MAPCLASS - C++ and parallelization

Diana Andreea Popescu
Rogelio Tomas Garcia

CERN

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Introduction

- C++ library
- Parallelization strategies
- Conclusions

MAPCLASS2

- A map is a polynomial representation of the transformation done over the particle coordinates from one point to another along the beamline
- Can load a file of map coefficients generated by MAD-X PTC
- Alternatively, it can generate the map directly from a twiss object (created by loading a Twiss file generated by MAD-X PTC)

Processing steps

- Constructing the map using all the elements sequentially
- For each element
  - we compute its formula using the known parameters' values
  - we obtain a map which is composed with the previous map
  - all the operations are polynomial operations - uses the polynomial library pytpsa
- Using the final map, we can compute the beam size, the beam offset, etc.
- Optimisation operation - changing the parameters of certain elements and recomputing the final map

Profiling results for map construction

```
 Ordered by: internal time

 ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  11406    0.856    0.000    1.328    0.000 ../../pytpsa/pol.py:251(fmulpol)
  14890    0.762    0.000    1.374    0.000 ../../pytpsa/pol.py:221(addpol)
  68978    0.426    0.000    0.860    0.000 ../../pytpsa/pol.py:161(truncate)
1761198    0.381    0.000    0.381    0.000 {method 'get' of 'dict' objects}
 653990    0.309    0.000    0.309    0.000 {zip}
  14022    0.215    0.000    1.671    0.000 ../../pytpsa/pol.py:235(mulpol)
 343225    0.203    0.000    0.278    0.000 ../../pytpsa/pol.py:10(abs)
 473700    0.132    0.000    0.132    0.000 {sum}
81366/79866    0.124    0.000    0.183    0.000 ../../pytpsa/pol.py:86(__init__)
  25972    0.086    0.000    0.412    0.000 ../../pytpsa/pol.py:205(mulcoef)
    405    0.079    0.000    2.802    0.007 ../../pytpsa/polmap.py:251(compose)
 343225    0.075    0.000    0.075    0.000 {abs}
    206    0.041    0.000    0.816    0.004 ../../transport.py:16(__call__)
  14228    0.040    0.000    0.202    0.000 ../../pytpsa/pol.py:198(addcoef)
  47524    0.033    0.000    0.033    0.000 {method 'update' of 'dict' objects}
  27316    0.031    0.000    1.594    0.000 ../../pytpsa/pol.py:293(__add__)
```

C++ map construction

- Polynomial library
- Formulas of elements
- Composition of maps
- Results are the same as with the ones obtained in Python

C++ map construction

- Used Boost Python to interface with the rest of the functions of MAPCLASS (calculations of sigma, etc.)
- The map can be constructed from a twiss file or from a twiss object constructed in Python
- Twiss object also in C++
- STL and Boost libraries used for data structures
- Optimized for speed based on Intel VTune Amplified analysis

C++ map construction

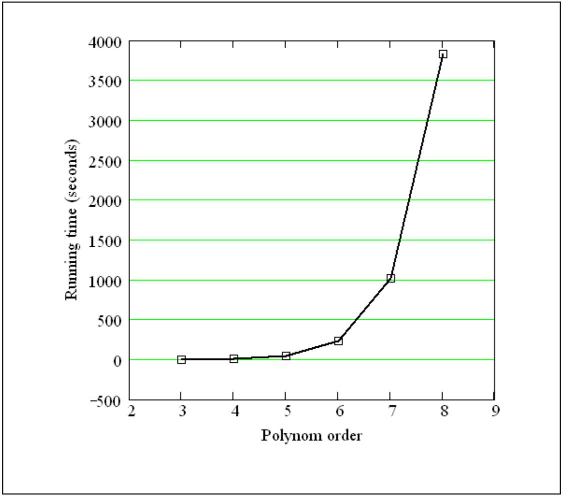
- Pol
  - truncation is done while doing the actual operations, not as before when it was perfomed at the end of the arithmetic operation
  - unordered map instead of Python dict
  - uses templates (for example coefficients can be double/complex numbers)
- Polmap
- Funset
- Transport
- Twiss
- MapBeamLine

C++ map construction

- The code can be found at https://github.com/pylhc/MapClass2
- The C++ part of the code is provided as a shared library
- Makefile for compilation
- Works on lxplus with Python 2.6
- On local machine if new Boost library is installed works with newer versions of Python

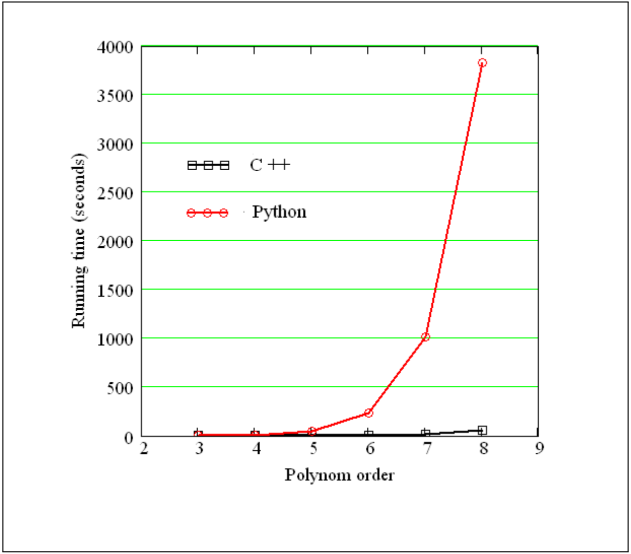
Initial timing map construction in Python

Final Focus System with 210 elements on quad core Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz



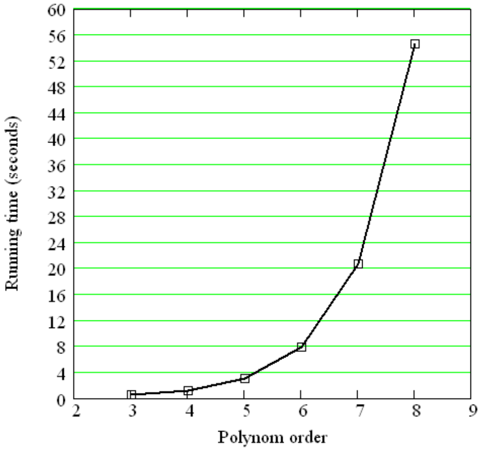
Comparison - Serial

Final Focus System with 210 elements on quad core Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz



C++ map construction from twiss file - timing

Final Focus System with 210 elements on quad core Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz



$$\begin{pmatrix} 3 & 0.488 \\ 4 & 1.101 \\ 5 & 2.976 \\ 6 & 7.817 \\ 7 & 20.772 \\ 8 & 54.602 \end{pmatrix}$$

Construction of map from twiss object - Serial

The elements that form the line are chained together iteratively

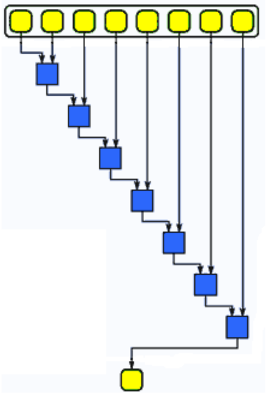


Figure: Serial version

Construction of map from twiss object - Parallel

- Problems
  - in serial version we have composition between a small map and a large one, here we will have composition between 2 large maps
  - load imbalance on cores

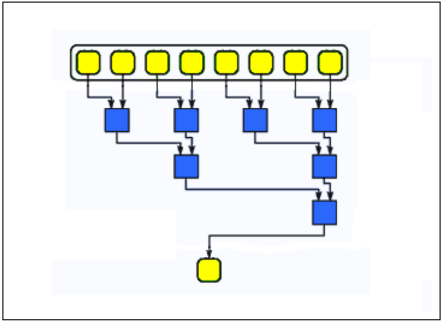


Figure: Parallel version

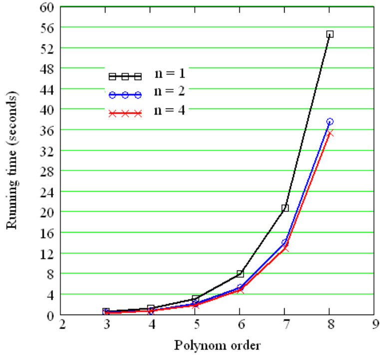
Construction of map from twiss object in parallel

- Each process has as input a continuous sequence of elements
- The results are combined by one processor
- The load on processors is inequal (composition with multipoles takes much longer than with other elements)
- OpenMP dynamic schedule cannot be used (composition not commutative)

Construction of map from twiss object in parallel

- Each process has as input a continuous sequence of elements
- The results are combined by one processor
- The load on processors is inequal (composition with multipoles takes much longer than with other elements)
- OpenMP dynamic schedule cannot be used (composition not commutative)
- Tree structure won't help due to load unbalance and composition between large maps
- Investigated Intel Thread Building Blocks

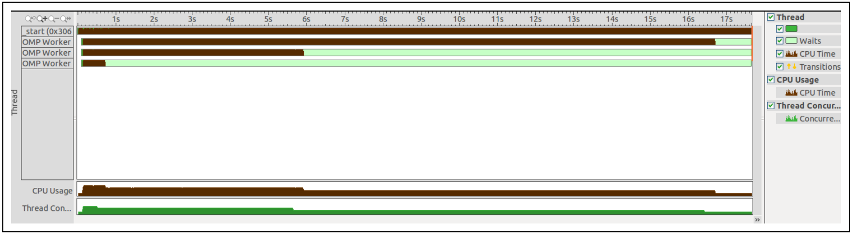
Final Focus System with 210 elements on quad core Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz



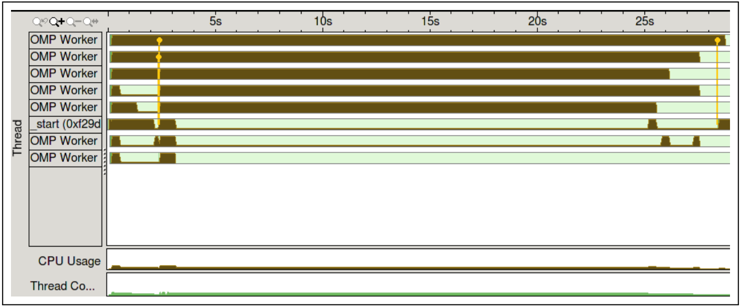
$$\begin{pmatrix} 3 & 0.488 & 0.325 & 0.303 \\ 4 & 1.101 & 0.701 & 0.652 \\ 5 & 2.976 & 2.05 & 1.78 \\ 6 & 7.817 & 5.182 & 4.768 \\ 7 & 20.772 & 13.942 & 12.932 \\ 8 & 54.602 & 37.551 & 35.387 \end{pmatrix}$$

Construction of map from twiss object

Intel VTune Amplifier - Concurrency Analysis



Intel VTune Amplifier - Concurrency Analysis

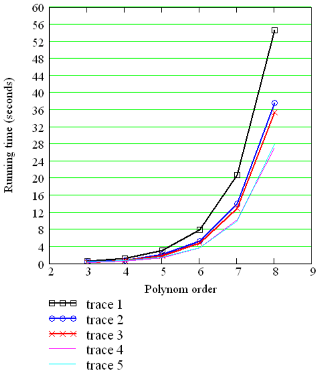


Composition of maps in parallel

- Serial version based on using a hash table which stores the intermediate products of polynoms in order to avoid recomputation
- The idea of parallelization is to have each core compute the composition for only one coordinate (x, px, y, py, d, s)
- Each core will have its own hash table
- Or can use one with concurrent access (concurrent unorderd map from Intel TBB)
- Small improvement due to dependency on the hash table
- Used when composing large maps

Construction of map with composition in parallel

Used with isolation of multipoles



$$\text{pol6} = \begin{pmatrix} 3 & 0.488 & 0.325 & 0.303 & 0.278 & 0.331 \\ 4 & 1.101 & 0.701 & 0.652 & 0.553 & 0.634 \\ 5 & 2.976 & 2.05 & 1.78 & 1.373 & 1.448 \\ 6 & 7.817 & 5.182 & 4.768 & 3.685 & 3.697 \\ 7 & 20.772 & 13.942 & 12.932 & 10.182 & 9.859 \\ 8 & 54.602 & 37.551 & 35.387 & 27.072 & 28.107 \end{pmatrix}$$

Conclusion

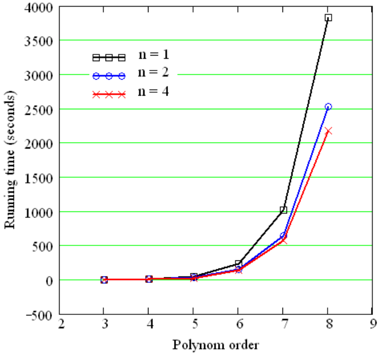
- Map construction in C++ takes a lot less time
- Cannot scale because of restrictions
- Investigate nested parallelism for having composition in parallel for larger elements (multipoles)

Parallelization in Python

- The GIL prevents multiple native threads from executing Python bytecodes at once
- Multiprocessing module - Limited API
- Cython
  - difficult to use in parallel context
  - need to release the GIL, but all the other functions (for polynomials) require the GIL !!!

Multiprocessing module - timing

Final Focus System with 210 elements on quad core Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz



$$\begin{pmatrix} 3 & 2.27 & 1.385 & 1.278 \\ 4 & 9.426 & 6.014 & 5.588 \\ 5 & 49.57 & 30.459 & 26.748 \\ 6 & 232.587 & 150.878 & 135.128 \\ 7 & 1.017 \times 10^3 & 637.469 & 577.185 \\ 8 & 3.827 \times 10^3 & 2.535 \times 10^3 & 2.178 \times 10^3 \end{pmatrix}$$