Parallel Computation of Greeks Using ADOL-C

Andrea Walther

Institute of Scientific Computing, TU Dresden

7th Euro AD Workshop, Oxford, Nov 24, 2008
Outline

1. ADOL-C 2.0
2. Parallelization of derivative calculation
3. Parallel computation of greeks
4. Conclusion and outlook
5. Advertisement :-(
Automatic differentiation by overloading in C++

ADOL-C 2.0 new version coming soon!!

- reorganization of taping
tape dependent information kept in separate structure

- different differentiation contexts ⇒
  - documented external function facility
  - documented fixpoint iteration facility
  - documented checkpointing facility based on revolve

- documented parallelization of derivative calculation

- coupled with ColPack for exploitation of sparsity

- probably available at COIN-OR
Parallelization of ADOL-C

Implementation based on operator-overloading:

- directives / functions for parallelization ignored
- loops enrolled $\Rightarrow$ complete loss of information about structure

$\Rightarrow$ no automated parallelization of loops!
Parallelization of ADOL-C

Implementation based on operator-overloading:

- directives / functions for parallelization ignored
- loops enrolled $\Rightarrow$ complete loss of information about structure

$\Rightarrow$ no automated parallelization of loops!

Solution so far:

- encapsulation of parallel loops in separate differentiation context
- available for OpenMP parallelization
Parallelization of derivative calculation

Approach (reverse mode)

- function
- derivative

Two nested contexts:
- Serial part (blue, red), “external” differentiated part (green)
- External differentiated part: user-assisted differentiated for loop body
Simulation of 1D plasma

- Simulation of two plasma periods
- $N$ Gaussian wave functions yield $N$ equations for $T$ time steps
- Physical properties, design, . . .
- Differentiation with ADOL-C
Simulation of 1D plasma

- Simulation of two plasma periods
- $N$ Gaußian wave functions yield $N$ equations for $T$ time steps
- physical properties, design, . . .
- differentiation with ADOL-C

programm structure:

For each time step
  For each wave function
    . . .
Simulation of 1D plasma

- Simulation of two plasma periods
- \( N \) Gaußian wave functions yield \( N \) equations for \( T \) time steps
- physical properties, design, . . .
- differentiation with ADOL-C

program structure:

For each time step
  For each wave function
    . . .

⇒ external differentiated functions, checkpointing, parallelization
   extensive interaction of user and developer!!
Simulation of 1D plasma: Speedup

SGI Altix 4700, TU Dresden, Intel Itanium II Montecito @ 1.6 GHz, 4 GB RAM (Bischof, Gürtler, Kowarz, Walther ’08)
Computation of Greeks

For given \( z_i, i = 1, \ldots, \text{npaths}, \) random variables, maturities, swap rates, LIBOR interval \( \delta, \) the LIBOR rates are defined by

\[
L_{i}^{n+1} = L_{i}^{n} \exp \left( \left( \sigma_{i-n-1} S_{i} - \frac{1}{2} \sigma_{i-n-1}^2 \right) \delta + \sigma_{i-n-1} z_n \sqrt{\delta} \right)
\]

\[
S_{i}^{n} = \sum_{j=n+1}^{i} \frac{\sigma_{j-n-1} \delta L_{j}^{n}}{1 + \delta L_{j}^{n}}
\]
Computation of Greeks

For given \( z_i, i = 1, \ldots, npaths \), random variables, maturities, swap rates, LIBOR interval \( \delta \), the LIBOR rates are defined by

\[
L_i^{n+1} = L_i^n \exp \left( \left( \sigma_{i-n-1} S_i - \frac{1}{2} \sigma_{i-n-1}^2 \right) \delta + \sigma_{i-n-1} z_n \sqrt{\delta} \right)
\]

\[
S_i^n = \sum_{j=n+1}^{i} \frac{\sigma_{j-n-1} \delta L_j^n}{1 + \delta L_j^n}
\]

and the payoff by

\[
P = \left( \prod_{i=0}^{Nmat-1} \frac{1}{1 + \delta L_i} \right) \left( \sum_{n=1}^{Nopt} 100(1 - B_{mat_n} - swap_n S_{mat_n}) + \right)
\]

\[
S_m = \sum_{i=1}^{m} \delta B_i \quad B_m = \prod_{i=1}^{m} \frac{1}{1 + \delta L_{Nmat+i-1}}
\]

Compute derivatives of averaged \( P \) with respect to \( L_i \) \quad (M. Giles 2007)
Serial computation of greeks:
Function evaluation

```c
for(l=0; l<npath; l++)
{
    for(k=0; k<N; k++)  L[k] = 0.05;

    path_calc(N, Nmat, delta, L, lambda, z[l]);
    portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, v[l]);
    // v = payoff
}

vtot = 0;
for (l=0; l<npath; l++)
    vtot += v[l];
vtot = vtot/npath;
```
Serial Computation of Greeks: Application of ADOL-C

trace_on(tag);
    for(k=0; k<N; k++) La[k] <<= 0.05;
    for(k=0; k<Nmat; k++) za[k] <<= z[0][k];

path_calc(N, Nmat, delta, La, lambda, za);
portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, va);

va >> = v[0];
trace_off();
Serial Computation of Greeks: Application of ADOL-C

trace_on(tag);
for(k=0; k<N; k++) La[k] <= 0.05;
for(k=0; k<Nmat; k++) za[k] <= z[0][k];

path_calc(N, Nmat, delta, La, lambda, za);
portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, va);

va >> = v[0];
trace_off();

for(l=0; l<npath; l++)
  gradient(tag,N+Nmat,xp[l],grad[l]); /* xp[l] = (Lp[l], z[l]) */
  /* N+Nmat components! */
Serial Computation of Greeks: Application of ADOL-C

trace_on(tag);
    for(k=0; k<N; k++) La[k] <<= 0.05;
    for(k=0; k<Nmat; k++) za[k] <<= z[0][k];

    path_calc(N, Nmat, delta, La, lambda, za);
    portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, va);

    va >> = v[0];
trace_off();

for(l=0; l<npath; l++)
    gradient(tag,N+Nmat,xp[l],grad[l]); /* xp[l] = (Lp[l], z[l]) */
        /* N+Nmat components! */

for (l=0; l<N; l++)
    { gradtot[l] = 0; for (k=0; k<npath; k++) gradtot[l] += grad[k][l]; }
Serial Computation of Greeks: Runtimes

The graph shows the runtime in seconds for function evaluation using different versions of ADOL-C. The x-axis represents the number of paths (npath), and the y-axis represents the runtime in seconds. The graph compares two versions:
- ADOL-C version 1.10.2 (red line)
- ADOL-C version 2.0 (green line)

The runtime increases linearly with the number of paths for both versions.
OpenMP-based computation of greeks: Function

```c
int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel
{ int index = omp_get_thread_num();
  int l,k; /* private variables */
  double *L = new double[N];
  for(l=index*slot; l<(index+1)*slot; l++)
    { for(k=0; k<N; k++) L[k] = 0.05;
      path_calc(N, Nmat, delta, L, lambda, z[l]);
      portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, v[l]);
    }
  #pragma omp barrier
  if (index==0)
    { vtot = 0;
      for (l=0; l<npath; l++)
        { vtot += v[l];
          vtot = vtot/npath; }
    }
```
OpenMP-based computation of greeks: Function

```c
int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel
{ int index = omp_get_thread_num();
  int l,k; /* private variables */
  double *L = new double[N];

  for(l=index*slot; l<(index+1)*slot; l++)
    for(k=0; k<N; k++) L[k] = 0.05;
  path_calc(N, Nmat, delta, L, lambda, z[l]);
  portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, v[l]);
}
```
Parallel computation of greeks

OpenMP-based computation of greeks: Function

```c
int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel
{ int index = omp_get_thread_num();
  int l,k; /* private variables */
  double *L = new double[N];

  for(l=index*slot; l<(index+1)*slot; l++)
    { for(k=0; k<N; k++) L[k] = 0.05;
      path_calc(N, Nmat, delta, L, lambda, z[l]);
      portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, v[l]); } }

#pragma omp barrier
if (index==0)
{ vtot = 0;
  for (l=0; l<npath; l++)
    vtot += v[l];
  vtot = vtot/npath; } }
```
OpenMP-based Computation of Greeks: ADOL-C

int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel firstprivate(ADOLC_OpenMP_Handler)
    { int index = omp_get_thread_num();
    int l,k; /* private variables */
    double *L = new double[N];
    adouble *La = new adouble[N]; adouble va;
    trace_on(index);
    for(k=0; k < N; k++) La[k] = 0.05;
    for(k=0; k < Nmat; k++) za[k] = z[0][k];
    path_calc(N, Nmat, delta, La, lambda, za);
    portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, va);
    va >>= v[index];
    trace_off();
    for(l=index*slot; l < (index+1)*slot; l++)
        gradient(index,N+Nmat,xp[l],grad[l]);
int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel firstprivate(ADOLC_OpenMP_Handler)
 { int index = omp_get_thread_num();
   int l,k; /* private variables */
   double *L = new double[N];
   adouble *La = new adouble[N]; adouble va;
   trace_on(index);
   for(k=0; k<N; k++) La[k] <<= 0.05;
   for(k=0; k<Nmat; k++) za[k] <<= z[0][k];
   path_calc(N, Nmat, delta, La, lambda, za);
   portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, va);
   va >>= v[index];
   trace_off();
}
OpenMP-based Computation of Greeks: ADOL-C

int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel firstprivate(ADOLC_OpenMP_Handler)
{ int index = omp_get_thread_num();
  int l,k; /* private variables */
  double *L = new double[N];
  adouble *La = new adouble[N]; adouble va;
  trace_on(index);
  for(k=0; k<N; k++) La[k] <<= 0.05;
  for(k=0; k<Nmat; k++) za[k] <<= z[0][k];
  path_calc(N, Nmat, delta, La, lambda, za);
  portfolio(N, Nmat, delta, Nopt, maturities, swaprates, L, va);
  va >> = v[index];
  trace_off();
  for(l=index*slot; l<(index+1)*slot; l++)
    gradient(index,N+Nmat,xp[l],grad[l]);
OpenMP-based Computation of Greeks: ADOL-C

```c
int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel firstprivate(ADOLC_OpenMP_Handler)
  { int index = omp_get_thread_num();
    int l,k; /* private variables */
    double *L = new double[N];
    adouble *La = new adouble[N]; adouble va;
    trace_on(index);
    ...
    trace_off();
    for(l=index*slot; l<(index+1)*slot; l++)
      gradient(index,N+Nmat,xp[l],grad[l]);
#pragma omp barrier
for (l=0; l<N; l++)
  { gradtot[l] = 0;
    for (k=0; k<npath; k++)
      gradtot[l] += grad[k][l]; }}
```

Andrea Walther
Parallel Computation of Greeks
7th Euro AD Workshop
Parallel computation of greeks

OpenMP-based Computation of Greeks: ADOL-C

int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel firstprivate(ADOLC_OpenMP_Handler)
  { int index = omp_get_thread_num();
    int l,k; /* private variables */
    double *L = new double[N];
    adouble *La = new adouble[N]; adouble va;
    trace_on(index);
      ...
    trace_off();
    for(l=index*slot; l<(index+1)*slot; l++)
      gradient(index,N+Nmat,xp[l],grad[l]);
  }
#pragma omp barrier
for (l=0; l<N; l++)
  { gradtot[l] = 0;
    for (k=0; k<npath; k++) gradtot[l] += grad[k][l]; }
int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel firstprivate(ADOLC_OpenMP_Handler)
  { int index = omp_get_thread_num();
    int l,k; /* private variables */
    double *L = new double[N];
    adouble *La = new adouble[N]; adouble va;
    trace_on(index);
    ...
    trace_off();
    for(l=index*slot; l<(index+1)*slot; l++)
      gradient(index,N+Nmat,xp[l],grad[l]);
  }
#pragma omp barrier
for (l=0; l<N; l++)
  { gradtot[l] = 0;
    for (k=0; k<npath; k++) gradtot[l] += grad[k][l]; }
OpenMP-based Computation of Greeks: ADOL-C

```c
int nthreads = 5;
int slot = npath/nthreads;
#pragma omp parallel firstprivate(ADOLC_OpenMP_Handler)
 { int index = omp_get_thread_num();
   int l,k; /* private variables */
   double *L = new double[N];
   adouble *La = new adouble[N]; adouble va;
   trace_on(index);
   ...
   trace_off();
   for(l=index*slot; l<(index+1)*slot; l++)
     gradient(index,N+Nmat,xp[l],grad[l]);
#pragma omp barrier
   for (l=0; l<N; l++)
     { gradtot[l] = 0;
       for (k=0; k<npath; k++) gradtot[l] += grad[k][l]; }
}
```

Andrea Walther
Parallel Computation of Greeks
7th Euro AD Workshop
Parallel Computation of Greeks: Runtimes

- npath=10000
- npath=20000
- npath=30000
- npath=40000
- npath=50000
- npath=60000
- npath=70000
- npath=80000
- npath=90000
- npath=100000

runtime in seconds

# processors
Parallel Computation of Greeks: Speedup

The diagram shows the speedup of parallel computation of Greeks as a function of the number of processors. The x-axis represents the number of processors, while the y-axis represents the speedup. The ideal speedup is shown by a straight line, and the actual speedup for different path lengths (npath) is represented by colored lines. The graph indicates that as the number of processors increases, the speedup also increases, although the speedup starts to plateaus at higher processor counts.
Summary and Outlook

- ADOL-C 2.0 coming soon
- stable integration of recent developments
  - external differentiated functions, checkpointing, fixpoint iterations
- improved exploitation of sparsity by coupling with ColPack
- easy-to-use OpenMP parallelization
- Further improvement of speedup?!

Contact:
WWW: http://www.math.tu-dresden.de/~adol-c
probably changing to COIN-OR
The Fakultät für Mathematik of the TU Chemnitz invites to the

22st Chemnitz FEM Symposium 2008

with the special topic

Sensitivity Analysis and Algorithmic Differentiation for FEM

September 28 – 30, 2009