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

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

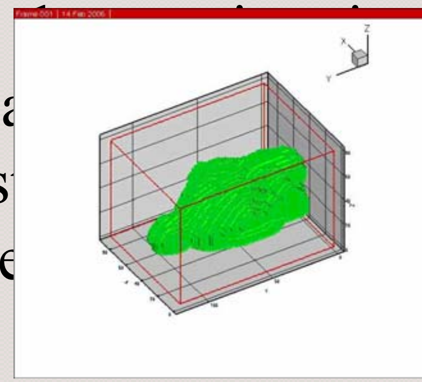
Anna M. Spagnuolo
Dept. of Mathematics and Statistics
Oakland University

Graduate Research Assistant: Michael DuChene
High Performance FPGA Systems Laboratory

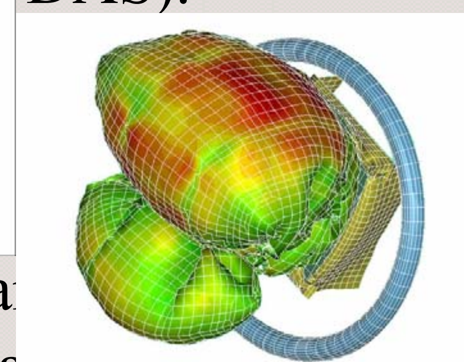
This work supported by NSF SGER grant #0636895
October 2006 – September 2007

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 … tes on a PC are often not adequate for real time a… DAS).



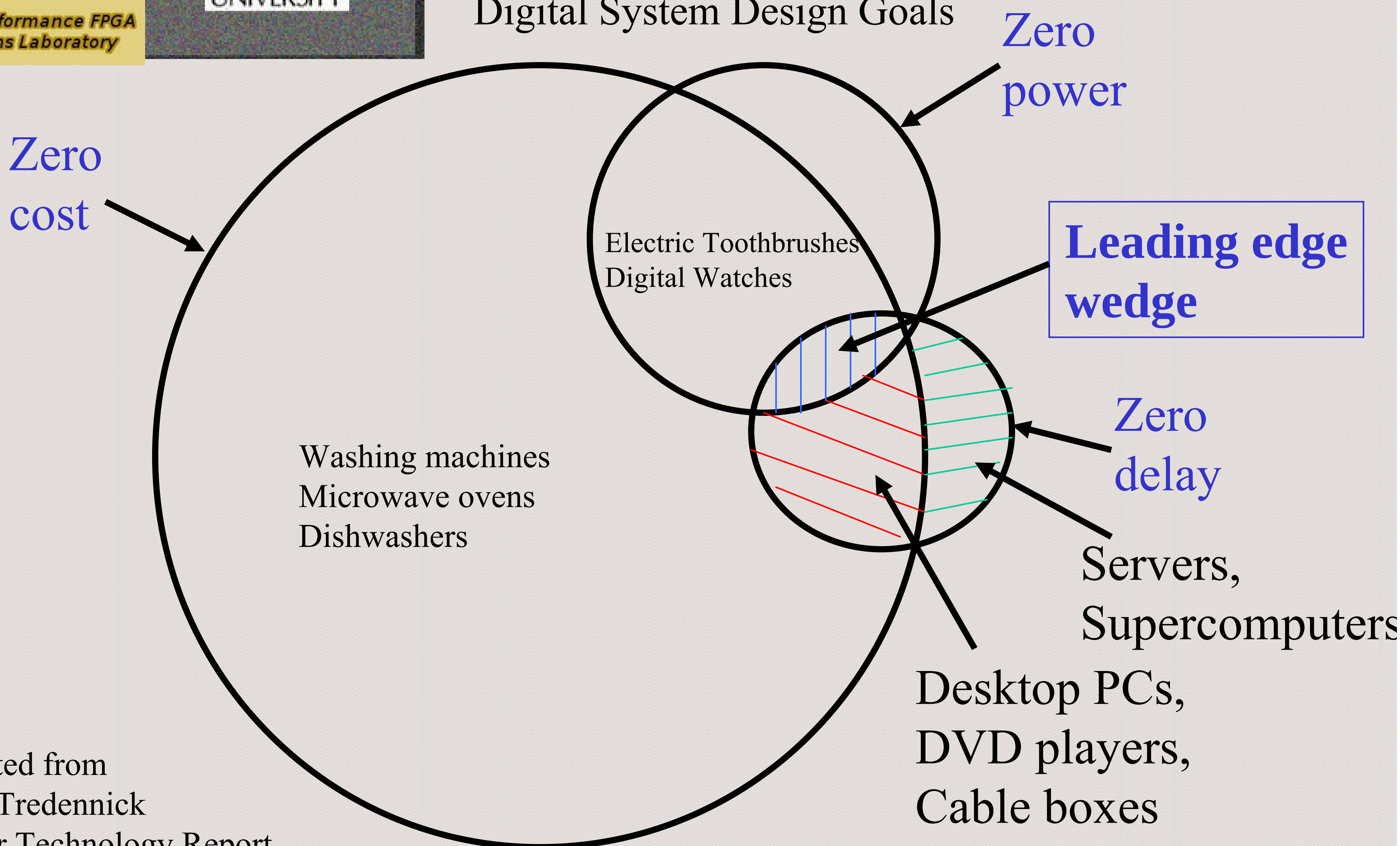
- Portable, low-power, mobile sys… methods have little high-speed te…



- Speeding up numerical simulations run on PCs a… Systems impacts small- and large-scale numerical computing

Research Objectives

Digital System Design Goals



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**

High Performance FPGA Systems Laboratory

Oakland UNIVERSITY

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

High Performance FPGA Systems Laboratory

Oakland University

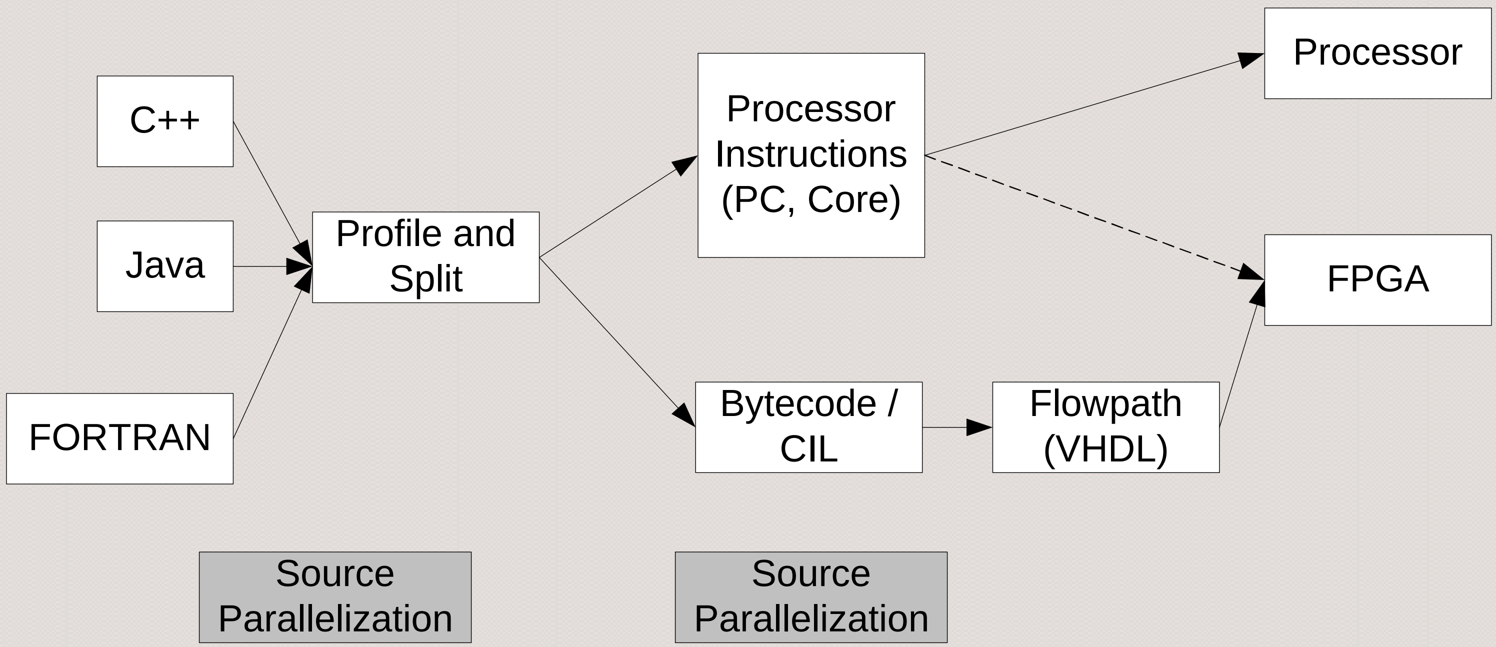
Research Objectives

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

Research Objectives

Design Flow



Current Results

- 3D diffusion problem solved using a Finite Difference Method

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

where f(c)/ =ρc+S(c)

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

$$u \cdot v = D \nabla c \cdot v = 0, \quad \text{on } \partial\Omega,$$

- Initial condition c(x, 0) = cinit(x), in Ω.

Current Results

- Discretization using Cell-Centered Finite Difference Method

$$f_{\mathbf{i}}^* = f((i+1/2)h,\ (j+1/2)h,\ (k+1/2)h),$$

$$c_{\mathbf{i}}^* = c((i+1/2)h,\ (j+1/2)h,\ (k+1/2)h),$$

$$D_{\mathbf{1,i}} = D(ih, (j+1/2)h, (k+1/2)h),$$

$$D_{\mathbf{2,i}} = D((i+1/2)h, jh, (k+1/2)h),$$

$$D_{\mathbf{3,i}} = D((i+1/2)h, (j+1/2)h, kh).$$

$$(\nabla \cdot D \nabla c^*)_{h,\mathbf{i}} = \frac{1}{h^2}\sum_{l=1}^{3}(D_{l,\mathrm{i}+h\mathrm{e}_l}(c^*_{\mathrm{i}+h\mathrm{e}_l} - c^*_{\mathrm{i}}) - D_{l,\mathrm{i}}(c^*_{\mathrm{i}} - c^*_{\mathrm{i}-h\mathrm{e}_l})$$

Current Results

- The Finite Difference Equation

$$\frac{c_{\mathbf{i}}^{*,n} - c_{\mathbf{i}}^{*,n-1}}{\Delta t} - (\nabla \cdot D \nabla c^{*,n})_{h,\mathbf{i}} = \frac{f_{\mathbf{i}}^{*,n}}{\phi} \text{ on } \Omega_h,$$

- Operator Splitting Method

Transport: We assume the special case that $u = 0$.

$$\overline{c}_{\mathbf{i}}^{*,n} = e^{\rho \Delta t} c_{\mathbf{i}}^{*,n-1}$$

Diffusion: Conjugate Gradient Method (bottleneck)

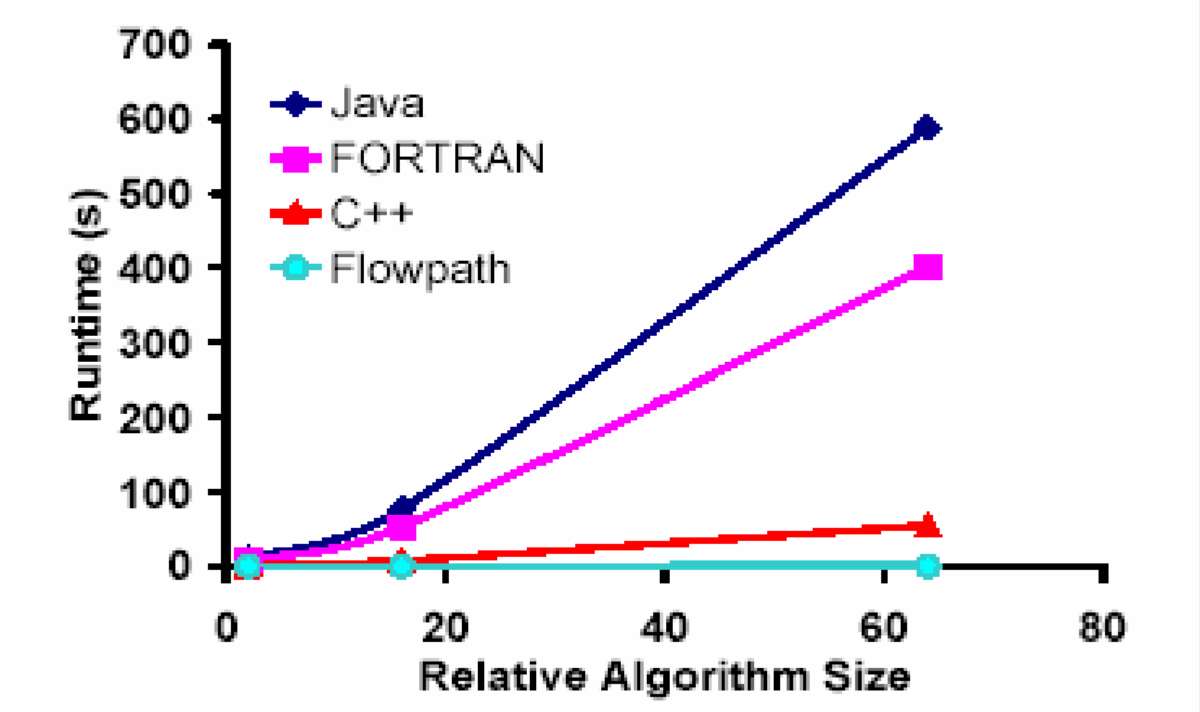
$$\frac{c_{\mathbf{i}}^{*,n} - \overline{c}_{\mathbf{i}}^{*,n}}{\Delta t} - (\nabla \cdot D \nabla c^{*,n})_{h,\mathbf{i}} = S_{\mathbf{i}}^{*,n}$$

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



High Performance FPGA Systems Laboratory

Oakland University

Current Results

Speedup relative to flowpath

# of Points	Flowpath Speedup			
	CPU – Java	CPU - C++	CPU - FORTRAN	Flowpath
1650	657	64	461	1
13200	690	65	471	1
105600	704	65	481	1

Time speedup

# of Points	Algorithm Runtime (milliseconds)			
	CPU – Java 1.1 GHz	CPU – C++ 1.1 GHz	CPU – FORTRAN 1.1 GHz	Flowpath (100 MHz)
1650	10,405	1,018	7,311	16
13200	76,991	7,202	52,545	112
105600	588,186	54,705	401,978	835

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

```
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 and Next Steps

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! ☺