Recent Activities in Automatic Differentiation and beyond

Christian Bischof

Institute for Scientific Computing and
Center for Computing and Communication
RWTH Aachen University
bischof@sc.rwth-aachen.de
www.sc.rwth-aachen.de

Joint work with Martin Bücker, Monika Petera, Arno Rasch, Andre Vehreschild

The Need for Derivatives

- Many numerical methods, sensitivity analysis, design optimization, inverse problems, data assimilation need derivatives (Gradients, Jacobians, Hessians, explicit or as operators).

Local Model based on Taylor Series	$f(x+h) = f(x) + \partial f / \partial x * h + O(h^2)$

- „f" can be a formula, but in most cases it is a computer program written in some programming language, and sometimes of substantial size.

The problem: Fast and accurate computation of derivatives of computer programs with little effort

Automatic Differentiation (AD)
- Theoretical Underpinnings -

Automatic Differentiaton (AD)

Given a computer program, AD generates a new program, which applies the chain rule of differential calculus, e.g.,

$$\frac{d\,f(w,v)}{dx} = \frac{\partial f}{\partial w} * \frac{dw}{dx} + \frac{\partial f}{\partial v} * \frac{dv}{dx}$$

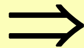
to elementary operations (i.e., $\frac{\partial f}{\partial w}, \frac{\partial f}{\partial v}$ are known).

- **AD does not generate truncation- or cancellation errors.**
- **AD generates a program for the computation of derivative values, not derivative formulae.**

The Forward Mode (FM)

Associate a "gradient" ∇ with every program variable and apply derivative rules for elementary operations.

```
t:=1.0
do i=1 to n step 1
  if (t>0) then
    t := t*x(i);
  endif
endif
```



```
t:=1.0;  ∇t:=0.0;
do i=1 to n step 1
  if (t>0) then
        ∇t:= x(i)*∇t+t*∇x(i);
        t := t*x(i);
  endif
endif
```

The computation of *p* (directional) derivatives requires gradients of length *p* → many vector linear combinations, e.g., ∇x(i) = (0,.,0,1,0,.,0) results in ∇t == *dt/dx(1:n).*

The Reverse Mode (RM)

Associate an "adjoint" α with every program variable and apply the following rules to the "inverted" program:
s = f(v,w) → α_v += ds/dv* α_s; α_w += ds/dw* α_s;

```
tval(0):=1.0; tctr:=0;
do i=1 to n step 1
  if (tval(tctr)>0) then
    jump(i):='true'; tctr=tctr+1;
    tval(tctr):=tval(tctr-1)*x(i);
```

1. Step:
Reversible program
(Single Assignment Form)

2. Step:
Adjoint computation
α_tval(tctr)=1.0,
all other α_*=0.
Upon exit,
α_x(i)==dt/dx(i)

```
do i=n downto 1 step -1
    if (jump(i) == 'true') then
        α_tval(tctr-1)+=x(i)*α_tval(tctr);
        α_x(i)+=tval(tctr-1)*α_tval(tctr);
        α_tval(tctr)=0; tctr=tctr-1;
```

Inherently, AD is a hard problem

Uwe Naumann recently proved that the task of computing derivatives of a program with minimal complexity is an NP-hard problem.

- So forward and reverse modes are just ends of an algorithmic spectrum.
- The challenge for AD tool developers and users is to exploit the associativity of the chain rule taking advantage of
  - program structure, and
  - knowledge about the mathematical underpinnings of a code
- List of available AD tools at **www.autodiff.org**

AD in Neutron Scattering

a little bit of insight may go a long way

AD in Neutron Scattering

- Nonlinear least-squares formulation to determine value of one (!) parameter from 12,237 data sets.
- Theoretical scattering function consists of about 3,000 lines of Fortran code.
- Originally used: NAG E04FDF, quasi-Newton method with divided-difference gradient:
  586 sec., 13 evals of f(.),
  warning about dubious quality of result!
- AD enables use of modified Gauss-Newton Method NAG E04GEF:
  498 sec., 6 evals of (f, ∇f), no warnings.
- Eval. of f takes 38.7 sec., (f, ∇f) takes 69.2 sec.

A Closer Look at g_voigt(...)

- The subroutine „voigt“ is a leaf in the call graph and accounts for almost 40% of the runtime of the code.

subroutine voigt(u,v,x,y)
⇓ Adifor 2.0
subroutine g_voigt(n,u,g_u,ldg_u,v,x,y,g_y,ldg_y)

From parameter list:

- (x,y) input (⇔complex number). (u,v) output.
- v and x are passive (no gradients g_v, g_x)
- u and y are active (gradients g_u, g_y)

ADIFOR‘s dependence analysis tells us that only δu/δy is needed!

Original Function voigt(..)

Let $i = sqrt(-1)$ and x, y, u, v real.
Then for given $z = x + i*y$

subroutine voigt(u,v,x,y)

computes $w = u + i*v$ defined by

$$w(z) = e^{-z^2} \cdot \operatorname{erfc}(-iz), \qquad \operatorname{Im}(z) > 0,$$

where $$\operatorname{erfc}(z) = 1 - \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} \mathrm{d}\, t.$$

This was not apparent from the 53 lines of code, which was rather convoluted (8 goto‘s!). The author told us.

„Handcoded“ Derivative

After some tedious, but elementary mathematics:

$$\nabla u = \left(2(uy + vx) - \frac{2}{\sqrt{\pi}} \right) \nabla y.$$

Thus, derivatives can be computed by executing original function voigt(..) first, followed by

```
dtmp = 2.0d0*(u*y + v*x) - 1.12837916709551d0
do j = 1, n
    g_u(j) = dtmp*g_y(j)
enddo
```

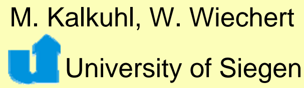
(n = 1 in this case, loop could be omitted)

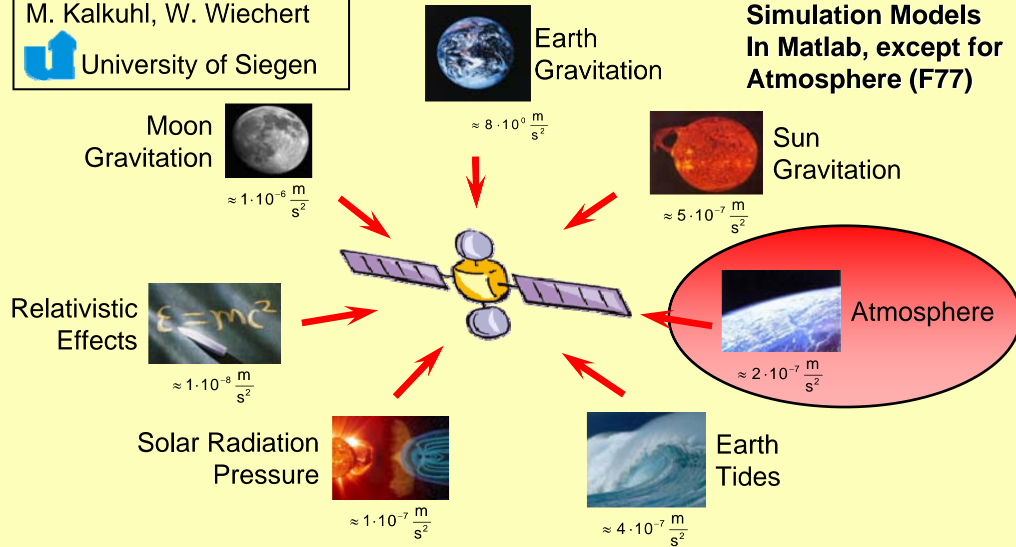
Impact of Strategic Use of Mathematics

- Out of roughly 3000 lines of code, one subroutine comprising 53 lines of code was treated „special", ADIFOR took care of the rest.
- Handcoded derivative for subroutine voigt() is 6.9 times faster due to exploitation of mathematical properties.
- Overall time to compute $(f, \nabla f)$ is reduced to 41.5 seconds from 69.2 seconds.
- Overall execution time of parameter identification is reduced to 333 seconds from 498 seconds.
- Information provided by the AD tool (variable activity) points to possible optimizations that the user can perform.
- Inspection and optimization is possible due to transparency of ADIFOR-generated output.
- An AD-enhanced library routine would have been even better!

A recent AD application: Satellite Simulation

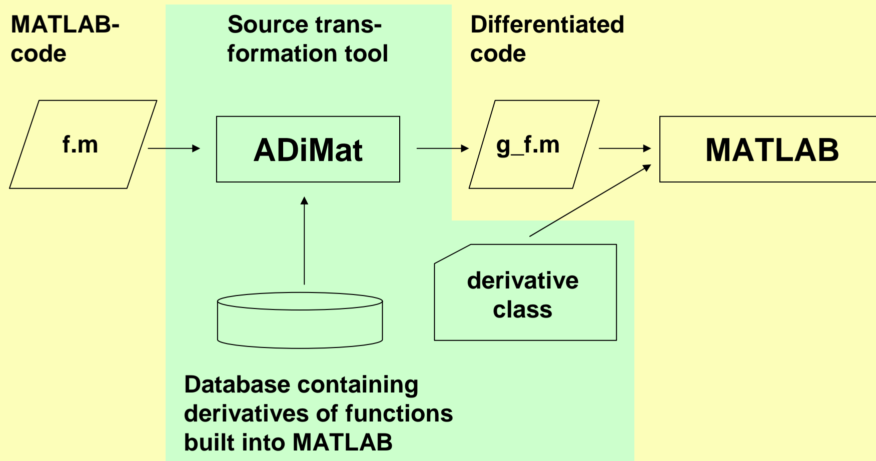
High Precision Satellite Simulation


M. Kalkuhl, W. Wiechert
University of Siegen



Kalman-filter for data fusion of simulated trajectory and GPS-data requires derivatives.

ADiMat: AD for MATLAB



ADiMat Processing Steps

```
function p = fit(x, d, m)
% FIT -- Given x and d, fit() returns p
% such that norm(V*p-d)=min,
% where V=[1, x, x.^2, ... x.^(m-1)].

dim_x = size(x, 1);

V = ones(dim_x, 1);
for count= 1: (m-1)
    V = [V, x.^ count];
end
P = V \ d;
```

Parsing

Activity analysis to determine variables needing derivatives

„Integer“-functions truncate dependencies

Canonicalization

Augmentation

Unparsing

ADiMat Output

```
function [g_p, p]= g_fit(g_x, x, d, m)

   dim_x= size(x, 1);

   g_V= adgradobj(getNDD, [dim_x, 1], 'zeros');
   V= ones(dim_x, 1);
   for count= 1: (m- 1)
      t2= x.^ (count- 1);
      g_t0= count.* t2.* g_x;
      t0= t2.* x;
      g_V= [g_V, g_t0];
      V= [V, t0];
   end

   t1= V\ d;
   g_p= V\ (-g_V* t1);
   p= t1;
```

Allocate zero derivative object of appropriate size for V since g_V appears first on right-hand side of an assignment

g_t0 and g_p are allocated automatically

Planned:
- Agressive optimization of derivative operators.
- Current use of cell arrays for derivative objects works, but is expensive.

Dealing with Matlab „Features“

```
function [z,y,optout]=foo(x,optin)

if nargin>1
   y= x*optin;
else
   y= x*5.0;
end
z= sqrt(x);
if nargout>2
   optout= y^2;
end
```

Want to compute dy/dx.
Leaving control flow governed by „nargin“ unchanged leads to error as in AD code alcode always nargin > 2.

```
function [z,g_y,y,optout]=g_foo(g_x,x,optin)

narginmapper=[ 0, 1, 2];
nargoutmapper=[ 1, 0, 2, 3];

if narginmapper(nargin)>1
   g_y= g_x*optin;
   y= x*optin;
else
   g_y= g_x*5.0;
   y= x*5.0;
end
z= sqrt(x);
if nargoutmapper(nargout)>2
   optout= y^2;
end
```

Similar solution for varargin. Inherently requires interprocedural dependence analysis.

Solution: For each occurrence of nargin and nargout, use indirect addressing to map a certain number of actual arguments in the differentiated code to the corresponding number in the original code.

Atmospheric Reference Models

- In Earth's atmosphere: chemical, thermodynamic and fluid dynamic effects
- Predict temperature and concentration profiles of species (N_2, O, O_2, H and He)
- Based on theoretical considerations and satellite drag data
  - MSIS (Mass Spectrometer Incoherent Scatter) suite of models developed at NASA's Goddard Space Flight Center
  - Based on insitu data from various rocket probes and satellites (OGO 6, San Marco 3, AEROS-A, …) and scatter radars (Jicamara, Arecibo, …)

MSIS-86 (sheet of air above 120km)

- 800 lines of Fortran 77
- Input: year, day of year, Universal Time, altitude (a), geodetic latitude (δ), longitude (λ), local apparent solar time (τ), solar F10.7 flux, magnetic Ap index.
- Output: densities of Ar, N, N_2, O, O_2, H, He, total mass density neutral temperature and exospheric temperature (T).

$$[c, T, \ldots] = \text{msis}(a, \delta, \lambda, \tau, \ldots)$$

- Used ADIFOR 2.0 (Automatic DIfferentiation of FORtran) developed at Rice and Argonne
- AD code for 4 derivatives generated by ADIFOR
  - storage increases by a factor of 2.7
  - runtime increases by a factor of 4.3 on Sun UltraSparc IV

AD Mexfunction Generator (AMG)

```
[g_D,D,g_T,T] = g_msis_f(iday, ut, g_alt, alt, g_xlat, xlat,
                    g_xlong, xlong, xlst, f107a, f107, ap, mass)
-size(D)={1,8}
-size(g_D)={numel(g_alt),8}
-size(T)={1,2}
-size(g_T)={numel(g_alt),2}
--header{
extern void g_msis_(int*, int*, double*, double*, double*, int*,
                    double*, double*, int*, double*, double*, int*,
                    double*, double*, double*, double*, int*,
                    double*, double*, int*, double*, double*, int*);
}
-kernel {
   int tmass = (int) mass[0];
   int tiday = (int) iday[0];
   int g_p = numel(g_alt);
   g_msis_(&g_p, &tiday, ut, alt, g_alt, xlat, g_xlat, xlong, g_xlong,
           xlst, f107a, f107, ap, &tmass, D, g_D, T, g_T);
}
```

AMG generated 130 lines of code from the 19 lines interface specification

AMG is a macro processor written in C, performing
• generation of MEX interfaces, including
• conversion of ADiMat's derivative objects to derivative formats used in ADIC and ADIFOR, and
• checks for the correct number of actual arguments and results when mex-Function is called.

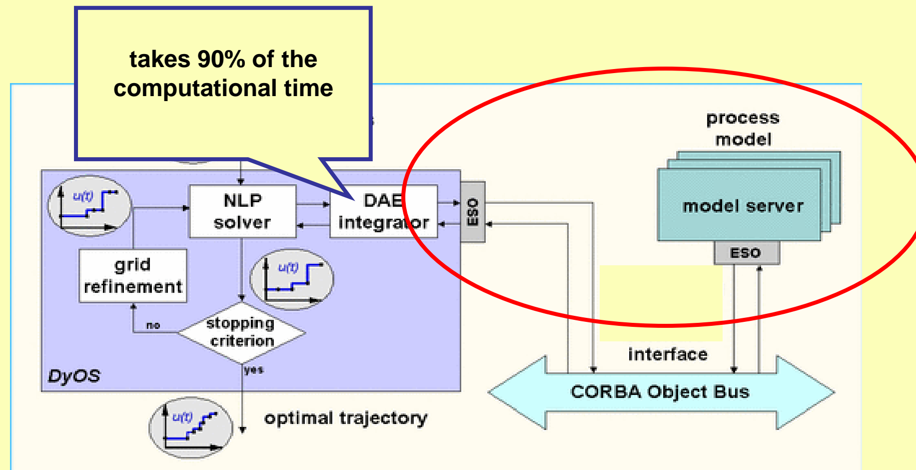
Conclusions (1)

- AD is very much about convenience, not just performance.
- In most instances, users will view AD as an ingredient to ensure „safer numerics“.
- Simulation of most phenomena typically involves several code modules, often written in different languages.
- To facilitate AD acceptance in a broader context:
  - AD tools must provide coverage for complete language specifications (even for unusual “features”)
  - Need to provide mechanisms to facilitate AD in a multilingual context.
- Unfortunately, no research funding for „convenience“.

AdiCape

AD of an XML-based „little language“ tailored for process control applications

DyOS- Dynamic Optimization Software



DyOS developed at Inst. for Process Control (Prof. Wolfgang Marquardt) at Aachen

CapeML

CapeML – A Model Exchange Language for Chemical Process Modeling (Lars von Wedel, RWTH-LPT, 2002).

• CapeML provides an **interoperability** between modeling tools (e.g. Modelica, gPROMS)

• Equation-oriented approach
  • does not make any assumptions about how to solve a model or
  • what quantities are considered known and unknown.

• XML Language definition

A CapeML fragment

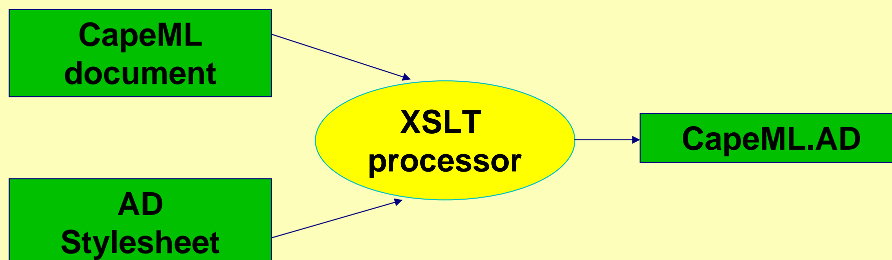
```
<BalancedEquation myID="V-0">
 <Expression>
  <Term>
   <Factor>
    <FunctionCall fcn.name="sin">
     <Expression>
      <Term>
       <Factor>
        <VariableOccurrence
              definition="V-car-alpha"/>
       </Factor>
      </Term>
     </Expression>
    </FunctionCall>
   </Factor>
  </Term>
 </Expression>
```

$$\sin(alpha) = beta$$

```
<Expression>
 <Term>
  <Factor>
   <VariableOccurrence
           definition="V-car-beta"/>
  </Factor>
 </Term>
 </Expression>
</BalancedEquation>
```

XSLT - Language Transformation

transforms from any XML-based markup language into another markup language



XSLT stylesheets contain functions and templates (patterns) and are interpreted by XSLT processor.
XSLT 2.0 offers significantly more functionality than XSLT 1.0

Auto Example in modeling language

```
model car
  Real velo;
  Real dist;
  Real time;
  Real accel; //=2.0;
  Real alpha; //=0.0025;
equation
  der(dist) = velo;
  der(time) = 1.0;
  der(velo) = 4/3.1415926 * arctan(accel) - alpha*pow(velo,2);
  accel:=2.0;
  alpha:=0.0025;
end car;
```

3 Equations
2 Assignments

In CapeML considered as 5 Equations

Auto Example- Equations (1st Deriv)

```
equation

  der(g_dist[1:q]) = g_velo[1:q];

  der(g_time[1:q]) = 0*Ident(q);

  der(g_velo[1:q]) = arctan(accel)*1/(3.14159)^2*(0*Ident(q))
                     + 4/3.14159*(1/(1 + accel^2)*g_accel[1:q])
                     - velo^2*g_alpha[1:q]
                     + alpha*2*velo^(2 - 1)*g_velo[1:q];

  g_accel[1:q] = 0*Ident(q);

  g_alpha[1:q] = 0*Ident(q);
```

Vector of ones of size q

CapeML extension required

Auto Example – Equations (2nd deriv)

```
equation
  der( h_dist[1:qq]) = h_velo[1:qq];
  der( h_time[1:qq]) = 0*Ident(qq);

  der( h_velo[1:3]) =
arctan(accel)*1/3.14159^2*0*Ident(qq) +(0-2*accel/((1 + accel^2)^2))*h_accel[1:qq])
+ 4/3.14159*((0-2*accel/((1 + accel^2)^2))*sym_outer_product(g_accel[1:q])
+ sym_plus_outer_product(1/3.14159^2*0*g_Ident[1:q], 1/(1 + accel^2)*g_accel[1:q])
- (velo^2*h_alpha[1:qq] + alpha*(2*velo^(2-1)*h_velo[1:qq]
+ 2*(2-1)*velo^(2-2)*sym_outer_product(g_velo[1:q]))
+ sym_plus_outer_product(g_alpha[1:q],2*velo^(2-1)*g_velo[1:q]));
  h_accel[1:qq] = 0*Ident(qq);
  h_alpha[1:qq] = 0*Ident(qq);
end car;
```

$$\nabla^2 z = \frac{\partial z}{\partial x}\nabla^2 x + \frac{\partial z}{\partial y}\nabla^2 y + \frac{\partial^2 z}{\partial x^2}(\nabla x \cdot \nabla x^T) + \frac{\partial^2 z}{\partial y^2}(\nabla y \cdot \nabla y^T) + \frac{\partial^2 z}{\partial x \partial y}(\nabla x \cdot \nabla y^T + \nabla y \cdot \nabla x^T)$$

ADiCape

- **ADiCape** - source transformation tool for automatic differentiation of CapeML programs using XSLT 2.0 transformations.
- First order directional derivatives
- Sparse seeding given Jacobian sparsity pattern (Curtiss-Powell-Reid coloring)
  Many processes are concatenations of components with feedback loops, e.g. Destillation Column – 206x410 Jacobian can be computed w/ 13 directional derivs.
- (Compressed) Symmetric projections of the Hessian matrix

Conclusions (2)

- XML provides a convenient mechanism for defining domain-specific languages, including a rich set of tools.
- Domain-specific languages are usually at a high level of abstraction, thus providing great potential for exploiting AD at a higher mathematical level.
- Typical programs are short, but derivatives are nontrivial (in part. 2nd order), yet they are numerically potentially very attractive.
- Experience with CapeML has also shown that AD typically requires language extensions to account for needed data structures

- If the possibility of AD was incorporated into language design from the start, the task would be much easier.

Exploiting Parallelism in AD

Parallel Hessian Computations

Idea:

- **Entries of the Hessian can be computed independently (Hessian-update is parallel loop)**
- **Loop-parallelization with OpenMP**
- **Drawback: synchronization before each Hessian update**
- **Solution: redundant execution of the whole code where the work on Hessians is explicitly distributed**

Parallel Hessians - Implementation

Illustration: first- and second-order derivatives for c=a*b

```
common /adomp/ LB, UB, my_i, my_j
c$omp threadprivate (/adomp/)
i=my_i
j=my_j
do k=LB,UB
  h_c(k)=b*h_a(k)+a*h_b(k)+g_a(i)*g_b(j)+g_a(j)*g_b(i)
  if (i.eq.j) then
    i=i+1
    j=1
  else
    j=j+1
enddo
do k=1,n
  g_c(k)=b*g_a(k)+a*g_b(k)
enddo
```

Shared variables: `h_a, h_b, h_c` (others are private)

Parallel execution with different values for `LB, UB, my_i, my_j`

$$\nabla^2 c = b \cdot \nabla^2 a + a \cdot \nabla^2 b + \nabla a \cdot \nabla b^T + \nabla b \cdot \nabla a^T$$

Redundant computation of the gradient

$$\nabla c = b \cdot \nabla a + a \cdot \nabla b$$

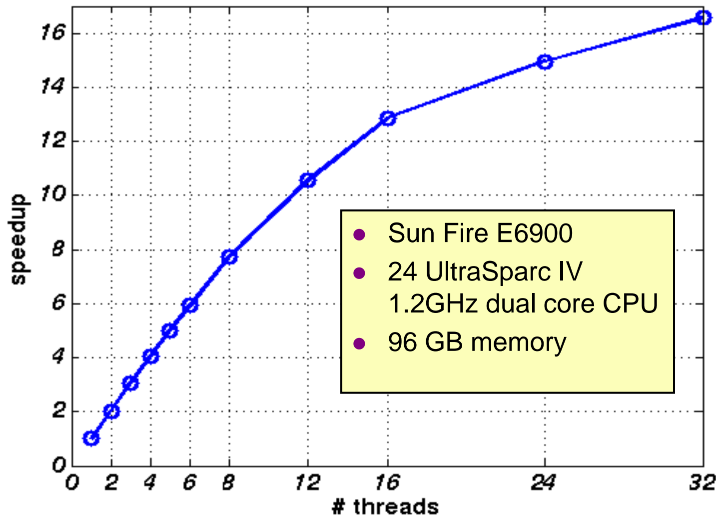
Parallel Hessians - Implementation

Main program:

```
common /adomp/ LB, UB, my_i, my_j
c$omp threadprivate (/adomp/)
c$omp parallel shared(h_x,...) firstprivate(x,g_x...)
  initialize(LB,UB,my_i,my_j)
  call h_msis(h_x,g_x,x,...)
c$omp end parallel
```

- **No synchronization (except for the main program)**
- **easy to implement:**
  - **make all Hessian variables shared using static allocation, (e.g., use the `save` attribute))**
  - **all other variables are private. In OpenMP, variables occuring outside the lexical scope of a parallel region are private by default. For static variables , we use `omp threadprivate`)**

Parallel Hessian for MSIS-86



300 independent variables

Performance limited by backplane memory bandwidth

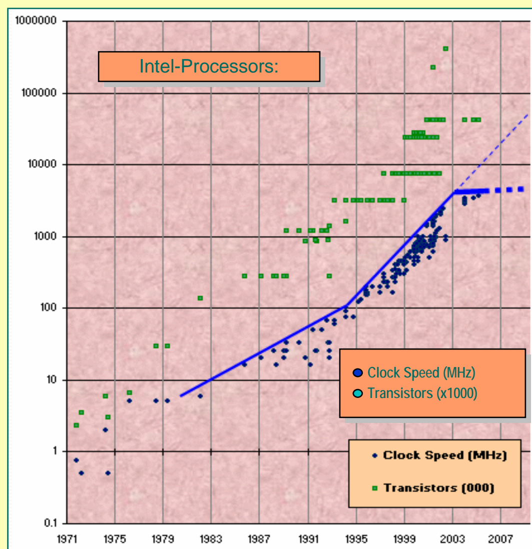
The Ubiquituous Parallel Computer

joint work w/
Dieter an Mey, Samuel Sarholz,
Christian Terboven

Center for Computing and Communication
RWTH Aachen University

(all performance numbers should be viewed as preliminary)

The Impact of Moore's Law



The number of transistors on a chip is still doubling every 18 months

… but the clock speed is no longer growing that fast.

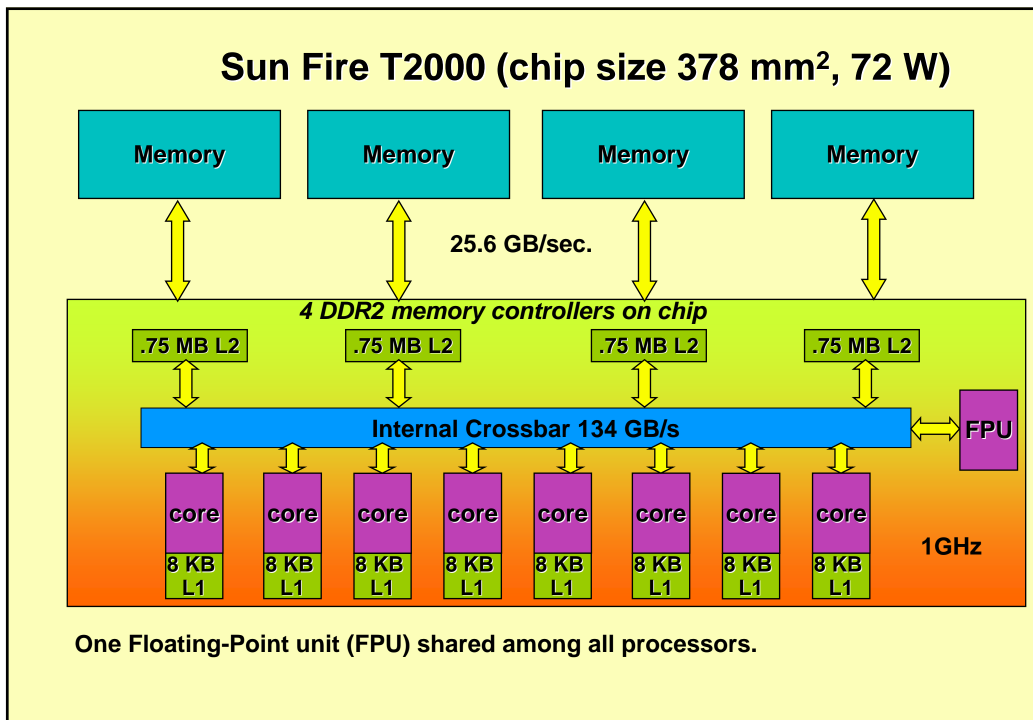
Instead we'll see many more cores per chip!

Source: Herb Sutter
www.gotw.ca/publications/concurrency-ddj.htm

Design Considerations

- Power consumption (and heat generated)
  - Scales linearly with #transistors but quadratically with clock rate
- There is no Moore's Law for memory latency
  - Memory latency halves only every six years.
- **Conclusion for commodity chips: Instead of striving for faster clock speed, tightly integrate multiple instances of slower processor cores on a chip.**
  - IBM Blue Gene is most radical example – 1024 CPUs/rack.

Sun Fire T2000 (chip size 378 mm², 72 W)



One Floating-Point unit (FPU) shared among all processors.

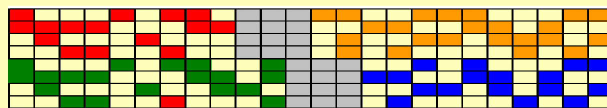
Terminology

- **Chip(-level) multiprocessing (CMP)** :
  multiple processor "cores" are included in the same integrated circuit, executing independent instruction streams.

- **Chip(-level) multithreading (CMT)**
  Multiple threads are executed within one processor core at the same
  time. Only one is active at a given time (time-slicing).

Chip Level Parallelism

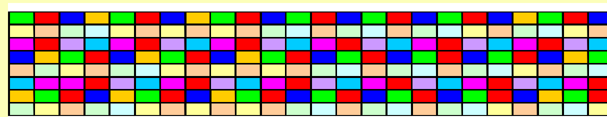
Chip-Level Multiprocessing



↔ = 0.66 ns

UltraSPARC IV+, CMP
superscalar, dual core
2 x 4 sparc v9 instr/cycle
1 active thread per core

Chip-Level Multithreading



↔ = 1.0 ns

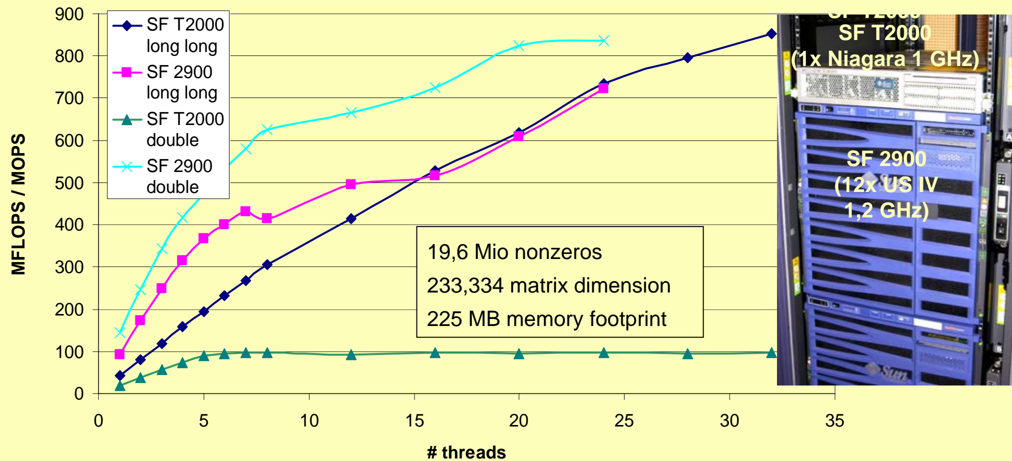
UltraSPARC T1, CMP+CMT
Single issue, 8 cores
8 x 1 sparc v9 instr/cycle
4 active threads per core
context switch comes for free

context switch

time

Sparse Matrix Vector Multiplication

What, if the Niagara could really compute with floating point numbers …



Concluding Remarks

- **Automatic Differentiation (AD) Tools: Efficient derivative programs with very little human effort.**
  Challenge: Make better use of compile-, runtime-and user-provided information, synergies w/ other tool efforts
- **AD + Brains: Methodologies for exploiting high-level user knowledge about program structure and algorithmic underpinnings.**
  Challenge: Harnesses for common paradigma, AD-enabled component frameworks
- **Program-based sensitivity analysis with AD is getting more important**
  - **Optimization, data assimilation, experimental design**
  - **Robustness and uncertainty analysis**
- **AD-enhanced codes can generate parallelism, parallel hardware is here anyway! Those who use it will win!**