Speedup using Flowpaths for a Finite Difference Solution of a 3D Parabolic PDE

Darrin M. Hanna
Dept. of Computer Science and Engineering
Oakland University

Anna M. Spagnuolo
Dept. of Mathematics and Statistics
Oakland University

Graduate Research Assistant: Michael DuChene
High Performance FPGA Systems Laboratory

This work supported by NSF SGER grant #0636895
October 2006 – September 2007
Motivation

• Many important numerical simulations take days, months, years
  • Applications include weather prediction, ADCIRC (UT-Austin with Clint Dawson), contaminant transport in porous media, oil recovery, medical device applications, and others.
• Individual parts of large-scale numerical computations are often run and tested on a PC before running on a supercomputer.
• Repeated numerical simulations that can take months on a PC are often not adequate for real time applications (DDAS).
• Portable, low-power, mobile systems that use numerical methods have little high-speed technology.
• Speeding up numerical simulations run on PCs and in Embedded Systems impacts small- and large-scale numerical computing.
Research Objectives

Digital System Design Goals

Zero cost

Zero power

Leading edge wedge

Zero delay

Washing machines
Microwave ovens
Dishwashers

Electric Toothbrushes
Digital Watches

Servers,
Supercomputers

Desktop PCs,
DVD players,
Cable boxes

Adapted from Nick Tredennick Gilder Technology Report
Motivation

• Currently many researchers use PCs to run numerical simulations. Supercomputer users often develop and test codes using PCs.
• Embedded (mobile) systems for running numerical codes are limited to processor cores.
• Take advantage of a reconfigurable, spatial computing paradigm for speeding up simulations
  • **Automated**
  • Uses *existing* codes written in common languages such as FORTRAN, C/C++, and Java
  • **Affordable and easy to use**
  • Generate hardware that has *higher execution frequencies*
  • Generate hardware description that is *human readable*
Research Objectives

- Develop a method optimized for speeding up execution of numerical codes using reconfigurable, spatial computing
  - Work is based on previous results in speedup using methodology for creating Flowpaths – SPPs from multi-threaded code
  - Flowpath optimization techniques based on constructs commonly found in numerical codes
- Current focus is towards an affordable, easy to use system for speedup compared with a PC
Why Spatial Computing? Why Flowpaths?

- Processors
  - Load-execute-store overhead
  - Stack operation overhead
  - Register and data manipulation overhead
  - OS overhead
  - Multithreading overhead including context switching, etc…
  - Fixed chip space
- Special-purpose processors (SPP)
  - Eliminate these overheads
  - Variable chip space
  - Critical path determines maximum execution frequency*
  - Very difficult to design, time consuming, requires specialized skill
Current Results

• 3D diffusion problem solved using a Finite Difference Method

\[
\frac{\partial c}{\partial t} + \nabla (uc) - \nabla (D \nabla c) = \frac{f(c)}{\phi}
\]

where \( f(c)/ = \rho c + S(c) \)

• The no-flow boundary conditions are imposed as follows:

\[
u \nabla c \cdot n = D \nabla c \cdot n = 0, \quad \text{on } \partial \Omega,
\]

• Initial condition \( c(x, 0) = c_{\text{init}}(x), \text{ in } \Omega.\)
Current Results

- Discretization using Cell-Centered Finite Difference Method

\[ f_i^* = f \left( (i + 1/2)h, (j + 1/2)h, (k + 1/2)h \right), \]
\[ c_i^* = c \left( (i + 1/2)h, (j + 1/2)h, (k + 1/2)h \right), \]

\[ D_{1,i} = D \left( (j + 1/2)h, (k + 1/2)h \right), \]
\[ D_{2,i} = D \left( (i + 1/2)h, jh, (k + 1/2)h \right), \]
\[ D_{3,i} = D \left( (i + 1/2)h, (j + 1/2)h, kh \right). \]

\[ \left( \nabla D \nabla c^* \right)_{h,i} = \frac{1}{h^2} \sum_{l=1}^{3} \left( D_{l,i} \left( c_{i+he_l}^* - c_i^* \right) - D_{l,i} \left( c_i^* - c_{i-he_l}^* \right) \right) \]
Current Results

- The Finite Difference Equation

\[
\frac{c_{i}^{*,n} - c_{i}^{*,n-1}}{\Delta t} - (\nabla D \nabla c_{i}^{*,n})_{h,i} = \frac{f_{i}^{*,n}}{\phi} \quad \text{on } \Omega_{h},
\]

- Operator Splitting Method

**Transport:** We assume the special case that \( u = 0 \).

\[
c_{i}^{*,n} = e^{\rho \Delta t} c_{i}^{*,n-1}
\]

**Diffusion:** Conjugate Gradient Method (bottleneck)

\[
\frac{c_{i}^{*,n} - c_{i}^{*,n}}{\Delta t} - (\nabla D \nabla c_{i}^{*,n})_{h,i} = S_{i}^{*,n}
\]
Current Results

• To Start
  • Create double arithmetic components
  • Non-optimized flowpaths
    • Inspect both extremes
      • Entire algorithm is a flowpath
      • A single line of code is a flowpath
• Next Steps
  • Employ flowpath optimizations
  • Use techniques to take advantage of code-level parallelism
  • Explore this methodology with Finite Element Methods
Current Results

- Entire code is a flowpath
  - 96.75 MHz
- PC - 1.10 GHz, 1.25 GB RAM
### Speedup relative to flowpath

<table>
<thead>
<tr>
<th># of Points</th>
<th>CPU – Java</th>
<th>CPU - C++</th>
<th>CPU - FORTRAN</th>
<th>Flowpath</th>
</tr>
</thead>
<tbody>
<tr>
<td>1650</td>
<td>657</td>
<td>64</td>
<td>461</td>
<td>1</td>
</tr>
<tr>
<td>13200</td>
<td>690</td>
<td>65</td>
<td>471</td>
<td>1</td>
</tr>
<tr>
<td>105600</td>
<td>704</td>
<td>65</td>
<td>481</td>
<td>1</td>
</tr>
</tbody>
</table>

### Time speedup

<table>
<thead>
<tr>
<th># of Points</th>
<th>Algorithm Runtime (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU – Java 1.1 GHz</td>
</tr>
<tr>
<td>1650</td>
<td>10,405</td>
</tr>
<tr>
<td>13200</td>
<td>76,991</td>
</tr>
<tr>
<td>105600</td>
<td>588,186</td>
</tr>
</tbody>
</table>
Current Results

- PowerPC on a Xilinx Virtex2 XC2VP30
- One line of code executing as a flowpath
  - 413,000,000 clock cycles to execute that line on a PowerPC
  - Emulated double arithmetic operations
- 4,538,887 clock cycles using a flowpath
- 82.315 MHz

```plaintext
    do i = 1,nx
    do j = 1,ny
    do k = 1,nz

        u(i,j,k) = adiag(i,j,k)*v(i,j,k) - aleft(i,j,k)*v(i-1,j,k) -
                   aright(i,j,k)*v(i+1,j,k) -
                   aup(i,j,k)*v(i,j,k+1) -
                   adown(i,j,k)*v(i,j,k-1) -
                   afront(i,j,k)*v(i,j+1,k) -
                   aback(i,j,k)*v(i,j-1,k)

...```
Challenges

- Synthesizing components to hardware takes time
  - One-time overhead for a given numerical code
- FPGA space is finite
  - Making use of reconfigurable real estate efficiently
- Creating a methodology that is both efficient and compatible with multiple, common languages
- Currently, busses between embedded microcores and on-chip processors are slow
- Bus interfaces can also be a limiting constraint (FPGA-FPGA, FPGA-PC)
- Temporary and persistent storage is limited
Conclusions

• Using reconfigurable, spatial computing, numerical codes can be sped up at least an order of magnitude *before* optimization or parallelism
• Hardware is generated from existing codes and is human readable
• Observations indicated that parallelism and optimization can lead to between two and three orders of magnitude of speedup.

Next Steps

• Develop methodology for generating flowpaths optimized specifically for constructs commonly occurring in numerical codes
• Use existing techniques for automated code-level parallelization for further speedup
• Compare the speed of this approach to using GPUs
• Compile the Java LINPACK to hardware
THANK YOU! 😊