Parallelising serial applications

Darryl Gove
Compiler Performance Engineering
Topics

• Process
• Tools
• Expectations
Profile

• Compile with debug info
  > -g [C/Fortran]
  > -g0 [C++]
  > Enables mapping of disassembly to source

• Profile with Performance Analyzer
  > collect <app> <params>
  > collect -P <pid>
### Application profile

![Sun Studio Analyzer](test.2.er)

**Functions**

<table>
<thead>
<tr>
<th>User CPU (sec.)</th>
<th>User CPU (sec.)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.838</td>
<td>11.838</td>
<td>&lt;Total&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User CPU (sec.)</th>
<th>User CPU (sec.)</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.818</td>
<td>11.838</td>
<td>main</td>
</tr>
<tr>
<td>0.020</td>
<td>0.020</td>
<td>_brk_unlocked</td>
</tr>
<tr>
<td>0.020</td>
<td>0.020</td>
<td>malloc</td>
</tr>
<tr>
<td>0.020</td>
<td>0.020</td>
<td>_malloc_unlocked</td>
</tr>
<tr>
<td>0.020</td>
<td>0.020</td>
<td>_morecore</td>
</tr>
<tr>
<td>0.020</td>
<td>0.020</td>
<td>sbrk</td>
</tr>
<tr>
<td>0.020</td>
<td>0.020</td>
<td>_sbrk_unlocked</td>
</tr>
<tr>
<td>0.020</td>
<td>11.838</td>
<td>_start</td>
</tr>
</tbody>
</table>
Source level profile

```c
20. int inset(double ix, double iy)

0. 0. 21. {

  <Function: inset>

  0. 0. 22. int iterations=0;
  0. 0. 23. double x=ix, y=iy, x2=x*x, y2=y*y;

  6.885 6.885 24. while ((x2+y2<4) && (iterations<1000))

  2.141 2.141 25. {
  0.430 0.430 26. y = 2 * x * y + iy;
  0.480 0.480 27. x = x2 - y2 + ix;
  1.411 1.411 28. x2 = x * x;
  0. 0. 29. y2 = y * y;
  0. 0. 30. iterations++;
  31. }

  0. 0. 32. return iterations;
  33. }
```
Parallelisation opportunities

• Large tasks
• No dependencies
• Avoid synchronisation

• Examples:
  > Independent transactions
  > Different iterations of loops
  > Different tasks
Expected gains from parallelisation

Maximum performance gain from parallelisation is determined by the time spent in code that can be parallelised.

Amdahl's Law.
POSIX Threads

- POSIX Threads (Pthreads)
  > Very flexible
  > Requires new code
  > May mean restructuring
Setting up Pthreads

```c
void main()
{
    pthread_t threads[2];
    int id[2];
    for (int i=0; i<2; i++) {
        id[i]=i;
        pthread_create(&threads[i],0,
            calculate,(void*)&id[i]);
    }
    for (int i=0; i<2; i++) {
        pthread_join(threads[i],0);
    }
}
```
Notes: Pthreads

- Solaris 9
  - `mt -lpthread`

- Solaris 10
  - `mt`
  - libc includes thread library
OpenMP

- OpenMP
  - Minimal source modification
  - Incremental parallelisation
  - Serial and parallel versions from same source
  - Some skill needed
  - Limited parallelisation options
    - 2.5 standard
      - Parallel for
      - Parallel sections
    - 3.0 standard
      - Tasks
void calculate()
{
    int x, y;
    double xv, yv;

    #pragma omp parallel for private(y, xv, yv)
    for (x=0; x<SIZE; x++) {
        for (y=0; y<SIZE; y++) {
            xv = ((double)(x-SIZE/2)) / (double)(SIZE/4);
            yv = ((double)(y-SIZE/2)) / (double)(SIZE/4);
            data[x][y] = inset(xv, yv);
        }
    }
}
Notes: OpenMP

• Compile with
  \> -xopenmp

• Warnings with
  \> -xloopinfo -xvpara

• Set number of threads threads with
  \> OMP_NUM_THREADS
Autoparallelisation

- Autoparallelisation
  > Easy to use
  > Limited range
  > Improve with:
    - Source changes
    - Compiler flags
Using autopar

```bash
$ cc -fast -xautopar -xreduction \
> -xloopinfo -xvpara m1.c

... "m1.c", line 40: PARALLELIZED, interchanged (inlined loop)
...```
Notes: autoparallelisation

- Compiler options
  - --xautopar --xreduction

- Warnings:
  - --xloopinfo --xvpara

- Set number of threads threads with
  > OMP_NUM_THREADS
Reductions

- Multiple threads cooperating to produce a single value.
- Order of calculation will be different to serial case

```c
for (i=0; i<N; i++)
{
    total += a[i];
}
```
Profiling a Multi-threaded application

Workload imbalance between the two threads
Data races

```bash
$ cc -fast -mt -o race race.c -lpthread

$ datarace
sum = 385158800

$ datarace
sum = 385071679
```
Data races

- Multiple threads reading/writing the same data without exclusive access
- Results in unpredictable behaviour

```
$ cc -fast -mt -o race race.c -lpthread
$ datarace
sum = 385158800
$ datarace
sum = 385071679
```
Thread Analyzer

Sun Studio Analyzer

Races

Total Races: 1

Race #1, Vaddr :0x21b10
Access 1: Write, calculate + 0x000001DC,
line 50 in "datarace.c"
Access 2: Write, calculate + 0x000001DC,
line 50 in "datarace.c"

Total Traces: 1

Data for Selected Race

Id: Race #1
Vaddr: 0x21b10

Access 1

Type: Write

calculate + 0x000001DC, line 50 in "datarace.c"

Access 2

Type: Write

calculate + 0x000001DC, line 50 in "datarace.c"
View source code

```c
for (y=0; y<SIZE; y++)
{
    xv = ((double)(x-SIZE/2))/(double)(SIZE/4);
    yv = ((double)(y-SIZE/2))/(double)(SIZE/4);
    sum+=inset(xv,yv);
}
```
Notes: Data races

Generate instrumented executable

```bash
$ cc -g -xinstrument=datarace -mt \n   -lpthread -o race race.c
$ collect -r on datarace
$ analyzer tha.1.er
```
Fixing data accesses

• Single thread access:
  > Mutex locks
  > Critical regions (OpenMP)

• Lock-less:
  > Atomic operations (man atomic_ops)

• Data sharing:
  > Thread local storage
  > OpenMP reduction directive
Summary

• Profile to find hot code

• Use multiple threads
  > Autoparallelisation
  > OpenMP easy to use
  > Pthreads gives more control

• Use the tools in Sun Studio
  > Performance Analyzer for profiling
  > Thread Analyzer for race condition detection
Parallelising serial applications

Darryl Gove
darryl.gove@sun.com
http://blogs.sun.com/d/