Parallel Programming in OpenMP

– Labs –

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 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!

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
```

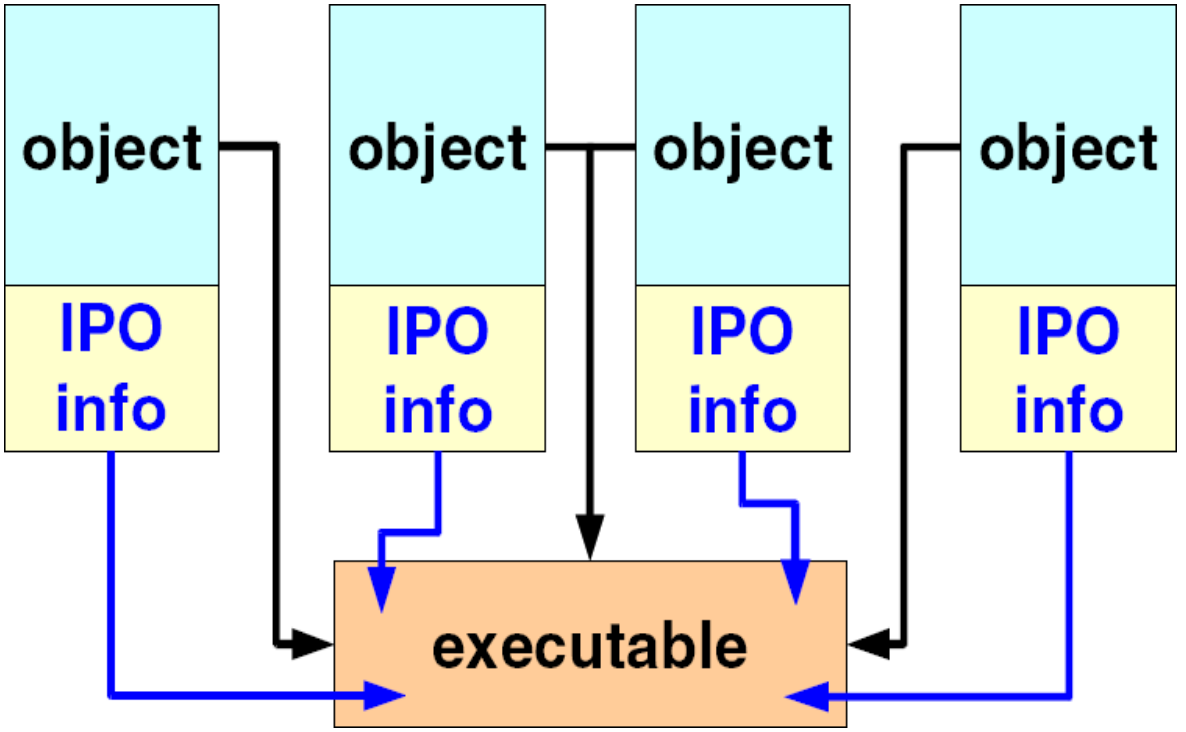
Same as with 'autopar'!

OpenMP Exercise – I

- Using more than one source file
  - if the function f() that should be integrated is not in the same source file, automatic parallelization does not work
  - workaround: use IPO (Interprocedural Optimization)
  - with the Studio compilers: -xipo

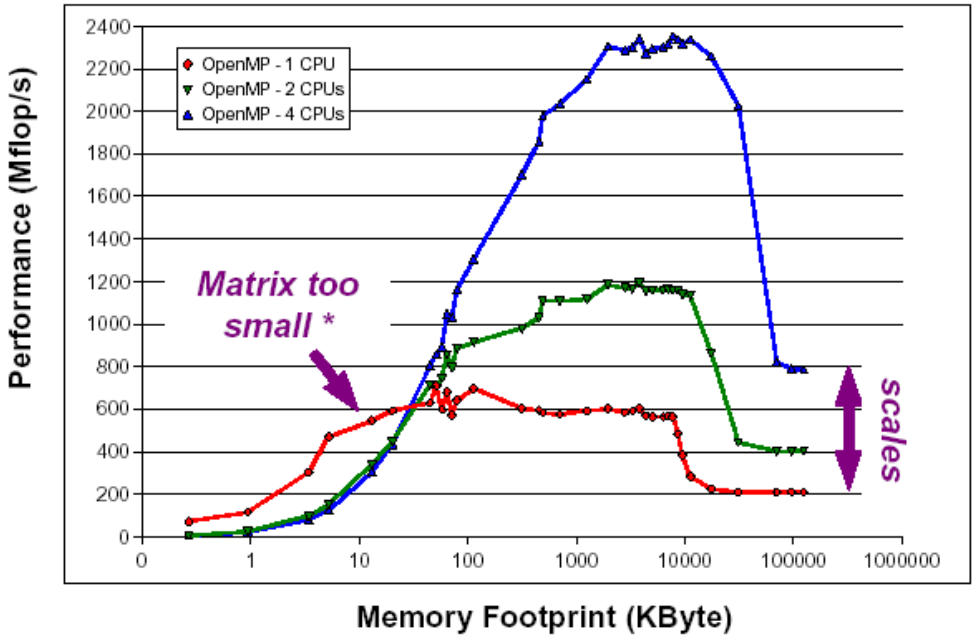
Re-cap: Inter Procedural Optimization

- With the -xipo option, the compiler stores additional information into the object files
- This information is used during the link phase to perform additional optimizations



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

Automatic Parallelization

```
 1 void
 2 mxv(int m,int n,double *a,double *b,double *c) {
 3
 4     int i, j;
 5     double sum;
 6
 7     for (i=0; i<m; i++) {
 8         sum = 0.0;
 9         for (j=0; j<n; j++)
10             sum += b[i*n+j]*c[j];
11         a[i] = sum;
12     }
13 }
```

```
cc -g -fast -xrestrict -xautopar -xloopinfo -c mxv.c
"mxv.c", line 7: PARALLELIZED, and serial version
generated
"mxv.c", line 9: not parallelized, unsafe dependence
(sum)
```

Automatic Parallelization

- Compiling with '-xautopar':

```
void mxv(int m,int n,double *a,double *b,double *c) {
  //... lines omitted ...
}

int main(...) {
    //... lines omitted ...
    // do max_it iteration for timing
    for(int k=0; k < max_it; k++) {
        mxv(m, n, a, b, c);
    }
    // ... lines omitted ...
}
```

- compiler reports “PARALLELIZED” in mxv()
- but we see no speed-up!?!?

Automatic Parallelization

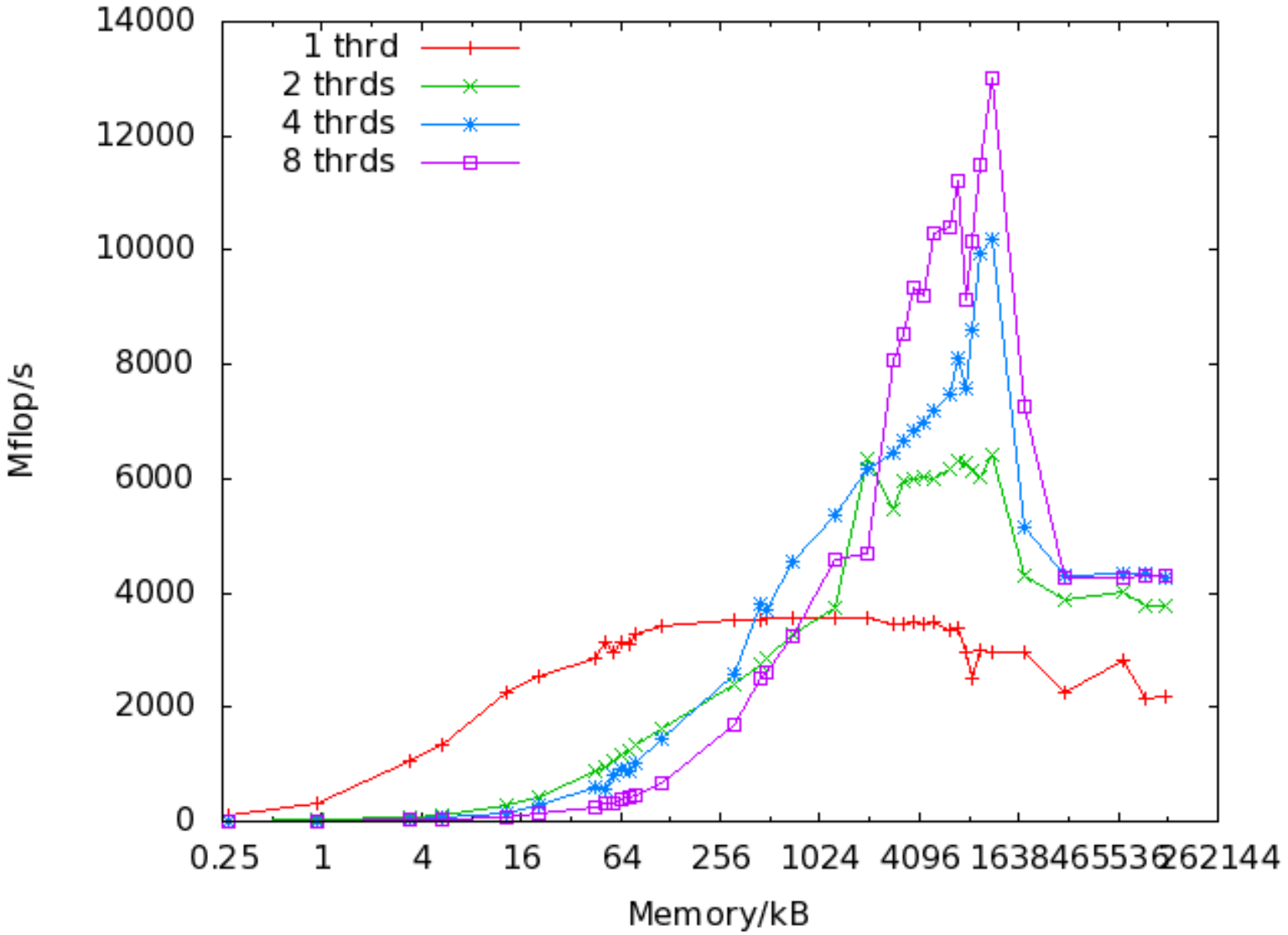
- Reason: the compiler “knows too much”!
- The compiler inlines mxv() in main(), and then there is a triple for-loop nesting, which prevents automatic parallelization! This can be seen in the compiler messages, too!
- It did create a parallel version, too, but does not use it!
- Fix: use '-xinline=no' => expected speed-up!
- Better: keep the source for main() and your subroutines in different source files!

OpenMP Parallelization

```
 1 void
 2 mxv(int m,int n,double *a,double *b,double *c) {
 3
 4     int i, j;
 5     double sum;
 6     #pragma omp parallel for private(i,j,sum)
 7     for (i=0; i<m; i++) {
 8         sum = 0.0;
 9         for (j=0; j<n; j++)
10             sum += b[i*n+j]*c[j];
11         a[i] = sum;
12     }
13 }
```

```
cc -g -fast -xopenmp -xloopinfo -c mxv.c
"mxv.c", line 7: PARALLELIZED, user pragma used
"mxv.c", line 9: not parallelized, loop inside
OpenMP region
```

OpenMP Parallelization



OpenMP Parallelization - if-clause