Using OpenMP with C++ in the Real World

Christian Terboven
terboven@rz.rwth-aachen.de

Center for Computing and Communication
RWTH Aachen University, Germany

Texas A&M University
November 25, College Station, TX, USA
Using OpenMP with C++ in the Real World

**25.11.2008 – C. Terboven**

**Agenda**

- Nested Parallelization
  - FIRE: Pattern Recognition
  - NestedCP: Computation of Critical Points
- OpenMP and "real" C++
  - DROPS: Navier-Stokes Solver
  - VRFEM: Realtime FEM for VR
- Challenges for OpenMP
- Conclusion
FIRE: Image Retrieval System

- FIRE = Flexible Image Retrieval Engine
  - Compare the performance of common features on different databases
  - Analysis of correlation of different features

Thomas Deselaers and Daniel Keysers,
RWTH I6: Chair for Human Language Technology and Pattern Recognition
FIRE: Image Retrieval System

\[ D(Q, X) := \sum_{m=1}^{M} w_m \cdot d_m(Q_m, X_m) \]

- Q: query image, X: set of database images
- Q_m, X_m: m-th feature of Q and X
- d_m: distance measure, w_m: weighting coefficient
- Return the k images with lowest distance to query image

- Well-suited for Shared-Memory parallelization: Data Mining in a large image database!

- Three levels to exploit parallelism:
  - Process multiple query images in parallel
  - Process database comparison for one query image in parallel
  - Computation of distances might be parallelized as well
FIRE: Nested OpenMP improves scalability

### Speedup Sun Fire E25K, 72 dual-core UltraSPARC-IV processors

<table>
<thead>
<tr>
<th># Threads</th>
<th>Only outer level</th>
<th>Only inner level</th>
<th>Nested OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>3.88</td>
<td>3.63</td>
<td>3.93</td>
</tr>
<tr>
<td>8</td>
<td>6.98</td>
<td>7.63</td>
<td>7.65</td>
</tr>
<tr>
<td>16</td>
<td>12.46</td>
<td>15.09</td>
<td>15.12</td>
</tr>
<tr>
<td>32</td>
<td>25.97</td>
<td>23.69</td>
<td>28.45</td>
</tr>
<tr>
<td>144</td>
<td></td>
<td></td>
<td>133.3</td>
</tr>
</tbody>
</table>

○ How can Nested OpenMP improve the scalability?
  – Scalability on outer level is limited because of output sync.
  – OpenMP overhead increases with the number of threads
  – Dataset might better fit to the number of threads
FIRE: Tasks outperforms Nested OpenMP

- Tasking with OpenMP 3.0 is even better: Barrier at the end of inner Parallel Region can be skipped!

Sun Fire X4600: 8s 2c AMD Opteron 885 @ 2.6 GHz

- Speedup vs. # threads graph
NestedCP: Parallel Critical Point Extraction

- VR in Aachen: Analysis of large-scale flow simulations
  - Feature extraction from raw data
  - Interactive analysis in virtual environment (e.g. a cave)
- Critical Point: Point in the vector field with zero velocity
NestedCP: Addressing Load Imbalance

- Algorithmic sketch of Critical Point extraction:
  - Loop over the time steps of unsteady datasets
  - Loop over the blocks of multi-block datasets
  - Loop checking the cells within the block for CP

- The time needed to check a cell may vary considerably!
NestedCP: Addressing Load Imbalance

- Solution in OpenMP is rather simple:

```c++
#pragma omp parallel for num_threads(nTimeThreads) \ schedule(dynamic,1)
for (cutT = 1; curT <= maxT; ++curT)
{
  #pragma omp parallel for num_threads(nBlockThreads) \ schedule(dynamic,1)
  for (curB = 1; curB <= maxB; ++curB)
  {
    #pragma omp parallel for num_threads(nCellThreads) \ schedule(guided)
    for (curC = 1; curC <= maxC; ++curC)
    {
      findCriticalPoints(curT, curB, curC);
    }
  }
}
```
NestedCP: Addressing Load Imbalance

- Speedup on Sun Fire E25k, 72 dual-core UltraSPARC-IV processors, execution with 128 threads:
  - Without load balancing: 10.3 (static scheduling)
  - With load balancing: 33.9 (dynamic scheduling)
  - Sun extension to guided sched.: 55.3 (weight factor = 20)

- The achievable speedup heavily depends on the dataset

- No load imbalance → almost perfect scalability
Using OpenMP with C++ in the Real World

25.11.2008 – C. Terboven

Agenda

- Nested Parallelization
  - FIRE: Pattern Recognition
  - NestedCP: Computation of Critical Points

- OpenMP and „real“ C++
  - DROPS: Navier-Stokes Solver
  - VRFEM: Realtime FEM for VR

- Challenges for OpenMP

- Conclusion
DROS: A Navier-Stokes Solver in C++

- Numerical Simulation of two-phase flow
- Modeled by instationary and non-linear Navier-Stokes equation
- Level Set function is used to describe the interface between the two phases
- Written in C++: is object-oriented, uses nested templates, uses STL types, uses compile-time polymorphism, ...
  - Solver achieves the same performance as same algorithm implemented in Fortran!
- (Adaptive) Tetrahedral Grid Hierarchy
- Finite Element Method (FEM)

Example: Silicon oil drop in D₂O (fluid/fluid)
DROPS: Parallelization of C++ Code

```cpp
PCG(const Mat& A, Vec& x, const Vec& b,
    const PreCon& M, int& max_iter,
    double& tol)
{
    Vec p(n), z(n), q(n), r(n);
    [...]
    for (int i=1; i<=max_iter; ++i) {
        [...]
        q = A * p;
        alpha = rho / (p*q);
        x += alpha * p;
        r -= alpha * q;
        [...]
    }
    y_Ax_par(&q.raw()[0],
              A.num_rows(), A.raw_val(),
              A.raw_row(), A.raw_col(),
              Addr( p.raw()));
    #pragma omp for reduction
    (+:alpha_sum)
    for (long j=0; j<n; j++)
        alpha_sum += p[j]*q[j];
    #pragma omp single {
        alpha = rho/alpha_sum;
    }
    #pragma omp for
    for (long j=0; j<n; j++)
        x[j] += alpha * p[j];
    #pragma omp for
    for (long j=0; j<n; j++)
        r[j] -= alpha * q[j];
}
```

- **Problems:**
  - Code Changes
  - Parallelization introduces additional overhead
PCG(const Mat& A, Vec& x, const Vec& b, const PreCon& M, int& max_iter, double& tol) {
    Vec p(n), z(n), q(n), r(n);
    [...]
    for (int i=1; i<=max_iter; ++i) {
        [...]
        q = A * p;
        alpha = rho / (p*q);
        x += alpha * p;
        r -= alpha * q;
        [...]
        y_Ax_par(&q.raw()[0], A.num_rows(), A.raw_val(), A.raw_row(), A.raw_col(),
                 Addr( p.raw()));

        #pragma omp for reduction (+:alpha_sum)
        for (long j=0; j<n; j++)
            alpha_sum += p[j]*q[j];

        #pragma omp single {
            alpha = rho/alpha_sum;
        }
        #pragma omp for
        for (long j=0; j<n; j++)
            x[j] += alpha * p[j];

        #pragma omp for
        for (long j=0; j<n; j++)
            r[j] -= alpha * q[j];
    }
}

DROPS: Parallelization of C++ Code

- Problems:
  - Code Changes
  - Parallelization introduces additional overhead

DROPS developers would not accept these code changes!
Parallelization Work may lead to a Dead End

- Parallelization efforts may lead to Diverging Code Versions:

DROPS: 3 developers work on the serial version, ½ developer works on MPI version

Modified Application

(tuning + parallelization)

Will they accept the modifications?

Modified Parallel Application

Continued Development

(porting)

Application

Modified Application

??? merge ???
A better approach?

- Specific strength of OpenMP: Incremental Parallelization!
- Goal: Find (parallelizable) Patterns or Kernels, that
  - can be used in ongoing programm development
  - can be tuned / parallelized independently
  - C++ → classes (abstraction vehicle)
- Based on the experience with DROPS, two codes of the Virtual Reality group, and other applications: laperf
  - C++ template library
  - Easily applicable (source code compatibility)
  - Offers a broad range of numeric data types and operations
  - Our research: Evaluate and Improve Parallel Programming Paradigms and Techniques + Ease of Use
Parallelization: New Approach with laperf

- All one needs to do is: Replace
  ```
typedef VectorBaseCL<double> VectorCL;
typedef SparseMatBaseCL<double> MatrixCL;
  ```
  with
  ```
typedef laperf::vector<double,
                   OpenMPIInternalParallelization> VectorCL;
  ```
  ```
typedef laperf::matrix_crs<double> MatrixCL;
  ```

- Extend existing abstractions to introduce parallelism!

- Currently supported parallelization types:
  - NoParallelization
  - OpenMPIInternalParallelization
Data Type Design

- Abstract data type for vector + CRS-matrix implementation to
  - Evaluate Parallelization Paradigms
  - Hide Parallelization from the User

- We extended the data types by adding
  - Two (optional) template parameters:
    - Parallelization: parallelization strategy
    - Alloc: STL-type allocator (cc-NUMA optimization)
  - One (optional) constructor argument:
    - Scheduling: Specify how work is distributed to the threads

- Our approach is easy to use and delivers full performance
  - while hiding the parallelization from the user
  - and hiding cc-NUMA tuning from the user.
Handling of cc-NUMA architectures

- All x86-based multi-socket system will be cc-NUMA!
  - Current Operating Systems apply first-touch placement.
  - If cc-NUMA is ignored, the speedup will be zero, typically.

- STL provides the concept of an allocator to encapsulate memory management
  → build on the same concept to optimize for cc-NUMA

- We created two allocators:
  - Can optionally be plugged into our data types
  - `dist_allocator`: Distribute data according to OpenMP schedule type (same scheduling as in computation)
  - `chunked_allocator`: Distribute data according to explicitly precalculated scheme to improve load balancing
Using OpenMP with C++ in the Real World

(Preliminary) Performance Measurements

- laperf-dev w/ GMRES Solver on 4-socket 2-core Opteron:

  - cc-NUMA architecture provides good memory bandwidth, if used correctly!

  ![Graph showing performance measurements](image-url)
Physically based simulation is indispensable component of many interactive virtual environments.

**Main challenge:** Realtime.

Higher computation costs than methods typically used in e.g. computer games.

- Single precision where possible,
- Double precision where needed.

Realtime cannot be achieved using sequential approaches:
No further (significant) improvements of single thread performance expected!

Lenka Jerabkova, Virtual Reality Center, RWTH Aachen
VRFEM: Speedup on Multicore Architectures

- The presented algorithm has been parallelized with focus on recent multicore architectures
  - Red bar: Realtime requirement. Has been reached on two-socket quad-core (Clovertown) system (pizza box!)

![Graph showing time vs. threads for different processors](image)

<table>
<thead>
<tr>
<th>#threads</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>time [s]</td>
<td>38</td>
<td>20</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

12500 FE mesh elements
Using OpenMP with C++ in the Real World

25.11.2008 – C. Terboven

**Agenda**

- Nested Parallelization
  - FIRE: Pattern Recognition
  - NestedCP: Computation of Critical Points
- OpenMP and „real“ C++
  - DROPS: Navier-Stokes Solver
  - VRFEM: Realtime FEM for VR
- Challenges for OpenMP
- Conclusion
OpenMP 3.0 fixed several C++ issues

- Improved applicability of `for`-Worksharing:
  - Loops over RandomAccessIterators have been allowed for the following operators: `< <= > >=
  - Loops over signed *and* unsigned integer variables have been allowed

- Thread privatization of static class member has been allowed
  - Use cases: Thread-specific Allocators, Singleton

- Lifetime and Initialization of non-POD types has been specified for `private, firstprivate, lastprivate, threadprivate, threadprivate+copyin, threadprivate+copyprivate` use cases
Can we bring OpenMP nearer to the language?

- Programmers prefer library-based approaches over compiler-based approaches:

```c++
#pragma omp parallel for
for (int i = 0; i < iNumElements; i++) {
    vec[i] = compute(vec[i], iNumIterations);
}
```

- How could the code look like without pragmas?

```c++
omp_pfor (0, iNumElements, [&] (int i) {
    vec[i] = compute(vec[i], iNumIterations);
});
```

- C++0x brings us lambda functions!
Implementing `omp_pfor` with lambda functions

- Making OpenMP’s pragma invisible:

  ```cpp
  template<typename F>
  void omp_pfor(int start, int end, F x) {
    #pragma omp parallel for
    for(int __i = start; __i < end; __i++) {
      x(__i);
    }
  }
  ```

- Currently in consideration:
  - How to bring OpenMP’s scheduling capabilities in there?
  - How to deal with various scoping issues? (→ attributes)

Experiments done together with Christopher Schleiden, student worker at the Computing Center.
○ What is really missing (my point of view):
  – Provide means to control thread and data affinity
    • Thread binding is simple to use, but specific to Operating System
    • Binding of OpenMP 3.0 tasks is still an open issue
    • cc-NUMA architecture: Data placement is crucial for performance
  – Define interaction with native threading models
    • C/C++0x: Compatibility of Memory Model
    • Fortran 20??: Compatibility of Memory Model
    • Define ways to pass a team or sub-team of threads as argument to a function call (→ library writers)
  – Integrate OpenMP better into the language
    • C++0x: Better binding based on Lambdas+Attributes+Concepts?
    • Solve some simple but long outstanding issues:
      – Reduction for non-POD datatypes and arrays
Agenda

- Nested Parallelization
  - FIRE: Pattern Recognition
  - NestedCP: Computation of Critical Points

- OpenMP and "real" C++
  - DROPS: Navier-Stokes Solver
  - VRFEM: Realtime FEM for VR

- Challenges for OpenMP

- Conclusion
Conclusion

- OpenMP can be an alternative to MPI, as parallelization might require less work!
  - Adding levels of parallelism can help to increase scalability!

- A clean C++ object-oriented coding style may be very helpful for OpenMP parallelization:
  - Encapsulation prohibits unintended data dependencies.
  - Encapsulation may improve data locality (think ccNUMA).

- OpenMP is useful for multi-core architectures!
  - But: Current support for C++ is limited.
  - But: Support for architecture aspects is still missing.

- Tasking concept in OpenMP 3.0 will greatly enhance usability!
Thank you for your attention!