Cerebral Blood Flow Simulation – A SEJITS friendly framework

Meriem Ben-Salah, Chris Chaplin, Razvan Carbunescu

Motivation

❖ Productivity layer programming language (e.g. Python) vs. efficiency layer programming language (e.g. C++):
  - ❖ 5x faster development and 3-10x fewer lines of code
  - ❖ BUT, 10x-100x less performance without exploring hardware model

❖ Scientists → Productivity layer Programmers

❖ ☹ Computationally expensive problems and lack of computer science knowledge -> Disaster!

❖ In need of a technique that permits:
  - ❖ high level knowledge of computer architecture abstraction
  - ❖ and simultaneously good performance

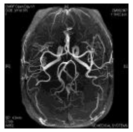
❖ SEJITS!

❖ Scientists/Productivity Layer Programmer input is needed

Cerebral Blood Flow Simulation

❖ Explore the SEJITS technique by means of a real application and define what the needs are?

❖ Personalized Medicine Application: Simulation of the blood flow in the main cerebral arteries (the Circle of Willis) based on patient data (CT scan, ultrasound) to save costs and save lives of stroke patients.

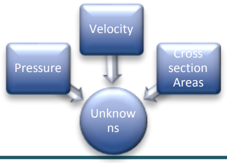


Theoretical Background: Physics

Blood
- Incompressible
- Inviscid

Interaction
- Resistance-Compliance Model

Vessel
- Elastic
- Thin Walled
- Isotropic



One dimensional physics on three dimensional geometry

System of PDEs: 1. Conservation of Mass , 2. Conservation of Momentum (+) Constitutive model

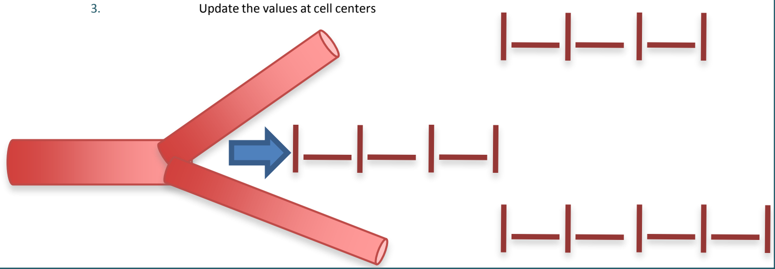
Numerics

❖ Numerical Scheme: Finite Volume Method in space and Forward Difference in time

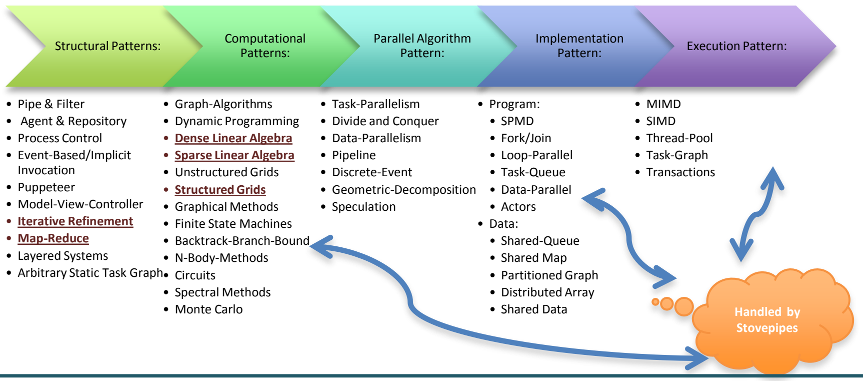
❖ Complexity:
  - ❖ O(10^6) time steps, each time step is 10 microseconds long.
  - ❖ Maximal time: ~ 5 to 10 seconds
  - ❖ O(10^4) grid points, spatial grid size is 0.1 millimeter
  - ❖ Total length of vasculature: 1- 1.5 meter
  - ❖ O(160) Gigaflops
  - ❖ Time alloted: 30 seconds
  - ❖ Gigafloprate: 240 Gigaflos/second

Solution Algorithm

1. Input geometry of cerebral arteries from medical image
2. Assume an initial state of blood flow
3. Generate spatial discretization
4. March in time
   1. Calculate the cell edges values AND
      1. Enforce inflow boundary conditions
      2. Enforce outflow boundary conditions
      3. Enforce continuity at bifurcation locations
   2. Combine the cell edges values
   3. Update the values at cell centers



Design Space



Structural Patterns:
- Pipe & Filter
- Agent & Repository
- Process Control
- Event-Based/Implicit Invocation
- Puppeteer
- Model-View-Controller
- **Iterative Refinement**
- **Map-Reduce**
- Layered Systems
- Arbitrary Static Task Graph

Computational Patterns:
- Graph-Algorithms
- Dynamic Programming
- **Dense Linear Algebra**
- **Sparse Linear Algebra**
- Unstructured Grids
- **Structured Grids**
- Graphical Methods
- Finite State Machines
- Backtrack-Branch-Bound
- N-Body-Methods
- Circuits
- Spectral Methods
- Monte Carlo

Parallel Algorithm Pattern:
- Task-Parallelism
- Divide and Conquer
- Data-Parallelism
- Pipeline
- Discrete-Event
- Geometric-Decomposition
- Speculation

Implementation Pattern:
- Program:
  - SPMD
  - Fork/Join
  - Loop-Parallel
  - Task-Queue
  - Data-Parallel
  - Actors
- Data:
  - Shared-Queue
  - Shared Map
  - Partitioned Graph
  - Distributed Array
  - Shared Data

Execution Pattern:
- MIMD
- SIMD
- Thread-Pool
- Task-Graph
- Transactions

SEJITS--Selective Embedded Just In Time Specialization

❖ Libraries are written in Low Level Programming Languages and are too specific

❖ Stick with a High Level Programming Language e.g. Python

❖ Hidden to the productivity layer programmer but good to know:
  - ❖ Computational Patterns (good for reuse) are selected (if it is worth it) and specialized at runtime
  - ❖ A specialized function is written in a low level language targeted to the "parallel" hardware architecture (so called stovepipe technique)
  - ❖ Use of Autotuning techniques (e.g. in Pyski, autotuned Linear Algebra methods embedded in Python)
  - ❖ The low level code is compiled at run time
  - ❖ The low level code is dynamically linked using the high level language interpreter

Implementation

❖ Separate prototyping in MATLAB for the verification of the correctness & stability of the numerical algorithm.

❖ MATLAB is a high level scripting language but has weak glue facilities.

❖ Implement algorithm in Python using data types and linear algebra operations that are intrinsic to NumPy (for now).

❖ Invoke calls/prospection by PySki and by SEJITS specialization

Hints from the Productivity Layer Programmer

❖ Do we need to approach programming differently?

❖ Use of functional programming techniques to give hints about specialization: map, filter, reduce

❖ Use semantics to give hints about specific computational and structural patterns, e.g. use of stencil operation or sparse matrix vector multiplication SpMV.

Requirements from a Productivity Layer Perspective

❖ Language Requirements: easy and familiar

❖ Composition: hierarchical composition of patterns to build frameworks, integration of legacy code

❖ Compilation: transparent conversion of data

❖ Quality of Service: real time service, best performance

Conclusion

❖ Hardware and implementation techniques are changing every day.

❖ Domain experts do not need to dive into detail into the computer science world.

❖ SEJITS will hopefully save lives of the productivity masses:
  - ❖ high level programming
  - ❖ performance (implicit use of new computer technologies)
  - ❖ Same program on all platforms: cluster, cloud, multicore etc.

❖ Input of productivity programmers is essential for success/applicability of this technology.

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, Samsung, and Sun Microsystems.

Conctacts: Meriem Ben-Salah bensalah@me.berkeley.edu, Chris Chaplin cmchap@berkeley.edu
Razvan Carbunescu carazvan@eecs.berkeley.edu