Programming and Language Challenges for Multi-core and Beyond

RK Shyamasundar
Tata Institute of Fundamental Research
shyam@tifr.res.in

(with R Govindarajan and S. Vadhiyar)
Agenda

• Performance, Productivity
  – Software Crises: Past and Emerging..

• CPU Technology Trends
  – 3 Walls for Unicore Performance
  – Advent Of MultiCores

• Performance and Productivity for MultiCore and Beyond: Challenges
  – Driving Programming Language Evolution to meet Hardware Evolution

• Ongoing Work
  – Work at Institutions in India
  – Work at IISc
  – Work at TIFR
Software Crisis

“To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”

*Edsger Dijkstra, 1972, Turing Award Lecture*
First Software Crisis: 1960s-70s

• **Problem:** Assembly Language Programming
  – Computers could handle larger more complex programs Needed to get Abstraction and Portability without losing Performance

• **Solution:** High-level languages for von-Neumann machines
  – FORTRAN and C Provided “common machine language” for uniprocessors
  – Single memory image. Single flow of control
Second Software Crisis: 1980s and ’90s

• **Problem:** Inability to build and maintain complex and robust applications requiring multi-million lines of code developed by hundreds of programmers
  – Needed to get Composability, Malleability and Maintainability
  – High-performance was not an issue • left to Moore’s Law

• **Solution:**
  – Object Oriented Programming - C++, C# and Java
  – Better tools:
    • Component libraries, Purify
  – Better software engineering methodology
  – Programming in the Large Vs Small
    • Design patterns, specification, testing, code reviews
Parallel Programming Models

• **Data parallel model.**:
  – same or similar computations are performed on different data repeatedly. Image processing algorithms that apply a filter to each pixel are a common example of data parallelism.
  – OpenMP is an API that is based on compiler directives that can express a data parallel model.

• **Task parallel model.** independent works are encapsulated in functions to be mapped to individual threads, which execute asynchronously. Thread libraries (e.g., the Win32 thread API or POSIX* threads) are designed to express task-level concurrency.

• **Hybrid models.** Sometimes, more than one model may be applied to solve one problem, resulting in a hybrid algorithm model. A database is a good example of hybrid models. Tasks like inserting records, sorting, or indexing can be expressed in a task-parallel model, while a database query uses the data-parallel model to perform the same operation on different data.
Methodology for Parallelization

• Basis for enhancement of performance via parallel processing basically lies with *decomposition techniques*. Dividing a computation into smaller computations and assigning these to different processors for execution are two key steps in parallel design. Two of the most common decomposition techniques

• **Functional decomposition** is used to introduce concurrency in the problems that can be solved by different independent tasks. All these tasks can run concurrently.

• **Data decomposition** works best on an application that has a large data structure. By partitioning the data on which the computations are performed, a task is decomposed into smaller tasks to perform computations on each data partition. The tasks performed on the data partitions are usually similar. There are different ways to perform data partitioning: partitioning input/output data or partitioning intermediate data.
How much do we gain in performance

- **Amdahl’s law** provides the theoretical basis to assess the potential benefits of converting a serial application to a parallel one. It predicts the speedup limit of parallelizing an application:
  \[
  T_{\text{parallel}} = \left\{ \left( 1 - P \right) + \frac{P}{N} \right\} \times T_{\text{serial}} + O_{\text{N}}
  \]
  where $T_{\text{serial}}$: time to run an application in serial version, $P$: parallel portion of the process, $N$: number of processors, $O_{\text{N}}$: parallel overhead in using $N$ threads. We can predict the speedup by looking at the scalability:
  - \[ \text{Scalability} = \frac{T_{\text{serial}}}{T_{\text{parallel}}} \]
  - We can get the theoretical limit of scalability, assuming there is no parallel overhead:
    - \[ \text{Scalability} = \frac{1}{\left\{ \left( 1 - P \right) + \frac{P}{N} \right\}} \]
    - When $N \to \infty$, Scalability $\to \frac{1}{1 - P}$
Challenges

• Decomposing applications to units of computation that can be executed concurrently.

• Continuing to meet the above as the number of processors increase – Scalability.
  • May not be enough concurrent jobs to keep the CPUs busy
  • Shared work queues become a bottleneck
  • Need to discover intrinsic parallelism (finding finer-grained parallelism is a challenge).
Parallel Programming Languages

• Parallel programming languages have not been able to address the issues of natural specification of parallel algorithms and handling of the architectural features to achieve performance concurrently.

• In other words, there is a large gap between programming languages that are too low level, requiring specification of many details that obscure the meaning of the algorithm, and languages that are too high-level, making the performance implications of various constructs unclear.

• However, in the context of sequential computing standard languages such as C or Java do a reasonable job of bridging such a gap.

• Main challenge is to bridge the gap between specification and realization (implementation) that would preserve natural specification of programs and at the same time enable realization of efficient programs on different architectures.
Oblivious about the Processors

• Solid boundary between Hardware and Software
• Programmers don’t have to know anything about the processor
• High level languages abstract away the processors
  – Ex: Java bytecode is machine independent
  – Moore’s law does not require the programmers to know anything about the processors to get good speedups
• Programs are oblivious to the processor
  – works on all processors
  – A program written in ’70 using C still works and is much faster today
  – This abstraction provides a lot of freedom for the programmers
Emerging Software Crisis: 2002 – 20??

- **Problem:** Sequential performance is left behind by Moore’s law
  - Needed continuous and reasonable performance improvements
    - To support new features to support larger datasets
    - While sustaining portability, malleability and maintainability without unduly increasing complexity faced by the programmer
  - Critical to keep-up with the current rate of evolution in software

Saman Amarasinghe, *6.189 Multicore Programming Primer, January (IAP) 2007*. (Massachusetts Institute of Technology: MIT OpenCourseWare).
Programmers View of Memory

Till 1985
- 386, ARM, MIPS, SPARC
- 1-10 MHZ
- ext mem Cyc – 1 cycle
- Most Inst -- 1 Cycle
- Lang. C -ops- 1 or 2 Cycles

On chip computation (Clk Sp) – sped up faster (till 2005) than off-chip communication (with memory) as feature sizes shrank
- Gap filled by transistor budgets on caches - filled the mismatch till 2005
- caches, deep-pipelining with bypasses, superscalar -- burned power to keep the illusion in tact till 2005 – Tipping point

Scraping single CPU Pentium

6/3/2010

Indo -Spain Workshop
Is this an Issue?

- If we don’t get any more performance
  - Stuck at the current functionality
  - Software will move from a frontier field with a lot of innovation to a mature one with lot of tweaking and optimization
- If the complexity of programming is too high
  - Only a few can be programmers
  - Hard to add innovative and novel functionality
- If the software becomes more unreliable
  - Already unreliable.
  - Scary consequences if even more unreliable
- If software becomes too costly to produce
  - Only a few software companies will be left in the world
  - Less innovations
Innovations in Architecture

Architecture is nothing but a way of thinking about programming
Edsgar Dijkstra

- 1943 ENIAC has accumulators operating in parallel
- 1965 Dijkstra describes critical regions problem
- 1966 Mike Flynn introduced SIMD/MIMD classification
- 1967 parallelizing FORTRAN compiler
- 1967 Amdahl describes what became the Amdahl’s law
- 1974 conditional critical regions and monitors introduced
- 1978 Tony Hoare introduces CSP
  .....
Where shall we search for the solution?

- Advances in Computer Architecture
- Advances in Programming Languages
- Advances in Compilers
- Advances in Tools
Computer Architecture: Uniprocessors

• General-purpose unicores have stopped historic performance scaling
  – Power consumption
  – Wire delays
  – DRAM access latency
  – Diminishing returns of more instruction-level parallelism
"Moore's Gap"

Performance (GOPS)

1. Diminishing returns from single CPU mechanisms (pipelining, caching, etc.)
2. Wire delays
3. Power envelopes
The March to Multicore: Uniprocessor Performance (SPECint)
Power Consumption (watts)
Power Efficiency (watts/spec)
Range of a Wire in One Clock Cycle

- 400 mm² Die
- From the SIA Roadmap
DRAM Access Latency

- Access times are a speed of light issue
- Memory technology is also changing
  - SRAM are getting harder to scale
  - DRAM is no longer cheapest cost/bit
- Power efficiency is an issue here as well

Images removed due to copyright restrictions.

<table>
<thead>
<tr>
<th>μProc</th>
<th>DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>60%/yr. (2X/1.5yr)</td>
<td>9%/yr. (2X/10 yrs)</td>
</tr>
</tbody>
</table>

Saman Amarasinghe, 6.189 Multicore Programming Primer, January (IAP) 2007. (Massachusetts Institute of Technology: MIT OpenCourseWare).

6/3/2010 Indo-Spain Workshop
Uniprocessors Don’t solve the problem

- A very active research effort on parallelization died down in the early ’90s, Why?
  - Everyone assumed uniprocessors will provide sustainable performance gains
  - Most customers were satisfied by uniprocessor gains
  - Small community of performance-at-any cost
    - They were not willing to compromise on performance
    - Willing to sacrifice bodies to get the best performance

- Now...
- Uniprocessors no longer scaling
- **But don’t have to contend** with uniprocessors
Diminishing Returns

• The ’80s: Superscalar expansion
  – 50% per year improvement in performance
  – Transistors applied to *implicit* parallelism
    • pipeline processor (10 CPI --> 1 CPI)

• The ’90s: The Era of Diminishing Returns
  – Squeaking out the last implicit parallelism
    • 2-way to 6-way issue, out-of-order issue, branch prediction
    • 1 CPI --> 0.5 CPI
  – performance below expectations
  – projects delayed & canceled

• The ’00s: The Beginning of the Multicore Era
  – The need for Explicit Parallelism
Multicore Model of Memory

- Will scale to 2, 4, 8, ... processors
- Ultimately Shared Memory Bottleneck -- 1024 ?? processors

Programming Languages without Uniform Memory Model

What about temporal order of writes by different observers?

- Shared Memory Algorithms parameterized by memory model
- Analysis? Refactoring

6/3/2010 Indo -Spain Workshop
Unicores are on the verge of extinction
Multicores are here

- MIT Raw
  - 16 Cores
  - Since 2002
- Intel Montecito
  - 1.7 Billion transistors
  - Dual Core IA-64
- Intel Tanglewood
  - Dual Core IA-64
- Intel Pentium D
  - (Smithfield)
- Intel Dempsey
  - Dual Core Xeon
- Intel Pentium Extreme
  - 3.2GHz Dual Core
- Intel Yonah
  - Dual Core Mobile
- AMD Opteron
  - Dual Core
- Sun Olympus and Niagara
  - 8 Processor Cores
- IBM Cell
  - Scalable Multicore
- IBM Power 4 and 5
  - Dual Cores Since 2001
- IBM Power 6
  - Dual Core

Saman Amarasinghe, 6.189 Multicore Programming Primer, January (IAP) 2007. (Massachusetts Institute of Technology: MIT OpenCourseWare).

6/3/2010 Indo-Spain Workshop
Multicores are Here

Saman Amarasinghe, 6.189 Multicore Programming Primer, January (IAP) 2007. (Massachusetts Institute of Technology: MIT OpenCourseWare).

6/3/2010 Indo -Spain Workshop
CPU Technology Trends Summary

• Microprocessor: Power Wall + Memory Wall + ILP Wall = Brick Wall
  ⇒ End of uniprocessors and faster clock rates
  ⇒ Every program(mer) is a parallel program(mer),
     Sequential algorithms are slow algorithms

• New “Moore’s Law” is 2X processors or “cores” per socket every 2 years, same clock frequency
  (parallel more power efficient: $W \approx CV^2F$)

  – Conservative: 2007, 4 cores, 2009 8 cores,
    2011 16 cores for embedded, desktop, & server
  – Sea change for HW and SW industries since changing programmer model, responsibilities
  – HW/SW industries bet farm that parallel successful
Novel Opportunities in Multicore

• Parallelism, i.e. multiprocessors, are not new
• But, not your same old multiprocessor problem
  – How does going from Multiprocessors to Multicores impact programs?
  – What changed?
  – Where is the Impact?
    – Communication Bandwidth
    – Communication Latency
Communication Bandwidth

- How much data can be communicated between two cores?

- What changed?
  - Number of Wires
  - Clock rate
  - Multiplexing

- Impact on programming model?
  - Massive data exchange is possible
  - Data movement is not the bottleneck
    → processor affinity not that important

32 Giga bits/sec  ~300 Tera bits/sec
Communication Latency

- How long does it take for a round trip communication?

- What changed?
  - Length of wire
  - Pipeline stages

- Impact on programming model?
  - Ultra-fast synchronization
  - Can run real-time apps on multiple cores

50X

~200 Cycles ~4 cycles
New way of Thinking about Architectures

Then

• Provide performance by hardware tricks
• Maintain compatibility
• Cannot rely on the compiler (Itanium debacle)

Now

• No simple tricks left; gain very low
• Need to work with compiler writers
• Architecture help for compiler writers
• If not done properly program productivity suffers
Beyond Diminishing Returns

Madison Itanium2
Released in 2002

Cache System

L3 Cache

Photo courtesy Intel Corp.
Aggressive movement by Industry to Multicore 2002

Multicore satisfies three properties

- Single chip
- Multiple distinct processing engines
- Multiple, independent threads of control (or program counters - MIMD)
Problem with Buses

Inherent architectural bottlenecks:
- No scalability
- Power inefficiency
- Primitive programming model

Timeline:
- Present Day
- 2006, 2007, 2010

Performance vs. Time Graph:
- Dual Cores
- Quad Cores
- 16 Cores
- n Cores
Opportunities

Technology
• 20MIPS CPU in 1987 to billion Transistor CPU

New application domains
• CPUs: too fast for desktop office apps
• Redefine a “general purpose processor” single-chip
  – media station for streaming data
  – compute server farm
  – handheld - a combination cell phone, camera, PDA, MP3 player, video player
  – a 10G router While running the gamut of existing desktop binaries
Tiled Multicore

Divide the silicon into an array of identical tiles

Combine processor with a switch (router) on each tile

Core + Switch = Tile
MIT RAW Chip (Anant Agarwal)

`.18 micron process, 16 tiles, 425MHz, 18 Watts (vpenta)`

Demonstrated in 2002
Beyond Multicore (1)

• While it is expected that transistor densities will continue to improve, it is also expected that starting with the 22nm node power densities, at constant frequency and switching factor (the percentage of transistors that switch per cycle), will start to increase or, more likely, frequencies will have to start to come down.
• While this does not signal the end of the multi-core era, it does imply that there is yet another gap that must be filled if the industry is to deliver performance growth at historical rates.
• Further adding to the concerns is that it seems likely that the delay of scaled wires will start to increase at about the same time, which implies that additional engineering will be required to make a core run at even the same frequency when migrated from one technology to the next.

(From Peter Hofstee)
Multicore and Beyond (2)

- We must therefore seek methods of further improving efficiency to deliver growing performance at a constant power budget with only modest improvements in power from technology with architectures that can be broadly applied. Fortunately the opportunity to deliver on this exists though it will come at the expense of requiring modifications to the algorithms and applications beyond the introduction of concurrency.
Hardware Evolution Effects on Programming

- Computation is increasingly cheap compared to communication
- More parallelism required for effective use of hardware
- Global shared memory model is less sustainable
- CPU-intensive processes need to be moved around to keep the chip cool by switching of unused parts of the chip
- Due to miniaturization, transient hardware errors (hence data errors) will become common
- Programming Language Mismatch hardware
The Multicore / Manycore Programming Challenge

Programmers cannot cope with thousands of threads and complex data flows using existing programming models.

Single Core
Single Thread
100% Serial Programming

Yesterday
6/3/2010

Multicore (2-16)
Multithread (10s – 100s)
80/20 Serial/Parallel Programming

Today
Indo-Spain Workshop

Manycore (32-100s)
20/80 Serial/Parallel Programming
Threading model breaks as complexity exceeds programmer capability

Tomorrow

1000s of HW threads in 2012
MultiCore Programming Challenges

• **Application Performance**
  – Maximize throughput and minimize response time
    • Task and data distribution and pipelining
    • Communication orchestration across cache hierarchies
    • Synchronization Issues: granularity
  – Scalability with increasing number of processors
  – Identification of performance bottlenecks

• **Programmer Productivity**
  – Simultaneous performance and functional correctness: Serious Productivity Impediment
  – Portable performance across diverse heterogeneous architectures
    • Algorithmic re-design for performance
  – Search for programming talent
  – Programming Models: MPI/Pthreads, X10/Chapel/Fortress, Map-Reduce/…

• **Software Reliability & Engineering**
  – Debugging of parallel application: deadlock, livelock, race-condition
  – Levels of testing before release
Options for Exposing Parallelism

- **Parallelism Fully Exposed**
  - Full exposure of machine details, e.g., MPI
  - Only usable by experts
  - High performance
  - Low productivity

- **Partial Exposure**
  - Limits exposure of unnecessary machine details, e.g., X10
  - Usable by larger number of programmers
  - High performance
  - Higher productivity
  - Bounds checks, ptr checks, strong typing, etc.

- **Parallelism Implicit**
  - No exposure of machine details, e.g., map/reduce
  - Usable by non-experts
  - High productivity
  - Limited domains or low performance
X10 Deployments with Bounded Resources

X10 language defines mapping from X10 objects & activities to X10 places.

X10 deployment defines mapping from virtual X10 places to physical processing elements.

Virtual level: unbounded resources per place.

Physical level: bounded resources per place.

Partial Global Address Space (PGAS)

6/3/2010 Indo-Spain Workshop
MultiCore Software Challenge

• “Every software maker out there ... has got to learn how to program parallel code ... to remain competitive.”
  — Dan Olds, Principal Analyst, Gabriel Consulting

• “This could become the biggest software remediation task of this decade.”
  — Multicore Will Induce Operational and Software Headaches, Gartner Group

• “There’s going to be a huge learning curve for developers to take on multi-threading in such a big way.”
  — Sharon Gaudin, Information Week
Enablers of Performance (1)

1. Concurrency
   - Cluster parallelism as in commodity scale-out clusters
   - Multi-core parallelism
   - Heterogeneous parallelism
Enablers of Performance (2)

2. Locality
   - Majority of the power required to execute a typical program is associated with moving the data around (Dally ... Stream Processing)
   - yet our programming languages ignore locality and treat memory as an infinite and infinitely close resource

3. Predictability
   - Vectors
   - Affinity of data
Performance: Growth

• Graphic Processors:
  – significantly increased the level of programmability and flexibility

• GPGPUs
  – highly threaded data-parallel programmable processors

• A significant no. of applications yield good performance
  – Not every application can be productively parallelized
  – Not every application will benefit from the performance improvements through locality and predictability
Opportunity for Efficiency

- When the energy per transistor switch stabilizes a much greater degree of hardware differentiation becomes economically feasible.
  - This will give reduced-function and appliance-type devices an increased advantage over systems built from general-purpose programmable hardware
- Multi-core designs no longer increase the efficiency gap compared to algorithms realized in hardware,
- Locality and predictability-aware programming and architectures may even reduce the gap somewhat,
  - Often there are three orders of magnitude (or more) of difference between a program executing a task and that same task realized directly in hardware.
- Increasing Specialization
Ongoing Work

• IIT (Kanpur, Delhi, Kharagpur, Madras, Bombay)
• CDAC – GARUDA Grid
• BARC – High Computational Cluster – Scale Out Networks
• NIC– Knowledge Networks
• Tier 2 Grid for LCG, CERN, at TIFR
• Tier 3 Grid for LCG, CERN, at DAE Units + Other Universities (Delhi, Panjab, IISc, …)
• IISc (Glimpse to follow -- R Govindarajan and Sathish Vadhiyar)
• TIFR -- Programming Lang Work
Mapping StreamIt for GPUs

- Work distribution across multiprocessors
  - GPUs have hundreds of processing pipes!
  - Exploit task-level and data-level parallelism
  - Multiple concurrent threads in SM to exploit DLP
- Identify Execution configuration: task granularity and concurrency
- Managing CPU-GPU memory bandwidth
  - Coalesced access
  - Transparent data transfer betn. CPU & GPU
- Scheduling tasks on GPUs (SMs) and CPU cores
  - Takes into account concurrency and communication

R Govindarajan, SERC, IISc

6/3/2010 Indo -Spain Workshop
Ongoing Work

- Look at other high level languages!
- Layered approach translation and execution

Look at other high level languages!
Layered approach translation and execution
Compiling OpenMP/MPI / X10

- Mapping the semantics
- Exploiting data parallelism and task parallelism
- Communication and synchronization across CPU/GPU/Multiple Nodes
- Accelerator-specific optimization
  - Memory layout, memory transfer, ...
- Performance and Scaling
Applications

Contact: Sathish Vadhiyar, SERC, IISc
Challenges

- Climate change analysis performed using multi-component climate models
- Many production systems in grids are batch systems
- Batch systems available for execution keep varying
- **Highly dynamic environment!**
- Need to coordinate different components on different batch systems
- Challenge is to sustain long climate runs on grids with multiple batch systems
Grid Middleware for Climate Modeling

- Have developed a generic grid middleware framework for execution of long-running multi-component applications on multiple distributed batch systems
- Coordinates distribution, execution and migration of climate components
- Also performs scheduling and dynamic rescheduling of components, fault-tolerance and synchronization

6/3/2010 Indo-Spain Workshop
Weather Modeling on Grids

- Critical climate events like cyclone tracking need to be simulated and visualized online by remote climate scientist.
- This provides for collaborative guidance.
- Climate data needs to be transferred from simulation to visualization site.
Challenges

• However, disk space at simulation and network bandwidths are constraints.
• Needed: adaptive grid middleware that dynamically adjusts simulation rate and output frequency based on resource constraints.
Grid Middleware for Weather Modeling

- Have developed an adaptive grid middleware for efficient online remote visualization of critical climate events
- Dynamically changes resources for simulations, and visualization output frequency

- Have used the framework to track cyclone Aila (May 2009)
- Inter-country experiments – Simulations in U.S., Visualization in India

6/3/2010 Indo-Spain Workshop
TIFR
Challenges in Affinity Driven Multi-Place Scheduling

- **Three Axes (performance, productivity, reliability)**
  - Deadlock Freedom with respect to bounded resources (Distributed work stealing)
  - Performance
    - Expose maximum concurrency
    - Load balancing across processors
    - Minimal Communication Cost
  - Space For Computation
- **Distributed Deadlock Challenges**
  - Distributed Deadlock Avoidance is Difficult!
    - Banker’s Algorithm using distributed agreement
      - Can be impractical
      - Can involve lot of communication
  - Distributed Deadlock Prevention can inhibit concurrency
  - Distributed Deadlock Detection
    - Detect deadlock at run-time and perform roll-backs
    - Can be very expensive in general
- **Space-Time Trade-off hard to optimize**
  - Maximal breadth first exploration of the DAG under space constraints
Affinity Driven Multi-Place Scheduling

- *Distributed Scheduling of Parallel Hybrid Computations*, 20th ISAAC, LNCS 5878, pp 1144-1154, Springer Verlag, full paper to appear, TCS, Dec 2009
- A Static Characterization of Affinity in a Distributed program, IEEE Int Conf. HPCC, Dalian, China, Sept 2008
May happen Parallelism Analysis

• (with Shivali Agarwal, Rajkishore Barik, Vivek Sarkar) May Happen-in-Parallel Analysis of X10 programs, PoPoPP, 2007

• Harshit Shah, RK Shyamasundar, Pradeep Varma, Concurrent SSA for Concurrent Generalized Barrier Synchronization Languages, IPDPS 2009, Rome, May 2009

• Saurabh Joshi and RK Shyamasundar, Refined MHP Analysis for X10 Programs with Clocks, under submission
Exceptions and Temporal/Spatial Error Analysis

- Backward-compatible constant-time exception-protected memory, 7th ACM SIGSOFT ESFC/FSE, pp. 274-275, 2009, Amsterdam
- Debugging PGAS Languages: Challenges and Approaches, ICS Workshop on Asynchrony in the PGAS Programming Model, ICS 09, NY, June 2009
- Static Detection of Place Locality and Elimination of Runtime Checks, in Programming Languages and Systems, 6th Asian Symposium, APLAS 2008, Bangalore, India, LNCS 5356, pp. 53-74, Springer 2008,
Barrier Synchronization with Clocks and Expressive Power

- Power of Barrier Synchronization with Clocks, under submission
- Realizing Global Non-determinism with X10, Under submission
- X10 is as powerful as Pi-Calculus, Under submission
- Generalized Dynamic Barrier Synchronization, under submission 2010
- (with Shivali Agarwal) Distributed Phase Synchronization of Dynamic Set of Processes, 28th ACM PODC, 274-275, August 2009
Cloud Computing

• Data Security
  – Biba + La Bell Padua Model + Labelling data
THANK YOU