Subject: Funny way how NOT to speedup sorting of small arrays
Posted by mirek on Sun, 01 Feb 2009 10:30:52 GMT
View Forum Message <> Reply to Message

Well, I got a nice idea that does not work, but I want to share anyway.

One of critical issues in polygon rasterizer (which I could not resist to work on in the end) is the scanline x positions sorting.

Usually the number of elements to sort is quite low. So I was thinking about optimizing selection sort:

```
template <class I, class Less>
inline I SelectMin2(I ptr, const Less& less)
{
 return less(ptr[0], ptr[1]) ? ptr : ptr + 1;
}

template <class I, class Less>
inline I SelectMin3(I ptr, const Less& less)
{
 I l = SelectMin2(ptr, less);
 I h = ptr + 2;
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin4(I ptr, const Less& less)
{
 I l = SelectMin2(ptr, less);
 I h = SelectMin2(ptr + 2, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin5(I ptr, const Less& less)
{
 I l = SelectMin4(ptr, less);
 I h = ptr + 4;
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin6(I ptr, const Less& less)
{
 I l = SelectMin4(ptr, less);
 I h = SelectMin2(ptr + 4, less);
```

```
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin7(I ptr, const Less& less)
{
 I l = SelectMin4(ptr, less);
 I h = SelectMin3(ptr + 4, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin8(I ptr, const Less& less)
{
 I l = SelectMin4(ptr, less);
 I h = SelectMin4(ptr + 4, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin9(I ptr, const Less& less)
{
 I l = SelectMin8(ptr, less);
 I h = ptr + 8;
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin10(I ptr, const Less& less)
{
 I l = SelectMin8(ptr, less);
 I h = SelectMin2(ptr + 8, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin11(I ptr, const Less& less)
{
 I l = SelectMin8(ptr, less);
 I h = SelectMin3(ptr + 8, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin12(I ptr, const Less& less)
{
 I l = SelectMin8(ptr, less);
 I h = SelectMin4(ptr + 8, less);
```

```
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin13(I ptr, const Less& less)
{
 I l = SelectMin8(ptr, less);
 I h = SelectMin5(ptr + 8, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin14(I ptr, const Less& less)
{
 I l = SelectMin8(ptr, less);
 I h = SelectMin6(ptr + 8, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin15(I ptr, const Less& less)
{
 I l = SelectMin8(ptr, less);
 I h = SelectMin7(ptr + 8, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
inline I SelectMin16(I ptr, const Less& less)
{
 I l = SelectMin8(ptr, less);
 I h = SelectMin8(ptr + 8, less);
 return less(*l, *h) ? l : h;
}

template <class I, class Less>
void FwSort(I begin, int len, const Less& less)
{
 switch(len) {
 case 16: IterSwap(begin, SelectMin16(begin, less)); begin++;
 case 15: IterSwap(begin, SelectMin15(begin, less)); begin++;
 case 14: IterSwap(begin, SelectMin14(begin, less)); begin++;
 case 13: IterSwap(begin, SelectMin13(begin, less)); begin++;
 case 12: IterSwap(begin, SelectMin12(begin, less)); begin++;
 case 11: IterSwap(begin, SelectMin11(begin, less)); begin++;
 case 10: IterSwap(begin, SelectMin10(begin, less)); begin++;
 case  9: IterSwap(begin, SelectMin9(begin, less)); begin++;
 case  8: IterSwap(begin, SelectMin8(begin, less)); begin++;
```

```
 case  7: IterSwap(begin, SelectMin7(begin, less)); begin++;
 case  6: IterSwap(begin, SelectMin6(begin, less)); begin++;
 case  5: IterSwap(begin, SelectMin5(begin, less)); begin++;
 case  4: IterSwap(begin, SelectMin4(begin, less)); begin++;
 case  3: IterSwap(begin, SelectMin3(begin, less)); begin++;
 case  2: IterSwap(begin, SelectMin2(begin, less));
 }
}
```

The final result: it works, but it is not faster than normal loop based sort:) Maybe if I could make C++ to emit CMOV, it would be better...

Mirek

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by Mindtraveller on Sun, 01 Feb 2009 15:59:48 GMT
View Forum Message <> Reply to Message

luzr wrote on Sun, 01 February 2009 13:30One of critical issues in polygon rasterizer (which I could not resist to work on in the end) Mirek, so you started working on image<->polygon converter as polygonized Da Vinci`s Mona Liza you`ve shown some time ago?

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by mirek on Sun, 01 Feb 2009 17:17:12 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Sun, 01 February 2009 10:59luzr wrote on Sun, 01 February 2009 13:30One of critical issues in polygon rasterizer (which I could not resist to work on in the end) Mirek, so you started working on image<->polygon converter as polygonized Da Vinci`s Mona Liza you`ve shown some time ago?

Well, I rather got tired of AGG bugs, design problems and limitations and (re)started new 2D sw renderer ("Painter 2.0") from scratch.

But heavily mining AGG sources - but I feel no shame, as AGG heavily mined others - it is actually funny to trace the code back - the bread and butter of AGG, antialiased polygon renderer, is based on Freetype code, which in turn is based on LibArt code.

That said, I am not sure whether the polygon rasterizing algorithm was invented by Raph Levien of LibArt, but if it was, he is really really smart guy.

Mirek

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by mr_ped on Sun, 01 Feb 2009 19:13:03 GMT
View Forum Message <> Reply to Message

I think with 10+ elements already quicksort can pay off. A well implemented quicksort will not hurt even with 2-3 elements that much.

Then the question is also, what do you sort, and how much the data can be pre-sorted already, in such case you can either avoid sort at all, or bubble sort can yield best results if every element is already close to it's sorted order position.

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by mirek on Sun, 01 Feb 2009 22:06:22 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Sun, 01 February 2009 14:13I think with 10+ elements already quicksort can pay off.

In U++, we maintain 16 as threshold.

Quote:
A well implemented quicksort will not hurt even with 2-3 elements that much.

There is only so much you can do with plain quicksort. All real quicksort algorithms switch to selection sort or insert sort when subsequence goes under certain threshold. It makes it quite faster.

Thus, if I could invent some faster variant for up to 16 elements, we would have a huge win...

Mirek

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by piotr5 on Mon, 11 May 2009 13:03:11 GMT
View Forum Message <> Reply to Message

I really am no expert in this sorting-thing, but I guess the real advantage of the sort-algorithm in this thread is that one can easily paralellize it. as far as I remember mmx had some possibility to sort aribatry bytes with this method in paralell: each 64-bit register can hold 8 bytes, and so comparing 2 registers will sort up to 16 bytes in a few assembler-mmx-commands. naturally larger values require more registers. of course with only 16 values starting a seperate thread (on some other processor or multicore) is quite an overhead, the same with switching from float to mmx on my amd. but if the overhead has already been taken of...

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by tojocky on Mon, 24 Aug 2009 15:29:22 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 02 February 2009 00:06mr_ped wrote on Sun, 01 February 2009 14:13I think with 10+ elements already quicksort can pay off.

In U++, we maintain 16 as threshold.

Quote:
A well implemented quicksort will not hurt even with 2-3 elements that much.

There is only so much you can do with plain quicksort. All real quicksort algorithms switch to selection sort or insert sort when subsequence goes under certain threshold. It makes it quite faster.

Thus, if I could invent some faster variant for up to 16 elements, we would have a huge win...

Mirek

Founded on Wikipedia that one of the fasted sorting algorithm of small number of elements is Shell sort.

Here is another interesting article of sorting.
Ion Lupascu (tojocky)

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by piotr5 on Sat, 29 Aug 2009 13:32:09 GMT
View Forum Message <> Reply to Message

I'm not sure how this would look like:

16 elements means 4x4 table. sorting each column iteratively means splitting it up into 2x2 tables and combining them into a 4-row column. however, with 9 elements to sort one could make a 3x3 table and sort each column with 3-4 comparisons instead of 1 comparison and a whole insert-sort of the 3 elements along with the whole overhead of it. also, after sorting the 4 columns, what next? should it now be 2x8 or 3x6 or immediately insertion-sort? according to the article it must be insertion-sort, but maybe in this particular case 2x8 would be better? and what to start with? when is 3x6 better than 4x4? what about 5x4?

sorry for the rant, I would be interested in some input though.
meanwhile I will do some testing...

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by piotr5 on Sun, 30 Aug 2009 23:38:24 GMT
View Forum Message <> Reply to Message

a small update since I wont get the time to make a full test:

with my particular test-data I can only confirm that nothing can drastically outperform a simple sort-loop. (i.e. exchange the minimum with the first element, and continue without that first element.) only through full speed-optimization the divide-and-conquer search for a minimum can really keep up with the loop. especially shell-search isn't much faster than even bubble-sort (all that with sorting only 16 integers)! however, with a trick I managed to find a sorting-algorithm which is about twice as fast: insert-sort. insert-sort in combination with binary search and a mem-copy for the insertion is faster (with my particular test-data) than any of them. it isn't much, but it's noticable. unfortunately memcpy() isn't fair since it doesn't take care of any pick-beaurocracy -- but it certainly is faster than any piece-by-piece swapping or shifting.

in my previous message I said something which is wrong:
shell-sort is done by applying insert-sort, and not
as I said by recursively applying the same method.
recursive shell-sort would only re-do what has already
been done in the previous step!

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by mirek on Wed, 02 Sep 2009 15:42:33 GMT
View Forum Message <> Reply to Message

piotr5 wrote on Sun, 30 August 2009 19:38
insert-sort in combination with binary search and a mem-copy for the insertion is faster (with my particular test-data) than any of them. it isn't much, but it's noticable. unfortunately memcpy() isn't fair since it doesn't take care of any pick-beaurocracy -- but it certainly is faster than any piece-by-piece swapping or shifting.

Well, but it IS compatible with Moveable.

Actually, this is an interesting point w.r.t. to moving some of algorithms to containers. Obviously, if Sort would be Vector's method, it would be OK for it to use memmove...

Mirek

Subject: Re: Funny way how NOT to speedup sorting of small arrays
Posted by Mindtraveller on Wed, 02 Sep 2009 18:33:05 GMT
View Forum Message <> Reply to Message

Also it would be easier to use and learn.