Subject: cpu times
Posted by Gianluigi Boca on Fri, 11 May 2012 16:36:41 GMT
View Forum Message <> Reply to Message

dear collaborators,
we all know at least from oral tradition, that the Standard Template Library containers are somewhat slower compared to the 'traditional' C code style arrays.

But how much slower are they actually ?

I checked the difference in cpu consumption when using a conventional C array or a Standard Template Library <vector> instead, using a very simple program.

I measured the cputime consumption
of 10,000,000,000 assignment operations [avoiding
though a calculation that can be optimized heavily by the compiler].

I wrote two almost identical simple loop programs :

1) Conventional C array program :

```
int main ()
{
 int v[10],b ;
 itmp = 500000;
for(int j=0;j<10000;j++){
 for(int i=0;i<itmp;i++){
 b=i+j;
 v[3]=j+i;
 b=v[3];
 }
}
```

2) Template <vector> program :

```
#include <vector>
int main ()
{
 vector <int> v(10,0) ;
 int b;
 int itmp = 500000;

for(int j=0;j<10000;j++){
 for(int i=0;i<itmp;i++){
 b=i+j;
 v.at(3)=i+j;
 b=v.at(3);
 }
}
  return 0;
};
```

I measured the cpu consumption of the two programs.
I also measured (and subtracted) the cputime consumption of the NON RELEVANT part of the code, namely :

```
int main ()
{
 int b;
 int itmp = 500000;

for(int j=0;j<10000;j++){
 for(int i=0;i<itmp;i++){
 b=i+j;
 }
}
  return 0;
};
```

THE FOLLOWING IS THE CpuTime CONSUMPTION OF THE TRADITIONAL C STYLE

```
 v[3]=j+i;
 b=v[3];
```

STATEMENTS : 9.352 sec

while THE FOLLOWING IS THE CpuTime CONSUMPTION OF THE Standard Template Library

```
 v.at(3)= i+j;
 b=v.at(3);
```

STATEMENTS : 166 sec

In other words, a factor almost 18 worse of the Template <vector>.

As you very well know, the Template <vector> gives the advantage
of the array boundary check (only when you use the at() function
though, NOT when you use the [] form!) but I am wondering if we can afford a factor of speed 18 slower in acces time for a code such as the PANDA code that is supposed to digest billion and billion of events in the future.

Please comment, thanks

Gianluigi

Subject: Re: cpu times
Posted by Ralf Kliemt on Sat, 12 May 2012 07:36:12 GMT

Hi Gianluigi,

Thanks for pointing out this important issue. When thinking about speeding up my code I'll give it more consideration. As you are at it, checking TArrayD and the boost arrays (we have boost but noone uses it, yet) would help us here, too!

I want to point out some features we use in the SDS package from the stl stuff:
- the iterators: they give dynamic and safe access to our data
- maps: storage by keys of arbitrary type
- maps: automatic sorting as you add data for common key types (here: integers)
I'm sure there is a faster way of doing all that with the proper knowledge of standard C and the patience to deal with it.

Cheers.
Ralf

Subject: Re: cpu times
Posted by Felix Boehmer on Sat, 12 May 2012 09:14:02 GMT

Dear Gianluigi,

while these are interesting measurements, I have to re-cite one those fun meetings we had last summer: You are comparing apples with pears!

If you want to compare the raw performance of the two data classes, you have to use similar functionality, e.g. the [] operator of the <vector> which does no implicit range check. It is unnecessary and bad practice to use at() in loops of the kind for(int i=0; i<vector.size(); i++) { meh = vector.at(i); //Completely unnecessary range-check
}over vectors anyway.

It would be interesting to directly compare assignment and reading performance like you did by replacing at() with []. Another thing you could look at which would have some real-world relevance is to compare array[] and (*<vector-pointer>)[] performance, that is the combined performance of a necessary de-referencing with following raw access.

Cheers

Felix

Subject: Re: cpu times
Posted by Gianluigi Boca on Thu, 31 May 2012 19:20:27 GMT

dear Felix,
I repeat here my reply since I don't see it in the discussion

and I fear it may have been lost.

So, the reason why I considered the at() function was beacause
in my opinion it is the more useful functionality of <vector>
(the boundary check).

Anyway I have measured also the [] operator and it turns
out to be 'only' about 4.5 times slower than the traditional
C array access (on a 64-bit Lenny machine here at GSI).

That in principle is still way too much, I believe.

However, after a discussion with Mohammad, he had promised to assess with Valgrind how much time is spent in the Pandaroot code on average in the STL library compared to the total process
time, for some 'typical' Panda event reconstruction etc.

If that fraction of time is negligible he says it is worthless
to bother.

Some computer gurus may disagree with his point (every fraction
of Cputime saved may translate in the long run in many days of Cputime saved). I have already heard this discussion in the past.
But anyway, let's stay tuned and see what he founds

cheers Gianluigi

Felix Boehmer wrote on Sat, 12 May 2012 11:14Dear Gianluigi,

while these are interesting measurements, I have to re-cite one those fun meetings we had last summer: You are comparing apples with pears!

If you want to compare the raw performance of the two data classes, you have to use similar functionality, e.g. the [] operator of the <vector> which does no implicit range check. It is unnecessary and bad practice to use at() in loops of the kind for(int i=0; i<vector.size(); i++) { meh = vector.at(i); //Completely unnecessary range-check
}over vectors anyway.

It would be interesting to directly compare assignment and reading performance like you did by replacing at() with []. Another thing you could look at which would have some real-world relevance is to compare array[] and (*<vector-pointer>)[] performance, that is the combined performance of a necessary de-referencing with following raw access.

Cheers

Felix

Subject: Re: cpu times

Hi,

First of all I am not sure if we really need this discussion here, but any way I run the profiler on the svn rev 15651 (yesterday). The standard sim, digi and then I profile the reco with valgrind and callgrind, all with 10 events. The result is shown below:

I normalize the time to the exec time, if you look at the picture above you will see that we spend:

Kalman filter:	57 %
STT track finding	13 %
STT+MVD tracking	15 %
MVD Riemann	7 %
STTMVDGEM	7 %

now if you look at the "calls" you will notice that the STT and Kalman (first three in the list above) are taking about 85 % of the time, however going down in the picture you will see that 70 % of the 85% are spend in Geane and glpk code, to make it clear:

we spend 70 % of processing time in external packages

The rest of the code which is 30 % of time has definitely not that much time spent in STL but in other algorithms and IO etc. So assuming that C arrays are much faster (Which I do not agree on!) it make no sense to contaminate the code with non-readable stuff because of speed. One has also to think about debugging and tracing the code e.g: out of bound problems which we already have with the c arrays in the tracking code.

best regards,

Mohammad

File Attachments

1) task.jpg, downloaded 376 times