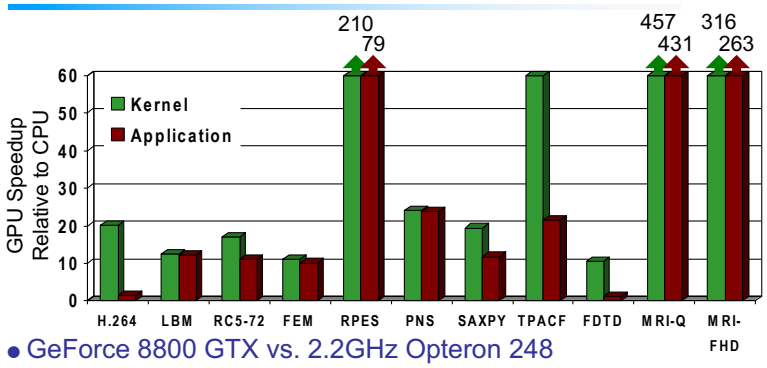
Sample Projects

Application	Description	Source	Kernel	% time
H.264	SPEC '06 version, change in guess vector	34,811	194	35%
LBM	SPEC '06 version, change to single precision and print fewer reports	1,481	285	>99%
RC5-72	Distributed.net RC5-72 challenge client code	1,979	218	>99%
FEM	Finite element modeling, simulation of 3D graded materials	1,874	146	99%
RPES	Rye Polynomial Equation Solver, quantum chem, 2-electron repulsion	1,104	281	99%
PNS	Petri Net simulation of a distributed system	322	160	>99%
SAXPY	Single-precision implementation of saxpy, used in Linpack's Gaussian elim. routine	952	31	>99%
TRACF	Two Point Angular Correlation Function	536	98	96%
FDTD	Finite-Difference Time Domain analysis of 2D electromagnetic wave propagation	1,365	93	16%
MRI-Q	Computing a matrix Q, a scanner's configuration in MRI reconstruction	490	33	>99%

Speedup of GPU-Accelerated Functions



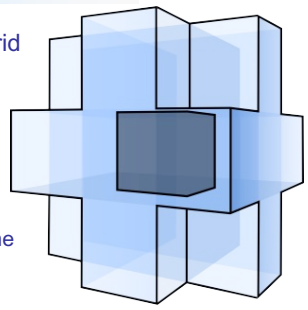
- GeForce 8800 GTX vs. 2.2GHz Opteron 248
- 10× speedup in a kernel is typical, as long as the kernel can occupy enough parallel threads
- 25× to 400× speedup if the function's data requirements and control flow suit the GPU and the application is optimized
- Keep in mind that the speedup also reflects how suitable the CPU is for executing the kernel

Sample Programs Scaling

App.	Archit. Bottleneck	Simult. T	Kernel X	App X
LBM	Shared memory capacity	3,200	12.5	12.3
RC5-72	Registers	3,072	17.1	11.0
FEM	Global memory bandwidth	4,096	11.0	10.1
RPES	Instruction issue rate	4,096	210.0	79.4
PNS	Global memory capacity	2,048	24.0	23.7
LINPACK	Global memory bandwidth, CPU-GPU data transfer	12,288	19.4	11.8
TRACF	Shared memory capacity	4,096	60.2	21.6
FDTD	Global memory bandwidth	1,365	10.5	1.2
MRI-Q	Instruction issue rate	8,192	457.0	431.0

LBM Fluid Simulation (from SPEC)

- Simulation of fluid flow in a volume divided into a grid
  - It's a stencil computation: A cell's state at time *t+1* is computed from the cell and its neighbors at time *t*
- Synchronization is required after each timestep – achieved by running the kernel once per timestep
- Local memories on SMs are emptied after each kernel invocation
  - Entire data set moves in and out of SMs for every time step
  - High demand on bandwidth
- Reduce bandwidth usage with software-managed caching
  - Memory limits 200 grid cells/threads per SM
  - Not enough threads to completely cover global memory latency



Flow through a cell (dark blue) is updated based on its flow and the flow in 18 neighboring cells (light blue).

What you will likely need to hit hard

- Parallelism extraction requires global understanding
  - Most programmers only understand parts of an application
- Algorithms need to be re-designed
  - Programmers benefit from clear view of the algorithmic effect on parallelism
- Real but rare dependencies often need to be ignored
  - Error checking code, etc., parallel code is often not equivalent to sequential code
- Getting more than a small speedup over sequential code is very tricky
  - ~20 versions typically experimented for each application to move away from architecture bottlenecks

Call to action on Project Work

- One page project description (pdf + trac) due end of March

  Introduction: A one paragraph description of the significance of the application.
  - **Description:** A one to two paragraph brief description of what the application really does
  - **Objective:** A sentence on what I would like to accomplish with the team on the application – we have to agree on this at the lab.
  - **Background :** Outline the technical skills (type of Math, Physics, Chemistry, etc) that one needs to understand and work on the application.
  - **Resources:** A list of web and traditional resources that students can draw for technical background, general information and building blocks. Give URL or trac/svn links. Only use our trac/svn.
  - **Contact Information:** Name, e-mail, group and master program the team members are part of.
- The remaining labs until the end of March are dedicated to presentation of project ideas by you and me and used to recruit teammates

67

Mathematical Modeling

- **Why** to model?
- **What** to model?
- **How** to model?
  - Ordinary Differential Equations
  - Partial Differential Equations
  - Etc.

67

68

Why to use models? Why not?

- To understand better the behavior of a system
- To predict the future development of a system or even to manipulate the system in order to achieve a desired result
- Models are usually cheaper and faster than conventional experiments
- Sometimes the only feasible way to find a solution is through modeling
  - Models can be too complicated or too restricted

68

69

What can be modeled?

- Mechanical structures: airplanes, buildings, skeletons
- Chemical reactions: batteries, pulp manufacturing
- Electromagnetic fields: mobile phones, human heart
- Fluid dynamics: airplanes, human heart
- Biological systems: simple population models, competition
- Not yet: macroscopic properties **from** atomic structure, **long time** weather forecasting (“never”...)

69

70

Modeling methods

- Differential equations: Ordinary & Partial DEs:
  - By far the most common method
  - Ample supply of numerical methods and computer programs
- Simulations: particle simulations, simulated annealing
  - Discrete dynamical models
  - Rules for interaction or changing from a state to another
- Game theory
- Statistical models
- Genetic algorithms

70

71

Things to consider

- Find the right mathematical description for the phenomenon to be modeled
  - Start with as simple a model as possible
  - Include only the most essential characters
- Decide how to analyze the model
- Try to estimate the reliability of your model AND your analysis
- Improve the model if needed
  - Change parameters
  - Add or remove something

71

72

Levels of models

- No model at all
- A model with no solutions
  - A frustrating situation, if it goes unrecognized for long
  - Mathematical analysis with existence theorems
- A model that can be solved numerically with computers
  - Problem: some numerical methods always give an answer - even when no solution exists!

72

73

Levels of models

- A model with analytical solutions
  - A relatively rare incidence with new, scientifically noteworthy models
    - The models that are easily analyzable have already been analyzed
    - Those that have defied analytical methods so far are usually very hard
  - Very useful as examples and test problems for computational methods.

73

74

Classical modeling

- Objectives
- Hypothesis – intuition, expertise
- Mathematical Formulation – experience
- Verification – hard work, testing over and over again
- Calibration – estimation, comparison
- Analysis and Evaluation – varies from mechanical work to very abstract proofs
- Usually, the modeling process starts from the top of this list and proceeds downwards until for some reason or another it is necessary to go back to a previous point

74

75

Objectives

- Before you start modeling, you should be able to answer these questions:
  - What is the system that you plan to model?
  - Which properties do you wish to find out about the system?
  - Are you after numbers (quantitative modeling) or just some characteristic behavior (qualitative modeling)?
  - How detailed must the model be?
  - How do you know when the model is adequate?
  - What are you going to do with the results?
    - Testing a new idea
    - Scientific publication: Journal, Conference, Newspaper article
    - Basis for a recommended action
    - Product design

75

76

Possible error sources

- New errors are introduced at every step of modeling
- Modeling error: the gap between the model and 'reality'
- Errors due to mathematical methods used in solving the model
  - This error can be zero in simple models (where the modeling error is usually larger)
  - Several methods in different stages of solution
    - The methods should be in balance with respect to their errors
  - Theoretical error estimates for numerical methods
- Errors induced by the use of computers
  - Computer arithmetic (rounding)
  - Errors in programming
  - Errors in post-processing (visualization, data analysis)
  - Wrong interpretation of the results

76

77

Table of Contents

- Motivation & Trends in HPC
- Mathematical Modeling
- **Numerical Methods used in HPSC**
  - **Automatic Differentiation**
  - Systems of Differential Equations: ODEs & PDEs
  - Solving Optimization Problems
  - Solving Nonlinear Equations
  - Basic Linear Algebra, Eigenvalues and Eigenvectors
  - Chaotic systems
- HPSC Program Development/Enhancement: from Prototype to Production
- Visualization, Debugging, Profiling, Performance Analysis & Optimization

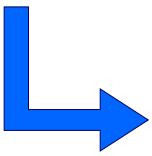
77

78

Automatic Differentiation – Motivation

- ***Design optimization***
- ***Sensitivity analysis & Parameter identification***
- ***Data assimilation problems***
- ***Inverse problems (data assimilation)***
- ***Solving ODE, PDE, DAE, …***
- ***Linear approximation & Curve fitting***

Derivative information required



Numeric Differentiation

Symbolic Differentiation

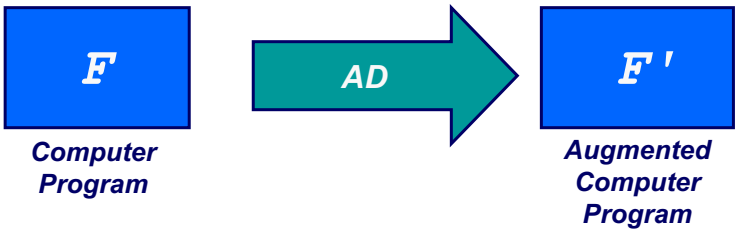
Automatic Differentiation

78

79

Automatic Differentiation (AD)

- Semantic transformation:
  - Given a computer code **F**, AD generates a new program **F'** which applies the chain rule of differential calculus to elementary operations for which the derivatives are known



79

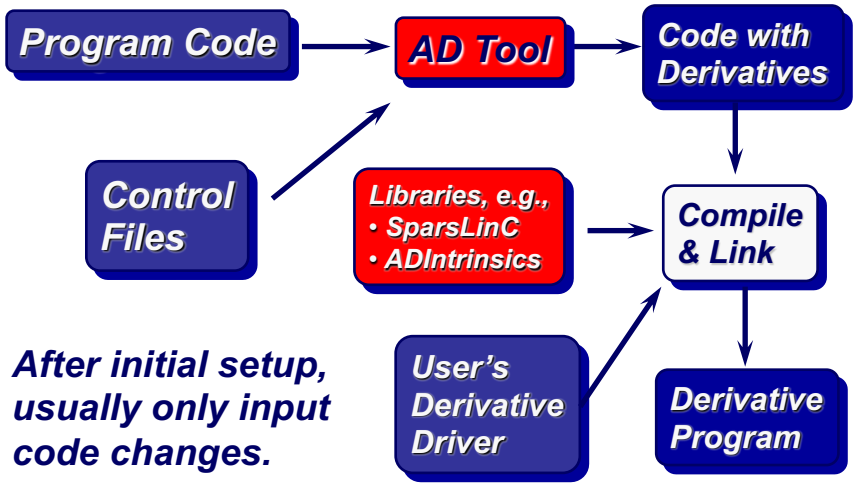
80

AD Advantages

- Generates truncation- and cancellation error-free derivatives
- Generates a program for the computation of derivative values, not of derivative formulae
- The associativity of the chain rule allows for a wide range of choices in accumulating derivatives → forward & reverse modes and generalizations

80

The AD Process



AD-Tool Implementations

- Source Transformation (ST)
  - Compiler-based technique → generate new code that **explicitly** computes derivatives
  - **Advantages**: entire context known at compile-time → efficient code, transparency
  - **Drawback**: difficult implementation
- Operator Overloading (OO)
  - Extends elementary operations & functions to also **implicitly** compute derivatives
  - **Requires**: redeclaration of active variables to the new overloaded types
  - **Advantages**: easy implementation & „same" source code
  - **Drawback**: granularity → inefficient code

Some AD Tools

http://www.autodiff.org

Fortran 77, some Fortran 90:

- ADIFOR 3.0/ADJIFOR (RM, ∂f , $\partial^2 f$): Alan Carle & Mike Fagan (Rice)
- Tapenade (RM, ∂f): Laurent Hascoet (INRIA)
- TAMC/TAF (RM, ∂f): Ralf Giering (FastOpt GmbH).

ANSI-C/C++:

- ADOL-C: (FM/RM, $\partial^k f$): Andreas Griewank & Co. (TU Dresden)
- ADIC: (FM/RM, $\partial^k f$): Hovland (ANL)

Application Specific:

- Cosy-Infinity (Remainder Differential Algebra): Martin Berz (U Michigan)
- TOMLAB/MAD (AD of Matlab): Shaun Forth (RMCS Shrivenham)
- ADiMat (AD of Matlab): Andre Vehreschild (RWTH Aachen)

OpenAD/F:

- AD Framework: J. Utke & U. Naumann (ANL & RWTH Aachen)