University of Illinois at Chicago

Department of Computer Science

Assessing Performance of Software Defined Radios on Multicore Hardware

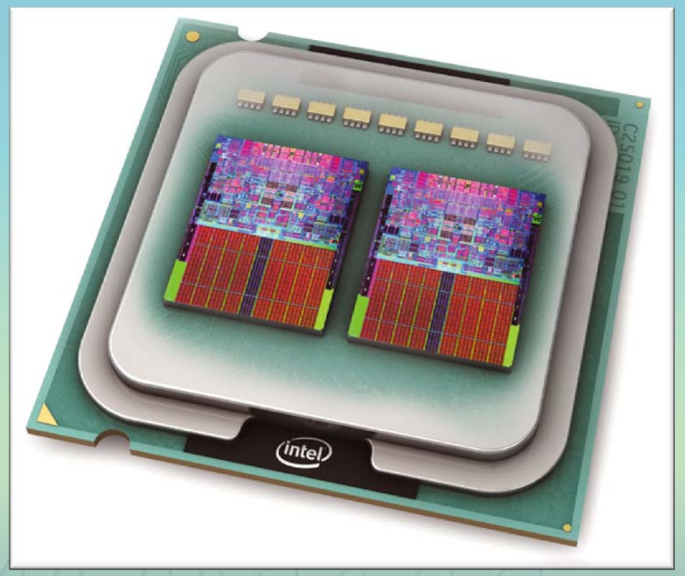
Nick Green, Ugo Buy
Dept. of Computer Science
University of Illinois
Chicago, Illinois

Redge Bartholomew
Rockwell Collins
Cedar Rapids, Iowa

12th June 2013

What is Multicore?

- Traditional, single core, processors use a serial pipeline
- The past: vast performance increases via increased transistor count
- The present: Single-core performance has plateaued
  - Instead, add multiple ‘cores’ on a chip to increase performance
  - Not a free lunch–your software must be multicore aware to exploit hardware capabilities



Project Aims

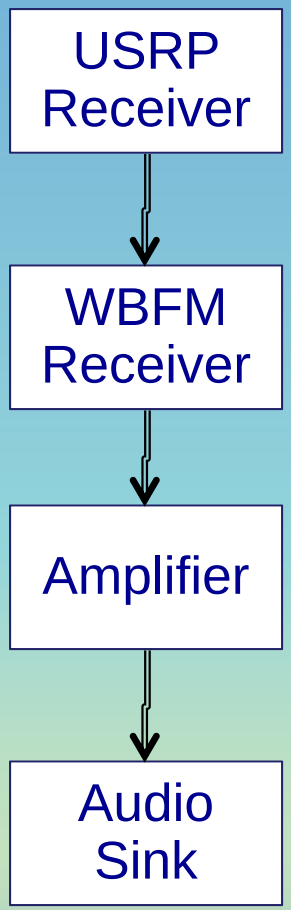
- Multicore is clearly the way forward, but there are some questions when it comes to radios…
  - Can an SDR benefit from execution on multicore h/w?
  - Can SDR functionality be extended to make use of multicore features?
  - Can multicore adversely affects performance?
  - Is it worth using multicore hardware for SDR applications?
- Our goal: Study empirically SDR performance on stock multicore hardware (e.g., 4-core PC running Linux)

Experiment Setup

- Ettus USRP1 WBX as the SDR transceiver
- The PC host setup (connected via USB) consists of various version of Ubuntu and multicore Intel CPUs
- Two host frameworks were evaluated:
  - OSSIE – JTRS compliant framework from Virginia Tech
  - GNU Radio – A mature radio framework with USRP support

SDR Software Architecture

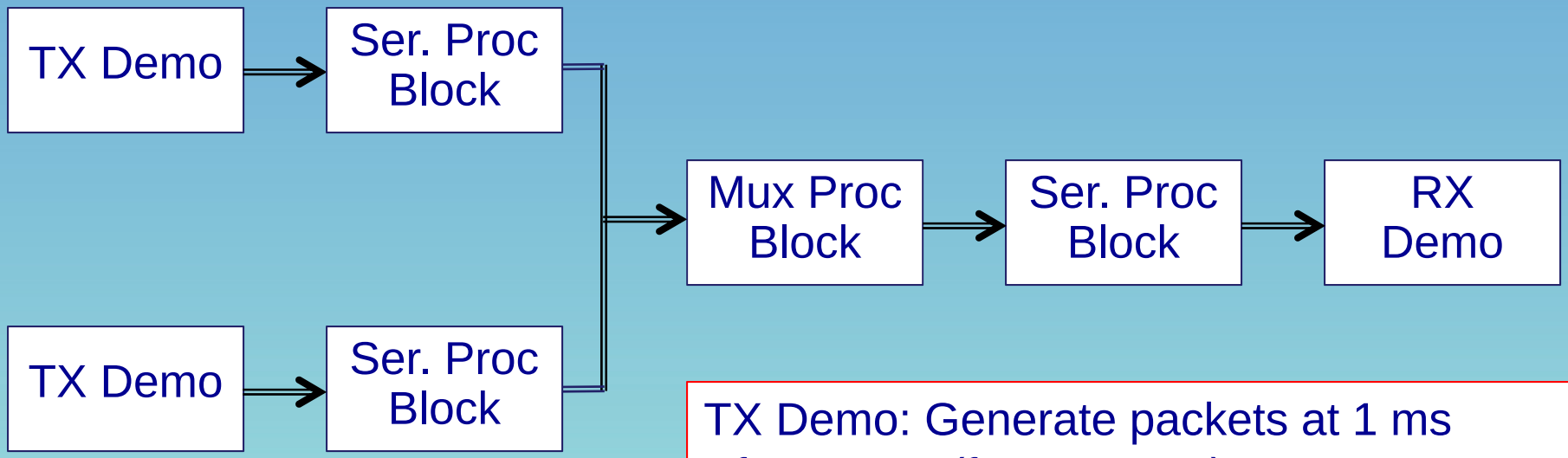
- Both OSSIE and GNU Radio are built on a graph based framework
- An application consists of source, destination and transform nodes
- Example – Source node receives a signal from the USRP, transformed (e.g. by demodulation), and then played on the PC speaker
- Important observation – Each node can run in their own software thread



OSSIE Experiments

- Used OSSIE to simulate SDR execution, with each node having its own thread
- Created set of simple graphs with custom '*processor blocks*', that perform simulated processor intensive workload (i.e., Floating-Point Divisions)
- Workload was varied over a number of components and graph throughput was recorded

Example of component configuration



TX Demo: Generate packets at 1 ms frequency (from OSSIE)

Serial Proc Block: repeat FP divisions

Mux Proc Block: same as above + multiplexing

RX Demo: Receive packets at 1 ms frequency (from OSSIE)

Empirical Data (many software threads)

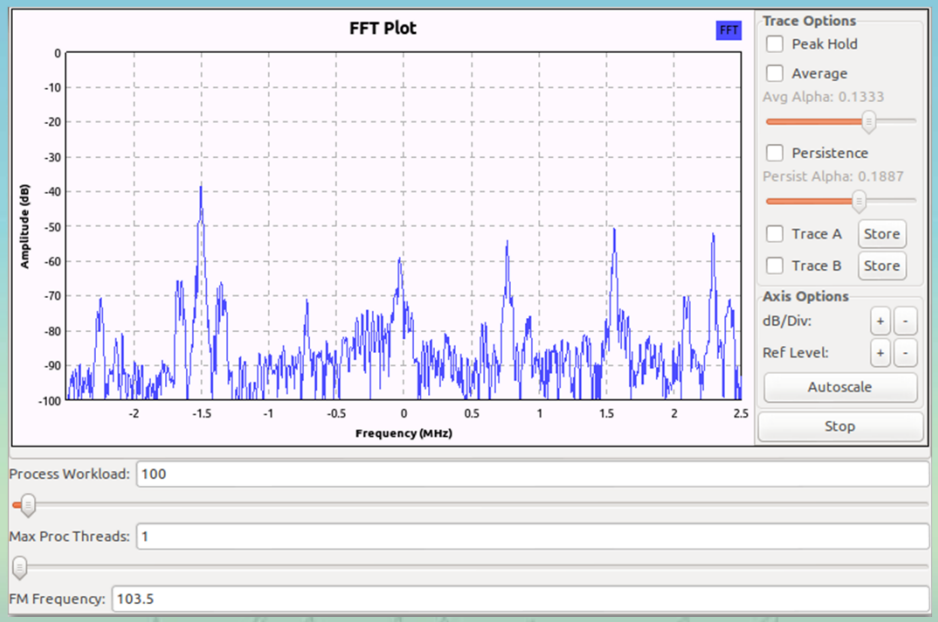
- Sample data for graph with multiplexing element (other graphs show similar behavior)

N1 (iter.)	N2 (iter.)	Mux (iter.)	N4 (iter.)	Dual+ (ms)	Mono (ms)	Mono+ (ms)	Dual+ / Mono	Dual+ / Mono+	Mono+ / Mono
1	1	1	1	1.20	1.20	1.20	1.00	1.00	1.00
1	1	1	1000	109	110	110	0.99	0.99	1.00
1	1	1	5000	548	548	549	1.00	1.00	1.00
1	1	5000	5000	547	1095	1088	0.50	0.50	0.99
1000	1000	5000	1000	655	876	873	0.75	0.75	1.00
1000	1000	1	1	110	220	218	0.50	0.50	0.99
1000	1000	1	5000	548	781	765	0.71	0.72	1.00
5000	5000	5000	5000	1084	2153	2176	0.50	0.50	1.00

+ Signifies hyper-threading being enabled

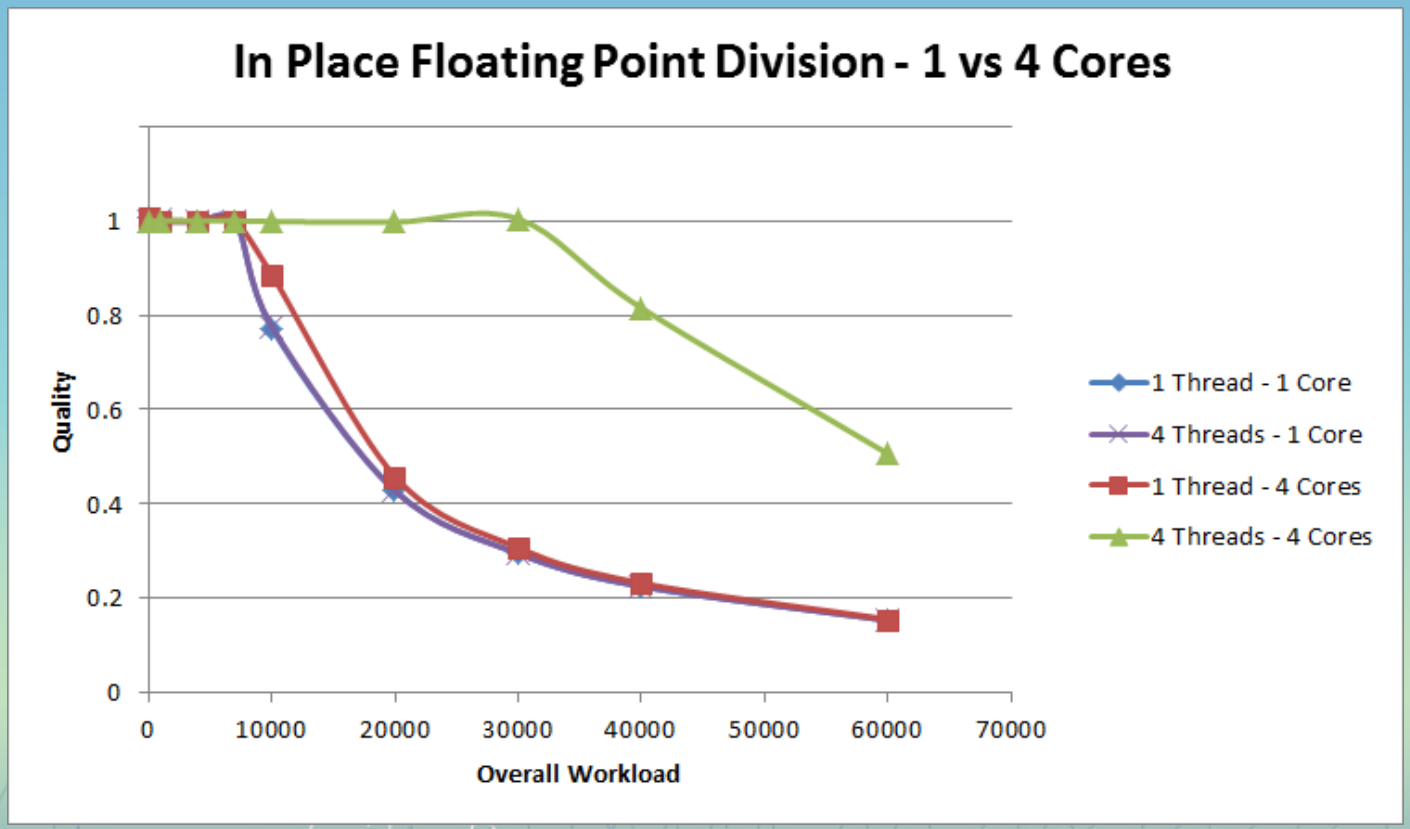
GNU Radio Experiments

- Created graphs in GNU Radio to receive transmissions via a USRP
- Graphs use a processor block and performing some processing (performing floating point division)
- Created a test dashboard to vary parameters



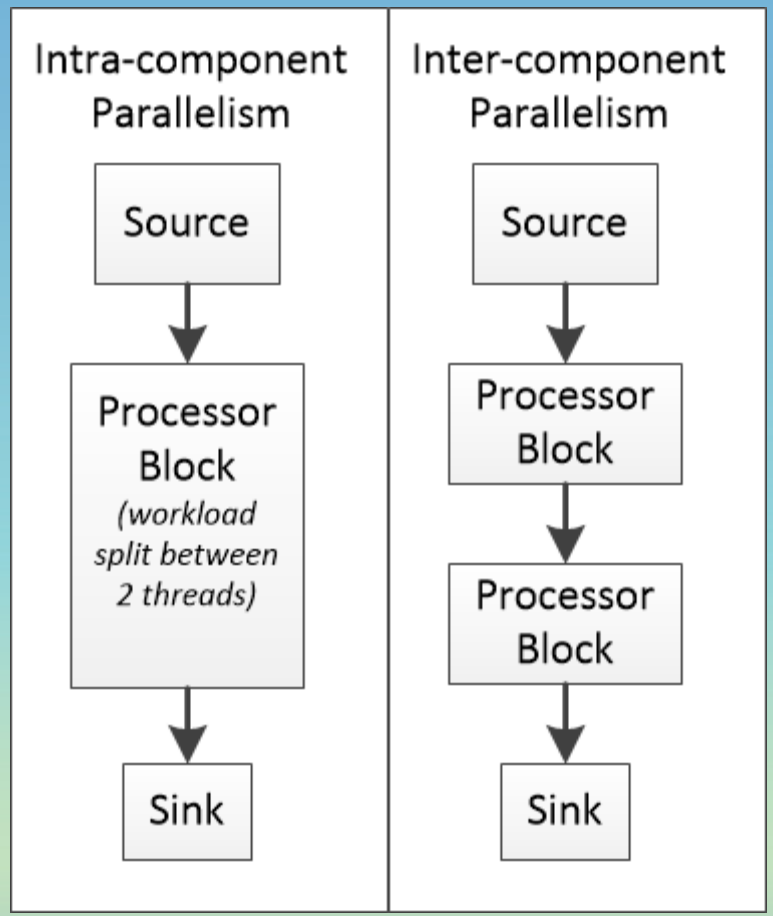
Empirical Data

- Measuring signal quality during FM reception where quality is measured by percentage of received packets



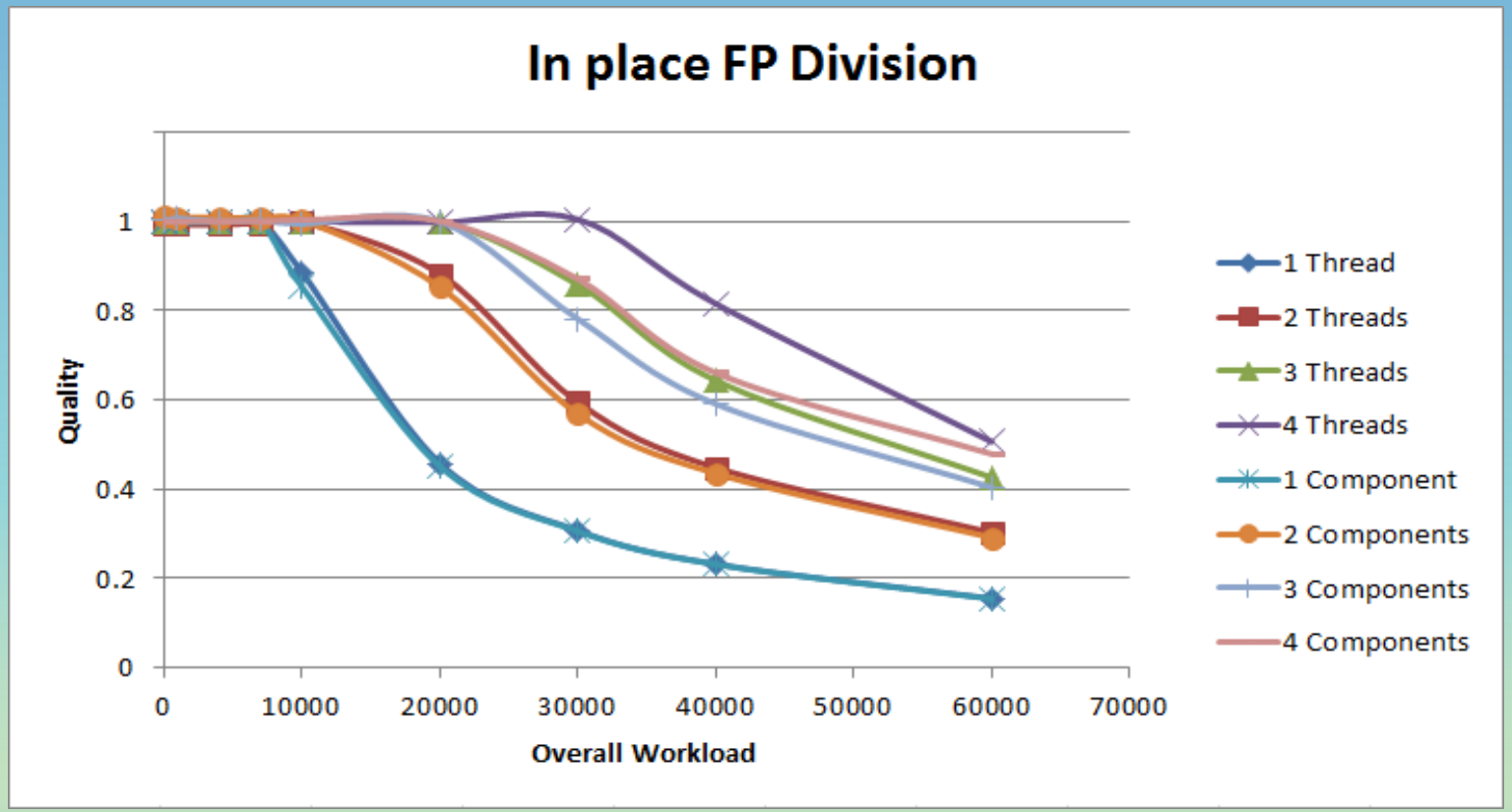
Inter vs Intra Component Parallelism

- First simulation: Performance can be enhanced with explicit concurrency within a component
- But with OSSIE, parallelism at component level also gives gains
- We can split work done by Processing Block into multiple blocks (architectural advantage)



Empirical Data (Threading Mode)

- Measuring signal quality (percentage of received packets)



Thread: Threads in one component

Component: Components each with one thread

GNU Radio Observations

- Unexpected degradation in case of inter-component parallelism vs. intra-component
- Intuition would say there is an overhead with inter-component transitions. However, after profiling, the culprit was OUR processing block!
- High floating point division workload did poorly when processing was split among components

Processor Block

- Processor Block operations show some interesting performance properties. More experiments were run
- Performance varied between processing type, typically multicore offering vast gains over single

Simulation Operation	Observation
In place floating point division	Good performance with intra-component parallelism, degraded with inter-component
Integer add/subtract	Optimal performance gains achieved
Integer array reversal	Substantial gains limited to 3 thread on quad core
Dynamic memory alloc.	Severely degraded when multiple threads used for allocation

Summary

- Multicore is a good thing!
- In the presented cases, multicore can offer instant benefits just by using the inherent multithreaded frameworks
- But performance varies based on what is being done and can be fragile
- Our test dashboard was very helpful in finding what worked, what didn't, and limitations

Thank you very much!

Questions:

- Nick Green, ngreen21@uic.edu
- Ugo Buy, buy@cs.uic.edu
- Redge Bartholomew rgbartho@rockwellcollins.com