Intervals and OpenMP:

Towards an Efficient Parallel Result-Verifying Nonlinear Solver

EWOMP 2003
Aachen University, Germany
September 24, 2003

Thomas Beelitz

Scientific Computing
University of Wuppertal
beelitz@math.uni-wuppertal.de

Basic branch–and–bound algorithm

function $\texttt{Check}([\mathbf{z}])$: /* Checks if the box $[\mathbf{z}]$ may contain a solution */

if $[\mathbf{z}]$ cannot be excluded from further consideration
then
  if $[\mathbf{z}]$ is small enough
  then
    add $[\mathbf{z}]$ to the list of possible solutions
  else
    split $[\mathbf{z}]$ into two subboxes $[\mathbf{z}^{(1)}]$, $[\mathbf{z}^{(2)}]$
    $\texttt{Check}([\mathbf{z}^{(1)}])$
    $\texttt{Check}([\mathbf{z}^{(2)}])$

Inter-box parallelism

$L_0 = \{$ the box $[\mathbf{z}]$ to be searched for singularities $\}$
for $k = 0, 1, \ldots,$ until $L_k = \emptyset$
    $L_{k+1} = \emptyset$
    for all boxes $[\mathbf{z}]$ in L_k
        if $[\mathbf{z}]$ cannot be excluded from further consideration
        then
            if $[\mathbf{z}]$ is small enough
            then
                add $[\mathbf{z}]$ to the list of the possible solutions
            else
                split $[\mathbf{z}]$ into two subboxes $[\mathbf{z}^{(\mathbf{1})}]$, $[\mathbf{z}^{(\mathbf{2})}]$
                add $[\mathbf{z}^{(\mathbf{1})}]$ and $[\mathbf{z}^{(\mathbf{2})}]$ to L_{k+1}

Inter-box parallelism

$L_0 = \{$ the box $[\mathbf{z}]$ to be searched for singularities $\}$
for $k = 0, 1, \ldots,$ until $L_k = \emptyset$
  $L_{k+1} = \emptyset$
  for all boxes $[\mathbf{z}]$ in L_k $\leftarrow$ parallelize this loop with **OpenMP**
    if $[\mathbf{z}]$ cannot be excluded from further consideration
    then
      if $[\mathbf{z}]$ is small enough
      then
        add $[\mathbf{z}]$ to the list of the possible solutions
      else
        split $[\mathbf{z}]$ into two subboxes $[\mathbf{z}^{(1)}]$, $[\mathbf{z}^{(2)}]$
        add $[\mathbf{z}^{(1)}]$ and $[\mathbf{z}^{(2)}]$ to L_{k+1}

Inter-box parallelism

$L_0 = \{$ the box $[\mathbf{z}]$ to be searched for singularities $\}$
for $k = 0, 1, \ldots,$ until $L_k = \emptyset$
  $L_{k+1} = \emptyset$
  for all boxes $[\mathbf{z}]$ in L_k $\leftarrow$ parallelize this loop with **OpenMP**
    if $[\mathbf{z}]$ cannot be excluded from further consideration
    then
      if $[\mathbf{z}]$ is small enough
      then
        add $[\mathbf{z}]$ to the list of the possible solutions
      else
        split $[\mathbf{z}]$ into two subboxes $[\mathbf{z}^{(1)}]$, $[\mathbf{z}^{(2)}]$
        add $[\mathbf{z}^{(1)}]$ and $[\mathbf{z}^{(2)}]$ to L_{k+1} $\leftarrow$ synchronization

Timings for solving a system with $n = 29$ unknowns

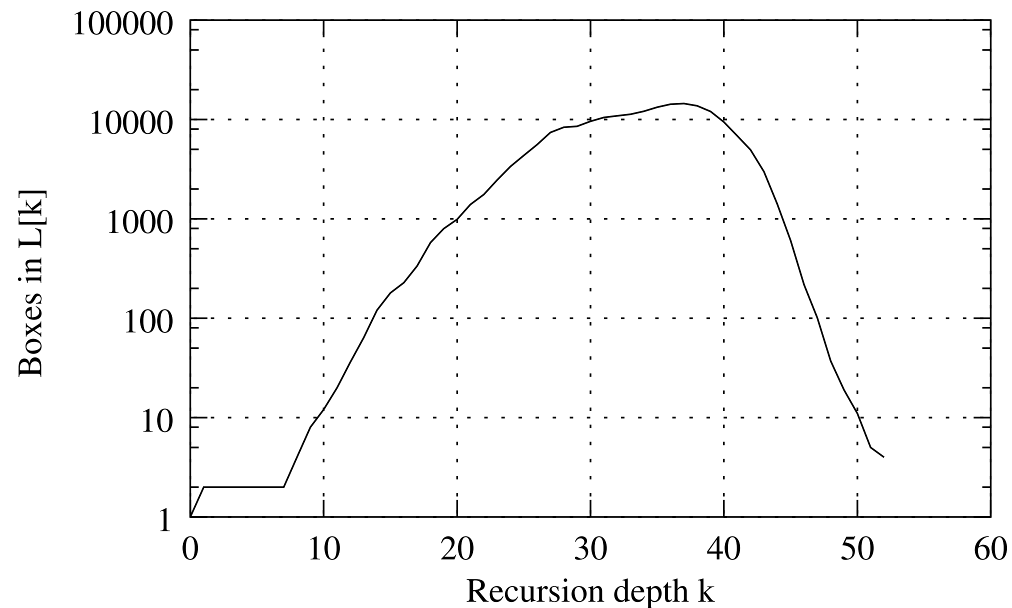
Program version	**#Procs**	**Time**	**Speedup**
Serial	1	17:32	
Serial, compiled with `-xopenmp`	1	21:18	
Parallel	1	22:55	1.00
Parallel, `schedule( static, 1 )`	2	13:52	1.65
Parallel, `schedule( static, 1 )`	3	11:20	2.02
Parallel, `schedule( static, 1 )`	4	10:46	2.13
Parallel, standard scheduling	4	11:41	1.96
Parallel, `schedule( dynamic, 1 )`	4	12:00	1.91

- compiled with version 5.5 of Sun C++ compiler
- performed on a Sun Fire 6800 server with 24 processors (900 MHz) and 24 GB of main memory

Possible limiting factors

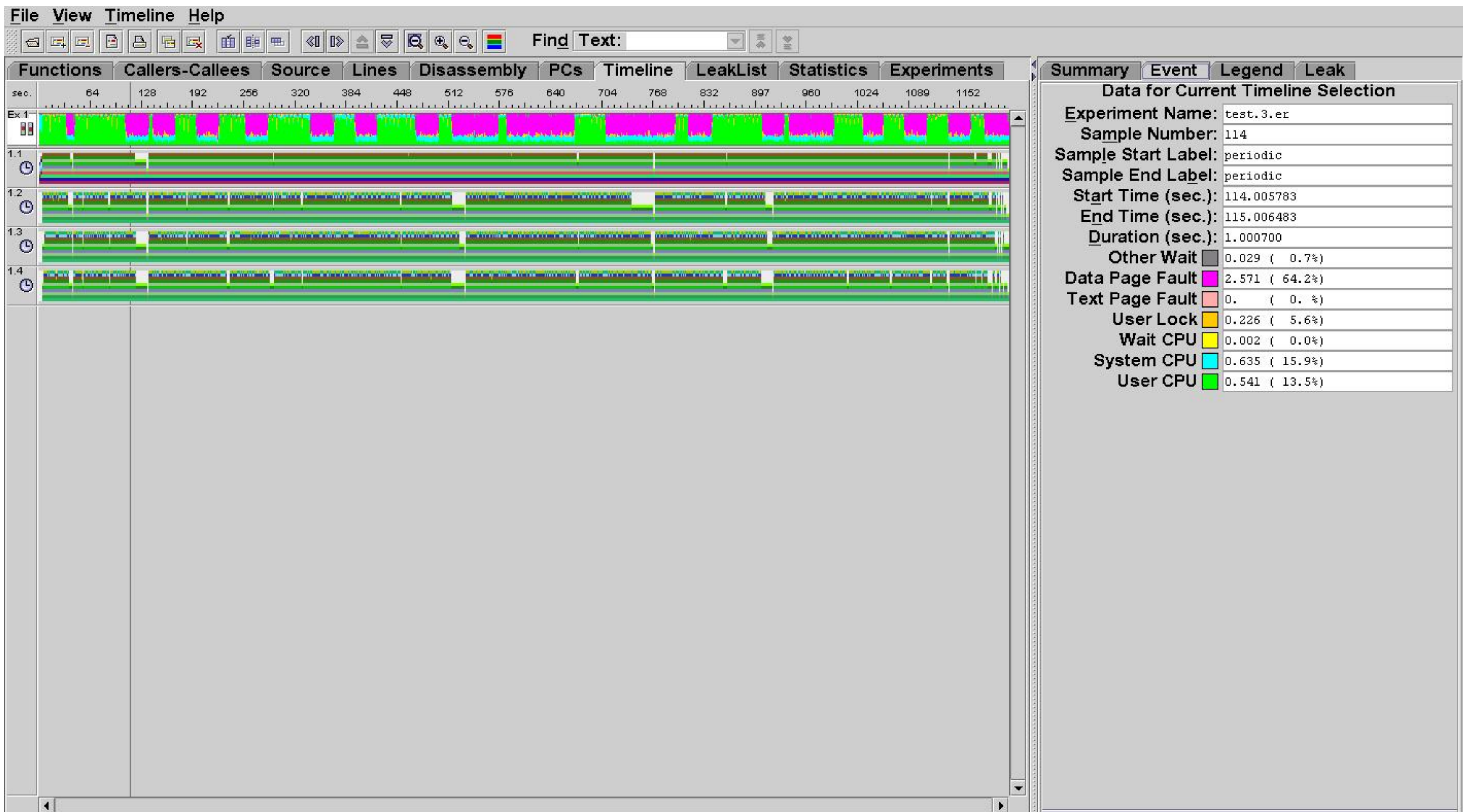
- **Granularity too small?**
  total number of boxes: $195\,381 \Rightarrow$ average of 5 milliseconds per box

- **Load imbalance?**



90.1% of boxes are handled at recursion levels $25, \ldots, 42$ (list length $> 4\,000$)

Performance Analyzer



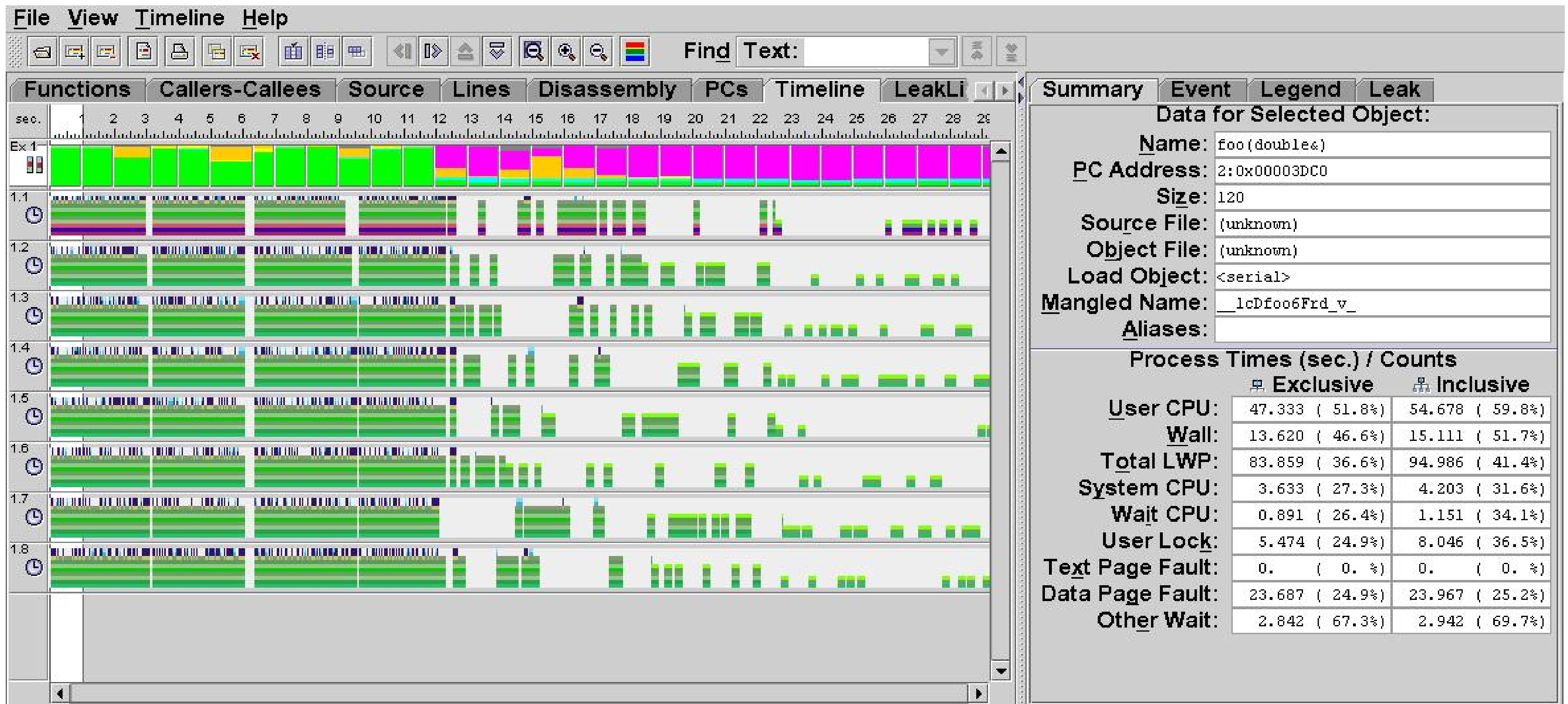
Thread-local objects – example1.cpp

```
#include <omp.h>
#include <iostream>

typedef double* vector;

void foo( double &sum ) {
    vector x = new double[ 2000 ];
    x[ 0 ] = 0.0;

    for ( int i=1; i<2000; i++ )
        x[ i ] = x[ i-1 ] + 1.0;
    sum += x[ 1999 ];

    delete[] x;
}

int main( void ) {
    int i;
    double sum = 0.0;

#pragma omp parallel for private( i ) shared( sum )
    for ( i=0; i<2500000; i++ )
        foo( sum );

    std::cout << sum << std::endl;
}
```

Version	P	Time [s]	Speedup
Serial	1	23.20	
Parallel	1	23.68	1.00
Parallel	2	14.05	1.68
Parallel	4	7.70	3.08
Parallel	8	10.23	2.31

compiled with 'CC -fast -xopenmp'

Performance Analyzer



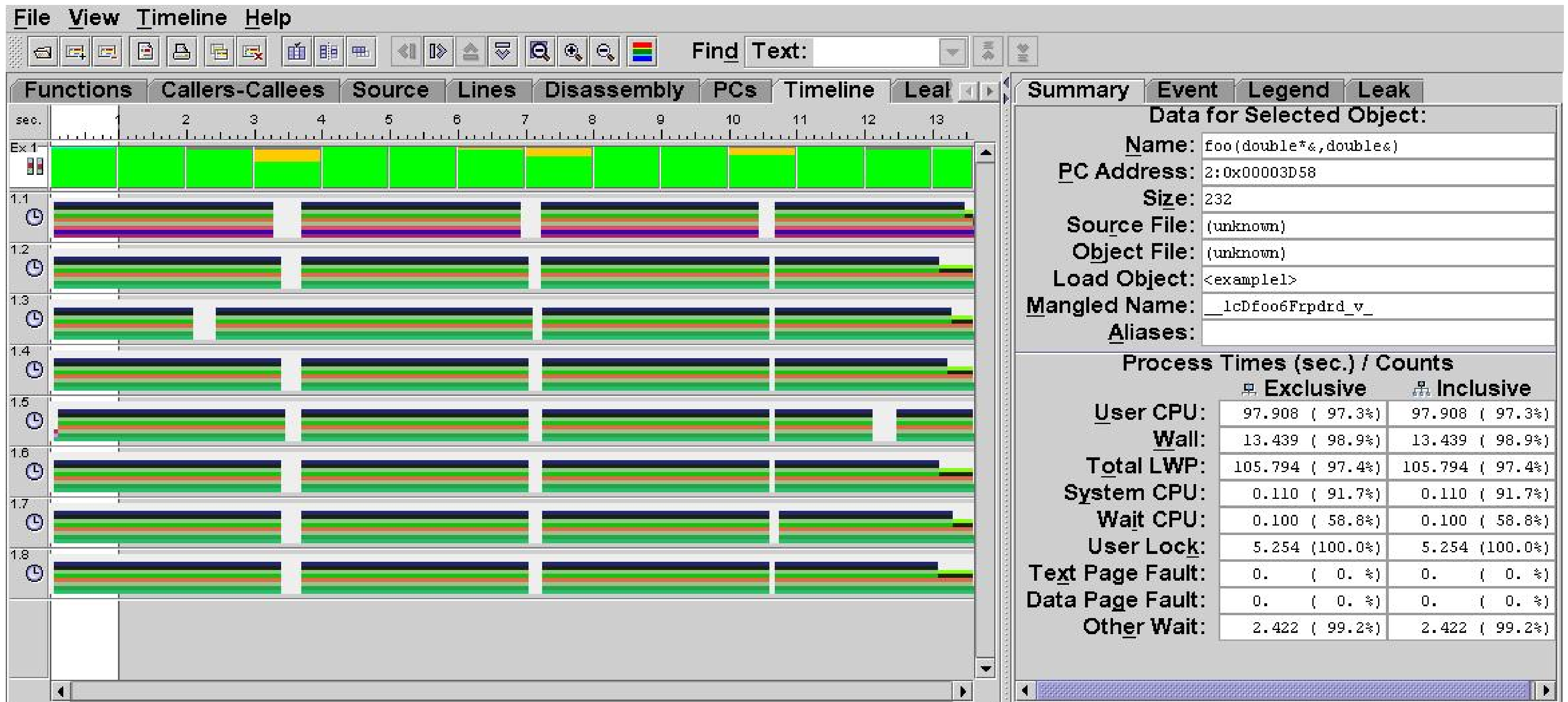
Thread-local objects – example2.cpp

```
#include <omp.h>
#include <iostream>

typedef double* vector;

void foo( vector &x, double &sum ) {
    for ( int i=1; i<2000; i++ )
        x[ i ] = x[ i-1 ] + 1.0;
    sum += x[ 1999 ];
}

int main( void ) {
    int i;
    double sum = 0.0;
#pragma omp parallel private( i ) shared( sum )
    {
        vector x = new double[ 2000 ];
        x[ 0 ] = 0.0;

#pragma omp for
        for ( i=0; i<2500000; i++ )
            foo( x, sum );

        delete[] x;

        std::cout << sum << std::endl;
    }
}
```

Version	P	Time [s]	Speedup
Serial	1	94.58	
Parallel	1	94.62	1.00
Parallel	2	47.85	1.97
Parallel	4	24.45	3.86
Parallel	8	12.24	7.73

compiled with 'CC -fast -xopenmp'

Performance Analyzer



example1.cpp – A matter of pre–allocation?

```
#include <omp.h>
#include <iostream>

typedef double* vector;

void foo( double &sum ) {
    vector x = new double[ 2000 ];
    x[ 0 ] = 0.0;

    for ( int i=1; i<2000; i++ )
        x[ i ] = x[ i-1 ] + 1.0;
    sum += x[ 1999 ];

    delete[] x;
}

int main( void ) {
    int i;
    double sum = 0.0;

#pragma omp parallel for private( i ) shared( sum )
    for ( i=0; i<2500000; i++ )
        foo( sum );

    std::cout << sum << std::endl;
}
```

Version	P	Time [s]	Speedup
Serial	1	23.20	
Parallel	1	23.68	1.00
Parallel	2	14.05	1.68
Parallel	4	7.70	3.08
Parallel	8	10.23	2.31

compiled with 'CC -fast -xopenmp'

example1.cpp – A matter of pre–allocation?

```
#include <omp.h>
#include <iostream>

typedef double* vector;

void foo( double &sum ) {
    vector x = new double[ 2000 ];
    x[ 0 ] = 0.0;

    for ( int i=1; i<2000; i++ )
        x[ i ] = x[ i-1 ] + 1.0;
    sum += x[ 1999 ];

    delete[] x;
}

int main( void ) {
    int i;
    double sum = 0.0;

#pragma omp parallel for private( i ) shared( sum )
    for ( i=0; i<2500000; i++ )
        foo( sum );

    std::cout << sum << std::endl;
}
```

Version	P	Time [s]	Speedup
Serial	1	23.20	
Parallel	1	23.68	1.00
Parallel	2	14.05	1.68
Parallel	4	7.70	3.08
Parallel	8	10.23	2.31

compiled with 'CC -fast -xopenmp'

Version	P	Time [s]	Speedup
Serial	1	94.58	
Parallel	1	94.62	1.00
Parallel	2	47.85	1.97
Parallel	4	24.45	3.86
Parallel	8	12.24	7.73

compiled with 'CC -xO3 -xopenmp'

Possible input from the lab session

- background information on Sun Compiler / OpenMP
- tests with other OpenMP implementations (our solver provides support for standard C++ together with the interval library C-XSC)
- analyses with diverse performance tools

Intervals and OpenMP:
Towards an Efficient Parallel Result-Verifying Nonlinear Solver

EWOMP 2003
Aachen University, Germany
September 24, 2003

Thomas Beelitz

Scientific Computing
University of Wuppertal
`beelitz@math.uni-wuppertal.de`