコマンド)

- ユーザーモード、カーネルモード
- 取得すべきイベントタイプ (キャッシュミス、TLBミスなど)
- サンプリング間隔
- 最大取得データ数

コマンドシンタックス

```
Usage: ./ebs (-u | -k) [-o OUTFILE] [-i INTERVAL] [-c COUNT] -t TYPE EXE_OR_PID
  options
    -u           - sample user-mode events
    -k           - sample kernel-mode events
    -o OUTFILE   - store sampled data to file
    -i INTERVAL  - sampling interval(default: 10000)
    -c COUNT     - max sampling count(default: 2000)
    -t TYPE      - event type to sample
    -m NAME,NAME... - event masks
  help options
    -h           - show event types
    -h TYPE      - show event masks
```

サポートするイベントタイプ

```
imprecise at-retirement event:
      instr_retired   - instruction retired
      uop_retired     - uops retired
precise front-end event:
      memory_loads    - memory loads
      memory_stores   - memory stores
      memory_moves    - memory loads and stores
precise execution event:
      packed_sp_retired - packed single-precision uop retired
      packed_dp_retired - packed double-precision uop retired
      scaler_sp_retired - scaler single-precision uop retired
      scaler_dp_retired - scaler double-precision uop retired
      64bit_mmx_retired - 64bit SIMD integer uop retired
      128bit_mmx_retired - 128bit SIMD integer uop retired
      x87_fp_retired  - floating point instruction retired
      x87_simd_memory_moves_retired - x87/SIMD store/moves/load uop retired
precise replay event:
      l1_cache_miss   - 1st level cache load miss
      l2_cache_miss   - 2nd level cache load miss
      dtlb_load_miss  - DTLB load miss
      dtlb_stor_miss  - DTLB store miss
      dtlb_all_miss   - DTLB load and store miss
      mob_load_replay_retired - MOB(memory order buffer) causes load replay
      split_load_retired - replayed events at the load port.
```

hardmeter使用例 (ebsコマンド)

```
$ /usr/src/hardmeter-030603/src/ebs -k -t l1_cache_miss ps
#eflags  liner_ip   eax      ebx      ecx      edx      esi      edi      ebp      esp
  PID TTY          TIME CMD
  982 pts/1    00:00:00 bash
 1005 pts/1    00:00:05 xemacs
 1182 pts/1    00:00:00 ebs
 1183 pts/1    00:00:00 ps
00000286 c016d554 c3974200 00001002 c3974200 f7b61900 ede27600 c3973480 ede27600 ee747e98
00000282 c016d554 c3974880 0000100f c3974880 f7b61900 ede27600 c3973480 ede27600 ee747e74
00000282 c016d554 c3974680 0000100b c3974680 f7b61900 ede27661 c397114d ede27600 ee747e74
00000246 c016d5ba 00000003 000010c6 00000003 f7b61900 ede27660 c3969b4c ede27600 ee747e98
00000217 c026064b eef1f063 eef1f064 00000000 c0279726 eef1f064 eef1f000 ffffffff ee747eac
00000282 c016ead0 00000014 f7946580 00000073 f77ee280 f79465a0 00000010 eef1f074 ee747ec8
00000286 c0147f7e ee746000 c374f000 00000ff2 0000000e c374f000 4002ed40 bffff538 ee747f7c
00000282 c015f650 c0332780 ede78680 00000000 ede78690 c0332780 c3975800 ede27680 ee747f50
00000206 c016d55c c3974400 00001005 c3974400 f7b61900 ede27661 c397464d ede27600 ee747e98
00000282 c0147e0e c03329e0 eea0e980 00000000 00000000 ede78680 ffffffe9 c3977300 ee747f58
00000282 c016d554 c3974680 0000100b c3974680 f7b61900 ede27661 c3974dcd ede27600 ee747e98
00000282 c016ead3 00000008 f734e980 406af000 f7117480 f734e9a0 00000168 ee82e08d ee747ec8
00000282 c016e1a0 00001000 f734ec80 f734eca0 f6ee7d80 f6dcc000 0191a000 416b7000 ee747e74
00000202 c0154aed 00000098 ee747f84 ee746000 f7fe4280 c374f000 4002ed60 ee747f84 ee747f20
00000246 c016eaf2 00001a30 f6dfd180 00000000 f6c81400 f6dfd1a0 000004d8 eeaf1096 ee747ec8
00000246 c016d5ba 00000003 00001018 00000003 f7b61900 ede27660 c3974dcc ede27600 ee747e98
00000206 c016d569 c3974700 0000100b c3974700 f7b61900 ede27661 c3974a4d ede27600 ee747e74
```

```
00000246 c016eaf2 00002dac f6dfda80 00000000 f1c13d80 f6dfdaa0 0000214c ee8f2093 ee747ec8
00000297 c016d569 c3924500 00001123 c3924500 f7b61900 ede27661 f7b6194d ede27600 ee747e98
00000286 c016e1a0 00001000 f6dfda80 f6dfdaa0 f1c13100 ee2c6000 01b28000 416bd000 ee747e74
00000246 c0145ec0 00000000 ffffffff f7090e80 00000000 c1df2ae8 f7090e80 c2100100 ee747ee8
#
# start time : Fri Sep 26 01:12:03 2003
# user       : no
# kernel     : yes
# interval   : 10000
# count      : 21
# tsc        : 24708464
#
# event name : l1_cache_miss
# description: 1st level cache load miss
# event mask : + nbogus
#              - bogus
#
```

hardmeter利用例 (2.4.20)

kernel build (make -j 100 bzImage) 時のL1 cache missの頻度

```
$ /usr/sbin/readprofile -r ;\
/usr/src/hardmeter-030603/src/ebs -k -t l1_cache_miss -c 50000 -o l1.001&
time make -j 100 bzImage >/dev/null;\
/usr/sbin/readprofile -m /boot/System.map > rp.001;\
kill -2 ‘ps auxw|grep ebs|grep -v grep|awk ’{print $2}’‘
$ awk ’{print $2}’ l1.001|/usr/src/hardmeter-030603/doc/eip2r\
|sort -nr |uniq -c|sort -nr|head
```

トップ10分析

イベント発生のトップ10を求める。論理アドレス(liner_ip)でソートする

```
頻度 アドレス ルーチン名
2500 c0132298 file_read_actor
1015 c011a843 schedule
 959 c01418e8 page_referenced
 905 c012e9c6 __constant_memcpy
 783 c0141d90 page_remove_rmap
 486 c013a34c scan_active_list
 324 c01090a5 ret_from_sys_call
 288 c01418fe page_referenced
 261 c0138eb3 lru_cache_add
 249 c012ec93 vm_account
```

L1 cache missを多発している場所を容易に計測できる。

ソースコード分析

1015 c011a843 schedule

objdump -S (逆アセンブル)

```
                p = list_entry(tmp, struct task_struct, run_list);
c011a840:       8d 4a c4                lea    0xffffffc4(%edx),%ecx
c011a843:       8b 51 28                mov    0x28(%ecx),%edx
```

```
/*
 * Default process to select..
 */
next = idle_task(this_cpu);
c = -1000;
list_for_each(tmp, &runqueue_head) {
        p = list_entry(tmp, struct task_struct, run_list);
        if (can_schedule(p, this_cpu)) {
                int weight = goodness(p, this_cpu, prev->active_mm);
                if (weight > c)
                        c = weight, next = p;
        }
}
```

メモリアクセス分析

キャッシュミスを発生した場所(c011a843)のレジスタ値を分析。ecxに注目する。(ソースコードより)

```
$ head -1 l1noo1.001
#eflags  liner_ip   eax      ebx      ecx      edx      esi      edi      ebp      esp
$ grep  c011a843 l1noo1.001|head
00003002 c011a843 ea29403c ea29403c ea712000 ea71203c ea344000 00000017 f749dee4 f749debc
00200002 c011a843 eaec803c eaec803c ea740000 ea74003c eb058000 00000016 f1ba1f2c f1ba1f04
00200002 c011a843 e9e6803c e9e6803c e9e8e000 e9e8e03c eabb6000 00000017 ea3abf34 ea3abf0c
00200002 c011a843 ea66a03c ea66a03c ea584000 ea58403c eb058000 00000016 f1ba1f2c f1ba1f04
00200002 c011a843 e9ff803c e9ff803c f2078000 f207803c eb058000 00000016 eaf4bf98 eaf4bf70
00200002 c011a843 e95ac03c e95ac03c e95b2000 e95b203c ea154000 00000015 eb755f2c eb755f04
00200002 c011a843 ea57a03c ea57a03c e9ff8000 e9ff803c ea77c000 00000000 eb755f2c eb755f04
00200002 c011a843 e953803c e953803c e9542000 e954203c ea154000 00000015 eb755f2c eb755f04
00200002 c011a843 f1ba003c f1ba003c e9382000 e938203c ea154000 00000015 eb197f98 eb197f70
00200002 c011a843 e9e5403c e9e5403c e9bce000 e9bce03c ea77c000 00000000 e9065f98 e9065f70
```

メモリパターンがxxxxx000となっていることに注意。下位13ビットが等しい。(8KBでalign)

キャッシュコンフリクトが発生している。

hardmeter利用例 (O(1)スケジューラ)

kernel build (make -j 100 bzImage) 時のL1 cache missの頻度

O(1)スケジューラ

頻度	アドレス	ルーチン名
2463	c0136438	file_read_actor
886	c0132af6	__constant_memcpy
660	c0145ec0	page_remove_rmap
349	c014a7e7	invalidate_bdev
266	c0145d90	page_add_rmap
260	c0140354	rmqueue
258	c010958d	ret_from_sys_call
237	c013230f	do_anonymous_page
227	c013d053	lru_cache_add
217	c0133b63	find_vma

従来のスケジューラ

頻度	アドレス	ルーチン名
2500	c0132298	file_read_actor
1015	c011a843	schedule
959	c01418e8	page_referenced
905	c012e9c6	__constant_memcpy
783	c0141d90	page_remove_rmap
486	c013a34c	scan_active_list
324	c01090a5	ret_from_sys_call
288	c01418fe	page_referenced
261	c0138eb3	lru_cache_add
249	c012ec93	vm_account

schedule(スケジューラ)のボトルネックが解消している。

ソースコード分析

```
int file_read_actor(read_descriptor_t * desc, struct page *page, unsigned long offset, u
{
        char *kaddr;
        unsigned long left, count = desc->count;

        if (size > count)
                size = count;

        kaddr = kmap(page);
        left = __copy_to_user(desc->buf, kaddr + offset, size);
        kunmap(page);

        if (left) {
                size -= left;
                desc->error = -EFAULT;
        }
        desc->count = count - size;
        desc->written += size;
        desc->buf += size;
        return size;
}
```

ソースコード分析

```
static inline unsigned long
__generic_copy_to_user_nocheck(void *to, const void *from, unsigned long n)
{
c0136423:       89 c6                   mov    %eax,%esi
        __copy_user(to,from,n);
c0136425:       89 d9                   mov    %ebx,%ecx
c0136427:       8b 44 24 28             mov    0x28(%esp,1),%eax
c013642b:       8b 7d 08                mov    0x8(%ebp),%edi
c013642e:       c1 e9 02                shr    $0x2,%ecx
c0136431:       01 c6                   add    %eax,%esi
c0136433:       89 d8                   mov    %ebx,%eax
c0136435:       83 e0 03                and    $0x3,%eax
c0136438:       f3 a5                   repz movsl %ds:(%esi),%es:(%edi)
↑ここでキャッシュミスが多発している
c013643a:       89 c1                   mov    %eax,%ecx
c013643c:       f3 a4                   repz movsb %ds:(%esi),%es:(%edi)
}
```

メモリアクセス分析

```
c0136438:       f3 a5                    repz movsl %ds:(%esi),%es:(%edi)
```

↑ここでキャッシュミスが多発している
(%ds:(%esi)から%es:(%edi)へ、ecx回,movslする)

```
$ head -1 l1.001
#eflags  liner_ip   eax      ebx      ecx      edx      esi      edi      ebp      esp
$ grep c0136438 l1.001 |head
00200246 c0136438 00000000 00001000 00000400 c1b34120 f3372000 40015000 f45fff74 f45ffef
00200246 c0136438 00000000 00001000 00000400 c1b22290 f2e54000 40015000 f45fff74 f45ffef
00200246 c0136438 00000000 00000400 000000f8 c03cdce0 fe046020 bfffe1a4 f4583f74 f4583ef
00200246 c0136438 00000000 00000400 000000f8 c03cdce0 fe046020 bfffe034 f4583f74 f4583ef
00200246 c0136438 00000000 00000400 000000f8 c03cdce0 fe028020 bfffda24 f4583f74 f4583ef
00200246 c0136438 00000000 00000400 000000f8 c03cdce0 fe046020 bfffd8b4 f4583f74 f4583ef
00200246 c0136438 00000000 00000400 000000f8 c03cdce0 fe046020 bfffd7b4 f4583f74 f4583ef
00200246 c0136438 00000000 00001000 00000070 c1b68ed0 f428ce40 083a8e08 f38a7f74 f38a7ef
00200246 c0136438 00000000 000009bc 0000026f c1b66770 f41d8000 083b9130 f38a7f74 f38a7ef
00200246 c0136438 00000000 00000400 00000100 c03cdce0 fe028000 bfffe324 f38a9f74 f38a9ef
```

ハードウェアプリフェッチの効果の測定

キャッシュミスのレイテンシーを隠蔽するテクニックとしてプリフェッチ(あらかじめデータをロードしておく)が知られている。

Pentium 4では、ハードウェアプリフェッチが実装された。

プリフェッチなし	プリフェッチあり	ルーチン名	キャッシュミスの比
10163	9812	file_read_actor	103.58
4442	4000	page_referenced	111.05
3327	3023	page_remove_rmap	110.06
3011	3306	__constant_memcpy	91.08
2376	2294	chedule	103.57
1614	1740	scan_active_list	92.76
1583	1395	page_referenced	113.48
1150	1183	ret_from_sys_call	97.21
1024	1004	__alloc_pages	101.99
980	981	page_referenced	99.9

カーネルプロファイリングツール

- readprofile
- oprofile
- kernprof
- lockmeter
- VTune for Linux

readprofileでのプロファイリング結果の比較

- boot時にprofile=数字と指定
- /usr/sbin/readprofileでデータを読む
- タイマ割り込みによるサンプリング (割り込み禁止時のデータ取得ができない)
- サンプリングの粒度が荒い (ルーチンレベル)
- メモリプロファイリング機能がない

readprofileの例

```
 1340 total                                      0.0009
  335 __constant_c_and_count_memset              2.0938
  199 do_page_fault                              0.1513
   67 page_referenced                            0.0974
   49 scan_active_list                           0.3828
   40 rmqueue                                    0.0510
   30 do_anonymous_page                          0.0493
   27 kmem_cache_free                            0.5625
   27 file_read_actor                            0.1205
   25 __kmem_cache_alloc                         0.0977
```

readprofileでのプロファイリング結果の比較

頻度	ルーチン名	頻度	ルーチン名
335	__constant_c_and_count_memset	2500	file_read_actor
199	do_page_fault	1015	schedule
67	page_referenced	959	page_referenced
49	scan_active_list	905	__constant_memcpy
40	rmqueue	783	page_remove_rmap
30	do_anonymous_page	486	scan_active_list
27	kmem_cache_free	324	ret_from_sys_call
27	file_read_actor	288	page_referenced
25	__kmem_cache_alloc	261	lru_cache_add

hardmeterの結果と異なる。プロファイリングの粒度がルーチン単位なので大きなルーチンほどサンプル数が多くなる。タイマ割り込みを利用したサンプリングなので、割り込み禁止時のサンプリングができない。

おおまかな目安程度の精度である。

oprofile

- 2.5.43から導入
- time/event base sampling
- プロファイリングの粒度が荒い(ルーチンレベル)
- メモリボトルネックの場所を特定できない
- http://oprofile.sourceforge,net/

kernprof

- SGIによる実装
- タイマ割り込みによるサンプリング
- メモリプロファイリングの機能がない
- http://oss.sgi.com/projects/kernprof/

lockmeter

- SGIによる実装
- ロック競合を測定する
- メモリプロファイリング機能はない
- http://oss.sgi.com/projects/lockmeter/

VTune for Linux

- Intelによる商用製品。機能は豊富。
- 各種ハードウェアイベントのプロファイリング機能

サポートされないカーネルのバージョン用にDriver Kitというのがある。

hardmeterの評価

ebsコマンドの実行オーバヘッド

	ebsあり	ebsなし	オーバヘッド
real	4m05.043s	4m04.343s	0.29％
user	3m46.703s	3m46.350s	0.16％
sys	0m12.883s	0m12.760s	0.96％

hardmeterの評価(続き)

従来のツールでは、(特別なハードウェアを利用しないかぎり)

- メモリアクセスのホットスポット(頻繁にメモリアクセスをしている場所)を特定することができなかった。
- どこのメモリにアクセスしているかを特定できなかった。
- キャシュコンフリクトなどの原因特定はできなかった。

シミュレーションによるメモリプロファイリングは実行オーバヘッドが非常に大きいため(数十倍から場合によっては数百倍以上)、実時間でのデータ収集は不可能であった。

hardmeterで初めて可能になった。

hardmeterの今後

- モジュールによる実装(カーネルパッチを不要とする)
- GUI(操作の視覚化、結果のビジュアル化)
- 性能問題分析と解決策の提示
- ユーザー層の拡大
- 商用化？

おまけ：YLUG カーネル読書会

hardmeterの着想のヒントは横浜Linux Users Group (YLUG) のメーリングリストでの議論および不定期に開催しているカーネル読書会で得た。実装に関しては、YLUGの久保氏の協力を得た。

http://www.ylug.jp/

まとめ

- 従来のツールでは不可能だった機能。(PEBS – 精密なイベントベースサンプリング)
  - メモリアクセスのホットスポットを発見する
  - どこのメモリにアクセスしているかを特定する
  - 原因究明のヒントを提供する
  - 実行オーバヘッドはほとんどない (実時間の計測が可能)
- オープンソース
- 特別なハードウェアを必要としない

hardmeterのリソース

コード、文書、メーリングリスト、掲示板

http://sourceforge.jp/projects/hardmeter

オープンソース(GPL)で公開

誰でも自由に、利用、変更、再配布可能

謝辞：hardmeterの実装にあたっては、平成14年度未踏ソフトウェア創造事業(東京大学教授喜連川優プロジェクトマネージャ)からの支援をうけた。