High-Performance Computing

Parallel Programming in OpenMP

– Labs: traps & pitfalls and results –

OpenMP Excercises – I

- Write an OpenMP code to calculate π, using

$$\pi = \int_0^1 \frac{4}{(1+x^2)} dx \approx \frac{1}{N} \sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-0.5}{N}\right)^2}$$

  - implement the integrand as a function
  - write your own reduction code
  - use the OpenMP reduction clause
  - compare the run-times

OpenMP Excercises – I

Automatic parallelization with GCC:

- compile with `-ftree-parallelize-loops=4`

```
$ gcc -g -Ofast -ftree-parallelize-loops=4 \
    -o pi.gcc pi.c

$ time -p ./pi.gcc 1000000000
Pi: 3.141593, Time Elapsed: 4.968379
real 4.97
user 4.97
```

No speed-up!

OpenMP Excercises – I

Automatic parallelization with GCC:

- here is the code

```
double pi_step(double i, double N) {
    double temp = ((i - 0.5) / N);
    return 4.0 / (1.0 + temp * temp);
}
double calc_pi(double N) {
    double pi = 0;
    for (int i = 0; i < N; i++) {
        pi += pi_step(i, N);
    }
    return pi / N;
}
```

OpenMP Excercises – I

Automatic parallelization with GCC:

- replace the “double N” with “size_t N”
- legal for loop!

```
$ gcc -g -Ofast -ftree-parallelize-loops=4 \
    -o pi.gcc pi.c

$ time -p ./pi.gcc 1000000000
Pi: 3.141593, Time Elapsed: 0.759578
real 0.76
user 2.92
```

Expected 4x speed-up!

OpenMP Exercise – I

Starting point: numerical integration of f(x)

- smart OpenMP solution

```
int i, n;
double h, x, sum;

h = 1.0 / (double)n;
sum = 0.0;

#pragma omp parallel for default(none) \
        shared(n,h) private(i,x) \
        reduction(+: sum)
for(i=1; i<=n; i++) {
   x = h * ((double)i + 0.5);
   sum += f(x);
}
```

OpenMP Exercise – I

Sequential version: compiled with '-g -fast'

- runtime: 4.12 secs

```
int i;
int N = 1000000000;
double pi = 0;




for( i=1 ; i<=N ; i++) {
   pi += 4.0 / ( 1.0 +
         ((i-0.5) / N) * ((i-0.5) / N) );
}

pi = pi * 1/N;
```

OpenMP Exercise – I

Automatic parallellization:

- compile with '-g -fast -xautopar -xreduction'

```
$ time OMP_NUM_THREADS=1 ./piauto
real    0m4.139s
user    0m4.125s

$ time OMP_NUM_THREADS=2 ./piauto
real    0m2.119s
user    0m4.211s

$ time OMP_NUM_THREADS=4 ./piauto
real    0m1.061s
user    0m4.153s
```

OpenMP Exercise – I

OpenMP version:

```
int i;
int N = 1000000000;
double pi = 0;

#pragma omp parallel for default(none) \
       shared(N) private(i) reduction(+: pi)
for( i=1 ; i<=N ; i++) {
   pi += 4.0 / ( 1.0 +
         ((i-0.5) / N) * ((i-0.5) / N) );
}

pi = pi * 1/N;
```

OpenMP Exercise – I

OpenMP parallellization:

- compile with '-g -fast -xopenmp'

```
$ time OMP_NUM_THREADS=1 ./piomp
real      0m8.362s
user      0m8.335s

$ time OMP_NUM_THREADS=2 ./piomp
real      0m4.256s
user      0m8.404s

$ time OMP_NUM_THREADS=4 ./piomp
real      0m2.125s
user      0m8.371s
```

But this is 2x slower!!!!

It scales!

What's going on here???

OpenMP Exercise – I

Looking at the problem:

- compiler comments:
  - almost no difference – same optimizations applied
- replaced loop body by a function call:
  - pi += f(x)
  - no effect!
- hmmm ... what now?

OpenMP Exercise – I

- What happens in the OpenMP version?
  - the code block following the “#pragma omp ...” gets “outlined” into a function
  - the compiler optimization is applied to this new, outlined function
  - in this process, some “information/knowledge” gets lost, and thus the compiler cannot apply advanced optimizations
  - in this case: the compiler does not optimize the division by N in the OpenMP version!

Programming OpenMP

OpenMP Exercise – I

OpenMP version: solution

```
int i;
int N = 1000000000;
double pi = 0;
double h = 1.0 / N;

#pragma omp parallel for default(none) \
        shared(N,h) private(i) reduction(+:pi)
for( i=1 ; i<=N ; i++) {
   pi += 4.0 / ( 1.0 +
         ((i-0.5) * h) * ((i-0.5) * h) );
}

pi = pi * h;
```

OpenMP Exercise – I

OpenMP parallellization - fixed:

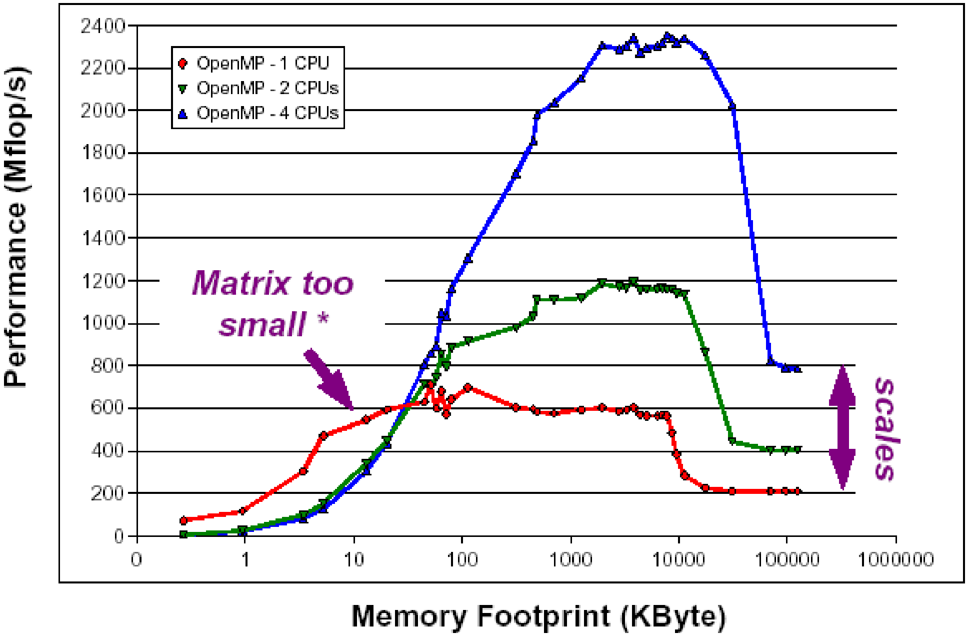
- compile with '-g -fast -xopenmp'

```
$ time OMP_NUM_THREADS=1 ./piompfix
real     0m4.137s
user     0m4.120s

$ time OMP_NUM_THREADS=2 ./piompfix
real     0m2.124s
user     0m4.177s

$ time OMP_NUM_THREADS=4 ./piompfix
real     0m1.061s
user     0m4.174s
```

Same as with 'autopar'!

OpenMP Excercises – II

- Improve the matrix times vector example by adding an if-clause to the omp pragma – experiment with the threshold value!



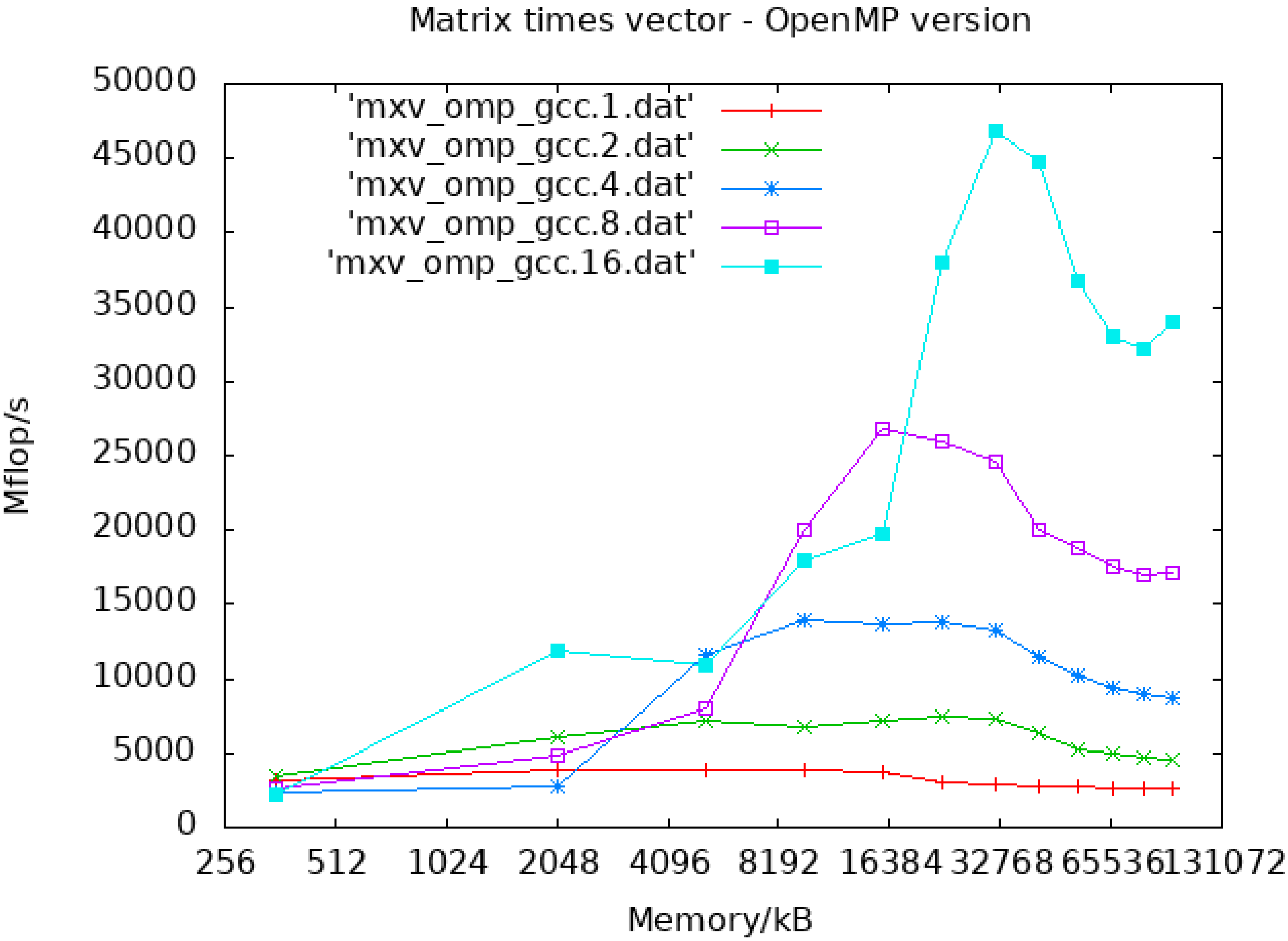
SunFire 6800
UltraSPARC III Cu @ 900 MHz
8 MB L2-cache

*) With the IF-clause in OpenMP this performance degradation can be avoided

OpenMP Parallelization

```
void
mxv(int m,int n,double * restrict a,
       double * restrict b,double * restrict c) {

   int i, j;
   double sum;
   #pragma omp parallel for private(i,j,sum)
   for (i=0; i<m; i++) {
      sum = 0.0;
      for (j=0; j<n; j++)
           sum += b[i*n+j]*c[j];
      a[i] = sum;
   }
}
```

OpenMP Parallelization


Matrix times vector - OpenMP version
Mflop/s
Memory/kB
'mxv_omp_gcc.1.dat'
'mxv_omp_gcc.2.dat'
'mxv_omp_gcc.4.dat'
'mxv_omp_gcc.8.dat'
'mxv_omp_gcc.16.dat'