Designing Efficient Libraries

Alexander Stepanov

July 21, 2003
What is STL?

STL is large, systematic, clean, formally sound, comprehensible, elegant, and efficient framework

*Bjarne Stroustrup, AT&T*

STL looks like the machine language macro library of an anally retentive assembly language programmer

*Pamela Seymour, Leiden University*
Design goals

- Well structured, comprehensive library of useful components
- Every component is as abstract as theoretically possible and as efficient as its hand-coded, non-abstract version in C
How fast is fast?

http://theory.stanford.edu/~amitp/rants/c++-vs-c/

<table>
<thead>
<tr>
<th>Data type</th>
<th>qsort</th>
<th>hand coded</th>
<th>Numerical Recipes</th>
<th>STL</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>5.90 - 5.92</td>
<td>1.54 - 1.65</td>
<td>1.46 - 1.50</td>
<td>1.11 - 1.14</td>
</tr>
<tr>
<td>short</td>
<td>9.03 - 9.03</td>
<td>1.73 - 1.80</td>
<td>1.58 - 1.59</td>
<td>1.17 - 1.19</td>
</tr>
<tr>
<td>byte</td>
<td>7.87 - 7.89</td>
<td>0.98 - 1.02</td>
<td>0.98 - 1.00</td>
<td>0.70 - 0.73</td>
</tr>
<tr>
<td>float</td>
<td>7.08 - 7.10</td>
<td>2.38 - 2.50</td>
<td>2.48 - 2.55</td>
<td>1.97 - 2.02</td>
</tr>
<tr>
<td>double</td>
<td>16.4 - 16.4</td>
<td>2.70 - 2.93</td>
<td>2.72 - 2.83</td>
<td>2.28 - 2.37</td>
</tr>
</tbody>
</table>
Lightweight interfaces

```c
int array[1000];
...
sort(array, array + 1000);

// use only parts you need
// works with C arrays
```
Ability to customize

// need descending order?

sort(array, array + 1000,
greater<int>());

// need to sort the second half only?

sort(array + 500, array + 1000);
Many related algorithms

- partial_sort, partial_sort_copy
  - find first 10 out of 1000
- stable_sort
  - sort by name, then by department
- min_element, max_element, nth_element
Complexity specifications

- Operation counts for algorithms
- Asymptotic complexity at the interface level

(see http://www.sgi.com/tech/stl/
in particular,
Controversial points

- not Object Oriented
- Copy semantics
- Unsafe
Performance pitfall 1

```cpp
vector<Record> v;
Record new_record;
while (get_record(new_record)) {
    v.reserve(v.size() + 1);
    v.push_back(new_record);
}
```
Performance pitfall 2

deque<double> d(10000000);
sort (d.begin(), d.end());
Bizarre algorithms

template <class Iter>
void sort(Iter f, Iter l) {
    while(next_permutation(f, l));
}

template <class Iter>
void maybe_sort(Iter f, Iter l) {
    while(!is_sorted(f, l))
        random_shuffle(f, l);
}
Conclusions

- To get performance, design for performance
- Performance tools require study and thinking
- Poor performance could mean sloppy design