HCDC Workloads, Evaluation & HCDC 2

Yipeng Huang

July 30, 2015

Workload Classes

- **Open loop**
  $x' = Ax + b$

- **Closed loop**
  $Ax = b$

- **These are the core of machine learning and scientific computation work**
  - Dense matrix: SVM, regression, optimization
  - Sparse matrix: PDEs, ODEs

State of the Art for Linear Systems

- **While linear systems are extremely well studied, some problems remain:**

- **Stiffness**
  - we need solvers that handle high dynamic range in values or frequency

- **Large scale**
  - we need ways to decompose large problems and solve in parallel

- **Economy**
  - we need ability to use cheap but unreliable or inaccurate algorithms, in addition to powerful and expensive ones

Can Continuous Time Computation Help?

- **All prevailing techniques rely on time stepping**
  - While SIMD vector processors and GPUs allow simultaneous operations, problems still done step-by-step
  - Time stepping introduces the problem of stiffness

- **Investigate <u>where</u> is the efficiency advantage of HCDC coming from:**

- **Dataflow?**
  - Easily replicated in FPGA, digital ASIC, DDA

- **Continuous time computation?**
  - Is the continuous time evolution tackling stiffness,
  - Is it doing a better job than preconditioners, implicit solvers, direct methods

Multigrid

- **Decompose problem into grid of grids**
  - Each grid has limited number of variables
- **In solving each grid, precision not required**
  - Each solution only has to precondition & provide initial guess for next, finer grid
- **Permits use of small, cheap, low dynamic range, unreliable linear solvers**
  - Jacobi iteration often used
  - Alternatively, just one step of conjugate gradients
  - How cheap can we go?
  - Steepest descent? Fixed point steepest descent? HCDC?

Comparison Targets

- **DDA**
  - Built fixed-point, floating-point, variable-order DDA
  - Also tried stochastic DDA
  - I plan on adding support for steepest descent, conjugate gradients
- **GPU**
  - Working my way through GPU multigrid and conjugate gradients code
- **CPU**
  - Built collection of optimization, linear systems, ODE solvers

What Analog Computing Needs

- **In building HCDC to obtain continuous-time computation, we gave up:**
- **Accuracy and precision**
- **Large problem size**
- **Programmability**

Accuracy

- **Hybrid digital-analog iteration for Ax = b**
  - Solve system of equations of residuals in analog computer, obtain correction
  - Add correction term to solution using digital computer
  - Each stage “zooms in” to accurate solution

- **Scaling everything to fit HCDC’s limited range**
  - A terms need to fit in multipliers, so scale by alpha
  - x terms need to fit in integrators and ADCs, so scale by beta
  - To keep problem correct, b is scaled by both alpha and beta
  - And finally make sure b terms fit in the DACs

- **Example:**
  - Ideal solution = [-1047.273926, -1679.667969]
  - HCDC solution = [-1047.772095, -1680.471802]

Large problem size

- **Digital computers easily handle all problems we currently do on HCDC**
  - Most powerful, precise, stiff solvers still 100x faster than HCDC, accounting for time to set up HCDC, measure results
- **HCDC may have advantage once scaled up**
  - Configuration of HCDC takes N^2 time, N is # of variables and/or integrators
  - Conjugate gradients takes N^3 time
- **Urgently need to understand efficient problem decomposition techniques like multigrid**

Programmability

- **HCDC 2 increases programming speed using**
  - Higher SPI clock speed
  - Hardware support for transmitting configuration bits
  - Software optimizations

Next Steps

- **HCDC 2 synthesis & simulation**
- **Try using HCDC to help multigrid methods**
- **Build GPU codes for comparison**
- **The nonlinear world…**
  - While linear problems have broad appeal, and therefore are good for computer architecture and workloads-based research,
  - Nonlinear problems are also possible in HCDC;
  - But specific nonlinear problems are only interesting to specific fields