Special Topics in Advanced Computing Technology: Programming and System Design

by

Dr. Lesley Shannon

Email: lshannon@ensc.sfu.ca

Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc894

Simon Fraser University

Slide Set: 1

Date: January 26, 2010

Slide Set Overview

- More on Course Readings
- Student Expertise for projects
- Graph theory basics
- This week's papers
- Future week's seminars

Course Readings

- Most week’s there are only three papers.
- When doing your presentation, you can focus on one of the papers if you wish.
  - I didn’t select all of these papers because they are the “best”; some are controversial
    - I don’t even like them.
- Highlight the interesting parts

Course Readings

- Next week's programming languages lecture tries to highlight a few different languages
  - However, programming languages and models are closely tied so you've seen a few this week as well.
- You do not have to teach the class four different languages during your seminar (highlight the interesting capabilities/functionality)

Course Readings

- The Computing Architectures class currently has 5 papers (2 SoCs and 3 Clouds)
- If we don’t split the lecture into two separate lectures (one on clouds and one on SoCs), you can focus on one of them

Course Readings

- I'm open to you changing the suggested reading list for your topic area ***WITH MY APPROVAL***
  - You must give the class at least one week's notice
  - My approval will be clear from my updating the course reading list web page and indicating that changes have been made on web ct/via email

Course Readings

- I cannot post copyrighted material on the course reading list web page that is not freely available on the web.
  - You have access to this material through SFU's (UBC's) online library
  - If there are serious problems in getting this to work (I've tried it without any), we can post material to the bulletin board or mail out to the course list.
    - Just let me know if there is a problem

Future week's Seminars

Topic Areas for Seminars

- Feb 2nd: Programming Languages -
- Feb 9th: Analysing Application Performance, Debugging, & Testing-
- Feb 23rd: Synthesis and Compilation-
- March 9th: Operating Systems-
- March 16th: Computing Architectures (SoC vs the Cloud)-
- March 23rd: Computing System Design-
- March 30th: Reliability and DFT, Silicon Debug, Verification-
- April 6th: Security-
- April 13th: Computing Technology-
- ???: Other Computer Architecture lecture (SoC vs the Cloud)-

Student's expertise for potential project collaborations

Student Expertise for Potential Projects

- Sergey: O/S Scheduling (contention in multicore/clusters)
- Sepher: Image Processing
- Farnaz: CAD/scalable placement
- Kevan: H/W support for O/S scheduling on SMP architectures
- Eric: Architecture and O/S support for O/S scheduling
- Zia: Architecture and O/S support for O/S scheduling
- Jian: photo-organics; energy harvesting
- Frank: 3D rendering
- Tian: Computer Vision and Machine Learning
- Kyle: Post-silicon Debug
- Reza: laser optics scanners: image processing & networking

Graph Theory Basics

Graph Theory Basics

- Graph theory classes offered at SFU:
  - Dr. Bojan Mohar, Math Dept:
  - Math 345 (Graph Theory) and Math 800/820 (Topological Graph Theory)
- It is the study of graphs
  - Used in mathematics and computer science to model pairwise relations within a set

Graph Theory Basics

- Key components:
  - The set of ***Nodes*** or ***Vertices*** representing the objects
  - The ***Edges*** that connect the nodes represent relationships between the respective nodes.

Graph Theory Basics

- Graphs may be:
  - Undirected graphs imply symmetric relationships
  - Directed graphs imply asymmetric relationships

Graph Theory Basics

- Graphs may be:
  - Mixed Graph: some edges may be directed and others may be undirected

Graph Theory Basics

- Graphs may be:
  - Graphs may be defined to allow loops and/or multiple edges from a vertex
  - Simple Graph: an undirected graph with no loops and no more than one edge between any two nodes

Graph Theory Basics

- Graphs may be:
  - Weighted graph: a number (weight) can be assigned to edges
  - Regular graph: each vertex has the same degree (same number of neighbours)

Graph Theory Basics

- Graphs theory vs computer science:
  - Graph Theory Tree: An undirected graph where any two vertices are connected via exactly one simple path
  - Computer Science Tree: Same as graph theory definition accept a directed graph (a directed tree in graph theory)
    - Binary Trees: any node that is not a leaf has at most 2 children

Graph Theory Basics

- A Forest: a disjoint union of trees
- Directed Acyclic Graph: a directed graph with no cycles
  - Unlike a directed tree there maybe more than one simple path to a vertex

Graph Theory Basics

- Directed Acyclic Graphs:
  - Common method of representing algorithms and circuits
    - Common terminology includes :
      - Root
      - Leaf
      - Parent
      - Child
      - Subgraph

Programming Models: How we picture the problem.

Programming Models – how we picture the problem

- Sequentially – “To Do lists”
  - How do we get parallelism

Programming Models – how we picture the problem

- Sequentially – “To Do lists”
  - How do we get parallelism

Programming Models – how we picture the problem

- Dataflow/Object-oriented
  - How do we get parallelism

Programming Models – how we picture the problem

- Constraint based:
  - How do we get parallelism

Programming Models – how we picture the problem

- Other options???

- Note: people commonly associate a language with the model
  - Provides a definitive representation
- However, languages have a defined syntax.
  - Individuals should be able to create new languages to map to a programming model (if they want to)

This Week's Papers

Programming Models – how we picture the problem

- G. Kahn's: "The semantics of a simple language for parallel programming." Proceedings of IFIP, 1974.

- **Key contribution:** A system model where concurrent processes communicate via unidirectional FIFO's. Well suited to data-intensive applications and highlighting parallelism.

Programming Models – how we picture the problem

- G. Kahn's: "The semantics of a simple language for parallel programming." Proceedings of IFIP, 1974.

- **Other details:**
  - Each process is itself sequential
  - Read operations are blocking (process stalls when there is no data)
  - Write operations are non-blocking

Programming Models – how we picture the problem

- G. Kahn's "The semantics of a simple language for parallel programming" Proceedings of IFIP, 1974.

- **Key limitations:**
  - Assumes that the internal links have unbounded capacity.
  - Kahn process networks are deterministic (the input data's path is **NOT** determined by the order of execution of the processes)

Programming Models – how we picture the problem

- E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, "Quantitative System Performance: Computer System Analysis Using Queueing Network Models", 1984. Only read Chapter 1 (the entire book is open source)

- **Key contribution:** Systems are modelled as a network of queues that enable analytical evaluation of system operations

Programming Models – how we picture the problem

- E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, "Quantitative System Performance: Computer System Analysis Using Queueing Network Models", 1984. Only read Chapter 1 (the entire book is open source)
- **Other details**:
  - Key components are service centres (which represent computing resources) and customers (which represent users and transactions
  - Parameters (such as workload intensity, service demand, etc) allow analytical evaluation of performances measures (eg throughput, utilization, etc)

Programming Models – how we picture the problem

- E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, "Quantitative System Performance: Computer System Analysis Using Queueing Network Models", 1984. Only read Chapter 1 (the entire book is open source)

- **Key limitation:** Does not map well to the complex, non-deterministic communication topologies currently available.

Programming Models – how we picture the problem

- B. Khailany, W. Dally, U. Kapasi, P. Mattson, J. Namkoong, J. Owens, B. Towles, A. Chang, and S. Rixner, "Imagine: Media Processing with Streams," IEEE Micro, 2001.

- **Key Contribution**: Works directly on the Stream Programming Model, unlike SIMD, vector processors using stream buffers.

Programming Models – how we picture the problem

- B. Khailany, W. Dally, U. Kapasi, P. Mattson, J. Namkoong, J. Owens, B. Towles, A. Chang, and S. Rixner, "Imagine: Media Processing with Streams," IEEE Micro, 2001.

- **Other Details**:
  - Data modelled as streams (“a sequence of similar elements”)
  - Computations expressed as kernels, they consume input streams and produce output streams
  - CUDA, etc representation to create multiple threads of execution based on kernels and streams

Programming Models – how we picture the problem

- B. Khailany, W. Dally, U. Kapasi, P. Mattson, J. Namkoong, J. Owens, B. Towles, A. Chang, and S. Rixner, "Imagine: Media Processing with Streams," IEEE Micro, 2001.

- **Key Limitation**: Fixed size/definition of Stream element (can't dynamically increase in length, like a linked list); Stream model in infancy (check out CUDA, OpenCL)

Programming Models – how we picture the problem

- K. Knobe, C. Offner, "TStreams: A Model of Parallel Computation" Technical Report form HP Cambridge, 2005.

- **Key Contribution**: A model that effectively exposes different levels of program parallelism (task, data, loop, pipeline)

Programming Models – how we picture the problem

- K. Knobe, C. Offner, "TStreams: A Model of Parallel Computation" Technical Report form HP Cambridge, 2005.

- **Key Limitation**: Very limited types: one data structure, one control structure, and one operation:

  "You cannot add 2 and 2 in Tstreams."

  Therefore not an efficient programming model depending on what you want to do.

Programming Models – how we picture the problem

- E.A. de Kock, G. Essink, W. Smits, R. van der Wolf, J.-Y. Brunei, W. Kruijtzer, P. Lieverse, and K. Vissers, "YAPI: application modeling for signal processing systems," DAC 2000, pps 402-405.

- **Key Contribution**: Extends Kahn Process networks to support non-deterministic events and decouples data types used for communication and computation

Programming Models – how we picture the problem

- E.A. de Kock, G. Essink, W. Smits, R. van der Wolf, J.-Y. Brunei, W. Kruijtzer, P. Lieverse, and K. Vissers, "YAPI: application modeling for signal processing systems," DAC 2000, pps 402-405.
- **Other Details:**
  - Uses a “probe” and assumes communication through unbuffered channels
  - Two communicating processes must complete their intercommunication actions at the same time
  - Probes indicate if a process is stalled because it cannot complete its communication action
  - Probes can be used for runtime channel selection to support non-deterministic events

Programming Models – how we picture the problem

- E.A. de Kock, G. Essink, W. Smits, R. van der Wolf, J.-Y. Brunei, W. Kruijtzer, P. Lieverse, and K. Vissers, "YAPI: application modeling for signal processing systems," DAC 2000, pps 402-405.

- **Key Limitation:** Does not guarantee deadlock free systems

Programming Models – how we picture the problem

- **Other interesting work in Dataflow**:
  - Dataflow Process Networks:
    - E. Lee and T. Parks. “Dataflow Process Networks.” Proceedings of the IEEE, 83(5):773-799, May 1995.
    - A special case of Kahn process networks, where processes are mapped as atomic actors that only fire (ie are scheduled) when input data has been available and does not stall.
    - Still assumes unbounded channels and determinism