Strategies for improving the performance of single threaded codes on a CMT system

Darryl Gove
Compiler Performance Engineering
darryl.gove@sun.com

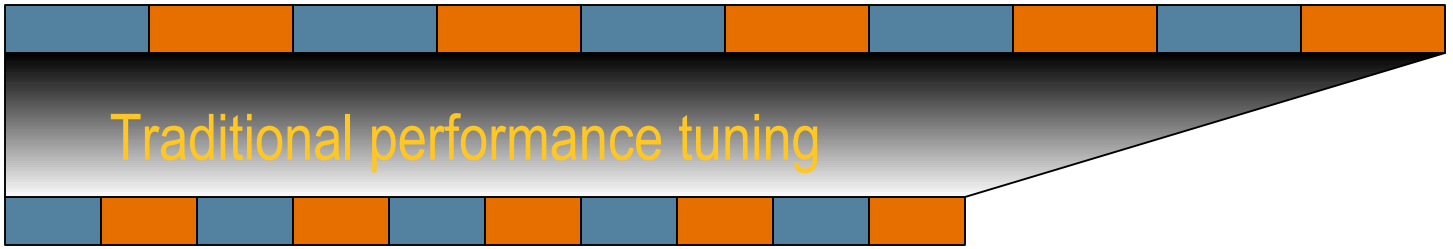
Outline

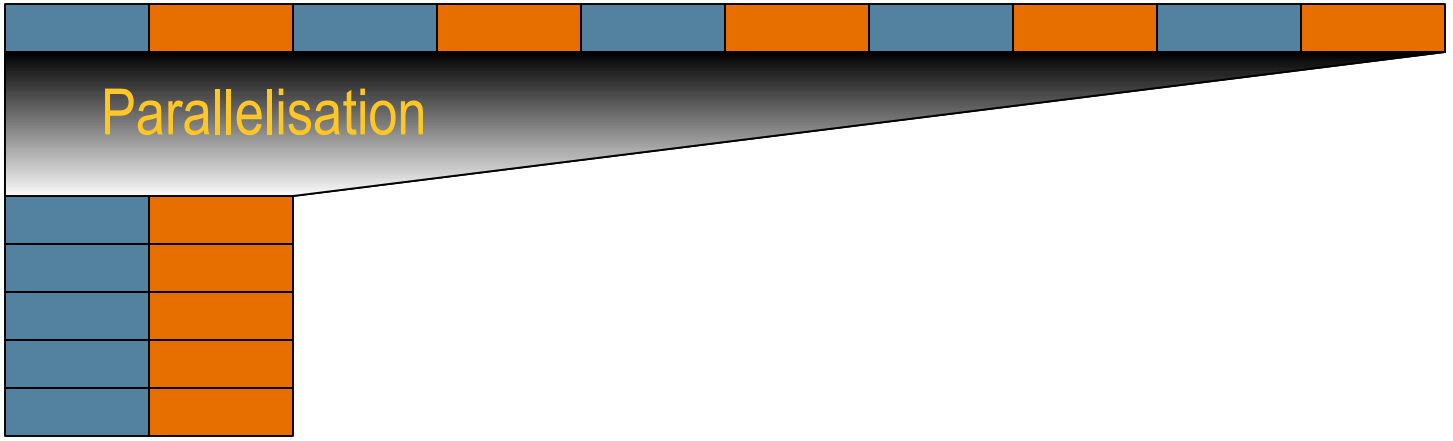
- Context
- Current options
- Example problem
- Atomic reduction
- Vector reduction
- Microparallelisation

Context – CMT processors

- CMT processors have:
  - Threads
  - Low synchronisation costs

Context - parallelisation





Current options

- Autoparallelisation
  - -xautopar -xreduction
  - Easy to use
  - Limited range of apps
- OpenMP
  - Some skill needed
  - Parallel for & sections
  - 3.0 introduces tasks
- Pthreads
  - Complex
  - Flexible

Reduction loop

```
for (i=1, i<n, i++)
{
 total += value[i];
}
```

More complex reduction

```
for (i=1, i<n, i++)
{
  j=index[i];
  result[j] += function(i,j);
}
```

Cannot be certain that the same index of result is not updated multiple times

OpenMP critical section

```
#pragma omp parallel for
for (i=1, i<n, i++)
{
   j=index[i];
   tmp = function(i,j);
   #pragma omp critical
   {
    result[j]+= tmp
   }
}
```

Atomic reduction

```
#pragma omp parallel for
for (i=1, i<n, i++)
{
   j=index[i];
   atomic_add(result[j], function(i,j));
}
```

Vector reduction (pseudo-code)

```
threadprivate tp_result=calloc(SIZE*sizeof(...))

#pragma omp parallel for
for (i=1, i<n, i++)
{
  j=index2[i];
  tp_result[j]+= function(i,j);
}

#pragma omp critical
{
  for (j=0; j<SIZE; j++)
  {
    result[j]+=tp_result[j];
  }
}
```

Microparallelisation

- Many threads, small chunks of work
  - Thread gets next chunk
  - Thread stalls until safe
  - Thread completes work
  - Update completed work
- Handles complex dependencies
- Issues:
  - Small chunks of work
  - Proportionally large overhead

Microparallelisation - splitting work

```
/*Get next iteration*/
while (my_iteration<total_iterations)
{
  this_iteration=
  compare_and_swap(my_iteration + 1,
                   my_iteration,
                   &current_iteration);
  if (this_iteration==my_iteration)
  {
   do_iteration(my_iteration);
  }
  else
  {
    my_iteration=this_iteration;
  }
}
```

Microparallelisation - check safety

```
/*Check that iteration can start*/
do
{
   ok=1;
   for (i=last_iteration+1; i<this_iteration; i++
   {
      if (index[i]==index[this_iteration])
      {
         ok=0;
      }
   }
}
while (!ok);
```

Do not start until no dependencies on older threads

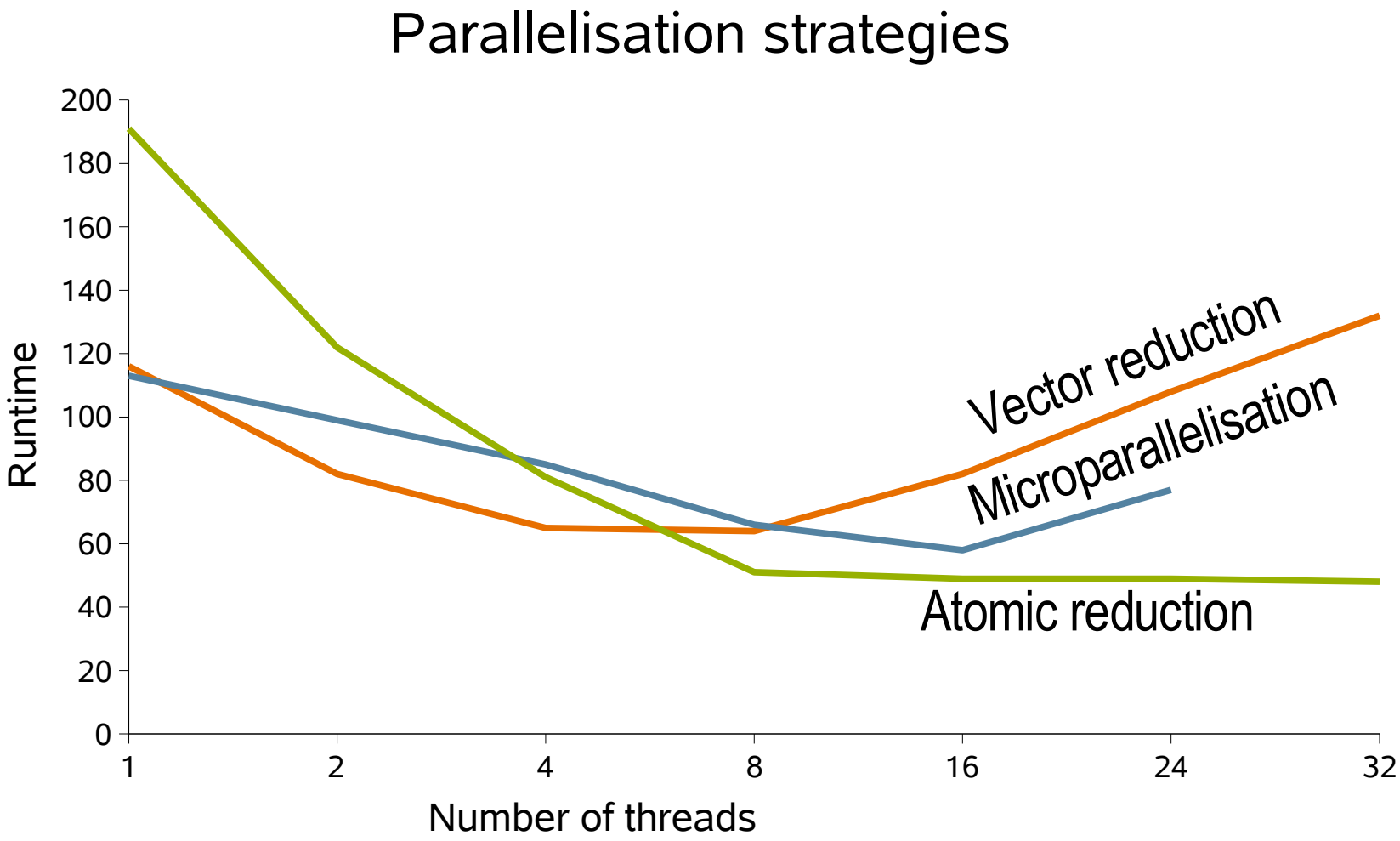
Microparallelisation – complete work

```
/*Do work*/

result[this_iteration]+=function(i,j);

/*Update completed iterations*/
while (compare_and_swap(this_iteration,
                        this_iteration-1,
                        &last_iteration)
       != this_iteration-1)
 {}
```

Benefits

Criteria	Atomic reduction	Vector reduction	Micro-parallelisation
Scales on SMP	No	Yes	No
Handles non-reductions	No	No	Yes
Low memory footprint	Yes	No	Yes

Results as graph



Runtimes from test code

Threads	Vector reduction	Micro-parallelisation	Atomic reduction
1	116	113	191
2	82	99	122
4	65	85	81
8	64	66	51
16	82	58	49
24	108	77	46
32	132		48
40			46

Conclusions

- Vector reduction wins
  - Good performance
  - Works on SMP systems
- Microparallelisation feasible
  - Handles dependency
  - But requires large iteration body

Strategies for improving the performance of single threaded codes on a CMT system

Darryl Gove
darryl.gove@sun.com
http://blogs.sun.com/d/