Performance Tuning and OpenMP

Matthias Müller
mueller@hlrs.de

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Outline

• Motivation
• Performance Basics
• General Performance Issues and OpenMP
• Special Performance Hints for OpenMP
Motivation

Reasons for parallel programming:

1. Higher Performance
   - Solve the same problem in shorter time
   - Solve larger problems in the same time

2. Higher Capability
   - Solve problems that cannot be solved on a single processor
     - Larger memory on parallel computers, e.g. 128 GB on hwrs8k
     - Time constraints limits the possible problem size
       (Weather forecast, turn around within working day)

In both cases performance is one of the major concerns.

---

Performance Basics: Speed Up

- Definition of speed up $S$

\[ S = \frac{T_s}{T_p} \]

- $T_s$: Serial Execution Time
- $T_p$: Parallel Execution Time

- Speed up versus number of used processors:

![Graph showing speed up versus number of processors]
Performance Basics: Amdahl’s Law

- Assumption:
  - Only a fraction $F$ of the algorithm is parallel with speed up $S_p$
  - A fraction $(1-F)$ is serial
- Total speed up:

$$S = \frac{1}{(1-F) + \frac{F}{S_p}}$$

- Even with infinite parallel speed up your total speed up is limited to:

$$S = \frac{1}{(1-F)}$$

Consequence of Amdahl’s law: necessary parallelization

- If you know your desired speed up $S$ you can calculate $F$:

$$F = 1 - \frac{1}{S}$$

- $F$ gives you the percentage of your program that has to be executed parallel in order to achieve a speed up $S$
- In order estimate the resulting effort you need to know in which parts of your program 100*(1-F)% of the time is spent.
Performance measurement: Profiling

- On most platforms:
  compile with option -p to get program counter profiling

<table>
<thead>
<tr>
<th>#calls</th>
<th>Time (%)</th>
<th>Accumulated Time (%)</th>
<th>Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>155648</td>
<td>31.22</td>
<td>31.22</td>
<td>Calc</td>
</tr>
<tr>
<td>603648</td>
<td>22.24</td>
<td>53.46</td>
<td>Multiply</td>
</tr>
<tr>
<td>155648</td>
<td>10.05</td>
<td>63.51</td>
<td>Matmul</td>
</tr>
<tr>
<td>214528</td>
<td>9.33</td>
<td>72.84</td>
<td>Copy</td>
</tr>
<tr>
<td>603648</td>
<td>7.87</td>
<td>80.71</td>
<td>Find</td>
</tr>
</tbody>
</table>

- Examples:
  - for a speed up of 2 you need to parallelize Calc and Multiply.
  - for a speed up of 5 you need to parallelize Calc, Multiply, Matmul, Copy and Find

Advantage of OpenMP: this incremental approach is possible

General issues: Problem size dependency of performance

- Example: Norm of a Matrix:
  \[ \|A\| = \max_j \sum_i |A_{ij}| \]

- Simple Algorithm:
  ```
  do j=1,n
    b(j)=0
    do i=1,n
      b(j) = b(j) + abs(a(i,j))
    end do
  end do
  do j=1,n
    result = max(result,b(j))
  end do
  ```

- Change Matrix size from 1 to 4096 and check the performance
OpenMP version of Matrix Norm

```plaintext
!$OMP PARALLEL
!$OMP DO PRIVATE(i)
    do j=1,n
        b(j)=0
        do i=1,n
            b(j) = b(j) + abs(a(i,j))
        end do
    end do
!$OMP DO REDUCTION(MAX:result)
    do j=1,n
        result = max(result,b(j))
    end do
!$OMP END PARALLEL
```

Performance on a PC (Dual Pentium II, 450 MHz)
Performance comparison (2 threads with OpenMP)

Use OpenMP only with sufficient workload: if-clause

- Only start parallel thread if there is enough workload, otherwise code is executed serial.

```c
!$OMP PARALLEL IF(n>32)
!$OMP DO PRIVATE(i)
   do j=1,n
      b(j)=0
   do i=1,n
      b(j) = b(j) + abs(a(i,j))
   end do
   end do
!$OMP DO REDUCTION(MAX:result)
   do j=1,n
      result = max(result,b(j))
   end do
!$OMP END PARALLEL
```
Performance with if-clause

Performance with OpenMP: Avoid thread creation

```
#pragma omp parallel for
for(i=0; i<size; i++)
a[i] = 1.0/a[i];

#pragma omp parallel for
for(i=0; i<size; i++)
b[i] = b[i]*2.0
```

The improved version only creates the threads once:

```
#pragma omp parallel
{
    #pragma omp for
    for(i=0; i<size; i++)
a[i] = 1.0/a[i];

    #pragma omp for
    for(i=0; i<size; i++)
b[i] = b[i]*2.0
}
```
Performance with OpenMP: Avoid barriers

1. Merge loops:
   Replace:
   ```c
   #pragma omp for
   for(i=0; i<size; i++)
       a[i] = 1.0/a[i];
   #pragma omp for
   for(i=0; i<size; i++)
       b[i] = b[i]*2.0
   ```
   with
   ```c
   #pragma omp for
   for(i=0; i<size; i++)
       a[i] = 1.0/a[i];
       b[i] = b[i]*2.0;
   ```

Performance with OpenMP: Avoid barriers (II)

2. Use the NOWAIT clause
   • An implicit barrier is put at the end of each work-sharing construct
   • It can be eliminated by using `nowait`, if the barrier is not required
   ```c
   #pragma omp parallel
   {
   #pragma omp for nowait
       for(i=0; i<size; i++)
           a[i] = 1.0/a[i];
   #pragma omp for
       for(i=0; i<n; i++)
           b[i] = b[i]*2.0
   }
Performance with OpenMP: load balancing

- Different scheduling mechanisms help to avoid load imbalance
- Be aware that dynamic scheduling has a larger overhead
- The default scheduling is implementation dependent, but probably very similar to \texttt{SCHEDULE(STATIC)}, with one chunk for each thread
- Use \texttt{SCHEDULE(DYNAMIC \[,chunksize\] )} if the workload for each iteration is large and the workload is not predictable
- Use \texttt{SCHEDULE(STATIC, chunksize)} if the load balance can be achieved by reducing the chunksize
- Use \texttt{SCHEDULE(GUIDED \[,chunksize\] )} as a compromise between \texttt{STATIC} and \texttt{DYNAMIC}
- Use \texttt{SCHEDULE(RUNTIME)} if the best scheduling depends strongly on the input data, set \texttt{OMP\_SCHEDULE} accordingly

Summary

- Do not forget serial performance
  - profiling helps to understand the effort to parallelize your program
  - the performance depends on the problem size, using parallelism reduces the problem size on each thread
- Avoid thread creation and unnecessary synchronization
- Use dynamic scheduling strategies only if you have load imbalance problems
- Try to find the best platform for your problem
Exercise: Matrix Norm Calculation

- Program calculates the norm of a rectangular matrix and a triangular matrix
- Fortran Version: norm_f90.f90
  C Version: norm_c.c
  hhrs_get_time.c: utility function for time measurement

Exercise: Matrix Norm Calculation on SR8000

Step 1: Compile and run the serial version of your program
On crosscompiler platform:
- Fortran:
  - xf90 -c norm_f90.f90 -OSS -nosearch norm_f90.f90
  - xcc -O4 -pvec +Op -nosearch hhrs_get_time.c
  - xf90 -o norm_f90 hhrs_get_time.o norm_f90.o
- C:
  - xcc -c -O4 -pvec +Op -nosearch norm_c.c
  - xcc -c -O4 -pvec +Op -nosearch hhrs_get_time.c
  - xcc -o norm_c norm_c.o hhrs_get_time.o
On Hitachi SR8000:
- run norm_c or norm_f90
Exercise: Matrix Norm Calculation on SR8000

Step 2: Profile the program

- Fortran:
  - xf90 -c norm_f90.f90 -OSS -noparallel -Xfuncmonitor
  - xcc -c -O4 -pvec +Op -noparallel hlrs_get_time.c
  - xf90 -o norm_f90 hlrs_get_time.o norm_f90.o -lpl -parallel

- C:
  - xcc -c -O4 -pvec +Op -noparallel norm_c.c -Xfuncmonitor
  - xcc -c -O4 -pvec +Op -noparallel hlrs_get_time.c
  - xcc -o norm_c norm_c.o hlrs_get_time.o -lpl -parallel

On Hitachi SR8000:
- run norm_c or norm_f90 and check pl_norm_XXXX.txt

Exercise: Matrix Norm Calculation on SR8000

Step 3: Add OpenMP directives

- Concentrate on the most time consuming parts of your program.
  Think about the speed up you like to achieve and the consequences
  of Amdahl’s law.

Step 4: Compile and run

- Fortran:
  - xf90 -c -OSS -parallel -omp norm_f90_omp.f90 -Xfuncmonitor
  - xcc -c -O4 -pvec +Op -parallel -omp hlrs_get_time.c
  - xf90 -parallel -omp -o norm_f90 hlrs_get_time.o norm_f90_omp.o -lpl
  - prun -p single ./norm_f90

- C:
  - xcc -c -O4 -pvec +Op -parallel -omp norm_c_omp.c -Xfuncmonitor
  - xcc -c -O4 -pvec +Op -parallel -omp hlrs_get_time.c
  - xcc -parallel -omp -o norm_c norm_c_omp.o hlrs_get_time.o -lpl
  - prun -p single ./norm_c
Exercise: Matrix Norm Calculation

Step 5: Optimize
- Try to get rid of redundant synchronization by adding nowait or merging loops
- Try to improve the performance for small matrices, compare the performance with the serial code
- Think about possible load imbalance

Hint: for your convenience there is a gnuplot script file.
- Save the output of the serial program in “scalar.log”
- Save the output of the OpenMP program in “parallel.log”
- Save the output of the optimized version in “optimized.log”
- Compare the results with “gnuplot makeplots.gnu”

Gnuplot

Hint: for your convenience there is a gnuplot script file.

Fortran
- Save one result in “result.log”
- Save the output of the optimized version in “solution_f90.log”
- Compare the results with “gnuplot makeplots_f90.gnu”

C:
- Save one result in “result.log”
- Save the output of the optimized version in “solution_c.log”
- Compare the results with “gnuplot makeplots_c.gnu”