Multithreaded Programming for Mere Mortals

Emery Berger

University of Massachusetts, Amherst
In the Beginning...
There was the Core.
And it was Good.
It was Very Good. Until...
...the Apocalypse.
And the Speed was no Moore.
More is Moore (?)…
color = □; row = 0; // globals
void nextStripe()
{
    for (c = 0; c < Width; c++) {
        drawBox (c,row,color);
    }
    color = (color == □)? □ : □;
    row++;
}
for (n = 0; n < 9; n++) {
    spawn nextStripe();
}
sync;
FALSE SHARING!
Invalidate
Thread 1

Invalidate

Thread 2

20x slower!
me = 1;
you = 1; // globals

me = new Foo;
you = new Bar; // heap

class X {
    int me;
    int you;
}; // fields

arr[me] = 12;
arr[you] = 13; // array indices
me = 1;
you = 1; // globals

me = new Foo;
you = new Bar; // heap

class X {
    int me;
    int you;
}; // fields

arr[me] = 12;
arr[you] = 13; // array indices
```java
me = 1;
you = 1; // globals

me = new Foo;
you = new Bar; // heap

class X {
    int me;
    int you;
}; // fields

arr[me] = 12;
arr[you] = 13; // array indices
```
me = 1;
you = 1; // globals

me = new Foo;
you = new Bar; // heap

class X {
    int me;
    int you;
}; // fields

arr[me] = 12;
arr[you] = 13; // array indices
me = 1;
you = 1; // globals

me = new Foo;
you = new Bar; // heap

class X {
    int me;
    int you;
}; // fields

arr[me] = 12;
arr[you] = 13; // array indices
### Basic Data Access Profiling (2010-07-12-09-33-05)

<table>
<thead>
<tr>
<th>Cacheline Address / Offset / Thread...</th>
<th>Coll...Refs</th>
<th>LL...s</th>
<th>A...y</th>
<th>T...y</th>
<th>Contention</th>
<th>INST_R... (refs)</th>
<th>M...S</th>
<th>MEM_LOAD.....L2_MISS</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xef35f340</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>45</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>Offsets: 3 Threads</td>
</tr>
<tr>
<td>0xed55c340</td>
<td>15</td>
<td>0</td>
<td>3</td>
<td>45</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>Offsets: 3 Threads</td>
</tr>
<tr>
<td>0x0804f080</td>
<td>12</td>
<td>0</td>
<td>4</td>
<td>99</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>0</td>
<td>Offsets: 6 Threads</td>
</tr>
<tr>
<td>Offset:0x08(8)</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>Threads: 1</td>
</tr>
<tr>
<td>Thread:00004598(0011)</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>Functions: 1</td>
</tr>
<tr>
<td>wordcount_reduce</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Offset:0x18(24)</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Threads: 1</td>
</tr>
<tr>
<td>Thread:0000459c(0014)</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Functions: 1</td>
</tr>
<tr>
<td>wordcount_reduce</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Offset:0x14(20)</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Threads: 1</td>
</tr>
<tr>
<td>Offset:0x0c(12)</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Threads: 1</td>
</tr>
<tr>
<td>Offset:0x1c(28)</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Threads: 1</td>
</tr>
<tr>
<td>Offset:0x10(16)</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Threads: 1</td>
</tr>
</tbody>
</table>

+ 850 lines...
No false positives

Object has 13767 interleaving writes (starts at 0xd5c8e160, length 32).
Allocation call stack:
  0: wordcount.c: 136
  1: wordcount.c: 444

Precise object identification
\[ t1 = \text{spawn } f(x); \\
\quad t2 = \text{spawn } g(y); \\
\quad \text{sync}; \]
\[
t_1 = \text{spawn } f(x); \\
t_2 = \text{spawn } g(y); \\
\text{sync}; \\
\]

\[
\text{if (!fork())} \\
\quad f(x); \\
\text{if (!fork())} \\
\quad g(y); \\
\]

GRACE [OOPSLA 2009]
Isolation

shared address space

disjoint address spaces
Performance: Processes vs. Threads

![Graph showing the normalized execution time for threads and processes with increasing thread execution time. The graph indicates that processes have a consistently lower normalized execution time compared to threads.]
Performance: Processes vs. Threads
Performance: Processes vs. Threads

![Graph showing normalized execution time versus thread execution time.

For thread execution times ranging from 1 to 1024 ms, the graph displays a trend where processes perform better than threads, with a normalized execution time of 1.0 at the start, dropping to about 0.8 for smaller thread execution times and then increasing slightly for larger thread execution times.]
“Shared Memory”
“Shared Memory”

Snapshot pages before modifications
“Shared Memory”

Snapshot pages before modifications

Write back diffs
for (i = 0; i < M; i++)
{
    x = 1;
}

for (i = 0; i < M; i++)
{
    y = 2;
}
for (i = 0; i < M; i++)
{
    x = 1;
}

for (i = 0; i < M; i++)
{
    y = 2;
}
for (i = 0; i < M; i++) {
    x = 1;
}

for (i = 0; i < M; i++) {
    y = 2;
}
for (i = 0; i < M; i++)
{
    x = 1;
}

for (i = 0; i < M; i++)
{
    y = 2;
}
Output: PTU vs. SHERIFF-DETECT

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>PTU (# shared lines)</th>
<th>SHERIFF-DETECT (# shared objects)</th>
</tr>
</thead>
<tbody>
<tr>
<td>canneal</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>kmeans</td>
<td>1916</td>
<td>2</td>
</tr>
<tr>
<td>linear_regression</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>468</td>
<td>0</td>
</tr>
<tr>
<td>pbzip2</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>pca</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>pfscan</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>reverse_index</td>
<td>N/A</td>
<td>5</td>
</tr>
<tr>
<td>streamcluster</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>word_count</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>swaptions</td>
<td>196</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>2,664</td>
<td>15</td>
</tr>
</tbody>
</table>
typedef struct {
    long long SX;
    long long SY;
    long long SXX;
    ...
} lreg_args;

void *linear_regression_pthread (void *args_in)
{
    lreg_args* args = (lreg_args*)args_in;
    ....
    for (i = 0; i < args->num elems; i++)
    {
        args->SX += args->points[i].x;
        args->SXX += args->points[i].x*args->points[i].x;
        ....
    }
/home/emery/sheriff$
/home/emery/sheriff$ gcc lreg.c -lsheriff_detect -o lreg
/home/emery/sheriff$ gcc lreg.c -lsheriff_detect -o lreg
/home/emery/sheriff$
/home/emery/sheriff$ gcc lreg.c -lsheriff_detect -o lreg
/home/emery/sheriff$ ./lreg
Sheriff-Detect has detected false sharing. Object 1: Allocation call stack:
   0: lreg.c: line number: 136
(1) Examine allocation site
Sheriff-Detect has detected false sharing.
Object 1: Allocation call stack:
  0: lreg.c: line number: 136

(1) Examine allocation site

```c
136 tid_args = (lreg_args *)calloc(sizeof(lreg_args), num_procs);
```

(2) Find references

```c
152 pthread_create(&tid_args[i].tid, &attr, linear_regression_pthread, (void*)&tid_args[i]) != 0);
```
void *linear_regression_pthread(void *args_in)
{
  lreg_args* args = (lreg_args*)args_in;
  ....
  for (i = 0; i < args->num_elems; i++)
  {
    args->SX += args->points[i].x;
    args->SXX += args->points[i].x*args->points[i].x;
    ....
  }

  "lreg_args" is not cache-line aligned
(3) Fix false sharing (here: *padding*)

typedef struct {
    ....
    char padding[128]; // Padding to avoid false sharing
} lreg_args;

linear_regression: before & after

9.2X
SHERIFF-DETECT performance

Normalized Execution Time

<table>
<thead>
<tr>
<th>blackscholes</th>
<th>canneal</th>
<th>dedup</th>
<th>ferret</th>
<th>fluidanimate</th>
<th>histogram</th>
<th>kmeans</th>
<th>linear_regression</th>
<th>matrix_multiply</th>
<th>pca</th>
<th>reverse_index</th>
<th>streamcluster</th>
<th>string_match</th>
<th>swaptions</th>
<th>word_count</th>
<th>geomean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
<td>0.8</td>
<td>1.2</td>
<td>1.0</td>
<td>0.8</td>
<td>0.9</td>
<td>0.7</td>
<td>0.8</td>
<td>0.7</td>
<td>0.7</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

20%
Prevents all false sharing
SHERIFF-PROTECT  SHERIFF-DETECT
for (i = 0; i < M; i++)
{
    x = 1;
}

for (i = 0; i < M; i++)
{
    y = 2;
}
false sharing detection

original program

Sheriff-DETECT
false sharing detection

Use Sheriff-DETECT reports to guide modifications (padding, alignment)

original program
Sheriff-DETECT

modified program
libpthread
false sharing detection

Use Sheriff-DETECT reports to guide modifications (padding, alignment)

original program

Sheriff-DETECT

modified program

libpthread

original program

libpthread

no false sharing

FALSE SHARING!
### False Sharing Detection

**Original Program**
- Sheriff-Detect

**Modified Program**
- libpthread
  - Use Sheriff-Detect reports to guide modifications (padding, alignment)
- Sheriff-PROTect
  - No false sharing
  - Source code unavailable or insufficient resources to act on Sheriff-Detect reports
  - Modifications degrade performance or cause excess memory consumption

---

**False Sharing Mitigation**
- Sheriff-PROTect
Dthreads

% g++ myprog.cpp -ldthreads -o myprog
Dthreads

- race conditions
- atomicity violations
- deadlock
- order violations
Dthreads

- race conditions
- atomicity violations
- deadlock
- order violations

Deterministic
DTHREADS = Deterministic Execution =

Race-free Executions

Replay Debugging w/o Logging

Replicated State Machines
Debuggable Races
Debuggable Races
Debuggable Races
Debuggable Races
Update in Deterministic Time & Order

Parallel

mutex_lock

cond_wait

pthread_create

Serial

"Thread" 1

"Thread" 2

"Thread" 3
**DTHREADS: Efficient Determinism**

Runtime relative to pthreads

- **CoreDet**: Usually faster than the state of the art
**DTHREADS: Efficient Determinism**

Generally as fast or faster than pthreads
race conditions
atomicity violations
deadlock
order violations
deterministic

DTHREADS

% g++ myprog.cpp -lpthread
SHERIFF-DETECT  SHERIFF-PROTECT
SHERIFF-DETECT  SHERIFF-PROTECT

[OOPSLA 2011]
SHERIFF-DETECT

SHERIFF-PROTECT

DTHREADS

[OOPSLA 2011]
https://github.com/plasma-umass/sheriff
https://github.com/plasma-umass/dthreads