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 hwwsr8k**
     - **Time constraints limits the possible problem size (Weather forecast, turn around within working day)**

In both cases performance is one of the major concerns.

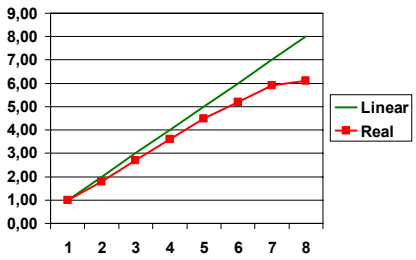
Performance Basics: Speed Up

- Definition of speed up ***S***

$$S = \frac{T_S}{T_P}$$

Ts: Serial Execution Time *Tp:* Parallel Execution Time

- Speed up versus number of used 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

#calls	Time (%)	Accumulated Time (%)	Call
155648	31.22	31.22	Calc
603648	22.24	53.46	Multiply
155648	10.05	63.51	Matmul
214528	9.33	72.84	Copy
603648	7.87	80.71	Find

- 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

Performance tuning basics

- There are no general rules!
- Things that help to achieve high performance:
  - Know your application
  - Know your compiler
  - Understand the performance tool
  - Know the characteristics of the hardware

General issues: Problem size dependency of performance

- Example: Norm of a Matrix:

$$\|A\| = \max_j \sum_i |A_{ij}|$$

- Simple Algorithm:

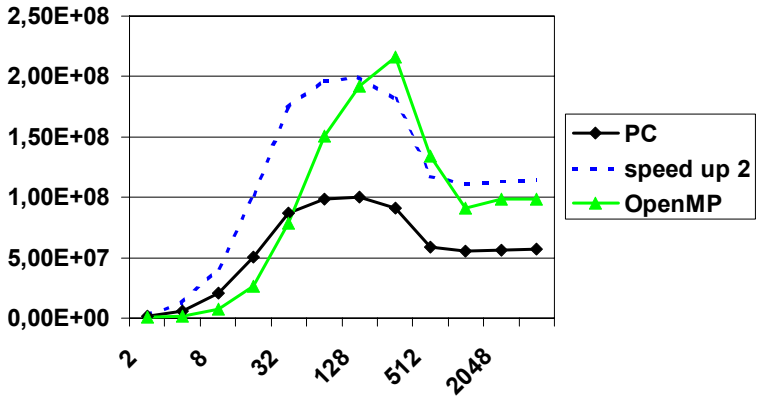
```
do j=1,n
    b(j)=0
    do i=1,n
        b(j) = b(j) + abs(a(i,j))
    end do
end do
do j=1,n
    result = max(result,b(j))
end do
```

- Change Matrix size from 1 to 4096 and check the performance

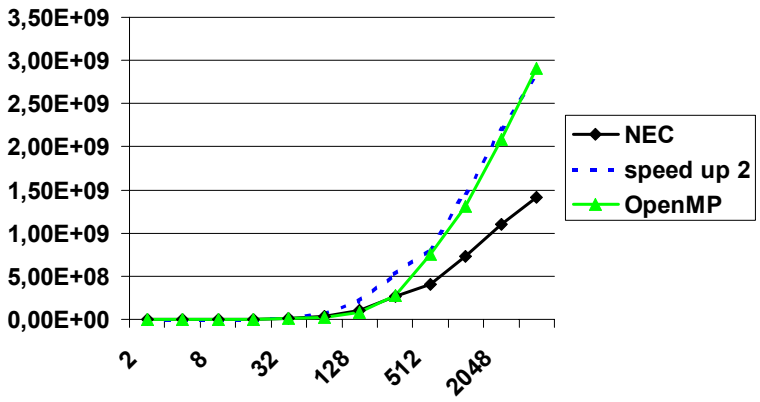
OpenMP version of Matrix Norm

```
!$OMP PARALLEL
!$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
```

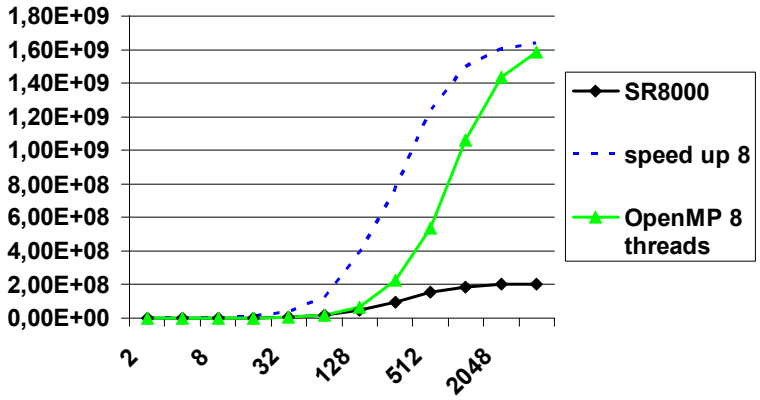
Performance on a PC (Dual Pentium II, 450 MHz)



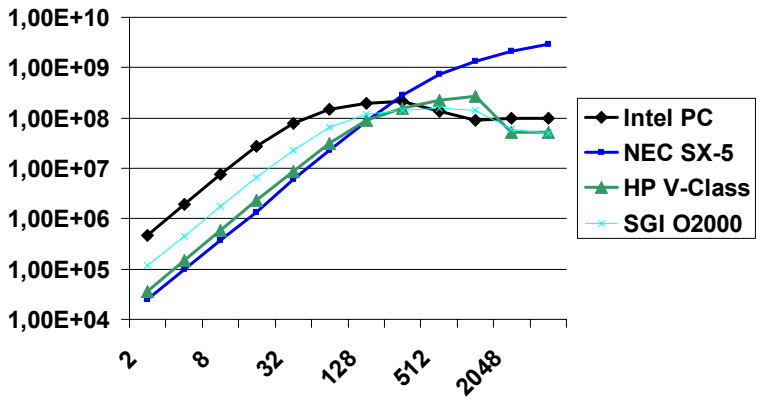
Performance on a Vectorcomputer (NEC SX-5Be)



Performance on the Hitachi SR8000



Performance comparison (2 threads with OpenMP)



Get a feeling for the involved overheads

Operation	Minimum overhead (cycles)	Scalability
Hit L1 cache	1-10	Constant
Function call	10-20	Constant
Thread ID	10-50	Constant, log, linear
Integer divide	50-100	Constant
Static do/for, no barrier	100-200	Constant
Miss all caches	100-300	Constant
Lock acquisition	100-300	Depends on contention
Dynamic do/for, no barrier	1000-2000	Depends on contention
Barrier	200-500	Log, linear
Parallel	500-1000	Linear
Ordered	5000-10000	Depends on contention

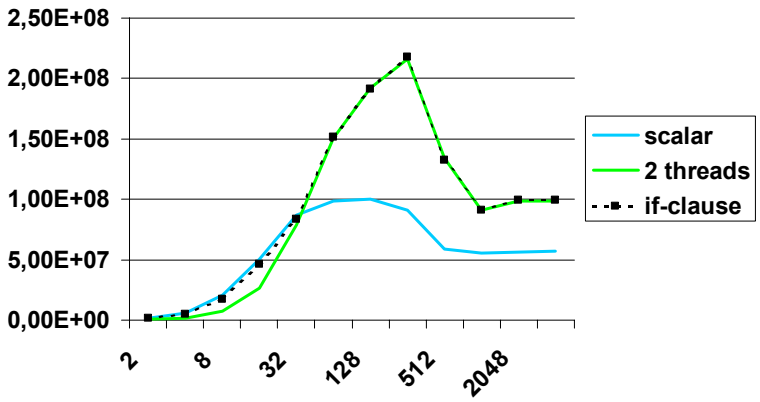
All numbers are approximate!! They are very platform dependant !!

Use OpenMP only with sufficient workload: if-clause

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

```
!$OMP PARALLEL IF(n>32)
!$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



Avoiding parallelism where it is harmful (II)

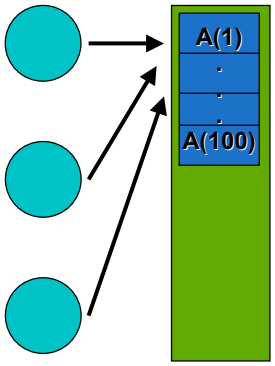
- When would parallelizing this loop help?

```
DO I = 1, N
   A(I) = 0
ENDDO
```

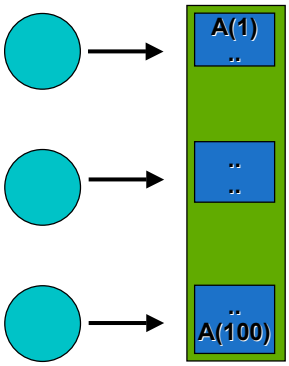
- Some issues to consider
  - Value of N
    - **Very large N, so A is not cache contained**
  - Placement of Object A
    - **If distributed onto different processor caches, or about to be distributed**
    - **On NUMA systems, when using first touch policy for placing objects, to achieve a certain placement for object A**

First touch algorithm on NUMA systems

```
DO I = 1, N
   A(I) = 0
ENDDO
```

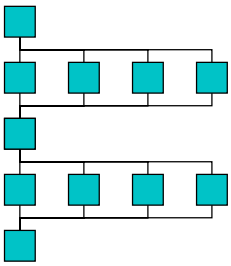



```
$!OMP DO
DO I = 1, N
   A(I) = 0
ENDDO
```



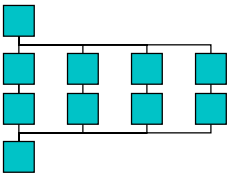
Performance with OpenMP: Avoid thread creation

```
#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:

```
#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

```
#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

```
#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 dependant, but probably very similar to `SCHEDULE(STATIC)`, with one chunk for each thread
- Use `SCHEDULE(DYNAMIC [,chunksize] )` if the workload for each iteration is large and the workload is not predictable
- Use `SCHEDULE(STATIC, chunksize)` if the load balance can be achieved be reducing the chunksize
- Use `SCHEDULE(GUIDED [,chunksize] )` as a compromise between `STATIC` and `DYNAMIC`
- Use `SCHEDULE(RUNTIME)` if the best scheduling depends strongly on the input data, set `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_omp.f90
  C Version: norm_c_omp.c
  hlrs_get_time.c: utility function for time measurement
- Source code in directory
  ~/OpenMP/#NR/performance:
  `cd ~/OpenMP/#NR/performance`

Exercise: Matrix Norm Calculation on HPN

Step 1: Compile and run the serial version of your program

- Fortran:
  - f90 +Oall -c norm_f90_omp.f90
  - cc -c -fast hlrs_get_time.c
  - f90 +Oall -o norm_f90 hlrs_get_time.o norm_f90_omp.o
  - ./norm_f90
- C:
  - cc -c -fast norm_c_omp.c
  - cc -c -fast hlrs_get_time.c
  - cc -fast -o norm_c norm_c_omp.o hlrs_get_time.o
  - ./norm_c

Exercise: Matrix Norm Calculation on HPN

Step 2: Compile OpenMP version and run

- Fortran:
  - guidef90 +Oall -c norm_f90_omp.f90
  - cc -fast -c hlrs_get_time.c
  - guidef90 +Oall -o norm_f90_omp hlrs_get_time.o norm_f90_omp.o
  - export OMP_NUM_THREADS=2
  - ./norm_f90_omp
- C:
  - guidec +K3 --backend -fast -c norm_c_omp.c
  - cc -fast -c hlrs_get_time.c
  - guidec +K3 --backend -fast -o norm_c_omp norm_c_omp.o hlrs_get_time.o
  - export OMP_NUM_THREADS=2
  - ./norm_c_omp

Exercise: Matrix Norm Calculation on HPN

Step 3: Compile OpenMP version with profiling and run

- Fortran:
  - guidef90 +Oall -c norm_f90_omp.f90
  - cc -fast -c hlrs_get_time.c
  - guidef90 **-WGstats** +Oall -o norm_f90_omp hlrs_get_time.o norm_f90_omp.o
- C:
  - guidec +K3 --backend -fast -c norm_c_omp.c
  - cc -fast -c hlrs_get_time.c
  - guidec **-WGstats** +K3 --backend -fast -o norm_c_omp norm_c_omp.o hlrs_get_time.o

Analyze the results with guideview: call `guideview` after execution of progamm

Exercise: Matrix Norm Calculation on AZUSA

Step 1: Compile and run the serial version of your program

- Fortran:
  - efc -O3 -ip -ipo -c norm_f90_omp.f90
  - ecc -O3 -c hlrs_get_time.c
  - efc -O3 -ip -ipo -o norm_f90 hlrs_get_time.o norm_f90_omp.o
  - ./norm_f90
- C:
  - ecc -O3 -c norm_c_omp.c
  - ecc -O3 -o norm_c norm_c_omp.o
  - ./norm_c

Bug fixes:

1. Replace CLK_TICK with 1000000
2. Replace **usere-users** with **reale-reals** in the main program
   (two occasions)
3. For C --- When linking, if You get a link-error; multiple definition of hlrs_get_time, You DO not have to relink with hlrs_get_time.o --- it‘s already in Your hlrs_get_time.h
4. For Fortran: ulimit –s 100000

Exercise: Matrix Norm Calculation on AZUSA

Step 2: Compile OpenMP version and run

- Fortran:
  - guideefc -O3 -ip -ipo -W0 -c norm_f90_omp.f90
  - ecc -O3 -c hlrs_get_time.c
  - guideefc -O3 -ip -ipo -W0 -o norm_f90_omp hlrs_get_time.o norm_f90_omp.o
  - export OMP_NUM_THREADS=2
  - ./norm_f90_omp
- C:
  - guidec -WGstats -c norm_c_omp.c
  - gcc -O3 -c hlrs_get_time.c
  - guidec -WGstats -o norm_c_omp norm_c_omp.o hlrs_get_time.o
  - export OMP_NUM_THREADS=2
  - ./norm_c_omp

Exercise: Matrix Norm Calculation on AZUSA

Step 3: Compile OpenMP version with profiling and run

Fortran:

- guideifc -O3 -ip -ipo -W0 -c norm_f90_omp.f90
- ecc -O3 -c hlrs_get_time.c
- guideifc -O3 -ip -ipo -W0 **-WGstats** -o norm_f90_omp hlrs_get_time.o norm_f90_omp.o

- C:
  - guidec +K3 -c norm_c_omp.c
  - gcc -O3 -c hlrs_get_time.c
  - guidec +K3 --backend **-WGstats** -o norm_c_omp norm_c_omp.o hlrs_get_time.o

Analyze the results with guideview: call `guideview` after execution of program

Exercise: Matrix Norm Calculation on SR8000

Step 1: Compile and run the serial version of your program

On crosscompiler platform:

- Fortran:
  - xf90 -c -OSS -noparallel norm_f90_omp.f90
  - xcc -c -O4 -pvec +Op -noparallel hlrs_get_time.c
  - xf90 -o norm_f90 hlrs_get_time.o norm_f90_omp.o
- C:
  - xcc -c -O4 -pvec +Op -noparallel norm_c_omp.c
  - xcc -c -O4 -pvec +Op -noparallel hlrs_get_time.c
  - xcc -o norm_c norm_c_omp.o hlrs_get_time.o

On Hitachi SR8000:

- run (results may vary due to timesharing)
  - ./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** norm_f90.f90
  - 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_omp.c -**Xfuncmonitor**
  - xcc -c -O4 -pvec +Op -noparallel hlrs_get_time.c
  - xcc -o norm_c norm_c_omp.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: Compile OpenMP version with profiling 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 4: 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 program in “result.log”
- Compare the results with “gnuplot makeplots_{c,f90}.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”