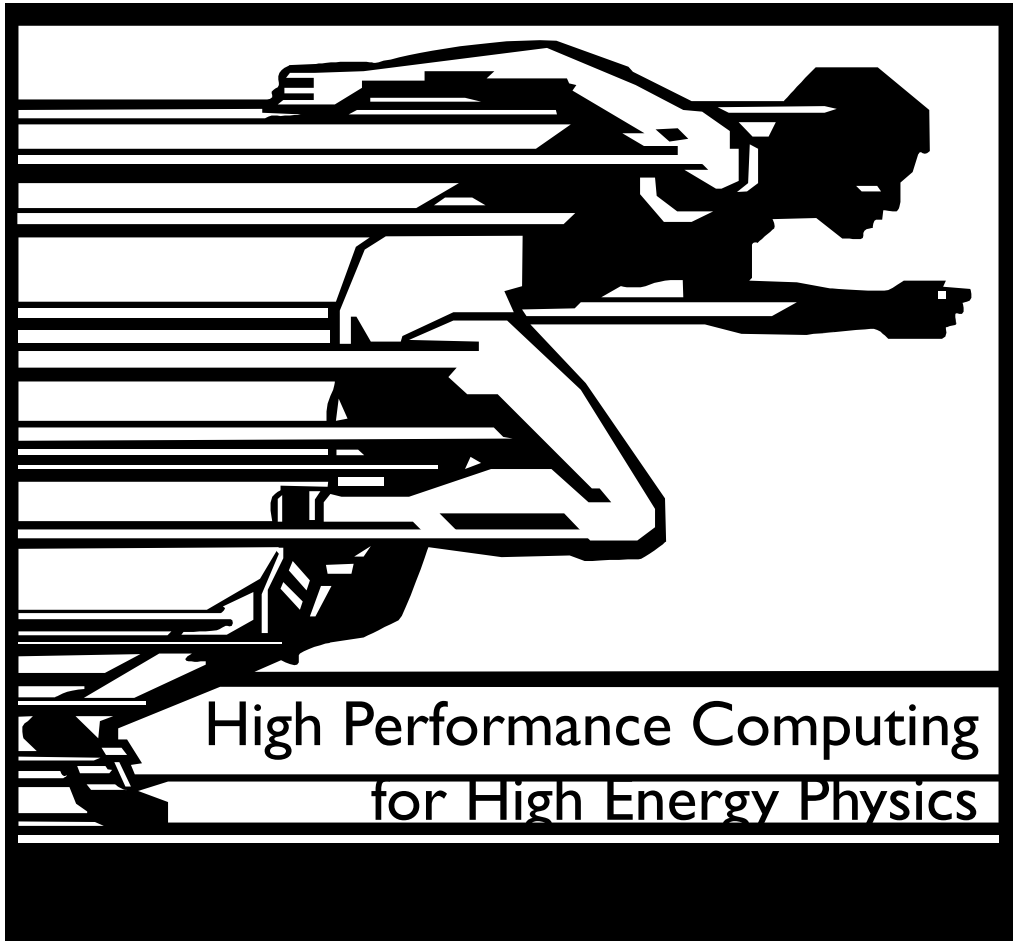
Concurrency in the minimization of unbinned Log-Likehood



Concurrency Forum
CERN, September `13

Vincenzo Innocente
SoFT

"It is a capital mistake to theorise before one has data.
Insensibly one begins to twist facts to suit theories instead of theories to suit facts." - Sherlock Holmes

Executive Summary

- What matters is MEMORY
- Hardware layout should always be taken into account
- Dynamic Task scheduling (of a DAG) is the most efficient solution for concurrency
- Retrofitting efficient concurrency in legacy code is hard, essentially impossible

- gcc is as efficient as icc
- “omp parallel” and “std::atomic” is all one needs to implement dynamic thread scheduling

Slides from recent Wouter's RooFit tutorial to ATLAS

See also Alfio Lazzaro at "Future computing in particle physics June 2011'

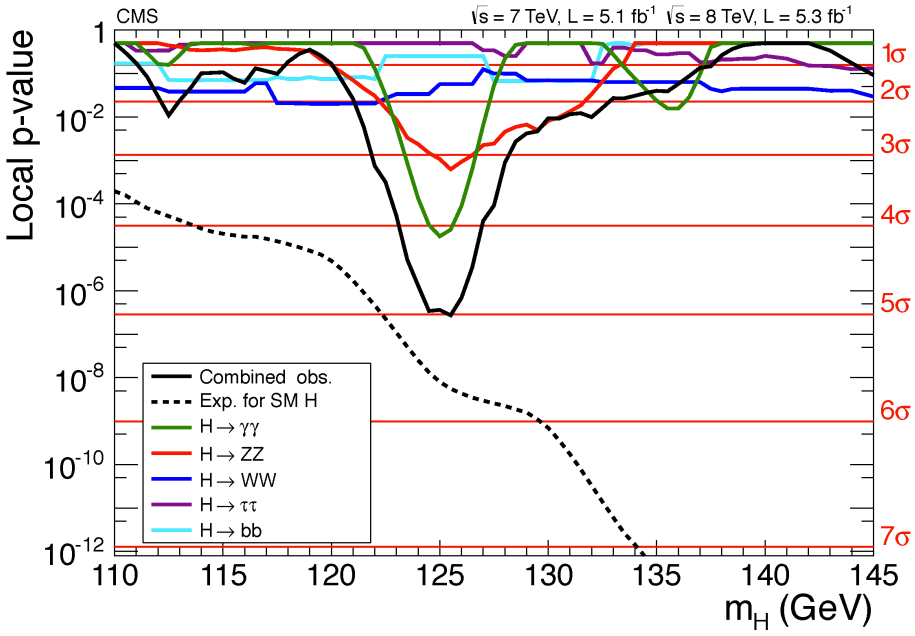
Model : likehood for B->eta' K0s from PhysRevD.80.112002

Illuminating discussions with A.Nowak, L.Moneta, J.Bendavid, G.Petrucciani

MOTIVATIONS

"It is a capital mistake to theorise before one has data.
Insensibly one begins to twist facts to suit theories instead of theories to suit facts." - Sherlock Holmes

The software used to decide to whom to award the Nobel Prize is awkwardly slow..



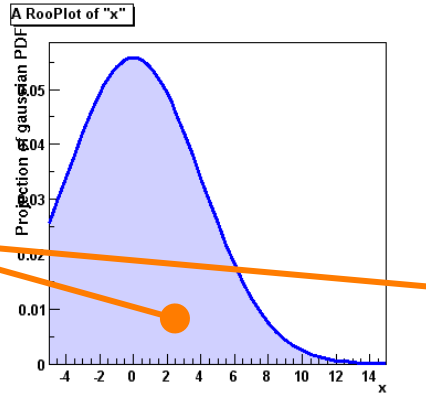
RooFit: How does it work – mathematical model

- Focus on specific models – **probability density functions**

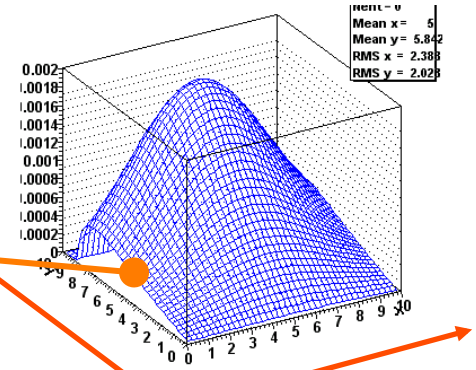
- Defining feature:

$$\int f(\vec{x},\vec{p})\,d\vec{x} \equiv 1,$$

$$f(\vec{x},\vec{p}) \geq 0$$



$$\int F(x)dx \equiv 1$$

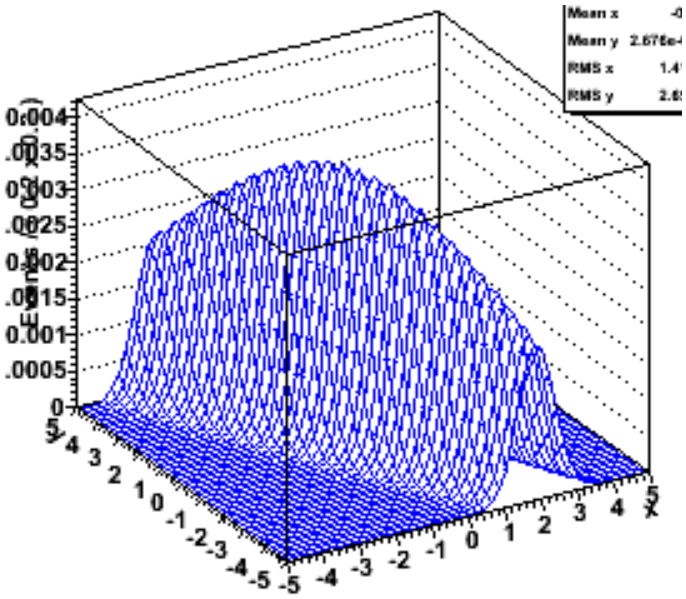


$$\int F(x,y)dxdy \equiv 1$$

- Normalization condition introduces extra complication in formulation of models, but has important advantages
  - Directly usable for formal statistical techniques (needed for low stats)
  - Easier interpretation of model parameters
  - Easier construction of complex models
- RooFit provides built-in support for normalization, taking away down-side for users, leaving upside

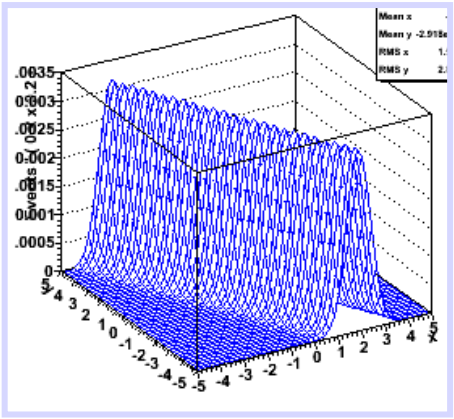
An example of model construction with pdfs

- Take following model f(x,y):
  what is the analytical form?

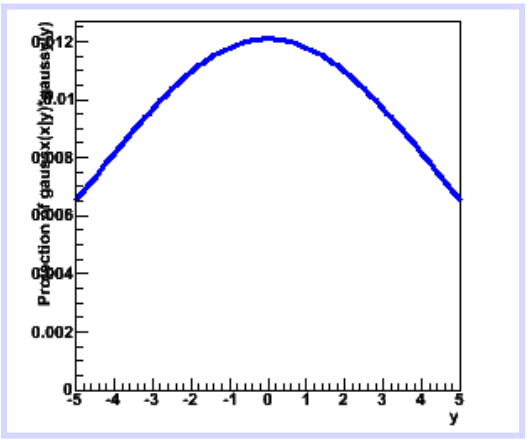


- Trivially constructed with (conditional) probability density functions!

Gauss f(**x**|a*y+b,1)



Gauss g(**y,0,3**)

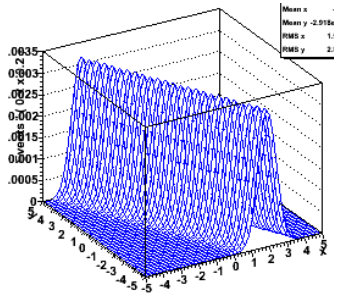


F(**x**,**y**) = f(**x**|**y**)*g(**y**)

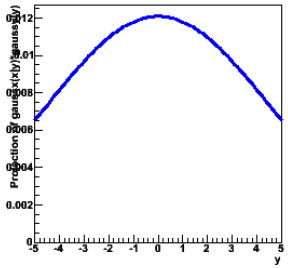
Coding a model in RooFit

- Construct each ingredient with a single line of code

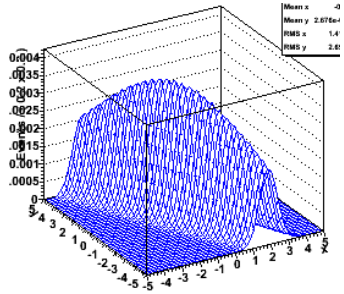
Gauss f(**x**,a*y+b,1)



Gauss g(**y,0,3**)



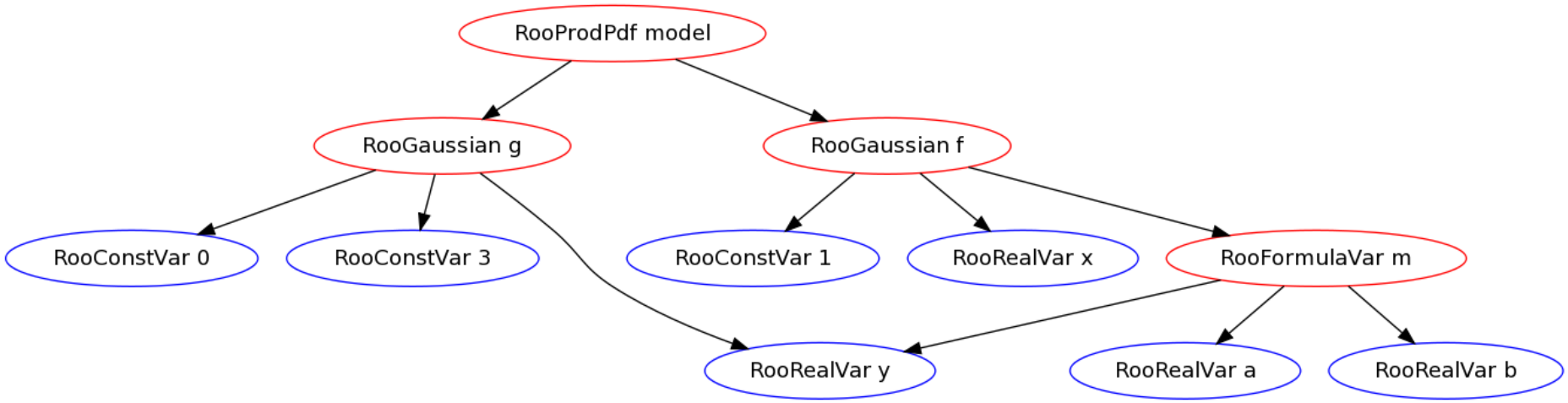
F(**x,y**) = f(**x**|**y**)*g(**y**)



```
RooRealVar x("x","x",-10,10) ;
RooRealVar y("y","y",-10,10) ;
RooRealVar a("a","a",0) ;
RooRealVar b("b","b",-1.5) ;

RooFormulaVar m("a*y+b",a,y,b) ;
RooGaussian f("f","f",x,m,C(1)) ;

RooGaussian g("g","g",y,C(0),C(3)) ;

RooProdPdf F("F","F",g,Conditional(f,y)) ;
```

The structure of a model visualized

- Graph of client-server connections between variables and functions



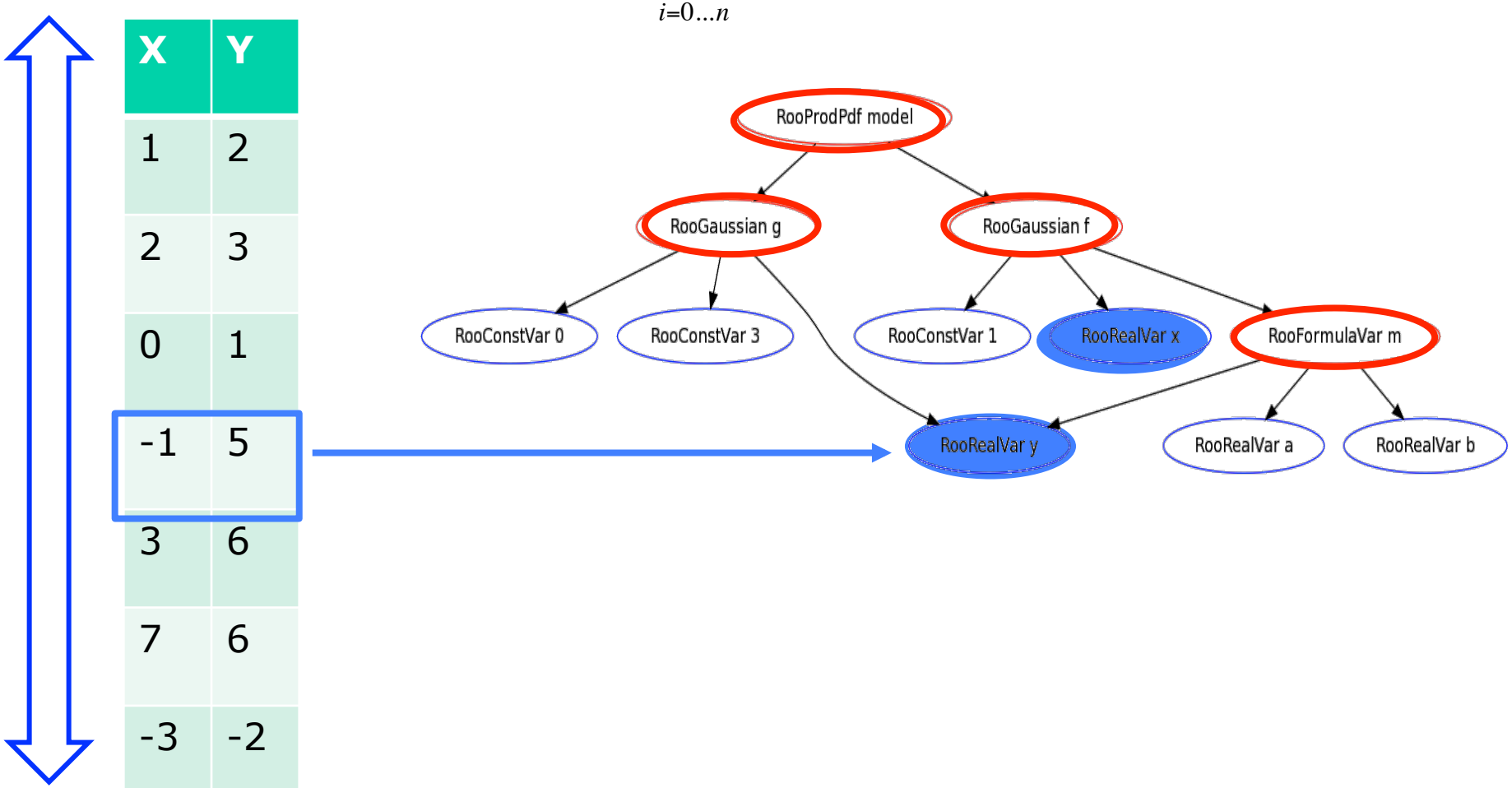

```
RooRealVar x("x","x",-10,10) ;
RooRealVar y("y","y",-10,10) ;
RooRealVar a("a","a",0) ;
RooRealVar b("b","b",-1.5) ;
RooFormulaVar m("a*y+b",a,y,b) ;
RooGaussian f("f","f",x,m,C(1)) ;
RooGaussian g("g","g",y,C(0),C(3)) ;
RooProdPdf model("model","model",g,Conditional(f,y)) ;
```

Graph made with F.graphVizTree("model.dot")

Optimization of likelihood calculations

- Likelihood evaluates pdf at all data points

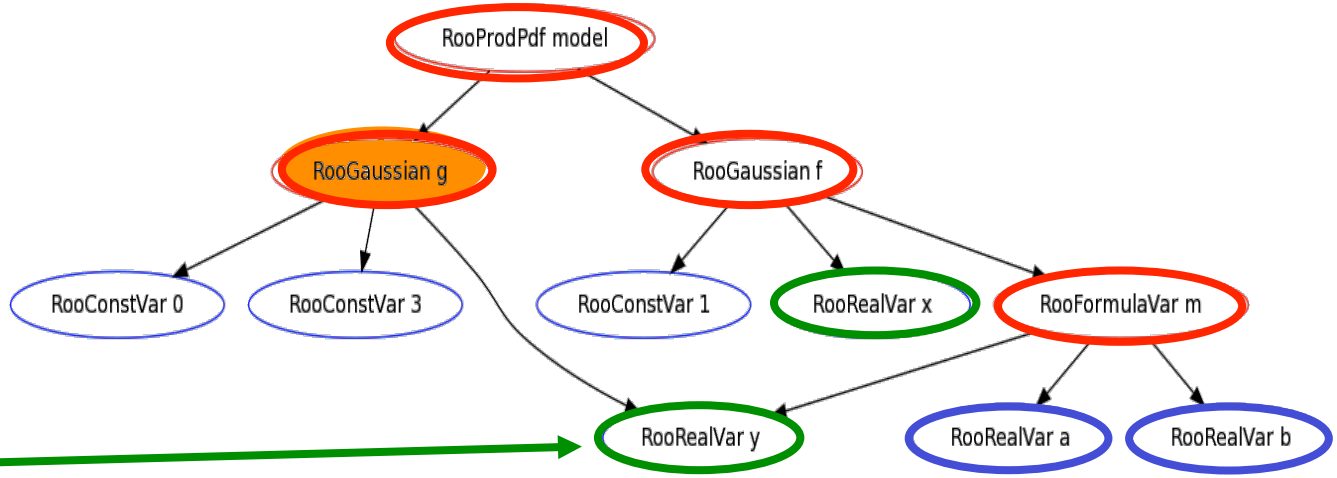
$$-\log L(\vec{p}) = -\sum_{i=0\ldots n} \log f(\vec{x}_i, \vec{p})$$



Level-1 optimization of likelihood calculation

- Constant terms (depend only on observables, not on parameters) are precalculated, added to dataset

X	Y	g
1	2	1.5
2	3	2.7
0	1	1.2
-1	5	0.6
3	6	9.8
7	6	3.4
-3	-2	5.7



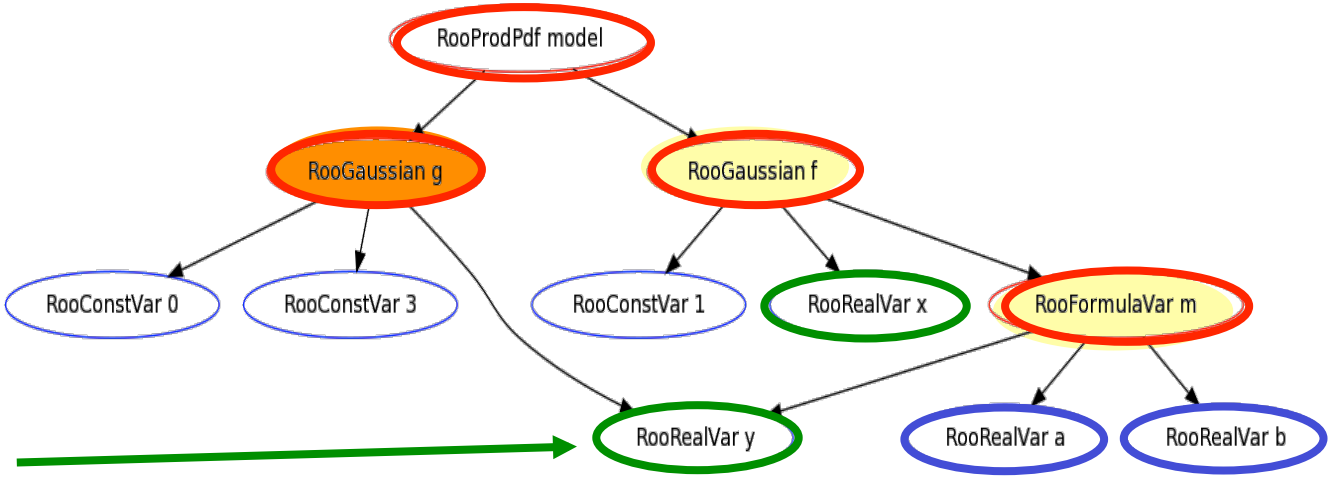
Level-2 optimization of likelihood calculation

- Precalculate all components, with conditional update strategy

Depends on m

Depends on a,b

X	Y	g	f	m
1	2	1.5	..	..
2	3	2.7	..	..
0	1	1.2	..	..
-1	5	0.6	..	..
3	6	9.8	..	..
7	6	3.4	..	..
-3	-2	5.7	..	..



Why does level-2 optimization help?

- Most minuit calls vary one parameter at a time (to calculate derivative of likelihood) → Computed cached values of most components will stay valid

prevFCN = 5170.289989 FCN=5170.53 FROM MIGRAD STATUS=INITIATE 6 CALLS 7 TOTAL
prevFCN = 4495.931306 a=0.9961, b=0.106, c=0.06274,
prevFCN = 3936.921265 a=0.9967,
prevFCN = 3936.938281 a=0.9954,
prevFCN = 3936.907905 a=0.9965,
prevFCN = 3936.933086 a=0.9956,
prevFCN = 3936.911321 a=0.9961, b=0.108,
prevFCN = 3937.05644 b=0.104,
prevFCN = 3936.790003 b=0.1074,
prevFCN = 3937.014478 b=0.1046,
prevFCN = 3936.829929 b=0.106, c=0.06845,
prevFCN = 3936.934463 c=0.05703,
prevFCN = 3936.911648 c=0.06688,
prevFCN = 3936.930463 c=0.05861,
prevFCN = 3936.913944 a=1, b=-0.02103, c=0.02074,
prevFCN = 3936.613348 a=0.9982, b=0.04018, c=0.04096,

Only a changes, caches depending on b,c remain valid

Only b changes, caches depending on a,c remain valid

Only c changes, caches depending on b,c remain valid

- Fraction of likelihood calls with 1-parameter change increases ~linearly with number of parameters

From level-2 optimization to vectorization

- Resequencing of calculation in full level-2 optimization results in 'natural ordering' for complete vectorization

Level-1 sequence

$m(y_0)$
$f(m_0)$
$g(x_0)$
$Model(f_0,g_0)$

$m(y_1)$
$f(m_1)$
$g(x_1)$
$Model(f_1,g_1)$

$m(y_2)$
$f(m_2)$
$g(x_2)$
$Model(f_2,g_2)$

Level-2 sequence

$m(y_0)$
$m(y_1)$
$m(y_2)$

$f(m_0)$
$f(m_1)$
$f(m_2)$

$g(x_0)$
$g(x_1)$
$g(x_2)$

$Model(f_0,g_0)$
$Model(f_1,g_1)$
$Model(f_2,g_2)$

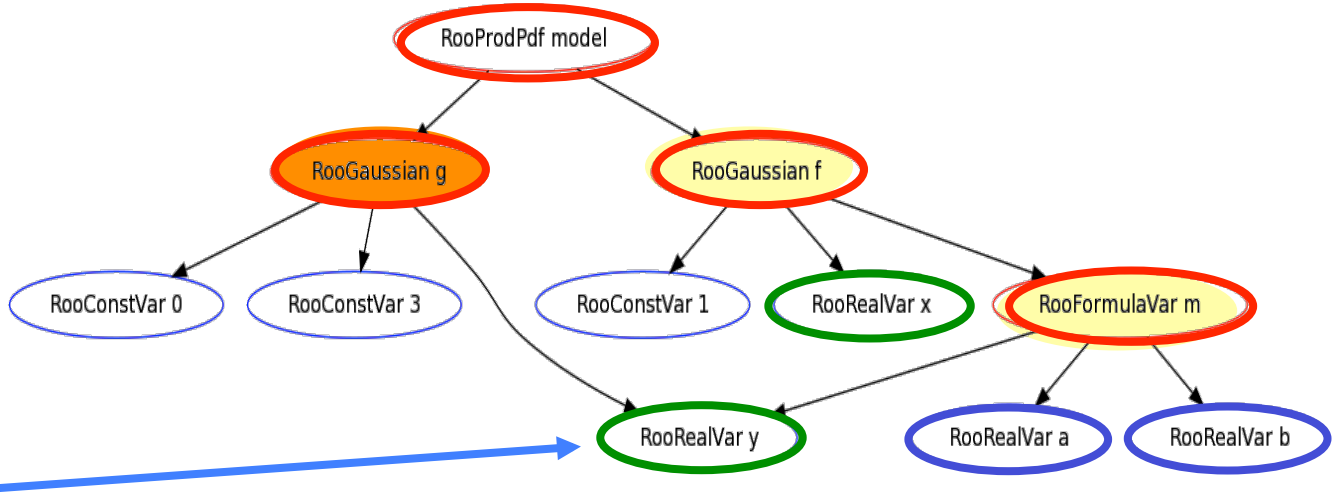
Level-2 optimization of likelihood calculation

- Pass complete vector x to m::evaluate()

Depends on m

Depends on a,b

X	Y	g	f	m
1	2	1.5	..	..
2	3	2.7	..	..
0	1	1.2	..	..
-1	5	0.6	..	..
3	6	9.8	..	..
7	6	3.4	..	..
-3	-2	5.7	..	..



Proxy class inside function object m can be trivially adapted to 'see' full vector x instead of 'current value'

Previous Work

- Alfio Lazzaro, in PH then in OpenLab, built a simplified prototype to test parallelization and vectorization options
  - » Bare-bone code, minimal functionalities and no user-friendly frills
  - » Implement caches at pdf level: fills them, does not reuse them…
  - » Latest development geared toward INTEL compiler and tools (ttb,cilk,mic)
- Results presented in several meetings
  - » CHEP, multicore, openlab
  - » In particular: Edinburgh June 2011
  - » Latest by summer student here in August
- I got the code to try to make it working with gcc
  - » I went much further that that…

Complex Model Test

$$n_a[f_{1,a}G_{1,a}(x) + (1 - f_{1,a})G_{2,a}(x)]AG_{1,a}(y)AG_{2,a}(z) +$$
$$n_b\, G_{1,b}\,(x)BW_{1,b}(y)G_{2,b}(z) +$$
$$n_c\, AR_{1,c}(x)P_{1,c}(y)P_{2,c}(z) +$$
$$n_d\, P_{1,d}(x)G_{1,d}(y)AG_{1,d}(z)$$

Model from B. Aubert *et. al.*,
Phys. Rev. Lett. 98, 031801, 2007

17 PDFs in total, 3 variables, 4 components, 35 parameters

- G: Gaussian
- AG: Asymmetric Gaussian
- BW: Breit-Wigner
- AR: Argus function
- P: Polynomial

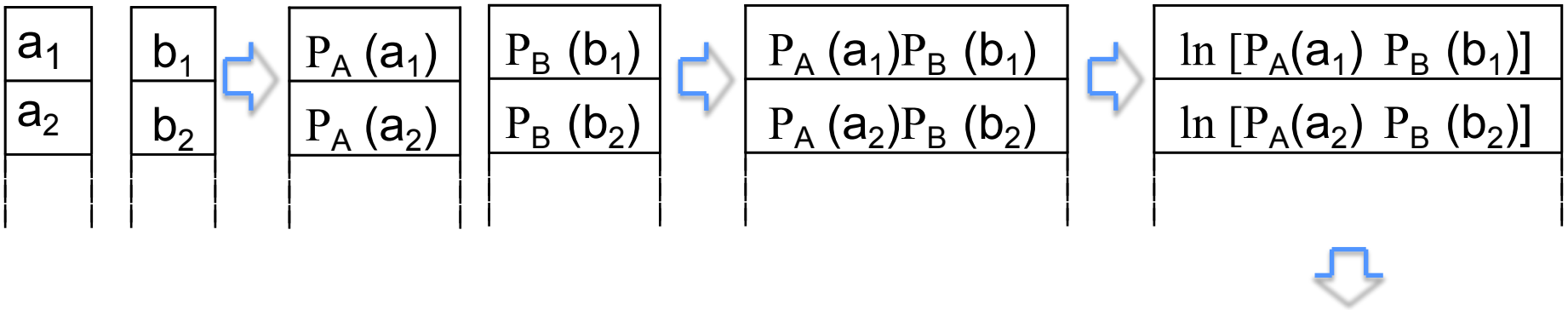
40% of the execution time is spent in exp's calculation

Note: all PDFs have analytical normalization integral

New algorithm and parallelization (1)

1. Read all events and store in arrays in memory
2. For each PDF make the calculation on all events
   - Corresponding array of results is produced for each PDF
   - Evaluation of the function inside the local PDF
3. Combine the arrays of results (composite PDFs)
4. Loop over the final array of results to calculate

Ex: $P = P_A(a_i)\, P_B(b_i)$



Final reduction in *NLL*

Parallel reduction

- The final reduction for the NLL evaluation done in parallel using block-wise algorithm
  - **Numerical approximation w.r.t. sequential reduction, which are number of threads dependent**
  - **Minuit is very sensitive to these approximation**
    - **Of course differences are negligible, but still they can worry people (and can be non deterministic)**

- We implemented a parallel reduction based on double-double algorithm which reduces the approximations (Y. He and C. H. Q.Ding, The Journal of Supercomputing, 18, 259–277, 2001; P. Kornerup *at al.*, IEEE Transactions on Computers, 01 Feb. 2011)
  - **We need to switch off any compiler optimization inside the reduction, using pragmas**
- Now the results are identical up $10^{-6,}$ no matter how many threads you are running

Test on CPU in sequential

- Dual socket Intel Westmere-based system: CPU (L5640) @ 2.27GHz (12 physical cores, 24 hardware threads in total), 10x4096MB DDR3 memory @ 1333MHz
- Linux 64bit, Intel C++ compiler version 12.0.2

# Events	10,000	25,000	50,000	100,000
RooFit				
# NLL evaluations	15810	14540	19041	12834
Time (s)	826.0	1889.0	5192.9	6778.9
Time per NLL evaluation (ms)	52.25	129.92	272.72	528.19
OpenMP (w/o vectorization)				
# NLL evaluations	15237	17671	15761	11396
Time (s)	315.1	916.0	1642.6	2397.3
Time per NLL evaluation (ms)	20.68	51.84	104.22	210.36
w.r.t. RooFit	2.5x	2.5x	2.6x	2.5x
OpenMP (w/ vectorization)				
# NLL evaluations	15304	17163	15331	12665
Time (s)	178.8	492.1	924.2	1536.9
Time per NLL evaluation (ms)	11.68	28.67	60.28	121,35
w.r.t. RooFit	4.5x	4.5x	4.4x	4.4x

4.5x faster!

Vectorization gives a 1.8x speed-up (SSE). Additional 12% using AVX on Intel Sandy Bridge

Alfio Lazzaro
(alfio.lazzaro@cern.ch)

Software:

gcc 4.9.0-gomp4 (gcc 4.8.1 “ok”)

vdt

numactl-devel

perf (2 and 3)

Hardware

Mostly: Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz (8+8)*2

(thanks to Openlab)

THE NEW PROTOTYPE

“It is a capital mistake to theorise before one has data.
Insensibly one begins to twist facts to suit theories instead of theories to suit facts.” - Sherlock Holmes

Moving to gcc

– Vectorization
  - » gcc does not implement "pragma ivdep" to vectorize loop no matter what
  - » Major limitation is that gcc currently vectorizes only loop with one basic block
  - » gcc can use INTEL svml library: I preferred to use vdt

```
#pragma ivdep
   for (Int_t idx = (Int_t)iPartialStart; idx<(Int_t)iPartialEnd; idx++) {
     resultsCPU[idx] = evaluateLocal(dataCPU[idx],
      m_mu->GetVal(),m_sigmaL->GetVal(),
       m_sigmaR->GetVal())*invIntegral;
   }
…
inline Double_t evaluateLocal(const Double_t x,
          const Double_t mu,
          const Double_t sigmaL, const Double_t sigmaR) const {

   Double_t arg = x - mu;
   Double_t coef = 0.0;

   if (arg < coef) {
     if (TMath::Abs(sigmaL)>1e-30)
        {coef = -((Double_t)0.5)/(sigmaL*sigmaL); }
   } else {
     if (TMath::Abs(sigmaR)>1e-30)
        {coef = -((Double_t)0.5)/(sigmaR*sigmaR)'}
   }

   return TMath::Exp(coef*arg*arg);
```

```
   auto invIntegral = GetInvIntegral();

   auto coeffL = -0.5/(m_sigmaL->GetVal()*m_sigmaL->GetVal());
   auto coeffR = -0.5/(m_sigmaR->GetVal()*m_sigmaR->GetVal());
   auto mu = m_mu->GetVal();
#pragma omp simd aligned(res, ldata : ALIGNMENT)
   for (auto idx = 0U; idx<bsize; ++idx) {
     auto x = ldata[idx];
     auto y = evaluateOne(x,mu,coeffL,coeffR)*invIntegral;
     res[idx] = y;
   }
 }
Static double evaluateOne(double x,  double mu,
                          double coeffL,  double coeffR) {

  auto arg = x - mu;
  auto coef = (arg<0) ? coeffL : coeffR;

  return TMath::Exp(coef*arg*arg);
 }
```

vdt in TMath

```
#ifdef STD_MATH
  inline Double_t Tan(Double_t x) { return std::tan(x); }
  inline Double_t Cos(Double_t x) { return std::cos(x); }
  inline Double_t Sin(Double_t x) { return std::sin(x); }
  inline Double_t Log(Double_t x) { return std::log(x); }
  inline Double_t Exp(Double_t x) { return std::exp(x); }
  inline Double_t ATan(Double_t x) { return std::atan(x); }
  inline Double_t ATan2(Double_t y, Double_t x) { return std::atan2(y, x);  }
#else
  inline Double_t Tan(Double_t x) { return vdt::fast_tan(x); }
  inline Double_t Cos(Double_t x) { return vdt::fast_cos(x); }
  inline Double_t Sin(Double_t x) { return vdt::fast_sin(x); }
  inline Double_t Log(Double_t x) { return vdt::fast_log(x); }
  inline Double_t Exp(Double_t x) { return vdt::fast_exp(x); }
  inline Double_t ATan(Double_t x) { return vdt::fast_atan(x); }
  inline Double_t ATan2(Double_t y, Double_t x) { return vdt::fast_atan2(y, x);  }
#endif
```

Prod, Sum, Polynomial

- Original Alfio's code vectorizes Polynomials and Prod/Sum of Pdfs swapping the loops over events and operands
- I templated the Pdfs and used template recursion
  - » gcc inlines easily till order 10, even more

```
template<int N>
class PdfPolynomial : public AbsPdf {
public:
  using Poly = HornerPoly<double, N>;
// N is the number of coefficient, not the order, still below one adds one...
....
 double coeffCPU[size+1];
 loadCoeff(coeffCPU,size);
 assert(m_coeff.GetSize()==N);

 Poly poly(coeffCPU);

#pragma omp simd aligned(res, ldata : ALIGNMENT)
 for (auto idx = 0; idx<bsize; ++idx) {
    auto x = ldata[idx];
    auto y = poly(x)*invIntegral;
    res[idx] = y;
  }
```

```
template<typename T, int N>
class HornerPoly {
public:
  HornerPoly(){}
  HornerPoly(std::initializer_list<T> il) : p(std::begin(il)+1), c0(*il.begin()){}
  HornerPoly(T const coeff[N+1]) : p(coeff+1), c0(*(coeff)){};
  T operator()(T x) const { return c0 + x*p(x); }
private:
  HornerPoly<T,N-1> p;
  T c0;
};
template<typename T>
class HornerPoly<T,0> {
public:
  HornerPoly(){}
  HornerPoly(T coeff) : c0(coeff){};
  HornerPoly(T const * coeff) : c0(*coeff){};
  T operator()(T) const { return c0; }
private:
  T c0;
};
```

NLL reduction

- To perform the sum of Logs, Alfio implemented *the Knuth algorithm (2Sum algorithm)*
  - » Requires precise floating point math: does not vectorize in gcc
- I used an algorithm I developed based on inlined *frexp*
  - » *https://twiki.cern.ch/twiki/bin/view/LCG/VIFastSumLog*
  - » Reduces the number of logs to ONE
  - » Vectorize
  - » Is as precise as using *float128*

Alignment

- For AVX data better be aligned to 32 bytes
  - » Compiler can cope with unknown-alignment by pealing loop
    - More code, more branches, typically slower
  - » Allocate aligned, partition data aligned, hint to the compiler alignment
    - **New: use omp4 pragma (in client code… as seen above)**
  - » Test with a prime number of events!

```
template<typename T>
inline size_t stride(size_t size) {
   auto mask = ALIGNMENT/sizeof(T)-1;// alignment is multiple of 2
   return (size&mask) ? (size|mask) + 1 : size;
}

Data::Data(const char_t* name, const char_t* title,
           uint_t size, uint_t nvars) :
  Named(name,title), m_stride(stride<Value_t>(size)),
m_capacity(nvars*m_stride) {

  m_data =
(Value_t*)memalign(ALIGNMENT,m_capacity*sizeof(Value_t));

}

Value_t * GetData(unsigned int index, unsigned int dataOffset)
{
   return (Value_t *)__builtin_assume_aligned(m_data+dataOffset
+index*m_stride,ALIGNMENT);
  }
```

```
// determine the number of elements per each process/thread (statically allocated)
// determine also the first element
inline int GetElements(int num, int rank,
        int nEvents, int& iStart, int& iEnd) {

  // allign to 4 double boudaries!
  int nEvents4 = nEvents/4;
  int numEventsIn = nEvents4/num;
  int numEventsOut = nEvents4%num;

   iStart = 4*rank*numEventsIn;
   if (rank<numEventsOut) {
    iStart += 4*rank;
    iEnd = iStart + 4*(numEventsIn + 1);
   } else {
    iStart += 4*numEventsOut;
    iEnd = iStart + 4*numEventsIn;
   }
   if (rank==num-1) iEnd+= (nEvents-4*nEvents4);

   return iEnd-iStart;
  }
```

ICC original code: performance

```
39.42%  main_XEON  main_XEON          [.] __svml_exp4_e9
10.08%  main_XEON  libMain_XEON.so    [.] _ZN10PdfArgusBG12evaluateSIMDERKjS1_d.R
10.06%  main_XEON  libMain_XEON.so    [.] _ZN11PdfGaussian12evaluateSIMDERKjS1_d.R
 8.10%  main_XEON  libMain_XEON.so    [.] PdfProd::RunEvaluate(int, unsigned int const&, unsigned int const&, Results const*, unsigned int)
 6.65%  main_XEON  libMain_XEON.so    [.] _ZN13PdfPolynomial12evaluateSIMDERKjS1_d.R
 6.41%  main_XEON  libMain_XEON.so    [.] _ZN6PdfAdd11RunEvaluateEiRKjS1_PK7Resultsjd.R
 4.30%  main_XEON  libMain_XEON.so    [.] NLL::RunParallelReductionOpenMP(Results const*, unsigned int)
 2.74%  main_XEON  libMain_XEON.so    [.] _ZN16PdfBifurGaussian12evaluateSIMDERKjS1_d.R
 2.51%  main_XEON  main_XEON          [.] __svml_log4_e9
 1.21%  main_XEON  libMain_XEON.so    [.] Data::GetCPUData(Variable const&) const
 1.04%  main_XEON  libiomp5.so        [.] __kmp_get_global_thread_id_reg
 0.87%  main_XEON  libiomp5.so        [.] __kmp_wait_sleep
 0.79%  main_XEON  libMain_XEON.so    [.] AbsPdf::GetValSIMD(unsigned int, unsigned int)
 0.67%  main_XEON  libiomp5.so        [.] omp_get_thread_num
```

Performance counter stats for './main_XEON -n 400000 -e -b 512 -i 5000':

```
       280 CPU-migrations            #    0.001 K/sec
372909.609063 task-clock             #    7.672 CPUs utilized
1162360849307 cycles                 #    3.117 GHz                      [45.44%]
990519955378 instructions            #    0.85  insns per cycle
                                     #    0.84  stalled cycles per insn [54.53%]
830226026970 stalled-cycles-frontend #   71.43% frontend cycles idle     [54.53%]
475526814138 stalled-cycles-backend  #   40.91% backend  cycles idle     [54.54%]
  1075312236 cache-misses            #   22.758 % of all cache refs      [54.54%]
  4724933836 cache-references        #   12.670 M/sec                    [54.55%]
   323396088 branch-misses           #    0.867 M/sec                    [36.38%]
 21701773660 L1-dcache-misses        #   58.196 M/sec                    [36.38%]
   276735892 L1-icache-misses        #    0.742 M/sec                    [36.37%]
    20600625 dTLB-misses             #    0.055 M/sec                    [36.37%]
     3766437 iTLB-misses             #    0.010 M/sec                    [36.36%]
```

48.606807743 seconds time elapsed

GCC original code

```
66.26%  main_XEON  libm-2.13.so        [.] __ieee754_exp
 8.03%  main_XEON  libm-2.13.so        [.] __ieee754_log
 6.67%  main_XEON  libm-2.13.so        [.] __GI___exp
 3.78%  main_XEON  libMain_XEON.so     [.] PdfBifurGaussian::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 3.51%  main_XEON  libMain_XEON.so     [.] PdfArgusBG::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 1.93%  main_XEON  libMain_XEON.so     [.] PdfPolynomial::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 1.81%  main_XEON  libm-2.13.so        [.] __finite
 1.81%  main_XEON  libMain_XEON.so     [.] PdfGaussian::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 1.62%  main_XEON  libMain_XEON.so     [.] PdfProd::RunEvaluate(int, unsigned int const&, unsigned int const&, Results const*, unsigned int)
 1.53%  main_XEON  libMain_XEON.so     [.] PdfAdd::RunEvaluate(int, unsigned int const&, unsigned int const&, Results const*, unsigned int, double)
 0.53%  main_XEON  libgomp.so.1.0.0    [.] gomp_team_barrier_wait_end
 0.35%  main_XEON  libm-2.13.so        [.] __log
 0.30%  main_XEON  [kernel.kallsyms]   [k] 0xffffffff8103ba4a
 0.30%  main_XEON  libMain_XEON.so     [.] exp@plt
 0.19%  main_XEON  libMain_XEON.so     [.] NLL::RunParallelReductionOpenMP(Results const*, unsigned int)
```

Performance counter stats for './main_XEON -n 400000 -b 512 -e -i 5000':

```
          3 cpu-migrations            #    0.000 K/sec
  1161886.230447 task-clock           #    7.998 CPUs utilized
  3341146800772 cycles                #    2.876 GHz                      [45.45%]
  4051525171837 instructions          #    1.21  insns per cycle
                                      #    0.51  stalled cycles per insn [54.54%]
  2052500215001 stalled-cycles-frontend #   61.43% frontend cycles idle    [54.53%]
   644329024276 stalled-cycles-backend  #   19.28% backend  cycles idle    [54.54%]
     1270572499 cache-misses           #   25.411 % of all cache refs     [54.54%]
     5000025161 cache-references       #    4.303 M/sec                   [54.55%]
     6955071432 branch-misses          #    5.986 M/sec                   [36.37%]
    25076495412 L1-dcache-misses       #   21.583 M/sec                   [36.37%]
      359741555 L1-icache-load-misses  #    0.310 M/sec                   [36.37%]
       14876971 dTLB-load-misses       #    0.013 M/sec                   [36.36%]
        2568987 iTLB-load-misses       #    0.002 M/sec                   [36.36%]

  145.280745065 seconds time elapsed
```

GCC modified code

```
37.19%  main_XEON  libMain_XEON.so      [.] PdfGaussian::evaluateSIMD(unsigned int const&, unsigned int const&, double)
21.68%  main_XEON  libMain_XEON.so      [.] PdfBifurGaussian::evaluateSIMD(unsigned int const&, unsigned int const&, double)
17.89%  main_XEON  libMain_XEON.so      [.] PdfArgusBG::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 5.80%  main_XEON  main_XEON            [.] PdfAdd<2>::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 5.12%  main_XEON  main_XEON            [.] PdfAdd3Prod<5>::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 2.86%  main_XEON  main_XEON            [.] PdfPolynomial<4>::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 2.80%  main_XEON  main_XEON            [.] PdfPolynomial<1>::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 1.83%  main_XEON  libgomp.so.1.0.0     [.] gomp_team_barrier_wait_end
 1.41%  main_XEON  main_XEON            [.] PdfPolynomial<0>::evaluateSIMD(unsigned int const&, unsigned int const&, double)
 0.78%  main_XEON  libMain_XEON.so      [.] NLL::PartialNegReduction(TMath::IntLog&, double const*, unsigned int, unsigned int)
 0.70%  main_XEON  libMain_XEON.so      [.] AbsPdf::GetValSIMD(unsigned int, unsigned int)
 0.44%  main_XEON  [kernel.kallsyms]    [k] 0xffffffff8103ba4a
 0.33%  main_XEON  libgomp.so.1.0.0     [.] gomp_barrier_wait_end
 0.29%  main_XEON  libc-2.13.so         [.] __strcmp_sse42
 0.14%  main_XEON  libgomp.so.1.0.0     [.] omp_get_thread_num
```

Performance counter stats for './main_XEON -n 400000 -b 512 -e -i 5000':

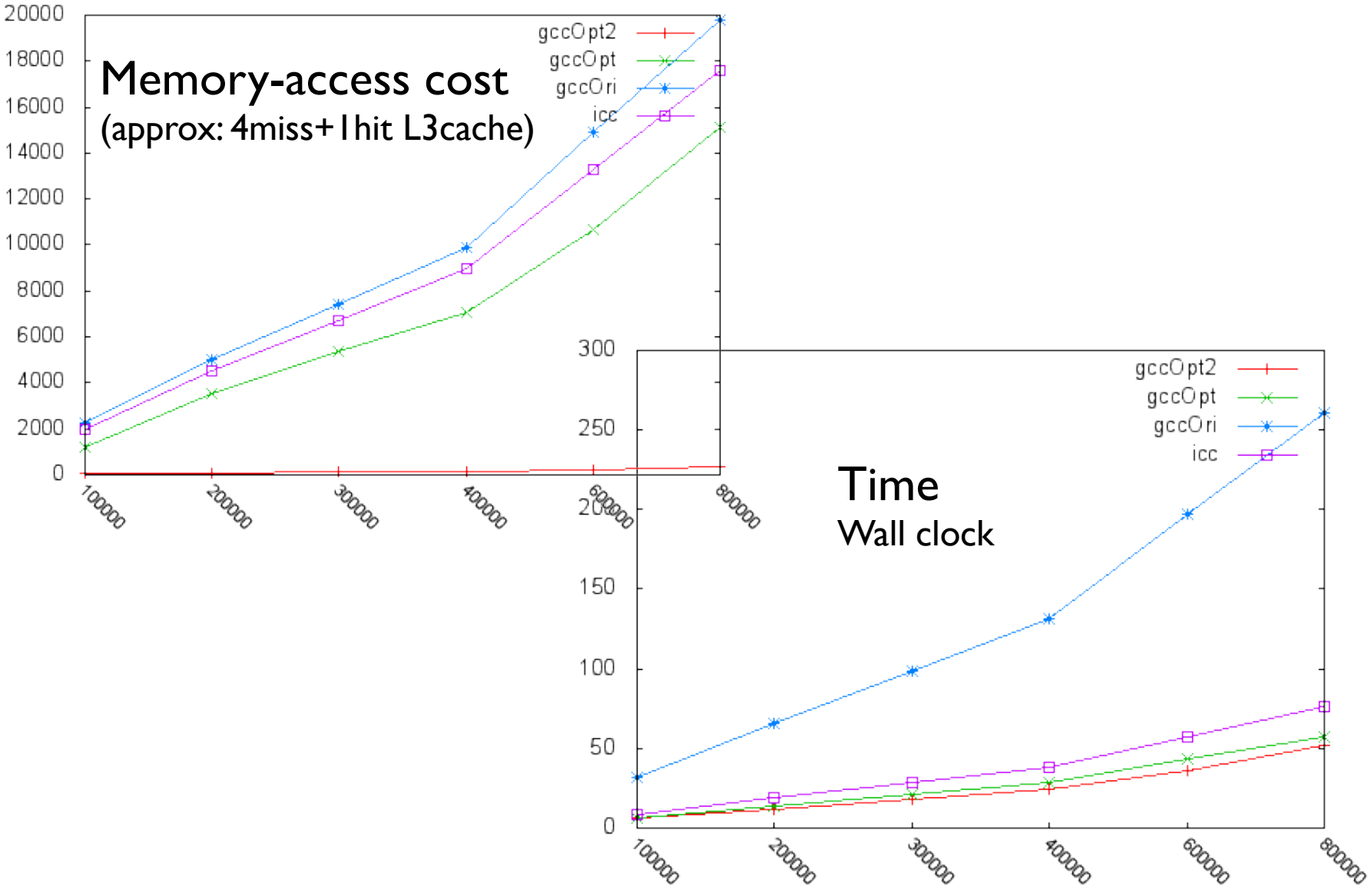
```
            0 cpu-migrations                #    0.000 K/sec
    250672.867448 task-clock                #    7.986 CPUs utilized
     719270518466 cycles                    #    2.869 GHz                      [45.45%]
     775978785874 instructions              #    1.08  insns per cycle
                                            #    0.63  stalled cycles per insn [54.53%]
     491039424152 stalled-cycles-frontend   #   68.27% frontend cycles idle     [54.53%]
     176362929875 stalled-cycles-backend    #   24.52% backend  cycles idle     [54.54%]
        864560863 cache-misses              #   23.913 % of all cache refs      [54.55%]
       3615515275 cache-references          #   14.423 M/sec                    [54.55%]
        326509045 branch-misses             #    1.303 M/sec                    [36.38%]
      16332933400 L1-dcache-misses          #   65.156 M/sec                    [36.38%]
         94278301 L1-icache-load-misses     #    0.376 M/sec                    [36.37%]
         14318840 dTLB-load-misses          #    0.057 M/sec                    [36.36%]
           868572 iTLB-load-misses          #    0.003 M/sec                    [36.36%]

31.390689657 seconds time elapsed
```

GCC: cache filling removed

```
41.76%  main_XEON  libMain_XEON.so    [.] PdfGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
25.78%  main_XEON  libMain_XEON.so    [.] PdfBifurGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
21.84%  main_XEON  libMain_XEON.so    [.] PdfArgusBG::GetVal(double*, unsigned int, Data const&, unsigned int) const
 3.08%  main_XEON  main_XEON          [.] PdfAdd3Prod<5>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 2.19%  main_XEON  libgomp.so.1.0.0   [.] gomp_team_barrier_wait_end
 1.59%  main_XEON  main_XEON          [.] PdfAdd<2>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 1.41%  main_XEON  main_XEON          [.] PdfPolynomial<4>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.93%  main_XEON  libMain_XEON.so    [.] NLL::RunEvaluationBlockSplittingStatic()
 0.65%  main_XEON  main_XEON          [.] PdfPolynomial<1>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.24%  main_XEON  main_XEON          [.] PdfPolynomial<0>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.23%  main_XEON  [kernel.kallsyms]  [k] 0xffffffff8103ba4a
 0.13%  main_XEON  libgomp.so.1.0.0   [.] omp_get_thread_num
 0.11%  main_XEON  libgomp.so.1.0.0   [.] gomp_barrier_wait_end
 0.01%  main_XEON  main_XEON          [.] omp_get_thread_num@plt
 0.01%  main_XEON  libMain_XEON.so    [.] omp_get_thread_num@plt
```

```
Performance counter stats for './main_XEON -n 400000 -b 512 -i 152':

          3 cpu-migrations                #    0.000 K/sec
  212250.350160 task-clock                #    7.978 CPUs utilized
  610763775623 cycles                     #    2.878 GHz                      [45.45%]
  746365726712 instructions               #    1.22  insns per cycle
                                          #    0.52  stalled cycles per insn [54.54%]
  390838561250 stalled-cycles-frontend    #   63.99% frontend cycles idle     [54.54%]
   93664171738 stalled-cycles-backend     #   15.34% backend  cycles idle     [54.54%]
        364193 cache-misses               #    0.066 % of all cache refs      [54.55%]
     555515897 cache-references           #    2.617 M/sec                    [54.55%]
     185045680 branch-misses              #    0.872 M/sec                    [36.37%]
   11924014968 L1-dcache-misses           #   56.179 M/sec                    [36.37%]
      34651487 L1-icache-load-misses      #    0.163 M/sec                    [36.37%]
      12293151 dTLB-load-misses           #    0.058 M/sec                    [36.37%]
         15306 iTLB-load-misses           #    0.072 K/sec                    [36.36%]

  26.606053633 seconds time elapsed
```

Time and memory-access cost



Using the caches

- Most of Minuit function evaluations are used to compute derivatives
  - » Only one parameter is changed each time.
  - » Most of the elementary Pdfs unaffected
- I have implemented cache reuse (in an ad-hoc way) and then simulated the Minuit behaviour (tested also with real Minuit)

```
for(int k=0; k!=nvar; ++k) {
                auto vr = pdfPars()[var[k]];
                auto v = vr->GetVal();
                auto e = vr->GetError();
                vr->SetAllVal(v+e);
                value += nll.GetVal();
                vr->SetAllVal(v-e);
                value -= nll.GetVal();
                vr->SetAllVal(v);
}
```

```
#pragma omp parallel
{
  int iStart=0, iEnd=0;
 unsigned int ntot = OpenMP::GetThreadElements(m_data
                        ->GetEntries(),iStart,iEnd);

  alignas(ALIGNMENT) double res[m_nBlockEvents];
  auto localValue = m_logs[OpenMP::GetRankThread()];
  for (UInt_t ie=0; ie<ntot; ie+= m_nBlockEvents) {
    auto offset = iStart+ie;
    auto bsize = std::min(m_nBlockEvents,ntot-ie);
    m_pdf->GetVal(res, bsize, *m_data, offset);
    PartialNegReduction(localValue,res,bsize);
  }
  m_logs[OpenMP::GetRankThread()] = localValue;
}
```

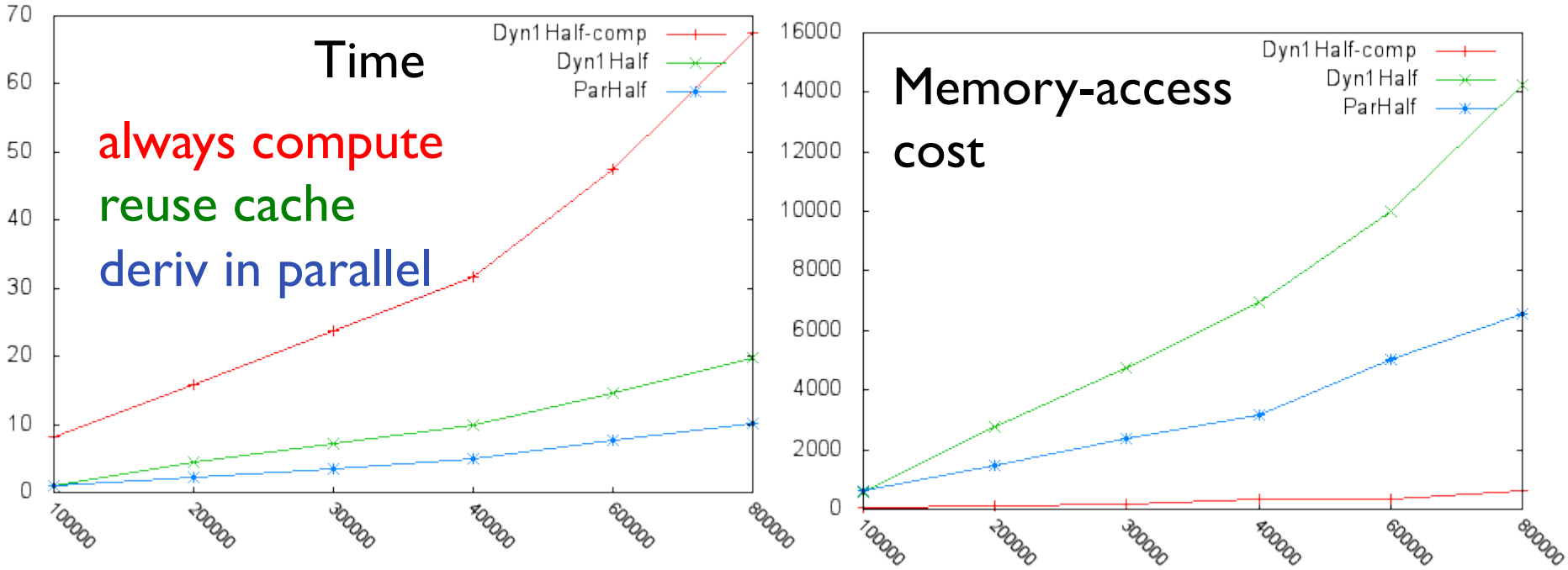
Reusing caches

```
67.18%  main_XEON  main_XEON          [.] PdfAdd3Prod<5>::GetVal(double*, unsigned int, Data const&, unsigned int) const
12.76%  main_XEON  libMain_XEON.so    [.] PdfGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 6.83%  main_XEON  libMain_XEON.so    [.] PdfBifurGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 3.97%  main_XEON  libMain_XEON.so    [.] PdfArgusBG::GetVal(double*, unsigned int, Data const&, unsigned int) const
 3.47%  main_XEON  libMain_XEON.so    [.] NLL::RunEvaluationBlockSplittingStatic()
 1.45%  main_XEON  libgomp.so.1.0.0   [.] gomp_team_barrier_wait_end
 1.12%  main_XEON  libgomp.so.1.0.0   [.] gomp_barrier_wait_end
 0.63%  main_XEON  main_XEON          [.] PdfAdd<2>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.57%  main_XEON  main_XEON          [.] PdfPolynomial<1>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.21%  main_XEON  main_XEON          [.] PdfPolynomial<0>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.20%  main_XEON  main_XEON          [.] PdfPolynomial<4>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.10%  main_XEON  libgomp.so.1.0.0   [.] omp_get_thread_num
 0.07%  main_XEON  [kernel.kallsyms]  [k] trigger_load_balance
 0.06%  main_XEON  [kernel.kallsyms]  [k] entity_tick
 0.06%  main_XEON  [kernel.kallsyms]  [k] hrtimer_interrupt
```

```
Performance counter stats for './main_XEON -n 400000 -b 512 -i 152 -c':

          2 cpu-migrations          #    0.000 K/sec
   64091.389207 task-clock          #    7.974 CPUs utilized
  182571479362 cycles               #    2.849 GHz                     [45.44%]
  100570669737 instructions         #    0.55  insns per cycle
                                    #    1.50  stalled cycles per insn [54.53%]
  150529703690 stalled-cycles-frontend  #   82.45% frontend cycles idle    [54.53%]
  114789874700 stalled-cycles-backend   #   62.87% backend  cycles idle    [54.54%]
     678074129 cache-misses         #   30.151 % of all cache refs      [54.57%]
    2248955994 cache-references     #   35.090 M/sec                    [54.57%]
      25924635 branch-misses        #    0.404 M/sec                    [36.39%]
    4942233830 L1-dcache-misses     #   77.112 M/sec                    [36.38%]
      21273768 L1-icache-load-misses #    0.332 M/sec                   [36.37%]
        496949 dTLB-load-misses     #    0.008 M/sec                    [36.36%]
          9181 iTLB-load-misses     #    0.143 K/sec                    [36.35%]

8.037609565 seconds time elapsed
```

Moving the parallel section “in Minuit”

- L3 Cache and memory access looks dominant:
- Cache reuse can be enhanced by computing all derivatives in parallel
  - » All threads will access pdfs’ and data caches for the same event block
  - » Obvious scaling issues for N-Thread>N-derivatives and for unbalanced computational load of pdfs



Code detail

```
                // outer parallel...
                auto nloops = 2*nvar;
                std::atomic<int> ok[nvar];for ( auto & o : ok) o=0;
                std::atomic<int> al(0);
#pragma omp parallel reduction(+ : value)
                {
                  int il=0;
                  while(true) {
                    if (il>=nloops) break;
                    while (il<nloops && !std::atomic_compare_exchange_weak(&al,&il,il+1));
                    if (il>=nloops) break;
                    auto ik = il/2; // hope optmize in >>1
                    bool pm = 0 == il%2;  // hope optmize in &1
                    auto vr = pdfPars()[var[ik]];
                    auto v = vr->GetVal();
                    auto e = vr->GetError();
                    auto nv = pm ? v+e : v-e;
                    vr->SetVal(nv);
                    if (pm)
                      value += nll.GetVal(var[ik]); // sequential
                    else
                      value -= nll.GetVal(var[ik]); // sequential

                    vr->SetVal(v);
                    // now the integral is wrong fir this thread...  (does not matter!)
                    nll.GetPdf()->CacheIntegral(var[ik]);
                    ok[ik]++;
                  }
                } // end parallel section
                for ( auto const & o : ok) assert(2==o);
```

Reuse cache. Derivatives in parallel

```
42.51%  main_XEON  main_XEON          [.] PdfAdd3Prod<5>::GetVal(double*, unsigned int, Data const&, unsigned int) const
24.00%  main_XEON  libMain_XEON.so    [.] PdfGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
12.87%  main_XEON  libMain_XEON.so    [.] PdfBifurGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 7.58%  main_XEON  libMain_XEON.so    [.] PdfArgusBG::GetVal(double*, unsigned int, Data const&, unsigned int) const
 6.05%  main_XEON  libMain_XEON.so    [.] NLL::GetVal(int)
 2.93%  main_XEON  libgomp.so.1.0.0   [.] gomp_team_barrier_wait_end
 1.14%  main_XEON  main_XEON          [.] PdfAdd<2>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.58%  main_XEON  main_XEON          [.] PdfPolynomial<1>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.39%  main_XEON  main_XEON          [.] PdfPolynomial<4>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.33%  main_XEON  libgomp.so.1.0.0   [.] gomp_barrier_wait_end
 0.33%  main_XEON  main_XEON          [.] PdfPolynomial<0>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.21%  main_XEON  libMain_XEON.so    [.] NLL::RunEvaluationBlockSplittingStatic()
 0.11%  main_XEON  libgomp.so.1.0.0   [.] omp_get_thread_num
 0.08%  main_XEON  [kernel.kallsyms]  [k] clear_page_c
 0.05%  main_XEON  main_XEON          [.] omp_get_thread_num@plt
 0.05%  main_XEON  [kernel.kallsyms]  [k] update_cfs_rq_blocked_load
 0.04%  main_XEON  libMain_XEON.so    [.] Variable::SetAllVal(double)
```

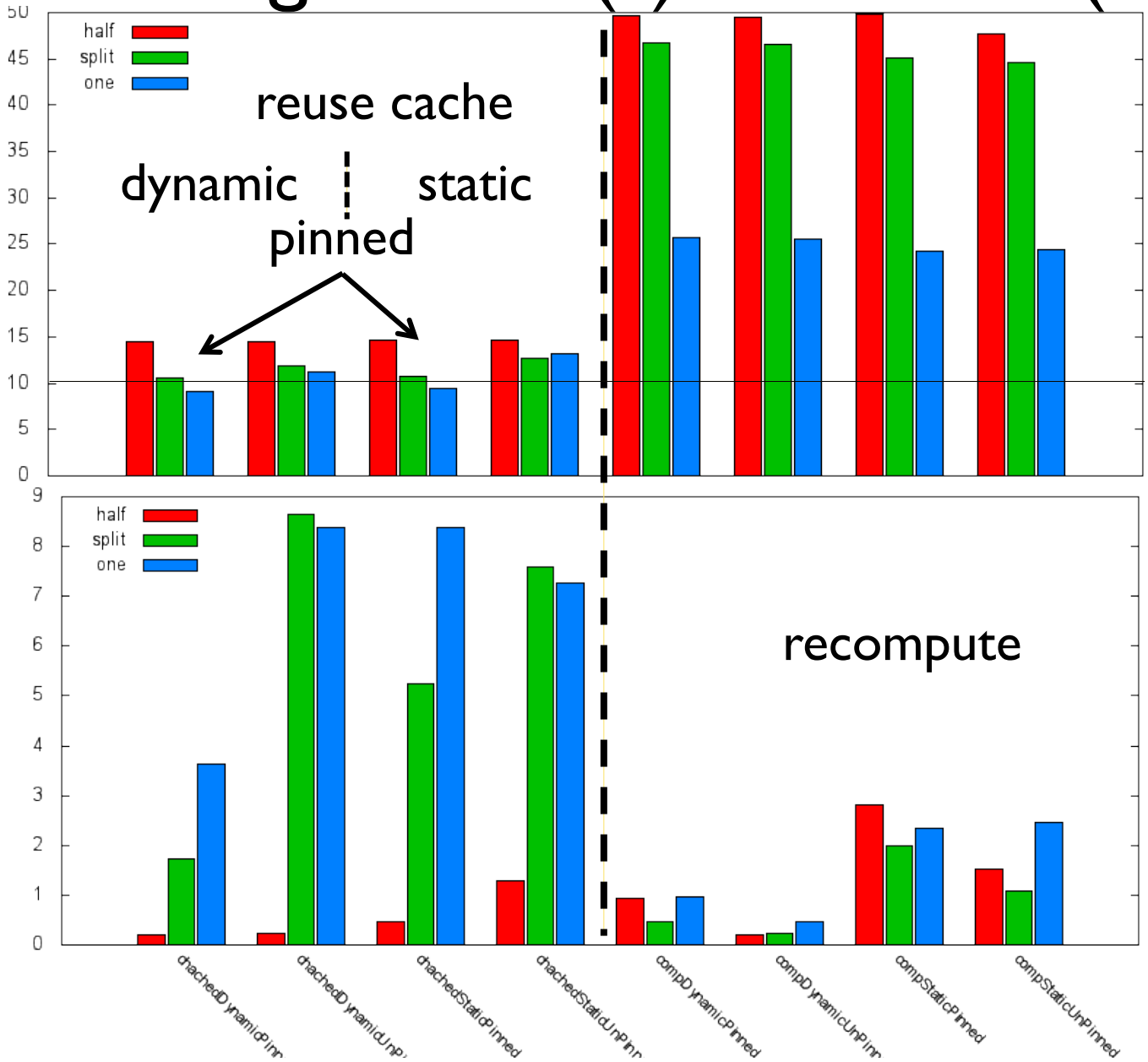
Performance counter stats for './main_XEON -n 400000 -b 512 -i 152 -p':

```
          3 cpu-migrations            #    0.000 K/sec
  32538.523152 task-clock             #    7.864 CPUs utilized
   93286042774 cycles                 #    2.867 GHz                      [45.43%]
   99825328477 instructions           #    1.07  insns per cycle
                                      #    0.62  stalled cycles per insn [54.53%]
   61922012782 stalled-cycles-frontend #   66.38% frontend cycles idle     [54.53%]
   28178464087 stalled-cycles-backend  #   30.21% backend  cycles idle     [54.56%]
      65247629 cache-misses            #    2.959 % of all cache refs      [54.60%]
    2205003881 cache-references        #   67.766 M/sec                    [54.59%]
      22924386 branch-misses           #    0.705 M/sec                    [36.39%]
    4922520242 L1-dcache-misses        #  151.283 M/sec                    [36.38%]
     6409681 L1-icache-load-misses     #    0.197 M/sec                    [36.36%]
     1943793 dTLB-load-misses          #    0.060 M/sec                    [36.34%]
       19566 iTLB-load-misses          #    0.601 K/sec                    [36.34%]

4.137624256 seconds time elapsed
```

Scheduling issues on Multi-socket

- All previous results where for 8 thread on a single socket
- Moving to two sockets one faces issues related to thread scheduling
  - » They exist also on a single socket in presence of other load such as on lxplus
- Timing for jobs with static partitioning suffers of a large variance
  - » Pinning Threads to CPUs alleviates somehow the problem as enhance memory affinity, does not solve load unbalance
  - » Standard dynamic scheduling plays against cache/memory affinity even with pinning
- I have implemented a dynamic scheduling based on work stealing that accounts for the Hardware organization
  - » The problem is statically partitioned among the sockets
  - » In each socket event-blocks are scheduled dynamically

Average Time (s) and RMS (%) 10 jobs



Half: cpu 0-7

Split cpu 0-3,8-11

One cpu 0-15

NUMA

- Memory Affinity is a know issue:
  - » How it affects our prototype?

```
numactl -C 0-7 ./main_XEON -n 768000 -b 512 -i -150 -c
OpenMP Real Time (s) = 14.41968
numactl -C 0-7 -m 0 ./main_XEON -n 768000 -b 512 -i -150 -c
OpenMP Real Time (s) = 14.59128
numactl -C 0-7 -m 1 ./main_XEON -n 768000 -b 512 -i -150 -c
OpenMP Real Time (s) = 30.61127

numactl -C 0-3,8-11 ./main_XEON -n 768000 -b 512 -i -150 -c
OpenMP Real Time (s) = 10.24753
numactl -C 0-3,8-11 -m 0 ./main_XEON -n 768000 -b 512 -i -150 -c
OpenMP Real Time (s) = 17.24593
numactl -C 0-3,8-11 -m 1 ./main_XEON -n 768000 -b 512 -i -150 -c
OpenMP Real Time (s) = 17.29603
numactl -C 0-3,8-11 --interleave=0,1 ./main_XEON -n 768000 -b 512 -i -150 -c
OpenMP Real Time (s) = 11.11212
numactl -C 0-3,8-11 --localalloc ./main_XEON -n 768000 -b 512 -i -150 -c
OpenMP Real Time (s) = 10.23134

numactl -C 0-3,8-11 ./main_XEON -n 768000 -b 512 -i -150 -c -a 1
OpenMP Real Time (s) = 17.17519
numactl -C 0-3,8-11 ./main_XEON -n 768000 -b 512 -i -150 -c -a 2
OpenMP Real Time (s) = 9.71343
```

It seems that memory is allocated in the "correct thread"
This is strange as all caches are allocated in the "main thread"
Executing "malloc" in a different thread made no difference

"read the small prints"

The page interleaving only occurs on the actual page fault that puts a new page into the current address space

Malloc allocates zero-pages: all pointing to the same one. (and it can be a huge-page!)

Numa aware allocation

```
#pragma omp parallel
  {
    // assume each thread will allocate in its own NUMA side
    // select one for each partition
    bool t0 = 0== omp_get_thread_num()%(omp_get_num_threads()/inPart());
    if (t0) {
      auto me = partition();
      int ls=0; int le=0;
      auto nev = Partitioner::GetElements(inPart(),me,size,ls,le);
      m_stride[me] = stride(nev);
      m_capacity[me]= nvars*m_stride[me];
#ifdef NUMACTL
      m_data[me]= (Value_t*)numa_alloc_local(m_capacity[me]*sizeof(Value_t));
#else
      m_data[me]= (Value_t*)memalign(ALIGNMENT,m_capacity[me]*sizeof(Value_t));
      // force the OS to allocate physical memory for the region
      memset(m_data[me], -1, m_capacity[me]*sizeof(Value_t));
#endif
      m_start[me]=ls;
    }
```

Caches are partitioned and each partition is allocated in the proper thread

To make affinity to help when making derivatives in parallel I need to split also their computation according to event location:
Becoming harder, more complex and quite ad-hoc

```
numactl -C 0-7 ./main_XEON -n 768000 -b 512 -i -150 –p          OpenMP Real Time (s) = 7.87001
numactl -C 0-3,8-11 ./main_XEON -n 768000 -b 512 -i -150 -p -a 0    OpenMP Real Time (s) = 8.80387
numactl -C 0-3,8-11 ./main_XEON -n 768000 -b 512 -i -150 -p -a 1    OpenMP Real Time (s) = 8.83677
numactl -C 0-3,8-11 ./main_XEON -n 768000 -b 512 -i -150 -p -a 2    OpenMP Real Time (s) = 7.46227
numactl -C 0-15 ./main_XEON -n 768000 -b 512 -i -150 -p -a 0        OpenMP Real Time (s) = 5.27249
numactl -C 0-15 ./main_XEON -n 768000 -b 512 -i -150 -p -a 1        OpenMP Real Time (s) = 5.67422
numactl -C 0-15 ./main_XEON -n 768000 -b 512 -i -150 -p -a 2        OpenMP Real Time (s) = 3.81434
numactl -C 0-31 ./main_XEON -n 768000 -b 512 -i -150 -p -a 4        OpenMP Real Time (s) = 3.12215
```

Messy code

```
                // outer parallel...
                auto nloops = 2*nvar;
                std::atomic<int> ok[nvar];for ( auto & o : ok) o=0;
                std::atomic<int> alP[Data::inPart()]; for ( auto & o : alP) o=0;
                double lval[OpenMP::GetMaxNumThreads()]={0.};
#pragma omp parallel
                {
                  auto par = Data::partition();
                  // auto start = data.startP();
                  // auto size  = data.sizeP();
                  std::atomic<int> & al = alP[par];
                  int il=0;
                  while(true) {
                    if (il>=nloops) break;
                   while (il<nloops && !std::atomic_compare_exchange_weak(&al,&il,il+1));
                    if (il>=nloops) break;
                    auto ik = il/2; // hope optmize in >1
                    bool pm = 0 == il%2;  // hope optmize in &1
                    auto vr = pdfPars()[var[ik]];
                    auto v = vr->GetVal();
                    auto e = vr->GetError();
                    auto nv = pm ? v+e : v-e;
                    vr->SetVal(nv);
                    if (pm)
                      lval[omp_get_thread_num()] += nll.GetVal(var[ik]);
                    else
                      lval[omp_get_thread_num()] -= nll.GetVal(var[ik]);

                    vr->SetVal(v);
                    // now the integral is wrong fir this thread...  (does not matter!)
                    nll.GetPdf()->CacheIntegral(var[ik]);
                    ok[ik]++;
                  }
                } // end parallel section
                for ( auto const & o : ok) assert(2*Data::inPart()==o);
                for (auto v: lval) value+=v;
```

```
double  NLL::GetVal(int lpar) {
  // assume to be in a thread..

  auto ntot = m_data->GetEntries();
  auto par = 0U;
  auto start =0U;
  auto size   =  ntot;

  // a hack
  if ( OpenMP::IsInParallel()) {
    par = Data::partition();
    start =  m_data->startP();
    size   =  m_data->sizeP();
  }

  m_pdf->CacheIntegral(lpar);


  alignas(ALIGNMENT) double res[m_nBlockEvents];
  TMath::IntLog localValue;
  for (auto ie=start; ie<start+size; ie+= m_nBlockEvents) {
    auto offset = ie;
    auto bsize = std::min(m_nBlockEvents,(start+size)-ie);
    m_pdf->GetVal(res, bsize, *m_data, offset);
    PartialNegReduction(localValue,res,bsize);
  }
  auto ret = -0.693147182464599609375*localValue.value();

  if (par==0 && m_pdf->IsExtended())
    ret += m_pdf->ExtendedTerm(ntot);

  return ret;
}
```

Memory bandwidth measurements

olsnba04 with localalloc
Sequential read (128-bit), size = 10 MB, loops = 11970, 23939.3 MB/s
Sequential read (128-bit), size = 12 MB, loops = 9975, 23936.6 MB/s
Sequential read (128-bit), size = 14 MB, loops = 8548, 23932.7 MB/s
Sequential read (128-bit), size = 15 MB, loops = 7872, 23609.6 MB/s
Sequential read (128-bit), size = 16 MB, loops = 7352, **23525.6 MB/s**
Sequential read (128-bit), size = 20 MB, loops = 4614, 18450.2 MB/s
Sequential read (128-bit), size = 21 MB, loops = 3780, 15874.1 MB/s
Sequential read (128-bit), size = 32 MB, loops = 1904, 12182.9 MB/s
Sequential read (128-bit), size = 48 MB, loops = 1229, 11795.9 MB/s
Sequential read (128-bit), size = 64 MB, loops = 923, 11807.4 MB/s
Sequential read (128-bit), size = 72 MB, loops = 821, 11815.6 MB/s
Sequential read (128-bit), size = 96 MB, loops = 616, 11816.5 MB/s
Sequential read (128-bit), size = 128 MB, loops = 462, **11822.5 MB/s**

olsnba04 with interleave=0,1
Sequential read (128-bit), size = 10 MB, loops = 12006, 24004.8 MB/s
Sequential read (128-bit), size = 12 MB, loops = 10005, 24002.5 MB/s
Sequential read (128-bit), size = 14 MB, loops = 8512, 23824.2 MB/s
Sequential read (128-bit), size = 15 MB, loops = 7948, 23837.3 MB/s
Sequential read (128-bit), size = 16 MB, loops = 7316, **23401.4 MB/s**
Sequential read (128-bit), size = 20 MB, loops = 3711, 14832.2 MB/s
Sequential read (128-bit), size = 21 MB, loops = 2901, 12177.6 MB/s
Sequential read (128-bit), size = 32 MB, loops = 1408, 9006.0 MB/s
Sequential read (128-bit), size = 48 MB, loops = 926, 8889.2 MB/s
Sequential read (128-bit), size = 64 MB, loops = 696, 8908.1 MB/s
Sequential read (128-bit), size = 72 MB, loops = 620, 8917.6 MB/s
Sequential read (128-bit), size = 96 MB, loops = 465, 8926.2 MB/s
Sequential read (128-bit), size = 128 MB, loops = 349, **8933.9 MB/s**

My memory cost approximation
is more for interleave than for local alloc….
Qualitatively results are anyhow very similar

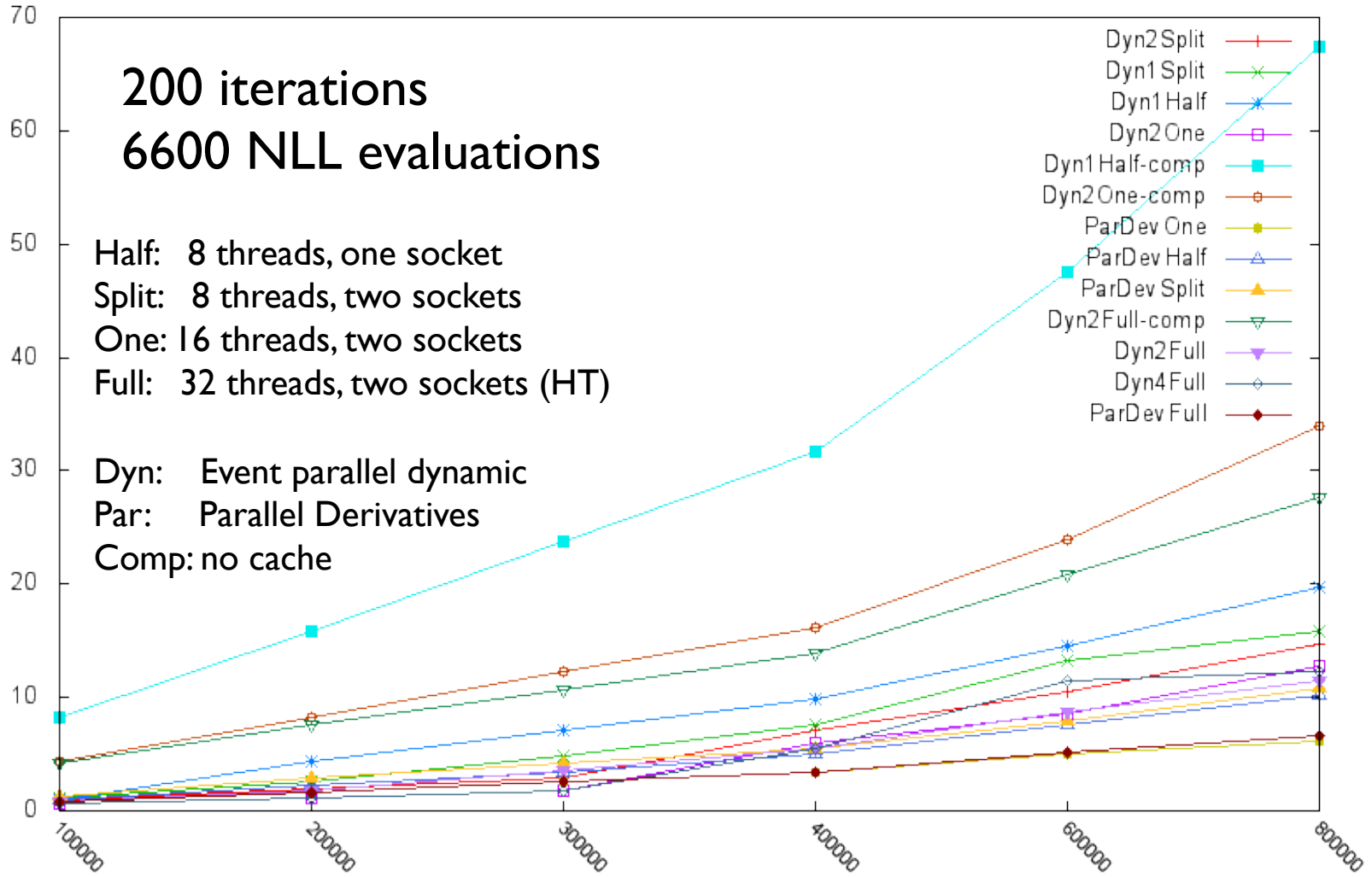
Detailed mem analysis

taskset –c 0-15 perf stat –e… ./main_XEON -n 600000 -b 512 -p -d 2 -i 152

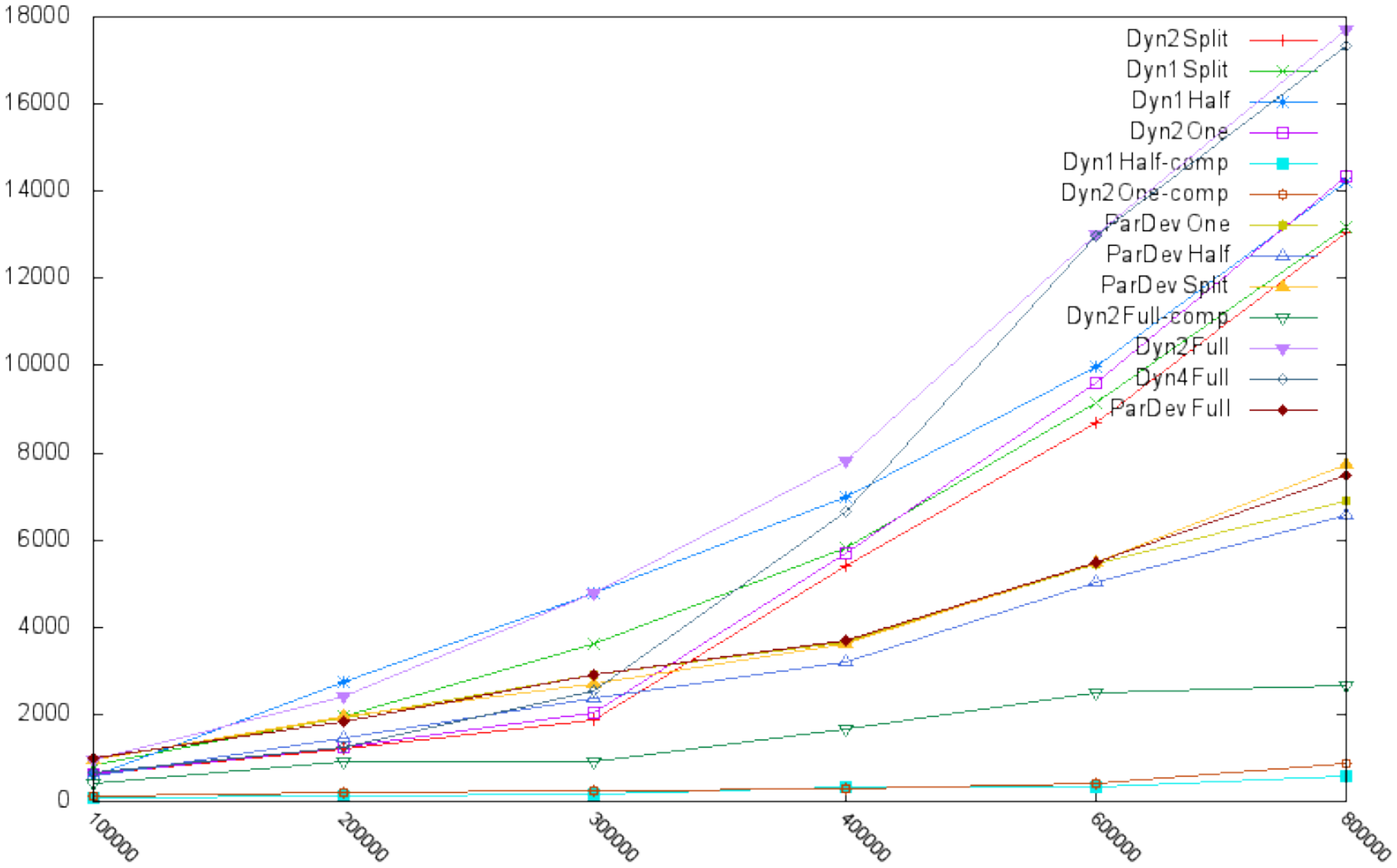
Default (one partition no numa alloc)	numa alloc two partitions (-a 2)	Two partitions alloc in main-thread (-a -2)
163036624 cache-misses	103213043 cache-misses	122604498 cache-misses
1413514986 cache-references	1483727744 cache-references	1498917241 cache-references
673610 r01D2	601285 r01D2	495444 r01D2
22522382 r02D2	23165946 r02D2	21901136 r02D2
83872 r04D2	68036 r04D2	84363 r04D2
131413592 r08D2	148643502 r08D2	139812626 r08D2
561324 r01D3	1081797 r01D3	510208 r01D3
1172361 r04D3	464 r04D3	535629 r04D3
449011 r10D3	92 r10D3	1476 r10D3
697892 r20D3	277 r20D3	3209 r20D3
3.179352909 seconds time elapsed	2.364013725 seconds time elapsed	3.118868190 seconds time elapsed

```
__PMC_EV_ALIAS("MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS",         IAP_EVENT_D2H_01H)
__PMC_EV_ALIAS("MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT",          IAP_EVENT_D2H_02H)
__PMC_EV_ALIAS("MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM",         IAP_EVENT_D2H_04H)
__PMC_EV_ALIAS("MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE”,         IAP_EVENT_D2H_08H)
__PMC_EV_ALIAS("MEM_LOAD_UOPS_LLC_MISS_RETIRED.LOCAL_DRAM”,       IAP_EVENT_D3H_01H)
__PMC_EV_ALIAS("MEM_LOAD_UOPS_LLC_MISS_RETIRED.REMOTE_DRAM”,      IAP_EVENT_D3H_04H)
__PMC_EV_ALIAS("MEM_LOAD_UOPS_LLC_MISS_RETIRED.REMOTE_HITM”,      IAP_EVENT_D3H_10H)
__PMC_EV_ALIAS("MEM_LOAD_UOPS_LLC_MISS_RETIRED.REMOTE_FWD”,       IAP_EVENT_D3H_20H)
```

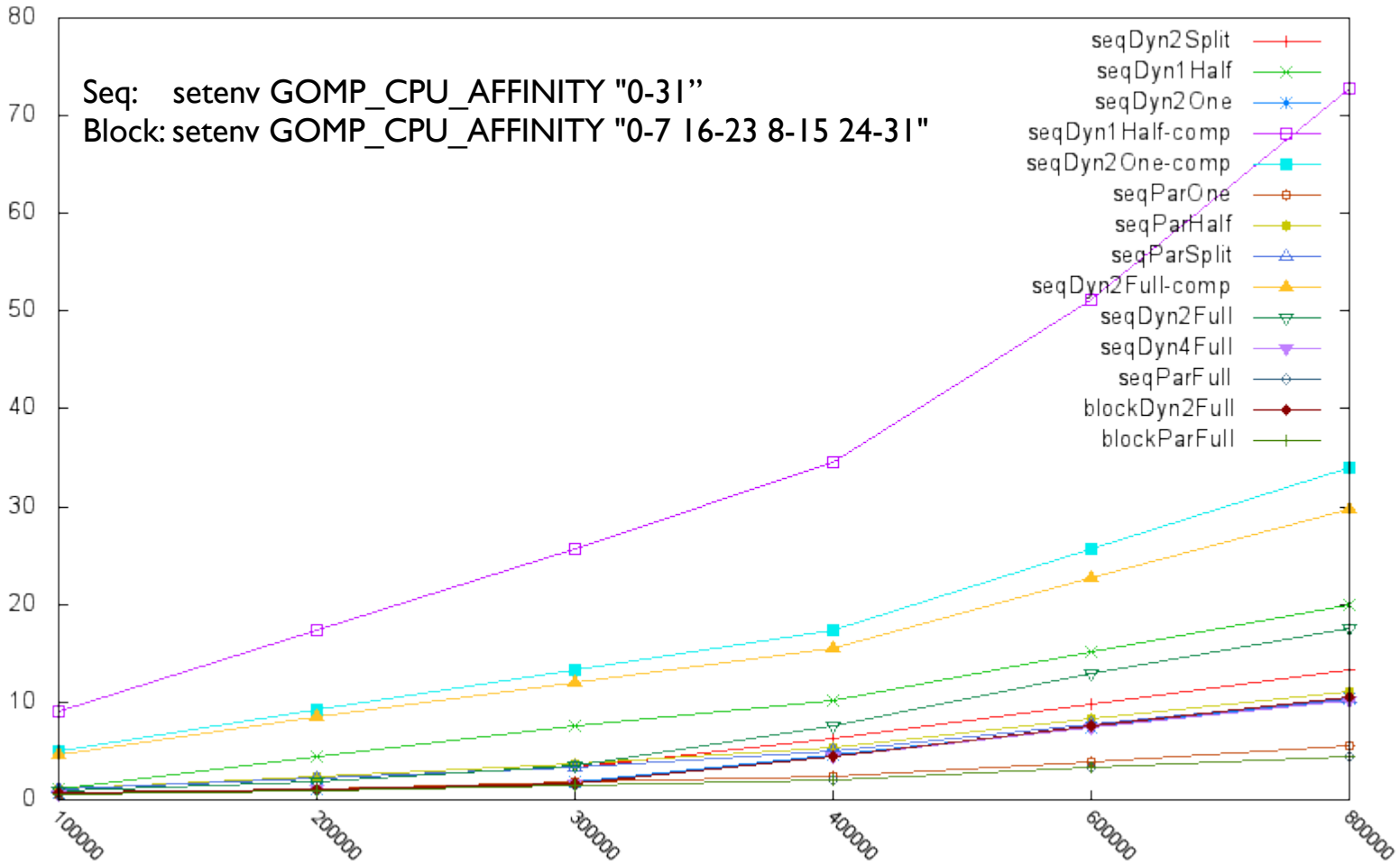
Time (no NUMA optimization)



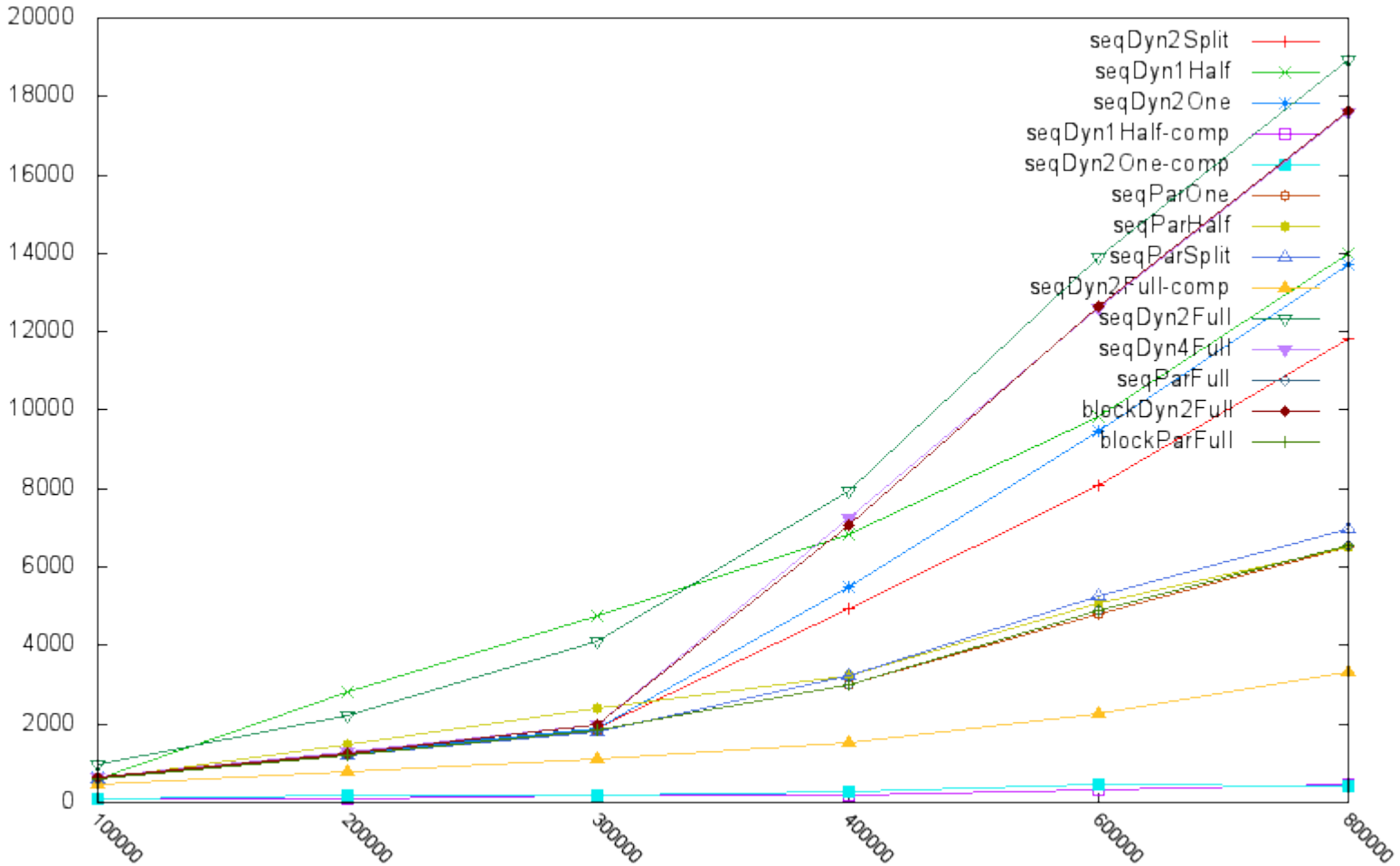
Memory cost (no NUMA optimization)



time (NUMA optimized)



Memory cost (NUMA optimized)



Cache reuse. Event parallel. 16 threads

```
58.53%  main_XEON  main_XEON          [.] PdfAdd3Prod<5>::GetVal(double*, unsigned int, Data const&, unsigned int) const
14.42%  main_XEON  libMain_XEON.so    [.] PdfGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 7.77%  main_XEON  libMain_XEON.so    [.] PdfBifurGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 4.52%  main_XEON  libMain_XEON.so    [.] PdfArgusBG::GetVal(double*, unsigned int, Data const&, unsigned int) const
 4.26%  main_XEON  libMain_XEON.so    [.] NLL::RunEvaluationBlockSplittingDynamic(std::atomic<int>*, int const*)
 4.04%  main_XEON  libgomp.so.1.0.0   [.] gomp_team_barrier_wait_end
 2.55%  main_XEON  libgomp.so.1.0.0   [.] gomp_barrier_wait_end
 0.75%  main_XEON  main_XEON          [.] PdfPolynomial<1>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.71%  main_XEON  main_XEON          [.] PdfAdd<2>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.24%  main_XEON  main_XEON          [.] PdfPolynomial<4>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.22%  main_XEON  main_XEON          [.] PdfPolynomial<0>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.15%  main_XEON  [kernel.kallsyms]  [k] update_cfs_rq_blocked_load
 0.11%  main_XEON  libgomp.so.1.0.0   [.] gomp_barrier_wait
 0.09%  main_XEON  [kernel.kallsyms]  [k] entity_tick
```

Performance counter stats for './main_XEON -n 400000 -b 512 -i 152 -c -d 2':

```
          4 cpu-migrations            #    0.000 K/sec
   64395.708308 task-clock            #   15.869 CPUs utilized
  183949764038 cycles                 #    2.857 GHz                      [45.44%]
  107448120541 instructions           #    0.58  insns per cycle
                                      #    1.35  stalled cycles per insn [54.51%]
  145441670962 stalled-cycles-frontend #   79.07% frontend cycles idle    [54.49%]
  107659468230 stalled-cycles-backend  #   58.53% backend  cycles idle    [54.49%]
     467236145 cache-misses           #   20.192 % of all cache refs     [54.50%]
    2313991354 cache-references       #   35.934 M/sec                   [54.53%]
      31911156 branch-misses          #    0.496 M/sec                   [36.44%]
    4994810105 L1-dcache-misses       #   77.564 M/sec                   [36.43%]
      23990329 L1-icache-load-misses  #    0.373 M/sec                   [36.42%]
        990736 dTLB-load-misses       #    0.015 M/sec                   [36.40%]
      19943340 iTLB-load-misses       #    0.310 M/sec                   [36.37%]

   4.057983758 seconds time elapsed
```

Parallel Deriv: 16 threads (numa no-opt)

```
46.45%  main_XEON  main_XEON          [.] PdfAdd3Prod<5>::GetVal(double*, unsigned int, Data const&, unsigned int) const
20.02%  main_XEON  libMain_XEON.so     [.] PdfGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
10.85%  main_XEON  libMain_XEON.so     [.] PdfBifurGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 6.67%  main_XEON  libgomp.so.1.0.0    [.] gomp_team_barrier_wait_end
 6.43%  main_XEON  libMain_XEON.so     [.] PdfArgusBG::GetVal(double*, unsigned int, Data const&, unsigned int) const
 5.17%  main_XEON  libMain_XEON.so     [.] NLL::GetVal(int)
 1.11%  main_XEON  main_XEON           [.] PdfAdd<2>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.70%  main_XEON  libgomp.so.1.0.0    [.] gomp_barrier_wait_end
 0.63%  main_XEON  main_XEON           [.] PdfPolynomial<1>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.47%  main_XEON  main_XEON           [.] PdfPolynomial<4>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.29%  main_XEON  main_XEON           [.] PdfPolynomial<0>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.19%  main_XEON  libMain_XEON.so     [.] NLL::RunEvaluationBlockSplittingStatic()
 0.11%  main_XEON  libgomp.so.1.0.0    [.] omp_get_thread_num
```

```
Performance counter stats for './main_XEON -n 400000 -b 512 -i 152 -p':

          4 cpu-migrations            #    0.000 K/sec
  39569.799971 task-clock             #   14.582 CPUs utilized
  113615840448 cycles                 #    2.871 GHz                      [45.40%]
  101420762979 instructions           #    0.89  insns per cycle
                                      #    0.79  stalled cycles per insn [54.51%]
   80376412195 stalled-cycles-frontend #   70.74% frontend cycles idle    [54.52%]
   45472941302 stalled-cycles-backend  #   40.02% backend  cycles idle    [54.56%]
     137067539 cache-misses            #    5.885 % of all cache refs     [54.58%]
    2328958537 cache-references        #   58.857 M/sec                   [54.63%]
      22437438 branch-misses           #    0.567 M/sec                   [36.42%]
    5000977587 L1-dcache-misses        #  126.384 M/sec                   [36.40%]
       8416272 L1-icache-load-misses   #    0.213 M/sec                   [36.38%]
       2348469 dTLB-load-misses        #    0.059 M/sec                   [36.35%]
        100091 iTLB-load-misses        #    0.003 M/sec                   [36.32%]

  2.713639482 seconds time elapsed
```

Event parallel: 16 threads (numa opt)

```
60.33%  main_XEON  main_XEON          [.] PdfAdd3Prod<5>::GetVal(double*, unsigned int, Data const&, unsigned int) const
14.11%  main_XEON  libMain_XEON.so     [.] PdfGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 7.67%  main_XEON  libMain_XEON.so     [.] PdfBifurGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 4.51%  main_XEON  libMain_XEON.so     [.] PdfArgusBG::GetVal(double*, unsigned int, Data const&, unsigned int) const
 4.36%  main_XEON  libMain_XEON.so     [.] NLL::RunEvaluationBlockSplittingDynamic(std::atomic<int>*, int const*)
 2.48%  main_XEON  libgomp.so.1.0.0    [.] gomp_barrier_wait_end
 2.36%  main_XEON  libgomp.so.1.0.0    [.] gomp_team_barrier_wait_end
 0.64%  main_XEON  main_XEON          [.] PdfAdd<2>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.62%  main_XEON  main_XEON          [.] PdfPolynomial<1>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.23%  main_XEON  main_XEON          [.] PdfPolynomial<4>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.22%  main_XEON  main_XEON          [.] PdfPolynomial<0>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.20%  main_XEON  [kernel.kallsyms]  [k] update_cfs_rq_blocked_load
 0.17%  main_XEON  libgomp.so.1.0.0    [.] gomp_barrier_wait
```

```
Performance counter stats for './main_XEON -n 400000 -b 512 -c -d 2 -i 153 -a 2':

          7 cpu-migrations          #    0.000 K/sec
   55579.505157 task-clock                #   15.819 CPUs utilized
  158700956939 cycles                    #    2.855 GHz                     [45.51%]
  104240035859 instructions              #    0.66  insns per cycle
                                         #    1.20  stalled cycles per insn [54.59%]
  124900452227 stalled-cycles-frontend   #   78.70% frontend cycles idle    [54.56%]
   89293775682 stalled-cycles-backend    #   56.27% backend  cycles idle    [54.54%]
     479123129 cache-misses              #   20.988 % of all cache refs     [54.52%]
    2282845726 cache-references          #   41.074 M/sec                   [54.50%]
      26450324 branch-misses             #    0.476 M/sec                   [36.34%]
    5063343989 L1-dcache-misses          #   91.101 M/sec                   [36.40%]
      20517899 L1-icache-load-misses     #    0.369 M/sec                   [36.42%]
        170085 dTLB-load-misses          #    0.003 M/sec                   [36.42%]
         77280 iTLB-load-misses          #    0.001 M/sec                   [36.42%]

  3.513412006 seconds time elapsed
```

Parallel Deriv: 16 threads (numa opt)

```
38.95%  main_XEON  main_XEON          [.] PdfAdd3Prod<5>::GetVal(double*, unsigned int, Data const&, unsigned int) const
25.75%  main_XEON  libMain_XEON.so     [.] PdfGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
14.00%  main_XEON  libMain_XEON.so     [.] PdfBifurGaussian::GetVal(double*, unsigned int, Data const&, unsigned int) const
 8.15%  main_XEON  libMain_XEON.so     [.] PdfArgusBG::GetVal(double*, unsigned int, Data const&, unsigned int) const
 6.62%  main_XEON  libMain_XEON.so     [.] NLL::GetVal(int)
 1.76%  main_XEON  libgomp.so.1.0.0    [.] gomp_team_barrier_wait_end
 1.23%  main_XEON  main_XEON          [.] PdfAdd<2>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.77%  main_XEON  libgomp.so.1.0.0    [.] gomp_barrier_wait_end
 0.56%  main_XEON  main_XEON          [.] PdfPolynomial<1>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.40%  main_XEON  main_XEON          [.] PdfPolynomial<4>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.37%  main_XEON  main_XEON          [.] PdfPolynomial<0>::GetVal(double*, unsigned int, Data const&, unsigned int) const
 0.24%  main_XEON  libMain_XEON.so     [.] NLL::RunEvaluationBlockSplittingStatic()
 0.15%  main_XEON  libgomp.so.1.0.0    [.] omp_get_thread_num
 0.11%  main_XEON  [kernel.kallsyms]   [k] update_cfs_rq_blocked_load
```

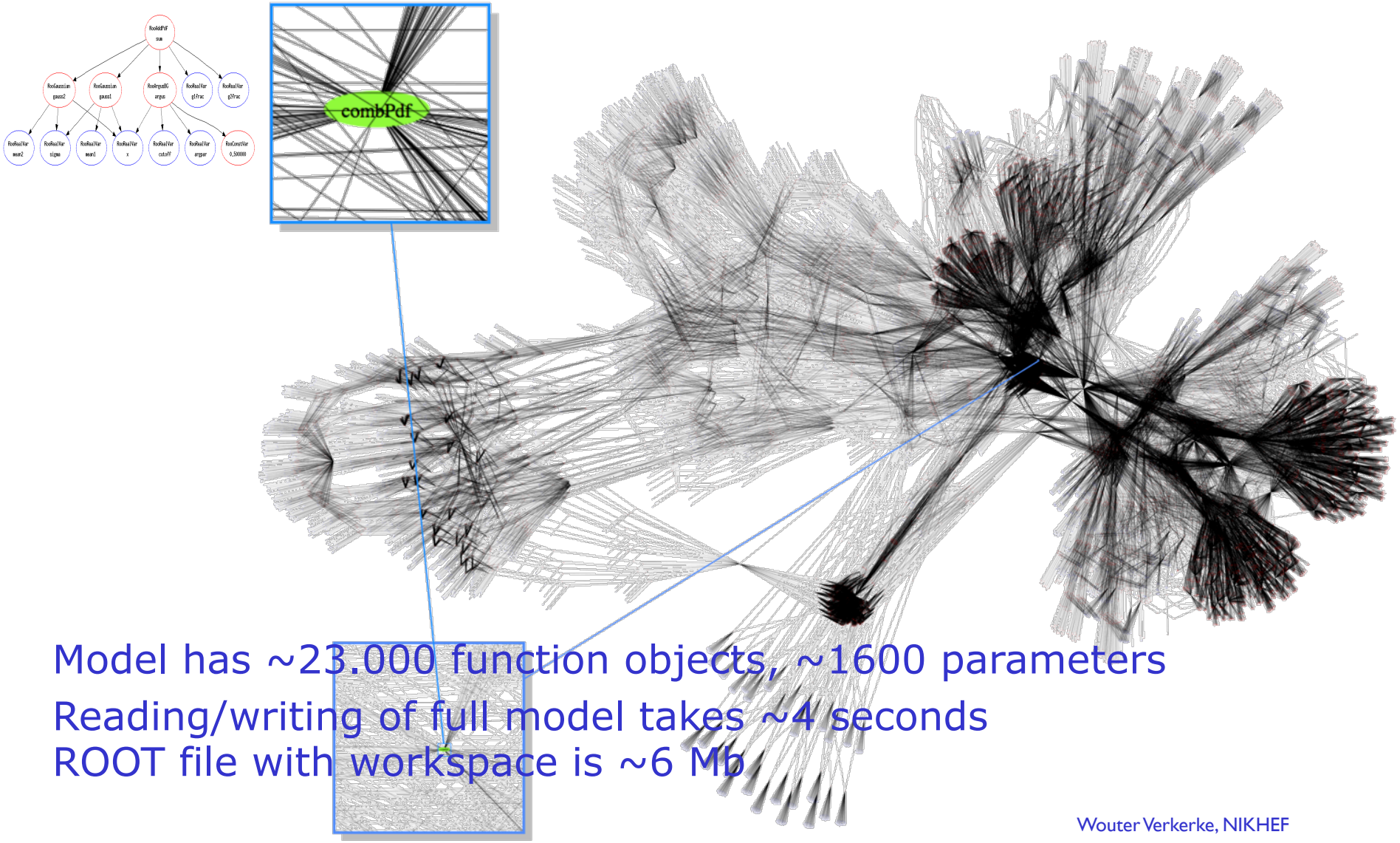
```
Performance counter stats for './main_XEON -n 400000 -b 512 -p -i 152 -a 2':

          7 cpu-migrations            #    0.000 K/sec
  28995.149402 task-clock             #   15.195 CPUs utilized
   83056570588 cycles                 #    2.864 GHz                     [45.48%]
  100513455533 instructions           #    1.21  insns per cycle
                                      #    0.52  stalled cycles per insn [54.59%]
   52639847662 stalled-cycles-frontend #   63.38% frontend cycles idle    [54.56%]
   19455589583 stalled-cycles-backend  #   23.42% backend  cycles idle    [54.54%]
      40120567 cache-misses            #    1.890 % of all cache refs     [54.58%]
    2122969272 cache-references        #   73.218 M/sec                   [54.60%]
      28760339 branch-misses           #    0.992 M/sec                   [36.46%]
    4965233308 L1-dcache-misses        #  171.244 M/sec                   [36.46%]
       6132900 L1-icache-load-misses   #    0.212 M/sec                   [36.44%]
         77762 dTLB-load-misses        #    0.003 M/sec                   [36.35%]
         14125 iTLB-load-misses        #    0.487 K/sec                   [36.30%]

  1.90821806l seconds time elapsed
```

What next

- And this was a "simple problem" 10 year old
  - 100K-1M events
  - Only one channel: 3 data variables
  - A 10^{th} of pdfs and of free parameters
    - Mostly exponents and polynomial
  - All analytical integrals

Today real life is by far more complex!

- Graph of the full ATLAS Higgs combination model



Model has ~23.000 function objects, ~1600 parameters

Reading/writing of full model takes ~4 seconds
ROOT file with workspace is ~6 Mb

New Infrastructure

- Split Data and Algorithms
  - » Pdfs made stateless
  - » Data, variables and caches are encapsulated in an external "State"
  - » The "State" encapsulate caching behavior as well
  - » State percolated through pdfs' calls (with callback)
    - Pdfs can be declared cacheble or to always recompute
  - » Fully compatible with previous schedulers
- Parallel scheduler
  - » Derivatives in parallel
    - One "Reference State" and 2xN "Modified States"
  - » Integrals in parallel (depend only on parameters, not data, not other pdfs)
  - » Followed (as soon as possible) by pdf evaluations event-block by event-block (respecting partitioning)
  - » Caching better be pre-empted as blocking
  - » No async queue used due to the complexity of scheduling

Parallel Deriv: 16 threads (new Infrastr)

```
32.10%  main_XEON  libMain_XEON.so      [.] PdfGaussian::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
17.62%  main_XEON  libMain_XEON.so      [.] PdfBifurGaussian::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
14.03%  main_XEON  main_XEON            [.] PdfAdd<5>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 9.68%  main_XEON  libMain_XEON.so      [.] PdfArgusBG::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 8.23%  main_XEON  main_XEON            [.] PdfProd<3>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 8.10%  main_XEON  libMain_XEON.so      [.] NLL::GetVal(PdfState&)
 2.19%  main_XEON  libgomp.so.1.0.0     [.] gomp_team_barrier_wait_end
 1.71%  main_XEON  main_XEON            [.] PdfAdd<2>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 1.07%  main_XEON  main_XEON            [.] PdfPolynomial<1>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 0.71%  main_XEON  main_XEON            [.] PdfPolynomial<4>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 0.68%  main_XEON  libgomp.so.1.0.0     [.] gomp_barrier_wait_end
 0.62%  main_XEON  libMain_XEON.so      [.] PdfModifiedState::pdfVal(unsigned long, double*, unsigned int, unsigned int) const
 0.51%  main_XEON  libMain_XEON.so      [.] PdfReferenceState::pdfVal(unsigned long, double*, unsigned int, unsigned int) const
 0.45%  main_XEON  main_XEON            [.] PdfPolynomial<0>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 0.36%  main_XEON  libMain_XEON.so      [.] NLL::RunEvaluationBlockSplittingStatic(PdfState const&, ….)
 0.32%  main_XEON  libMain_XEON.so      [.] PdfReferenceState::cachePdf(unsigned long, unsigned int, unsigned int) const
 0.27%  main_XEON  libMain_XEON.so      [.] PdfModifiedState::paramVal(unsigned long) const
```

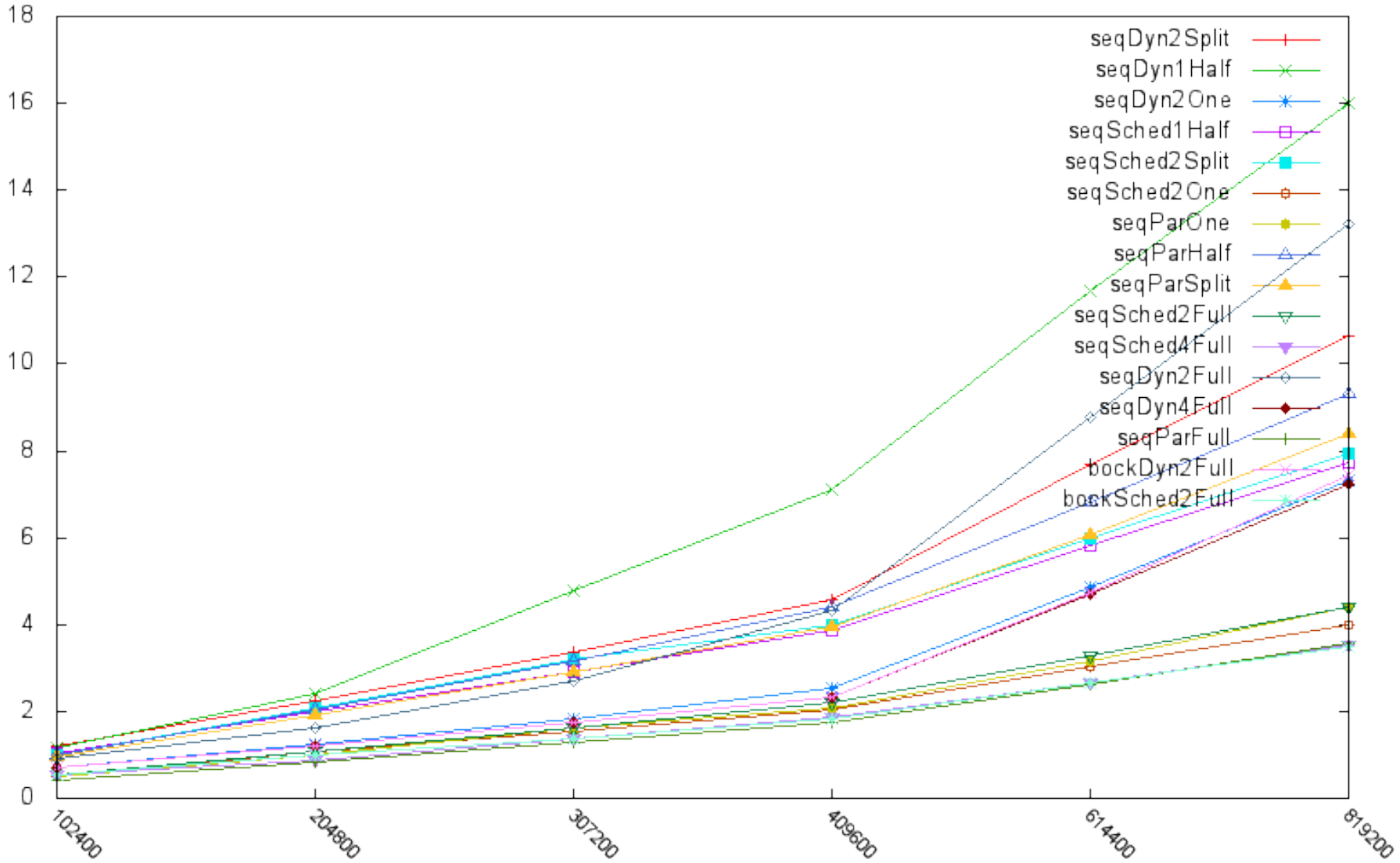
```
Performance counter stats for './main_XEON -n 400000 -b 512 -p -i 152 -a 2':

      23649.913865 task-clock                #   15.568 CPUs utilized
               243 context-switches          #    0.010 K/sec
                 5 cpu-migrations            #    0.000 K/sec
              2177 page-faults               #    0.092 K/sec
       68394000138 cycles                    #    2.892 GHz
       41717806046 stalled-cycles-frontend   #   61.00% frontend cycles idle
       13523788909 stalled-cycles-backend    #   19.77% backend  cycles idle
       89899595357 instructions              #    1.31  insns per cycle
                                             #    0.46  stalled cycles per insn
        3595201013 branches                  #  152.018 M/sec
          24440458 branch-misses             #    0.68% of all branches

      1.519183528 seconds time elapsed
```

Parallel Scheduler

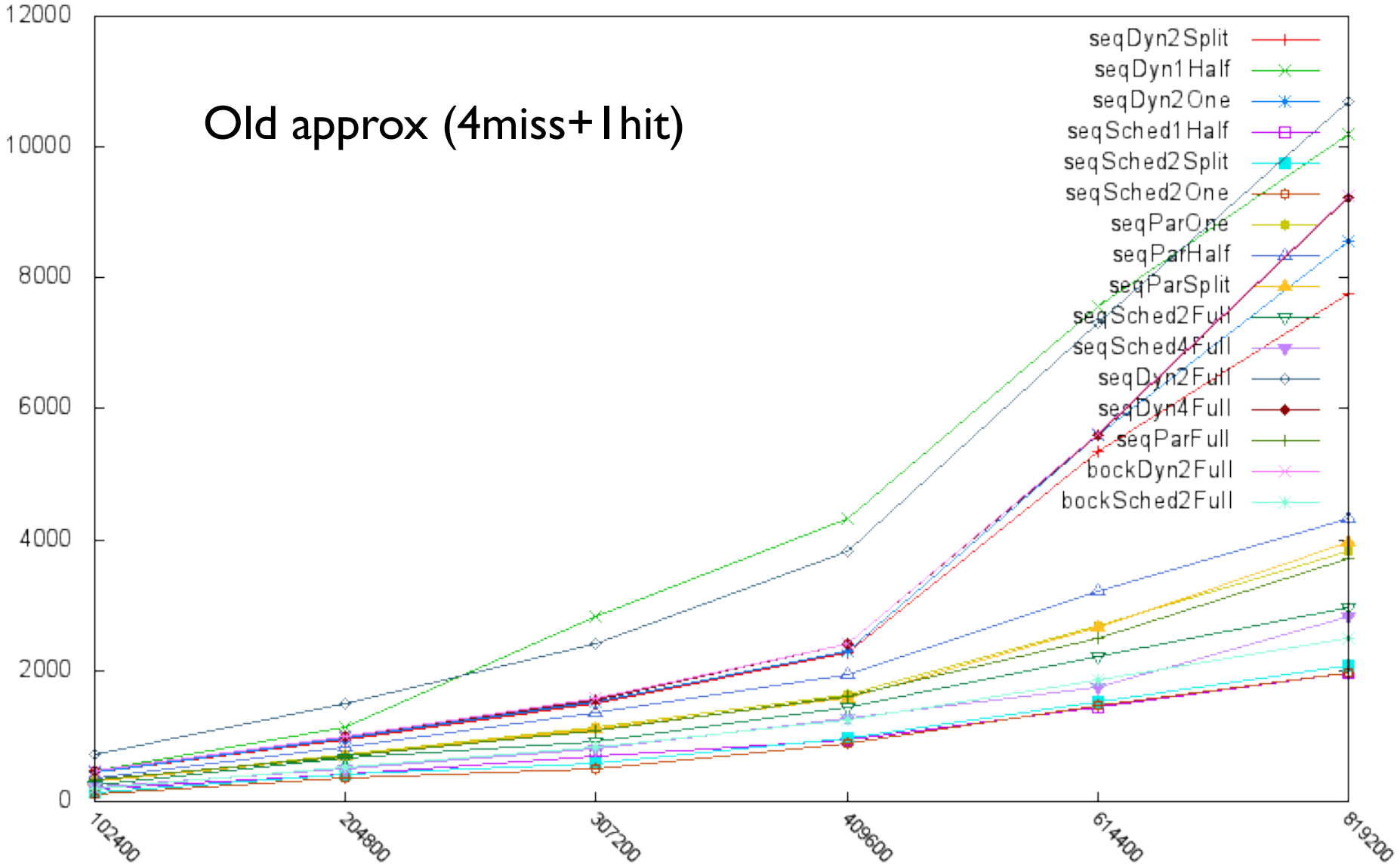
```
32.15%  main_XEON  libMain_XEON.so     [.] PdfGaussian::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
17.91%  main_XEON  libMain_XEON.so     [.] PdfBifurGaussian::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
10.33%  main_XEON  main_XEON           [.] PdfAdd<5>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 9.94%  main_XEON  libMain_XEON.so     [.] PdfArgusBG::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 8.39%  main_XEON  libMain_XEON.so     [.] PdfScheduler::computeChunk(unsigned int, unsigned int)
 6.28%  main_XEON  main_XEON           [.] PdfProd<3>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 4.47%  main_XEON  libgomp.so.1.0.0    [.] gomp_team_barrier_wait_end
 1.79%  main_XEON  main_XEON           [.] PdfAdd<2>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 1.16%  main_XEON  libgomp.so.1.0.0    [.] gomp_barrier_wait_end
 0.93%  main_XEON  libMain_XEON.so     [.] PdfModifiedState::pdfVal(unsigned long, double*, unsigned int, unsigned int) const
 0.83%  main_XEON  main_XEON           [.] PdfPolynomial<1>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 0.73%  main_XEON  libMain_XEON.so     [.] PdfScheduler::doTasks()
 0.73%  main_XEON  libMain_XEON.so     [.] PdfScheduler::chunkResult(unsigned long,TMath::IntLog)
 0.71%  main_XEON  main_XEON           [.] PdfPolynomial<4>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 0.67%  main_XEON  libMain_XEON.so     [.] PdfReferenceState::pdfVal(unsigned long, double*, unsigned int, unsigned int) const
 0.54%  main_XEON  main_XEON           [.] PdfPolynomial<0>::values(PdfState const&, double*, unsigned int, Data const&, unsigned int) const
 0.35%  main_XEON  libMain_XEON.so     [.] NLL::RunEvaluationBlockSplittingStatic(PdfState const&)
 0.35%  main_XEON  libMain_XEON.so     [.] PdfModifiedState::paramVal(unsigned long) const
 0.33%  main_XEON  libMain_XEON.so     [.] PdfReferenceState::cachePdf(unsigned long, unsigned int, unsigned int) const
 0.19%  main_XEON  libMain_XEON.so     [.] PdfModifiedState::invIntegral(unsigned long) const
```

```
Performance counter stats for './main_XEON -n 400000 -b 512 -s -i 152 -a 2':

      23585.102988 task-clock                #   15.570 CPUs utilized
               180 context-switches          #    0.008 K/sec
                 5 cpu-migrations            #    0.000 K/sec
              2605 page-faults               #    0.110 K/sec
       68212554860 cycles                    #    2.892 GHz
       39771855497 stalled-cycles-frontend   #   58.31% frontend cycles idle
       13611786320 stalled-cycles-backend    #   19.95% backend  cycles idle
       91045521921 instructions              #    1.33  insns per cycle
                                             #    0.44  stalled cycles per insn
        3859380141 branches                  #  163.636 M/sec
          34133110 branch-misses             #    0.88% of all branches

      1.514821603 seconds time elapsed
```

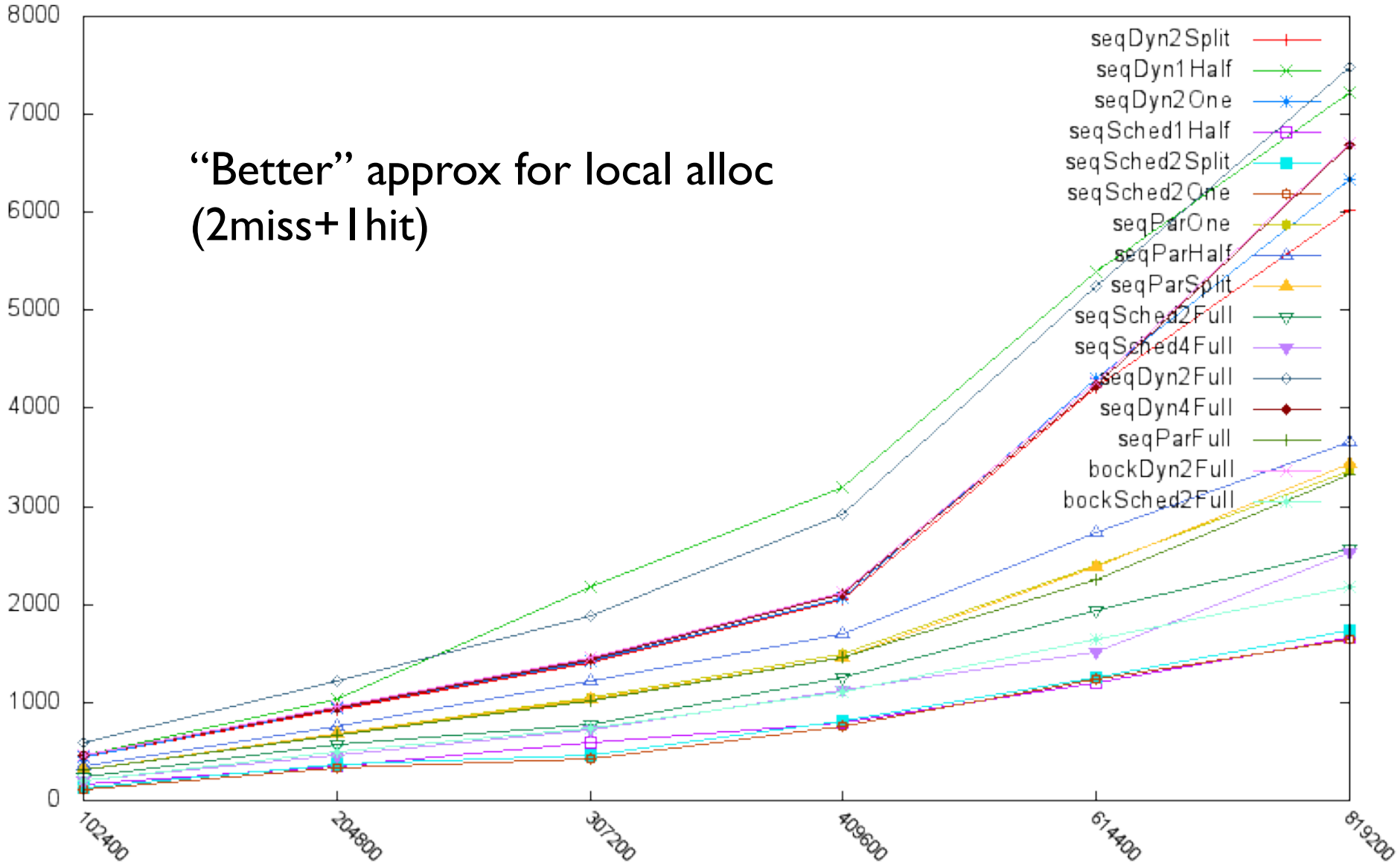
Time new Infrastructure



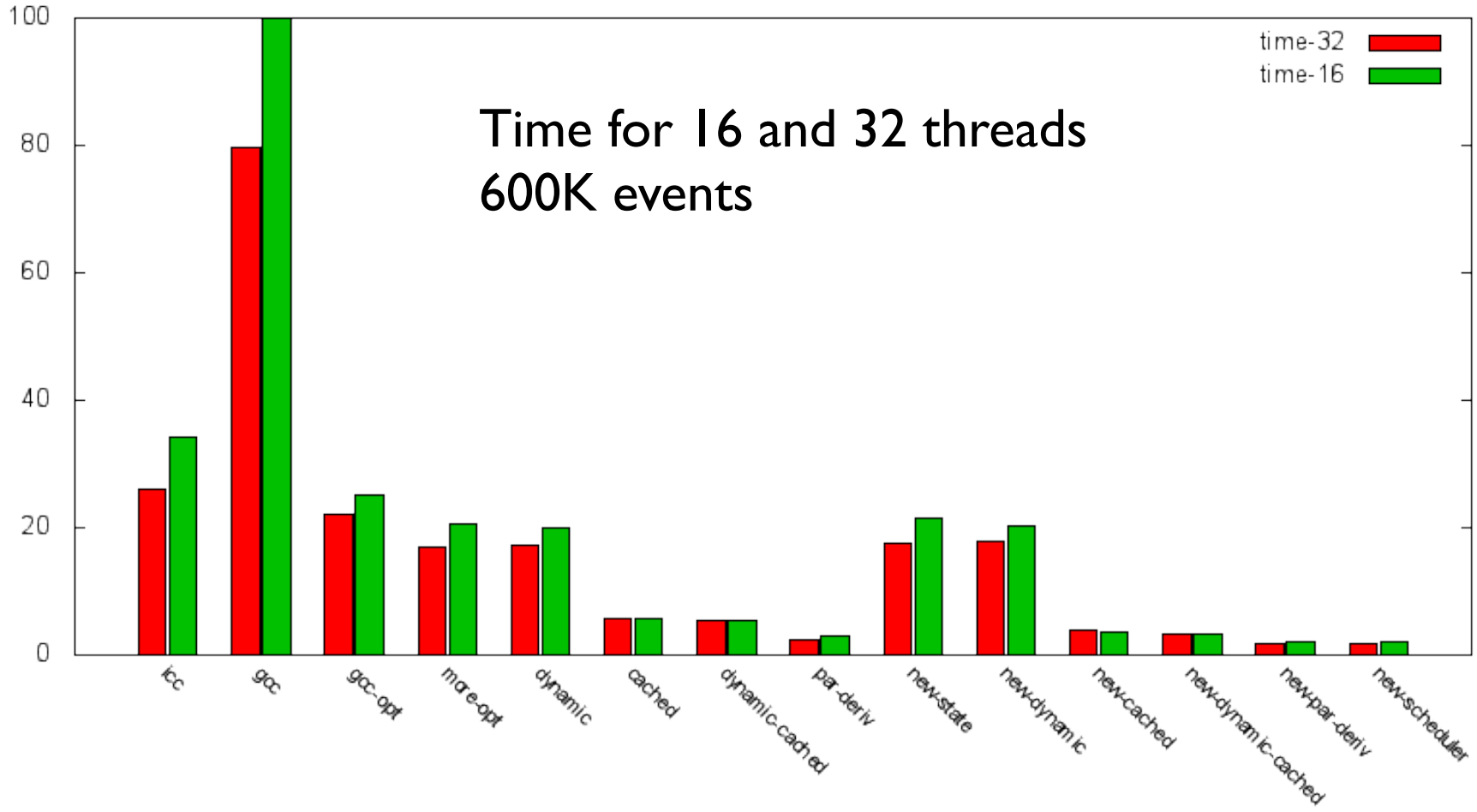
Memory cost (new infr)



Memory cost (new infr)



Summary of results



Summary of results (zoom)