CGT 581I - Parallel Graphics and Simulation

Knights Landing Tasks and Threads

Bedrich Benes, Ph.D.
Professor
Department of Computer Graphics
Purdue University

Tasks and Threads

- Use
  - OpenMP
  - Threading Building Blocks (TBB)
  - Intel Math Kernel Library (MKL)
- Do not use pthreads (unless necessary)
  - Similar to program in assembly language
  - Can be very efficient
  - But huge programming overhead

© Bedrich Benes

Tasks and Threads

- Think in tasks not threads
- Threads are mapped by the API
- OpenMP, MKL, TBB all use pthreads (Linux) and threads (Windows)

© Bedrich Benes

What? When?

Library/Language	C	C++
OpenMP	Excellent	Good
TBB	Excellent	Excellent

© Bedrich Benes

Thread pools

- Creating/destroying threads is expensive
- Create threads only once and reuse them
- Automatically by OpenMP and TBB
- They also have good schedulers and load balancing
- Leave some threads for the OS

© Bedrich Benes

Nesting and Hot Teams

- Nesting:
  - Creating threads inside threads
  - May be too many threads
- OpenMP default: no nesting
- env variable: `OMP_NESTED TRUE/FALSE`
- C/C++ : `omp_set_nested(1); //enable`

© Bedrich Benes

omp nesting

```
#include <omp.h>
int main(){
 int i,n;
 omp_set_num_threads(8);
 omp_set_nested(1); // enabled
 #pragma omp parallel{ // parallel
   printf("tid:%d\n", omp_get_thread_num());
   #pragma omp parallel num_threads(2){
    printf("tid:%d\n", omp_get_thread_num());
    #pragma omp for
    for(i=0;i<n;i++)//work here
   }
  }
}
```

© Bedrich Benes

Nesting and Hot Teams

- Intel has an omp extension called hot teams
- `KMP_HOT_TEAMS_MODE=1`
- `KMP_HOT_TEAMS_MAX_LEVEL=2`

- Keeps the thread alive but idle
- Avoids overhead for creation
- Handled only from the environment

© Bedrich Benes

Intel Threading Building Blocks

- portable template C++ library
- concurrency-safe STL-compatible data structures, memory allocations, atomics
- task scheduler, efficient for embedded algorithms such as `tbb::parallel_for`

© Bedrich Benes

TBB

- no compiler changes – fully portable
- no need for compiler support
- low-level services (malloc, atomics)
- open source project

```
#include <tbb/tbb.h>
using namespace tbb;
```

© Bedrich Benes

tbb::parallel_for

maps a functor across a range of values

```
tbb::parallel_for(first,last,f)
```

where `f` is the functor

it is a parallel version of:

```
for (auto i=first;i<last;i++)
    f(i);
```

© Bedrich Benes

tbb::parallel_for

```
#include <iostream>
#include <tbb/tbb.h>
using namespace tbb;
class Hello{
public:
void operator()(int x) const {
    std::cout << "Hello" << x << std::endl;
   }
};
```

© Bedrich Benes

tbb::parallel_for

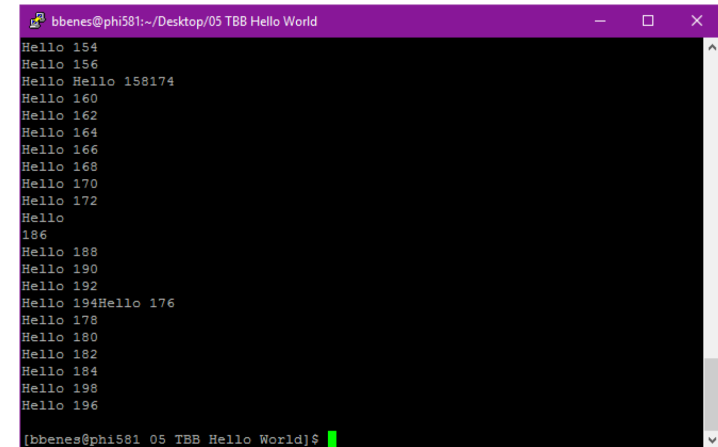
```
int main()
{
    tbb::parallel_for(0, 200, 1, Hello());
    return 0;
}
```

Compile:

```
icpc main.cpp -o main -ltbb
```

© Bedrich Benes

tbb::parallel_for



© Bedrich Benes

tbb::parallel_for - stride

maps a functor across a range of values

```
tbb::parallel_for(first,last,stride,f)
```

where f is the functor

it is a parallel version of:

```
for (auto i=first;i<last;i+=stride)
    f(i);
```

© Bedrich Benes

tbb::blocked_range

defines range of values

```
tbb::blocked_range<int>(-2,10)
```

will be range of -2,-1,0,…,9

Can be used with `parallel_for`

© Bedrich Benes

PURDUE UNIVERSITY

tbb::parallel_for - rabge

maps a functor across a range of values

`tbb::parallel_for(range,f)`

where f is the functor

Decomposes `range` into subranges and applies `f` in parallel

Subrange is sequential

© Bedrich Benes

PURDUE UNIVERSITY

tbb::blocked_range

```
#include <iostream>
#include <vector>
#include <tbb/parallel_for.h>
#include <tbb/blocked_range.h>

std::vector<double> dta(102400, 0);
```

© Bedrich Benes

PURDUE UNIVERSITY

tbb::blocked_range

```
class IncrDta{
public:
   void operator()(const
tbb::blocked_range<size_t>& range) const
    {
 for (size_t i = range.begin();
      i < range.end(); ++i)
       dta[i] += 1; //just increase the #
    }
};
```

© Bedrich Benes

PURDUE UNIVERSITY

tbb::blocked_range

```
int main()
{
  blocked_range<size_t> range(0, dta.size());
  parallel_for(range, IncrDta());
}
```

© Bedrich Benes

PURDUE UNIVERSITY intel

tbb::task_scheduler_init

```
int n =
task_scheduler_init::default_num_threads();
//gets the default number of threads

//initialize with n
tbb::task_scheduler_init init(n);
```

© Bedrich Benes

PURDUE UNIVERSITY intel

Partitioners

Partitioners will optimize performance by dividing the range

```
auto_partitioner
simple_paritioner
affinity_partitioner
```

e.g.,

```
parallel_for(range, f, simple_partitioner());
```

© Bedrich Benes

PURDUE UNIVERSITY intel

Partitioners

`auto_partitioner`
divide to balance the load, but not more

`simple_paritioner`
divide as small as possible

`affinity_partitioner`
use previously invoked `parallel_for/reduce` pattern

© Bedrich Benes

PURDUE UNIVERSITY intel

tbb::parallel_reduce

`parallel_reduce`

is… well… parallel reduce

© Bedrich Benes

tbb::parallel_reduce - example

```
#include <iostream>
#include "tbb/parallel_reduce.h"
#include "tbb/blocked_range.h"
using namespace tbb;
```

© Bedrich Benes

tbb::parallel_reduce

```
class Sum{
public:
 float val;
 Sum() : val(0) {} //default sum is 0
 Sum( Sum& s, split ) {val = 0;}
 void operator()( const blocked_range<float*>& r ) {
     float tmp = val;
     for( float* a=r.begin(); a!=r.end(); ++a )
        tmp += *a;
     val = tmp;
    }
    void join( Sum& rhs ) {val += rhs.val;}
};
```

© Bedrich Benes

tbb::parallel_reduce

```
float ParallelSum( float a[], size_t n ) {
 Sum total;
 parallel_reduce( blocked_range<float*>
      ( a, a+n ), total );
 return total.val;
}
```

© Bedrich Benes

tbb::parallel_reduce

```
void main(){
  const int max = 1024000;
  float dta[max];

  for (int i=0;i<max;i++) dta[i]=1.f/(i+1);

  std::cout << "The sum is " <<
  ParallelSum(dta, max) << std::endl;

}
```

© Bedrich Benes

tbb::parallel_reduce

```
bbenes@phi581:~/Desktop/07 TBB Parallel Reduce
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.4086
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.411
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.4163
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.4086
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.4181
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.4143
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.4147
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.4139
[bbenes@phi581 07 TBB Parallel Reduce]$ ./main
The sum is 14.4139
[bbenes@phi581 07 TBB Parallel Reduce]$
```

© Bedrich Benes

tbb::parallel_reduce

the same with std::vector

```
#include <iostream>
#include <algorithm>
#include <numeric>
#include <tbb/tbb.h>
```

© Bedrich Benes

tbb::parallel_reduce

```
class Sum {
public:
 int val;
 Sum() : val(0) {}
 Sum(Sum&s, tbb::split) : val (0) {}
void operator()(const
tbb::blocked_range<std::vector<int>::iterator>& r) {
 val = std::accumulate(r.begin(), r.end(), 0);
}
 void join(Sum& rhs) { val += rhs.val; }
};
```

© Bedrich Benes

tbb::parallel_reduce

```
int main(){
 std::vector<int> a(100);
 std::fill(a.begin(), a.end(), 1);
 Sum sum;
tbb::parallel_reduce(
tbb::blocked_range<std::vector<int>::iterator>
  (a.begin(), a.end()), sum);
 std::cout << sum.val << std::endl;
 return 0;
}
```

© Bedrich Benes

tbb::parallel_invoke

```
tbb::parallel_invoke(f1(), f2(), f3())
```

evaluates `f1()`, `f2()`, and `f3()` in parallel

and waits for the completion

2-10 functions are supported

© Bedrich Benes

tbb::parallel_invoke

Example:

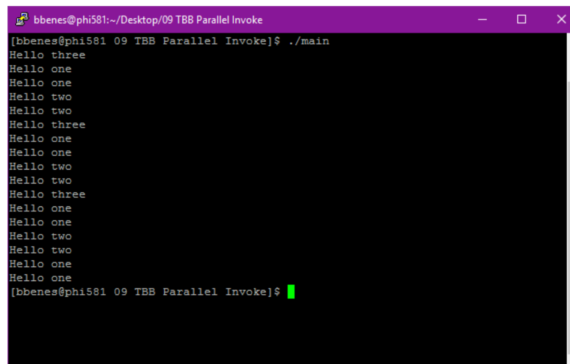
```
#include  <iostream>
#include <tbb/tbb.h>

void f1(){std::cout << "Hello one\n";}
void f2(){std::cout << "Hello two\n";}
void f3(){
 std::cout << "Hello three\n";
 tbb::parallel_invoke(f1, f1, f2, f2);//nested
}
```

© Bedrich Benes

tbb::parallel_invoke

```
int main()
{
    tbb::parallel_invoke(f3, f3, f3, f1, f1);
    return 0;
}
```



© Bedrich Benes

tbb flow graph

tbb supports graph-like communication and synchronization

Tasks are coordinated via message passing

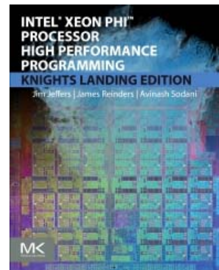
See more

https://www.threadingbuildingblocks.org/tutorial-intel-tbb-flow-graph

© Bedrich Benes

Reading

- Intel Xeon Phi Processor High Performance Programming Knights Landing Edition
- James Jeffers, James Reinders, and Avinash Sodani
- ISBN: 9780128091944
- Morgan Kaufmann
- Chapter 8



© Bedrich Benes