Code profiling on Graham

Sergey Mashchenko
(SHARCNET / Compute Ontario / Compute Canada)
Outline

- Introduction
- Simple profiling
- Profiling serial, MPI and OpenMP codes with MAP
- CUDA profiling
- Questions?
Introduction
What is profiling?

- Profiling is the task of timing a code.
- It is used primarily as a part of the iterative process of improving the efficiency (reducing the wallclock runtime) of the code.
- It is often done using simple means (like inserting time measurement lines in your code), but for serious profiling work one has to use dedicated profiling tools.
Simple profiling
Timing the whole code

• On SHARCNET clusters, one can use the Linux shell command “time” to time the whole code.

  $  time  ./your_code , or
  $  time  mpirun -np 32 ./your_mpi_code

• This has to be done on an empty node, to improve the accuracy of timing.

• A node can be reserved with salloc command (gives interactive access to compute resources for up to 3 hours), e.g.

  $  salloc --time=0-03:00 -c 32 -A def-account --mem=120000M
  (for serial and multi-threaded codes), and

  $  salloc --time=0-03:00 -n 32 -A def-account --mem-per-cpu=4000M
  (for MPI codes).
Timing the whole code (cont.)

- On Graham, cpu cores take non-negligible time to spin up from the idle state (1200 MHz) to the maximum speed under full load (2600 MHz).
- As a result, one has to time the code multiple times in a loop, choosing the best timing, e.g.:
  
  ```
  $ for ((i=0; i<10; i++)); do { time -p ./code ; } |& grep real ; done |sort -k 2 -gr
  ```
- This is obviously not ideal. A much better way is to place timers inside your code, to time specific parts of the code.
- This again should be ideally done in an internal (code) loop, to eliminate the cpu spin-up effect.
Timers inside your code

- `gettimeofday`: high precision (10μs) cpu-based timer (just google for the function `timeval_subtract`).

```c
#include <sys/time.h>
...
struct timeval tdr0, tdr1, tdr;
double delta_t;
gettimeofday (&tdr0, NULL);

< The code to time >

gettimeofday (&tdr1, NULL);
tdr = tdr0;
timeval_subtract (&delta_t, &tdr1, &tdr);
printf ("Time: %e\n", delta_t);
```
Timers inside your code (cont.)

- **OpenMP code**: `omp_get_wtime()` can be used to time (in seconds) both entire parallel regions, or individual threads inside a parallel region.

```c
#include <omp.h>
...
double t1 = omp_get_wtime();

< The code to time >

double t2 = omp_get_wtime();
printf ("Time: %e\n", t2-t1);
```

- This can also be used to time non-OpenMP codes (as it is more convenient than gettimeofday), just don't forget to add the “#include <omp.h>” line, and compile the code with “-qopenmp” (icc) or “-fopenmp” (gcc) switches.
Timers inside your code (cont.)

- **MPI code:** `MPI_Wtime()` can be used the same way as `omp_get_wtime()` on the previous slide:

```c
#include <mpi.h>
...
double t1 = MPI_Wtime();
    < The code to time >

double t2 = MPI_Wtime();
printf ("Time: %e\n", t2-t1);
```
Timers inside your code (cont.)

- **CUDA code**: to time a specific CUDA kernel, the best approach is to use CUDA events:

```c
cudaEvent_t start, stop;
float time;
cudaEventCreate (&start);
cudaEventCreate (&stop);
cudaEventRecord (start, 0);
kernel_to_time <<<grid, threads>>> ();
cudaEventRecord (stop, 0);
cudaEventSynchronize (stop);
cudaEventElapsedTime (&time, start, stop);
cudaEventDestroy (start);
cudaEventDestroy (stop);
```
Timers inside your code (cont.)

- For timing CUDA code consisting of multiple kernels, and/or concurrent GPU and CPU computations, and/or concurrent GPU operations (using streams), one has to use cpu-based timers (like gettimeofday or omp_get_wtime).
Profiling serial, MPI and OpenMP codes with MAP
Overview

- Parallel profiler MAP (along with the parallel debugger DDT) are now a part of the software package Forge.
- The original company behind MAP was Allinea. In 2016 it was acquired by the CPU maker ARM.
- SHARCNET has been using (and paying for) Allinea/ARM products since 2006.
- Though the debugger DDT was a success from the beginning, the Allinea's first attempt at parallel profiling (OPT) was a failure.
- The replacement MAP (came about in 2013; originally only for serial/MPI codes) used a much better approach to parallel profiling, and is now widely used in HPC community.
Overview (cont.)

- On Graham, profiler MAP is provided via module “ddt-cpu” (or aliases “allinea-cpu” and “arm-forge-cpu”).
- The Graham's license is for up to 512 concurrent cpu cores across all users (for both MAP and DDT).
- Niagara cluster (operated by SciNet) has a smaller license (up to 128 cpu cores). Neither MAP nor DDT are available on Cedar.
How to use MAP

- MAP (and DDT) are GUI applications, so one has to enable X11 forwarding in the SSH connection to be able to use them.
  - One has to add “-Y” switch to the usual ssh command:
    ```
    $ ssh -Y user@graham.computecanada.ca
    ```
  - Windows users: use free software MobaXterm, which comes with both SSH client and X window server (required for X11 forwarding).
  - Mac users: install free app XQuartz (has X windows server).
  - Linux users: everything you need is already installed on your box.

- As an alternative, one could use VNC connection to graham (google for “VNC compute canada” for details).
  - It has a better performance, but takes longer to set up.
How to use MAP (cont.)

- When using the X11 forwarding method, you need to add "--x11" switch to your salloc command, e.g.
  
  ```bash
  $ salloc --x11 --time=0-03:00 -c 32 -A def-account --mem=120000M
  ```

- After allocating the node(s) with salloc, load the module:

  ```bash
  $ module load ddt-cpu/7.1
  ```

- The code to be profiled has to be compiled with "-g" switch which adds symbolic information to the binary. You should use all your usual optimization flags (e.g. -O2).

- Run the code under MAP like this:

  ```bash
  $ map ./your_code <optional code arguments>
  ```
Some details

- Use the 7.1 version for now, as the newer one has some issues.
- Request one more cpu core than your code needs with salloc, as MAP uses one cpu core at 100% inside salloc session (it is likely a bug; we'll try to fix it).
- If you need more resources than available with salloc (>3h runtime, or hundreds of cpu cores), submit the MAP session as a job, e.g.
  
  ```sh
  module load ddt-cpu/7.1
  map --profile -n 16 ./code
  ```

- The profiling results (*.map files) can be analyzed offline with MAP:
  
  ```sh
  $ map results.map
  ```
We always start from the same (0-th) city:
\[ \text{perm}(0) = 0; \]

Cycle for Monte Carlo steps:
\[ \text{for } (k=0; k<N_MCLOCAL; k++) \]
\[ \quad \text{// Generating a random permutation:} \]
\[ \quad \text{// Initially we have an ordered list:} \]
\[ \quad \text{// rand \( \leq N \) returns a random integer \( \leq N \).} \]
\[ \quad \text{// Then we reschedule it randomly, starting with \( l=1 \):} \]
\[ \quad \text{d = 0.0;} \]
\[ \quad \text{for } (l=1; l<N\_CITIES-1; l++) \]
\[ \quad \quad \text{// This generates a random integer in the range \( [1 ... N\_CITIES-1] \):} \]
\[ \quad \quad \quad \text{ll = l + \text{rand}(0, \text{seed}) \% (N\_CITIES-1);} \]
\[ \quad \quad \text{// Swapping the \( l \) and \( ll \) cities:} \]
\[ \quad \quad \quad \text{il = perm(ll);} \]
\[ \quad \quad \quad \text{perm(ll) = perm(l);} \]
\[ \quad \quad \quad \text{perm(l) = il;} \]

// At this point, cities in perm[l-1] and perm[l] have already been rescheduled, so we
// can compute their contribution to the total distance:
Zooming in

we always start from the same (0-th) city:
perm[0] = 0;

// generating a random permutation:
for (i=1; i<N_CITIES; i++)
perm[i] = i;

// then we rehersal it randomly, starting with 1:
d = 0.6;
for (i=1; i<N_CITIES-1; i++)
{
    // this generates a random integer in the range [1 ... N_CITIES-1]:
    li = 1 + rand_r(seed) % (N_CITIES-1);
    // swapping the i and li cities:
    tperm = perm[i];
    perm[i] = perm[li];
    perm[li] = tperm;
}

// at this point, cities in perm[1] and perm[i] have already been reshuffled, so we
// can compute their contribution to the total distance:
December 19, 2018

“Code profiling on Graham”
Sergey Mashchenko, SHARCNET
Serial code profiling

- Example: a typical efficiency issue is when a loop in the code reads elements of the vector/array not in the order the data is stored in the memory. This makes CPU-memory caching inefficient.

- For C/C++ codes, data is stored in a row-major order, so it is the last index in multidimensional arrays which should correspond to the inner-most loop:

```plaintext
for (int i=0; i<N; i++)
    for (int j=0; j<N; j++)
        A[i][j] = 0.0;
```

```plaintext
\begin{array}{ccc}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{array}
```
Serial code profiling (CPU instructions preset).

Bad memory access case.

The problem
Serial code profiling (CPU instructions preset).

Good memory access case.
Profiling MPI codes

- Dynamic workload balancing (DWB) is frequently used by MPI programs.
- We use it when the length of time spent on computing different parts of a large workload by different MPI ranks is hard or impossible to predict ahead of time.
- Well written DWB code should have a way to adjust the size of the workload quantum. (In other words – number of chunks.)
Dynamic workload balancing example

- Example code:
  - `dynamic_workload_balancing.c`: using “nanosleep” function to emulate different processing time for different elements of a large input array
  - On 15 graham cpu cores, I got the following wall clock times:

<table>
<thead>
<tr>
<th>N_chunks / N_CPUs</th>
<th>Wall clock time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28.8</td>
</tr>
<tr>
<td>10</td>
<td>16.7</td>
</tr>
<tr>
<td>100</td>
<td>16.2</td>
</tr>
<tr>
<td>1000</td>
<td>16.1</td>
</tr>
<tr>
<td>200,000</td>
<td>20.6</td>
</tr>
</tbody>
</table>

- Severe workload imbalance
- Optimal performance
- Latency becomes important
Code profiling on Graham

Sergey Mashchenko, SHARCNET

Dynamic workload balancing example.

Chunks are too small = latency dominated.
MPI profiling, dynamic workload balancing example.

Chunks are too large = severe workload imbalance.
MPI profiling, dynamic workload balancing example.

Chunks have just the right size.

```
117  // Then it receives the current chunk of workload (array B):
118  MPI_Recv(B, K, MPI_DOUBLE, root, tag1, MPI_COMM_WORLD, &status);
119  MPI_Get_count(&status, MPI_DOUBLE, &k1);
120  // The computing part:
121  for (k=0; k<k1; k++)
122  {
123  
124  B[i] = compute2(B[i]);
125  
126  count++;
127  Nk += 1;
128  }
129  printf("Rank %d processed %d chunks (%d elements)\n", rank, count, Nk);
130  }  
131  MPI_Finalize();
```

“Code profiling on Graham”
Sergey Mashchenko, SHARCNET
OpenMP profiling

- In OpenMP, critical regions have a significant overhead and should be used sparingly.
- In particular, if used inside a loop for reduction, there should be a pre-selection statement, outside of the critical region:

```c
if (x > x_max)  // Pre-selects only plausible candidates
#pragma omp critical
    if (x > x_max)  // Very infrequently threads would enter the critical
        x_max = x;  // region, for the definitive “if” clause application
```
OpenMP profiling, dynamic critical region's impact.

No pre-selection = huge performance hit.
OpenMP profiling, dynamic critical region's impact.

Pre-selection = good performance.

---

“Code profiling on Graham”
Sergey Mashchenko, SHARCNET
CUDA profiling

- Recently MAP became capable of CUDA (GPGPU) code profiling.
- Unfortunately, SHARCNET's license doesn't cover this feature.
- But we do have Nvidia provided visual profilers for CUDA programs – nvvp and nsight.
- Unfortunately, they don't provide line-by-line kernel analysis (the way MAP does). But they provide plenty of detailed info on kernel performance.
- nvvp and nsight are bundled with cuda modules.
CUDA profiling (cont.)

- No code re-compilation is needed for nvvp profiling.
- Using nvvp interactively (on graham and cedar):
  
  ```
  $ salloc --x11 --time=0-03:00 --ntasks=1 --gres=gpu:1
  -A def-account --mem-per-cpu=4G
  $ module load cuda/10
  $ nvcc -O2 your_code.c -o your_code
  $ nvvp ./your_code
  ```

- The app will provide a step-by-step profiling setup. You choose which kernels to profile, and what specific details you need.
- The app will often provide useful descriptive suggestions regarding which parts of your code have efficiency issues.
CUDA profiling (cont.)

- Non-coalesced access of the device memory is a significant efficiency issue in GPU programming (similar to the row-major memory access requirement on CPUs).

**Good (coalesced) access pattern:**

![Access Pattern Diagram]

- Address: 0 to 128
- Thread ID: 0 to 31
Sign of non-coalesced memory access
Good memory access pattern
Questions?

• You can always contact me directly (syam@sharcnet.ca) or send an email to help@sharcnet.ca.