New algorithm for Cholesky Decomposition

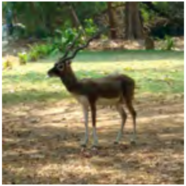
Rahul Balasubramanian

CERN

30th August 2016

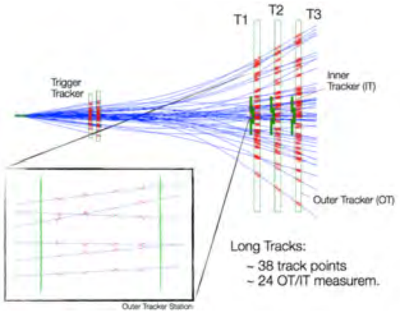
about me

- B.Tech in Engineering Physics from Indian Institute of Technology Madras, Chennai, India



- Physics Masters at Université Paris-Sud, Orsay, France

tracking



- estimating parameters from detector hits to describe tracks or vertex
  - track represented by state vector $\vec{s}_i = (x, y, tx, tz, q/p)^T$
- straight line description for fit in y-z direction
  - $y(z) = b_y z + a_y$
- parabolic model with three track parameters (a, b, c) for x-y projection
  - $x(z) = c(z - \texttt{zReference})^2 + b(z - \texttt{zReference}) + a$

parameterized fits

- standard weighted least-squares fit carried out, χ^2 is minimized

$$\chi^2 = \sum_{\text{hits } i} \left(\frac{x_i - x_{\text{track}}(z_i) - y_{\text{track}}(z_i)(\frac{dx}{dy})_i}{\sigma_i} \right)$$

- given a good estimate of $\vec{x}_i$, $\chi^2(\vec{x}_{i+1})$ can be expanded upto quadratic order in $\delta\vec{x} = \vec{x}_{i+1} - \vec{x}_i$

$$\chi^2(\vec{x}_i + \delta\vec{x}) = \chi^2(\vec{x}_i) + \nabla\chi^2(\vec{x}_i)|_{\vec{x}_i}.(\delta\vec{x}) + \sum_{k,l}(\delta\vec{x})_k \frac{\partial^2\chi^2}{\partial x_k \partial x_l}|_{\vec{x}_i}(\delta\vec{x})_l$$

- updated parameter $\delta\vec{x}$ calculated by setting derivative of $\chi^2(\vec{x}_i + \delta\vec{x})$ to zero

$$(-\nabla\chi^2 |_{\vec{x}_i})_k = \frac{\partial^2\chi^2}{\partial x_k \partial x_l}|_{\vec{x}_i}(\delta\vec{x})_l$$

matrix form

- have to solve for $\mathbf{x}$ in $\mathbf{Mx} = \mathbf{r}$, where $\mathbf{M} = \frac{\partial^2 \chi^2}{\partial x_k \partial x_l}|_{\vec{x}_i}$, $\mathbf{x} = \delta\vec{x}$ and $\mathbf{r} = -\nabla\chi^2|_{\vec{x}_i}$

$$\begin{pmatrix} \langle 1\rangle & \langle dz\rangle & \langle \eta\rangle & \langle \zeta\rangle & \langle -\zeta dz\rangle \\ \langle dz\rangle & \langle dz^2\rangle & \langle \eta dz\rangle & \langle -\zeta dz\rangle & \langle -\zeta dz^2\rangle \\ \langle \eta\rangle & \langle \eta dz\rangle & \langle \eta^2\rangle & \langle -\zeta\eta\rangle & \langle -\zeta\eta dz\rangle \\ \langle -\eta\rangle & \langle -\eta dz\rangle & \langle -\eta\zeta\rangle & \langle \zeta^2\rangle & \langle \zeta^2 dz\rangle \\ \langle -\eta dz\rangle & \langle -\zeta dz^2\rangle & \langle -\eta\zeta dz\rangle & \langle \zeta^2 dz\rangle & \langle \zeta^2 dz^2\rangle \end{pmatrix} \begin{pmatrix} \delta a \\ \delta b \\ \delta c \\ \delta a_y \\ \delta b_y \end{pmatrix} = \begin{pmatrix} \langle dx\rangle \\ \langle dxdz\rangle \\ \langle dx\eta\rangle \\ \langle -dx\zeta\rangle \\ \langle -dx\zeta dz\rangle \end{pmatrix}$$

where $dz_i = z_i - \texttt{zReference}$, $\eta_i = dz_i^2$, $\zeta_i = (\frac{dx}{dy})_i$ and $\langle q\rangle = \sum_i \frac{q_i}{\sigma_i^2}$

definition

- for positive definite real symmetric matrix $\mathbf{M}$, Cholesky decomposition $\rightarrow \mathbf{M} = \mathbf{L}\mathbf{L}^T$
- Solving $\mathbf{x}$ in $\mathbf{M}\mathbf{x} = \mathbf{r} \equiv \mathbf{y}$ in $\mathbf{L}\mathbf{y} = \mathbf{r}$ then $\mathbf{x}$ in $\mathbf{L}^T\mathbf{x} = \mathbf{y}$

  simple forward subsitution and backward substitution
- divide and conquer method for the decomposition
- incorporating SIMD vectorization and template metaprogramming

CERN

divide and conquer

- solving $\mathbf{L}$ in $\mathbf{L}\mathbf{L}^{\mathsf{T}} = \mathbf{M}$, solving sub-block matrices of $\mathbf{L}$

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{pmatrix}, \mathbf{L} = \begin{pmatrix} \mathbf{L}_0 & \mathbf{0} \\ \mathbf{L}_1 & \mathbf{L}_2 \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{L}_1 & \mathbf{0} \\ \mathbf{L}_2 & \mathbf{L}_3 \end{pmatrix} \begin{pmatrix} \mathbf{L}_1^{\mathsf{T}} & \mathbf{L}_2^{\mathsf{T}} \\ \mathbf{0} & \mathbf{L}_3^{\mathsf{T}} \end{pmatrix} = \begin{pmatrix} \mathbf{M}_0 & \mathbf{M}_1 \\ \mathbf{M}_2 & \mathbf{M}_3 \end{pmatrix}$$

- solving three sub-problems sequentially, decomposition calculated recursively

$$\mathbf{L}_1\mathbf{L}_1^{\mathsf{T}} = \mathbf{M}_0$$
$$\mathbf{L}_1\mathbf{L}_2^{\mathsf{T}} = \mathbf{M}_1$$
$$\mathbf{L}_3\mathbf{L}_3^{\mathsf{T}} = \mathbf{M}_3 - \mathbf{L}_2\mathbf{L}_2^{\mathsf{T}}$$

- universal for all sizes and convinient to debug

invert by blocks

- computing $\mathbf{L}_1^{-1}$ with $\mathbf{L}_1$ sub-blocks

$$\mathbf{L}_1 = \begin{pmatrix} (\mathbf{L}_1)_1 & \mathbf{0} \\ (\mathbf{L}_1)_2 & (\mathbf{L}_1)_3 \end{pmatrix}$$

$$(\mathbf{L}_1)^{-1} = \begin{pmatrix} ((\mathbf{L}_1)_1)^{-1} & \mathbf{0} \\ -((\mathbf{L}_1)_3)^{-1}(\mathbf{L}_1)_2((\mathbf{L}_1)_1)^{-1} & ((\mathbf{L}_1)_3)^{-1} \end{pmatrix}$$

- matrix inversion called recursively

use of dot product

- c_{ij} of $\mathbf{C} = \mathbf{AB}$ given by,

$$c_{ij} = \sum_{n=1}^{k} a_{ik} b_{kj} = \mathbf{A}_i . \mathbf{B}_j^\top$$

simd vectorization

- **S**ingle **I**nstruction **M**ultiple **D**ata present in most modern CPU's acting on small vectors
- single assembly instruction invoked on all elements of SIMD vector

$$c[0] = a[0] + b[0]$$

$$c[1] = a[1] + b[1]$$

$$c[2] = a[2] + b[2]$$

$$c[3] = a[3] + b[3]$$

- SIMD vectorization used to calculate dot product of two vectors

template metaprogramming

- compiler generates temporary source codes which is merged with rest of source code and compiled

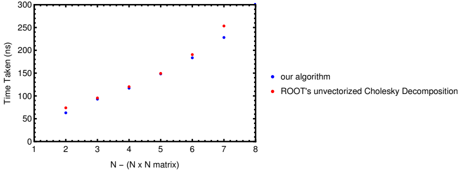
```
template <unsigned N> struct fac_find{
        enum { value = N * fac_find<N - 1>::value };
};
template <> struct fac_find<0u>{ enum { value = 1 }; };

// fac_find<3>::value goes through these steps:
// enum { value = 3 * fac_find<2>::value };
// enum { value = 3 * 2 * fac_find<1>::value };
// enum { value = 3 * 2 * 1 * fac_find<0>::value };
// enum { value = 3 * 2 * 1 * 1; };
```

- loops involved in vector/matrix operations unrolled

Results

- working code can be found at Cholesky in GitLab



- maximum absolute relative difference of calculated matrices of same order of magnitude in comparison to `CholeskyDecomp.h`
- first version of code matches already existing hand tuned scalar code
- reduce current control flow out of runtime path by using template metaprogramming to write it out at control time

Thank You!