Tuning/Event Generation/Monte Carlo/Kitchen Sink meeting

Faster Parton Distribution Evaluation in Monte Carlos

Zack Sullivan

Fermi National Accelerator Laboratory

June 11, 2004

Based on ZS, hep-ph/0403055 and new studies

Contents:

1. How much time is spent retrieving CTEQ PDFs?
   (i.e. Why do we care?)
2. Speed enhancements to the evaluations:
   How do they work?
   How well do they work?
3. Potpourri (a few other irregularities to watch)

http://home.fnal.gov/∼zack/pdf/ has improved code and details.

Why should we care?

More time is spent retrieving PDFs than in any other routine in event generation (by a long way).

I first noticed this when profiling **ZTOP**, my fully differential NLO coding of s- and t-channel single-top-quark production, but it seems to be universal.

Fraction of time spent inside PDF functions using default CTEQ PartonX# on single-top

	CTEQ4/5	CTEQ6
ZTOP	90%	60%
HERWIG	70%	33%
PYTHIA	30%	16%

You should care because CTEQ5L and CTEQ5M1 are the default PDF choices for Run 2.

Why does **HERWIG** spend more time than **PYTHIA**?

PYTHIA calls **STRUCTM** ∼100 times/event requested.
HERWIG calls **STRUCTM** ∼1100-1800 times/event requested.

Each call of **STRUCTM** calls the PDF evaluation 8 times.

This means you can call the PDFs up to 13,000,000 times to get 1000 events using CTEQ5 and **HERWIG**!

Why is this so expensive? Why is CTEQ4/5 more expensive than CTEQ6?

CTEQ4/5 and 6 use completely different interpolation algorithms.

I don't want to change results, so how can I speed these up?

POLINT and CTEQ PDFs

What is **POLINT**?

It is a routine from "Numerical Recipes" that performs a polynomial fit of degree $n-1$ to a set of n points based on *Neville's algorithm*.

Why is it used?

This is used by CTEQ to smoothly interpolate between values of x and Q^2 that are read in form a best-fit table.

Most of the time spent in CTEQ4/5 occurs in **POLINT**, while CTEQ6 only uses **POLINT** at the endpoints of x or Q^2. (which is sometimes expensive).

Why is **POLINT** so slow?

1. A line that is never called disables compiler optimizations.

2. $n = 3(4)$ for CTEQ4/5(6). **POLINT**(n points) is too general for most compilers to fully optimize.

Fix this by removing the unreached line, and writing versions of **POLINT** for CTEQ4/5(6) that set $n = 3(4)$. $\Longrightarrow$

ZS, hep-ph/0403055

There is still a lot of redundancy in **POLINT3**. Therefore, I've also written a version that removes all unnecessary calls. $\Longrightarrow$

```
      SUBROUTINE POLINT (XA,YA,N,X,Y,DY)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C     Adapted from "Numerical Recipes"
      PARAMETER (NMAX=10)
      DIMENSION XA(N),YA(N),C(NMAX),D(NMAX)
      NS=1
      DIF=ABS(X-XA(1))
      DO 11 I=1,N
        DIFT=ABS(X-XA(I))
        IF (DIFT.LT.DIF) THEN
          NS=I
          DIF=DIFT
        ENDIF
        C(I)=YA(I)
        D(I)=YA(I)
11    CONTINUE
      Y=YA(NS)
      NS=NS-1
      DO 13 M=1,N-1
        DO 12 I=1,N-M
          HO=XA(I)-X
          HP=XA(I+M)-X
          W=C(I+1)-D(I)
          DEN=HO-HP
          IF(DEN.EQ.0.)PAUSE
          DEN=W/DEN
          D(I)=HP*DEN
          C(I)=HO*DEN
12      CONTINUE
        IF (2*NS.LT.N-M)THEN
          DY=C(NS+1)
        ELSE
          DY=D(NS)
          NS=NS-1
        ENDIF
        Y=Y+DY
13    CONTINUE
      RETURN
      END
```

```
cz This specialized recoding assumes N=3.
      SUBROUTINE POLINT3 (XA,YA,N,X,Y,DY)
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C     Adapted from "Numerical Recipes"
      DIMENSION XA(3),YA(3),C(3),D(3)
      NS=1
      DIF=DABS(X-XA(1))
      DO 11 I=1,3
        DIFT=ABS(X-XA(I))
        IF (DIFT.LT.DIF) THEN
          NS=I
          DIF=DIFT
        ENDIF
        C(I)=YA(I)
        D(I)=YA(I)
11    CONTINUE
      Y=YA(NS)
      NS=NS-1
      DO 13 M=1,2
        DO 12 I=1,3-M
          HO=XA(I)-X
          HP=XA(I+M)-X
          W=C(I+1)-D(I)
          DEN=HO-HP
cz          IF(DEN.EQ.0.)PAUSE
          DEN=W/DEN
          D(I)=HP*DEN
          C(I)=HO*DEN
12      CONTINUE
        IF (2*NS.LT.3-M)THEN
          DY=C(NS+1)
        ELSE
          DY=D(NS)
          NS=NS-1
        ENDIF
        Y=Y+DY
13    CONTINUE
      RETURN
      END
```

```
cz This is a specialized recoding that
cz   assumes N=3.
cz Written by Z. Sullivan, 5/14/04
      SUBROUTINE POLINT3 (XA,YA,N,X,Y,DY)
C     Modified "Numerical Recipes" routine.
      IMPLICIT NONE
      DOUBLE PRECISION XA(3),YA(3),X,Y,DY
      DOUBLE PRECISION C1,HO,HP,HP2,W,D1,D2,DEN
      INTEGER N

      HO=XA(1)-X
      HP=XA(2)-X
      w=ya(2)-ya(1)
      DEN=HO-HP
      DEN=W/DEN
      D1=HP*DEN
      C1=HO*DEN

      HP2=XA(3)-X
      w=ya(3)-ya(2)
      DEN=HP-HP2
      DEN=W/DEN
      D2=HP2*DEN

      W=HP*DEN-D1
      DEN=HO-HP2

      if((x+x-xa(2)-xa(3)).gt.0d0) then
         y=ya(3)+d2+hp2*w/den
      elseif((x+x-xa(1)-xa(2)).gt.0d0) then
         y=ya(2)+d1+ho*w/den
      else
         y=ya(1)+c1+ho*w/den
      endif

      RETURN
      END
```

STRUCTM and the Monte Carlos

Both **PYTHIA** and **HERWIG** call PDFs by using the CERNLIB routine **STRUCTM**. This in turn loops over 8 uniquely defined PDFs: $u_v, d_v, u_s, d_s, s, c, b, g$.

Much of the code that sets up **POLINT**, or the CTEQ6 algorithm, is repeated with identical results between runs.

An obvious algorithmic improvement is to do 1 of 2 things:

1. **SAVE** the values of x, Q^2, and the results of functions applied to them; and bypass that setup code unless x or Q^2 change.

2. Write a custom **STRUCTM** to do the direct calls itself.

I won't show these here, but you can find them at:

http://home.fnal.gov/∼zack/pdf/

How much do each of these changes help?. . .

Benchmarks give a rough feeling

There are large discrepancies in the amount of execution time between different compilers and compiler flags.

1. **ifc** is ALWAYS faster than **g77** by a factor of 1–3.
2. Using **POLINT3** removes most of the difference between compiler flags.
3. Using **POLINT3**, CTEQ4/5 is $\sim 20\%$ faster than CTEQ6.
4. Using **POLINT3**, CTEQ5 is faster and more accurate than the functional fit in **PYTHIA**.

Typical speed gains relative to **POLINT3/4** (looping over x)

	g77 3.1(2.95)		**ifc** 6.0	
Optimization	CTEQ4/5	CTEQ6	CTEQ4/5	CTEQ6
Default CTEQ	1/(1.1–1.2)	1.0	1/(1.5–2.7)	1/1.03
POLINT3/4	1.0	1.0	1.0	1.0
POLINT3 (fast)	1.5	—	1.2	—
SAVE x, Q^2	1.26	2.6	1.12	2.4
SAVE x, Q^2 (fast)	2.3	—	1.9	—
fastest **STRUCTM**	3.1	3.1	4.6	2.7
fastest times:	40 s	50 s	17 s	35 s

While these numbers are a nice guide, a simple benchmark misses the fact that real programs interact in subtle ways with their routines. E.g., the CPU has more cache misses, memory alignment is different, . . .

We care about real code $\Rightarrow$

Speedup for NLO single-top code **ZTOP**

Typical speed gains relative to **POLINT3/4**

Optimization	**g77** 3.1(2.95) CTEQ4/5	**g77** 3.1(2.95) CTEQ6	**ifc** 6.0 CTEQ4/5	**ifc** 6.0 CTEQ6
Default CTEQ	1/1.2	1.0	1/(1.6–2.2)	1.0
POLINT3/4	1.0	1.0	1.0	1.0
POLINT3 (fast)	1.3	—	1.25	—
SAVE x, Q^2	1.2	2.0	1.13	2.0
SAVE x, Q^2 (fast)	1.7	—	1.7	—
fastest **STRUCTM**	1.9	2.15	1.9, 2.7	2.15
fastest times:	86 s	98 s	60, 42 s	62 s

Using the fastest routines makes **ZTOP** run 2–5 times faster!
Publication-quality runs drop from over day to a few hours.

This is probably the biggest gain a theory calculation can achieve, but factors of 2–3 should be attainable.

I actually got another factor of 1.5 by hard-coding some calls to **CTQNPDF**. Removing unnecessary PDF calls can help.

Speedup for HERWIG and PYTHIA

Typical **HERWIG** speed gains relative to **POLINT3/4**

Optimization	**g77** 3.1(2.95) CTEQ4/5	**g77** 3.1(2.95) CTEQ6	**ifc** 6.0 CTEQ4/5	**ifc** 6.0 CTEQ6
Default CTEQ	1/1.12	1.0	1/(1.2–1.7)	1.0
POLINT3/4	1.0	1.0	1.0	1.0
POLINT3 (fast)	1.25	—	1.1	—
SAVE x, Q^2	1.14	1.6	1.12	1.3
SAVE x, Q^2 (fast)	1.5	—	1.3	—
fastest **STRUCTM**	1.65	1.7	1.53	1.4
fastest times:	64 s	50 s	75 s	55 s

Strangely, **g77** was faster than **ifc**, and CTEQ6 was faster than CTEQ5. I do not know why yet, but it worries me (see next page).

Typical **PYTHIA** speed gains relative to **POLINT3/4**

Optimization	**g77** 3.1(2.95) CTEQ4/5	**g77** 3.1(2.95) CTEQ6	**ifc** 6.0 CTEQ4/5	**ifc** 6.0 CTEQ6
Default CTEQ	1/1.03	1.0	1/(1.15–1.25)	1.0
POLINT3/4	1.0	1.0	1.0	1.0
POLINT3 (fast)	1.06	—	1.05	—
SAVE x, Q^2	1.04	1.13	1.05	1.13
SAVE x, Q^2 (fast)	1.1	—	1.1	—
fastest **STRUCTM**	1.14	1.15	1.18	1.15
fastest times:	55 s	58 s	42 s	43 s

HERWIG is 1.5–2.6 times faster.

PYTHIA is 1.15–1.5 times faster ($\sim$3.3 times faster than **HERWIG**).

Some “gotchas” I’ve discovered

I often use **PFTOPDG** rather than **STRUCTM** because it directly provides $u, \bar{u}, d, \bar{d}, s, \bar{s}, c, \bar{c}, b, \bar{b}, t, \bar{t}, g$.

Warning: The CERNLIB version just calls **STRUCTM**. New PDFs will have $s \neq \bar{s}$, so **PFTOPDG** will not return the correct PDFs as it is currently written.

We’ve already seen **HERWIG** calls the PDFs much too often.

Before EVERY call to **STRUCTM**, **HERWIG** calls **PDFSET**!

1. To use new PDFs, you must write your own **PDFSET** to fill the necessary common blocks. **HERWIG** mysteriously overwrites these common blocks, and so must call **PDFSET** to fix it’s mistake.

2. Switching to **LHAPDF** will require writing a CERNLIB-like **PDFSET** that secretly stores an extra copy of the variables, and refills the common blocks with each call.

3. **LHAPDF** currently includes the CTEQ6 evolution code (not CTEQ5). The code in EVLCTEQ is essentially **PFTOPDG**, so the same gains in speed can come from updating **evolvePDF**.

Conclusions

1. Retrieving PDFs is still the single most time consuming operation in **HERWIG** and theoretical codes (including those with fast detector simulations).

	CTEQ4/5	CTEQ6
ZTOP	90⟶42%	60⟶48%
HERWIG	70⟶30%	33⟶23%
PYTHIA	30⟶9%	16⟶9%

2. Additional improvements can still be made

   - You can replace the binary search for x and Q^2 with a hash $\Rightarrow$ factor of 2 for that piece, but only 5% overall.
   - We may have reached the point of diminishing returns.

3. I've made the validated routines available at
   http://home.fnal.gov/∼zack/pdf/

   At least use **POLINT3**. I'll use the fastest version of **STRUCTM**.

4. Watch out for the interface bug in **PFTOPDG**.

5. It is more important to write correct code, than fast code.

   That does not mean it is unimportant to write fast code. . .