MANCHESTER 1824

The University of Manchester

# From Transactions to Dataflow and Back Again

Mikel Luján
Advanced Processor Technologies Group
University of Manchester
http://www.cs.manchester.ac.uk/apt

MANCHESTER 1824

The University of Manchester

# From Transactions to Dataflow and Back Again

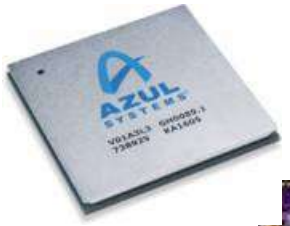
M. Ansari, C. Kotselidis, B. Khan, M. Horsnell, K. Jarvis, S. Khan, D. Goodman, C. Seaton, C. Kirkham, I. Watson & M. Lujan

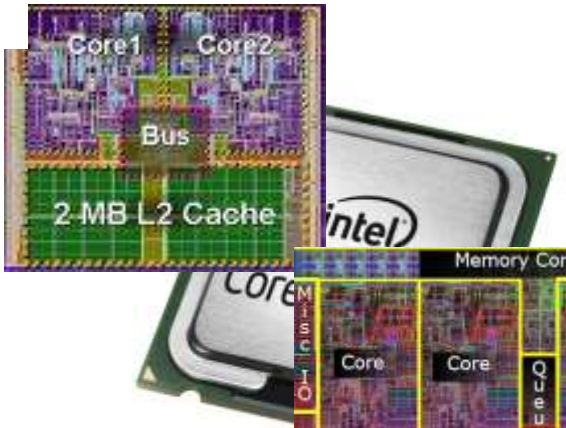
Advanced Processor Technologies Group

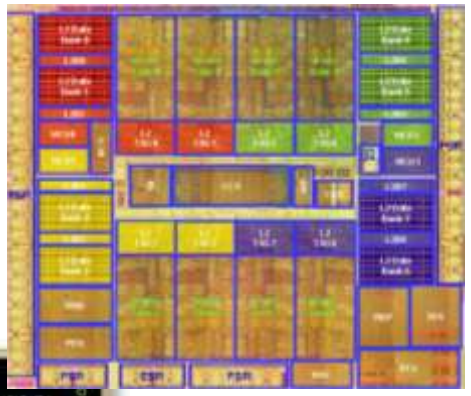
University of Manchester

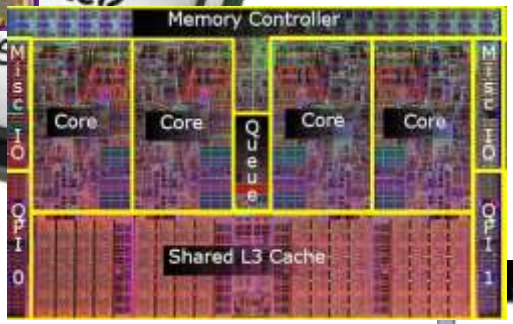
http://www.cs.manchester.ac.uk/apt

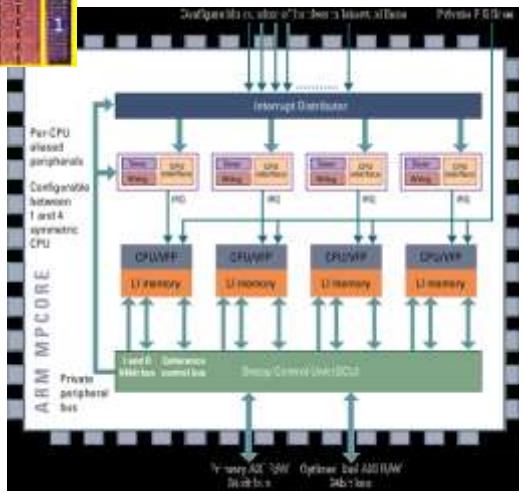
# Multi-cores


AZUL
SYSTEMS


Core1
Core2
Bus
2 MB L2 Cache
intel


Sun
solaris
ULTRASPARC T2


Memory Controller
Core
Core
Queue
Core
Core
Shared L3 Cache


Interrupt Distributor
CPU/VFP
L1 memory
ARM MPCORE

# APT Group and Manchester

Multi-Core Chips

Advanced P

Asynchronou

Networks-on-Chip

Sw/Hw/ML
*Mikel Lujan*
*Gavin Brown*
*Ian Watson*

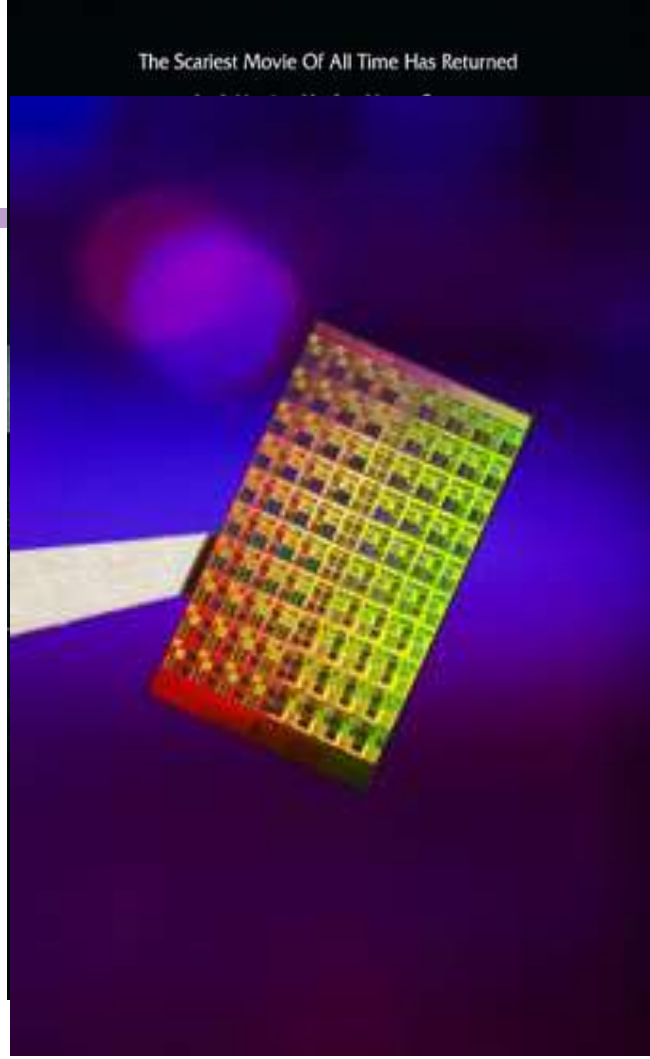
3D VLSI
*V. Pablidis*

Neural Systems Engineering
*Steve Furber*
*Jim Garside*
*Dave Lester*

*A. Rawsthorne*

# Multi-cores == Terror Movie?

- Business volume
  - Hardware $200K millons
  - Software $2K billons

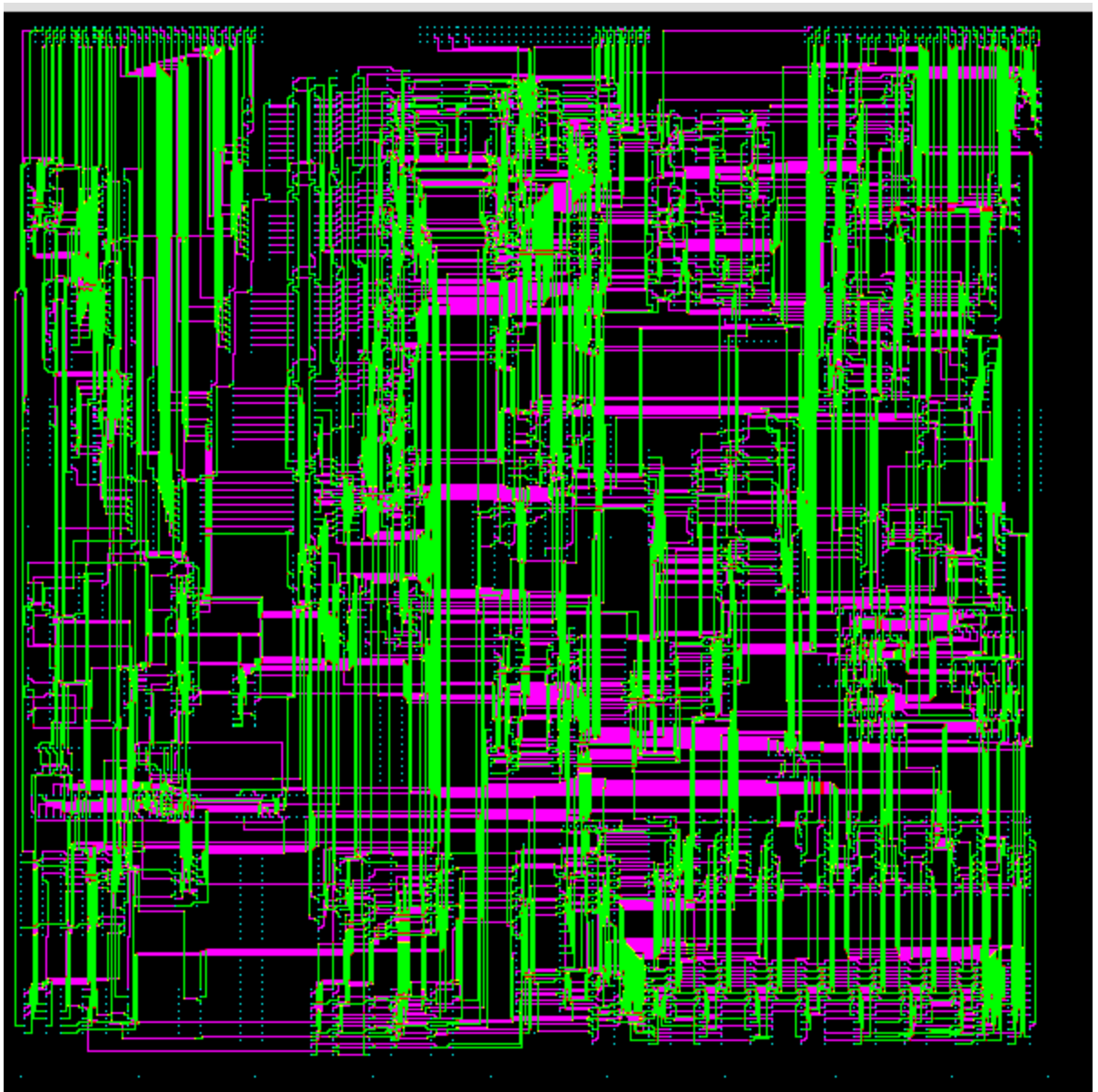
# Roadmap for today

- APT Group Intro & Need for SW/HD co-design
- Lee's algorithm
  - Understand the problem
- Parallel implementations
  - Different choices
  - Lessons
- Transactional Memory
  - Basic concept
  - Lee with transactions
- Performance analysis
- Improving performance
- Teraflux

MANCHESTER 1824

The University of Manchester

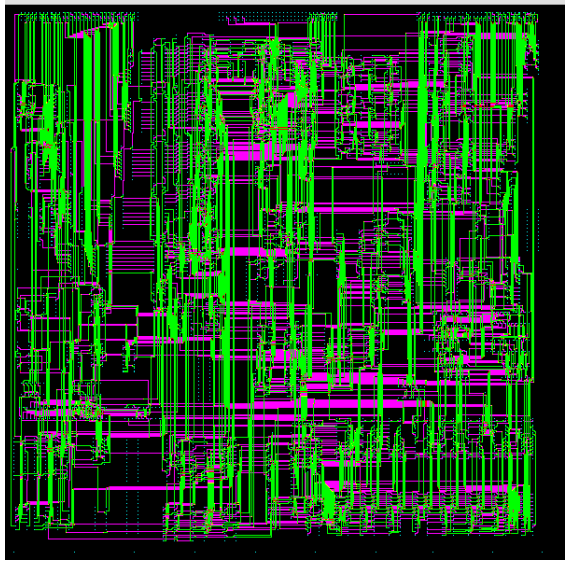
# Circuit Routing



# Definitions

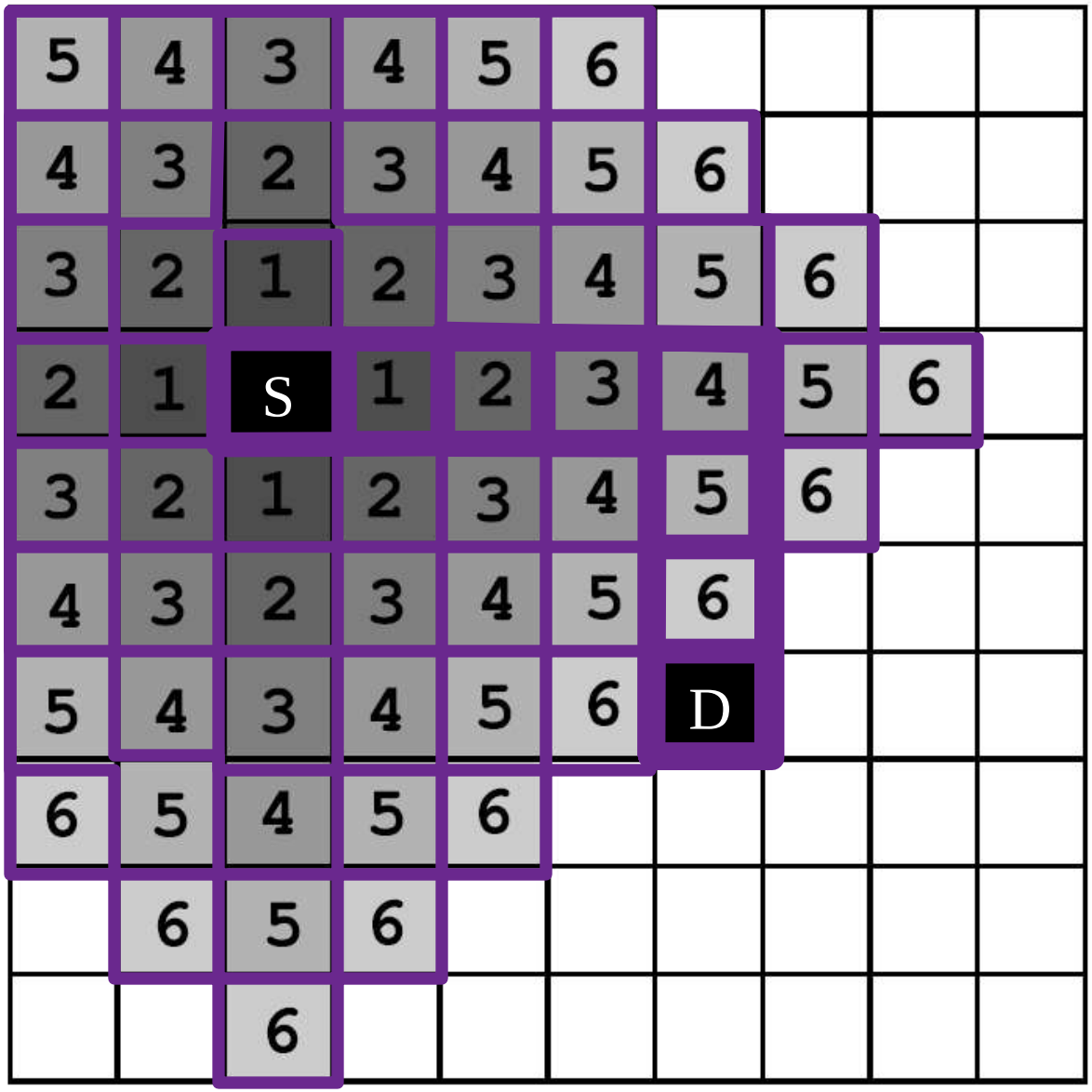
- Grid: a three dimensional
- Layer: is the combination of a conductive layer and non-conductive one
- Via: connection among the different layers
- Cell: a point in the grid
- Route: a set of contiguous cells that reach from the source cell to the destination cell
- Obstacle: one cell (or set of cells) that cannot belong to any route

# Problem definition



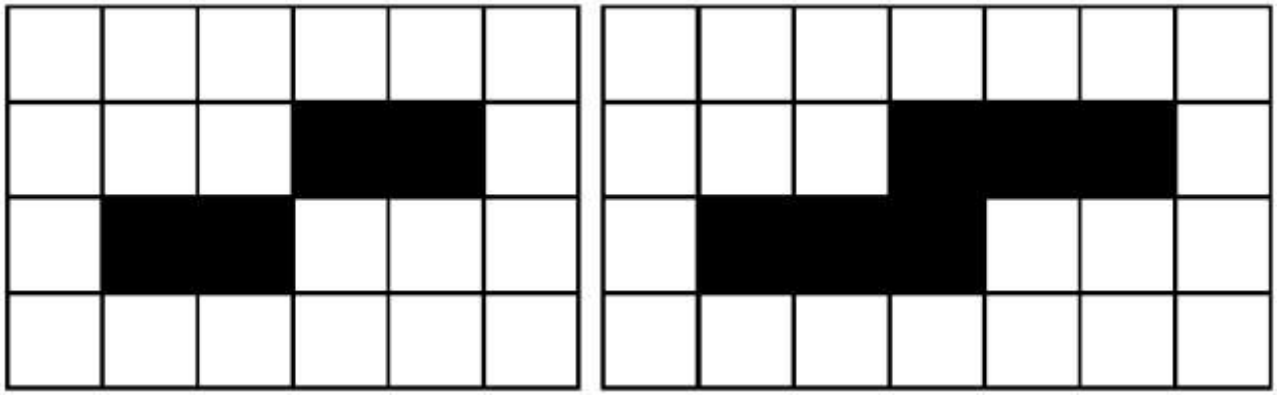
- Input:
  - Description of the board
  - List of cell pairs
    - (source, destination)
- Output:
  - list of routes
- Program:
  - Automatically generate the routes so that the routes do not contain cells in common while offering the best "electrical properties".

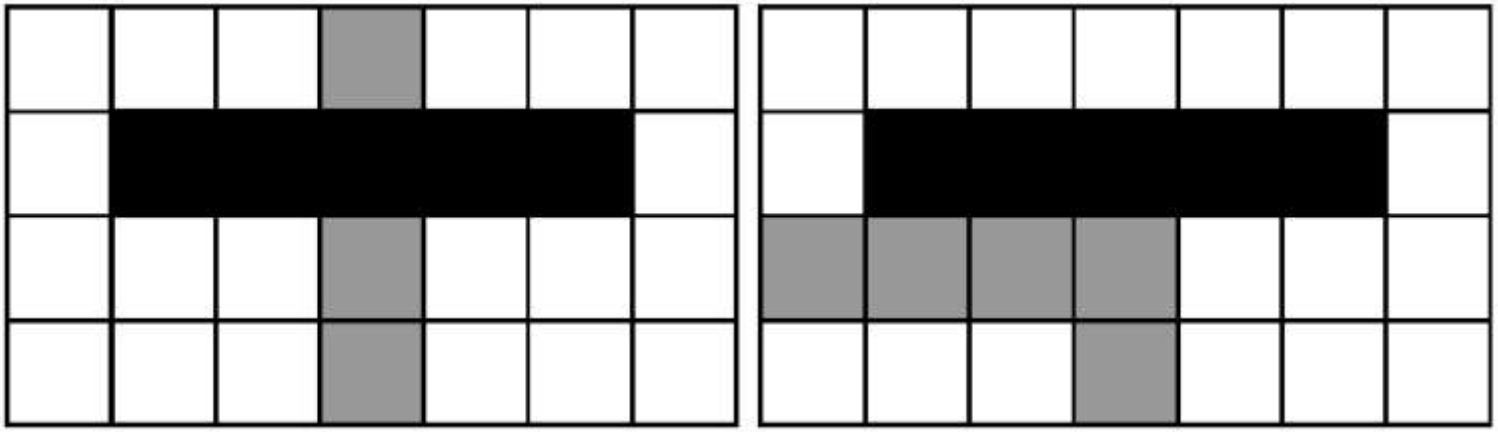
# Lee's algorithm



# What kind of routes can we guarantee to have found?

# Example of routes: disallowed vs allowed





# Lee's algorithm (pseudo code)

```
Grid grid

for i in list of routes {
        expand (from source to destination)
        traceBack (from destination to origin)
        cleanup(expansion)
}
```

# Roadmap for today

- APT Group Intro & Need for SW/HD co-design
- Lee's algorithm
  - Understand the problem
- Parallel implementations
  - Different choices
  - Lessons
- Transactional Memory
  - Basic concept
  - Lee with transactions
- Performance analysis
- Improving performance
- Teraflux

# My turn – Parallel Lee

```
Grid grid
ListOfRoutes myRoutes // subset of routes

for my_i in myRoutes  {
        acquire lock(grid)
        expand(from origin to destination)
        traceBack (from destination to origin)
        cleanup(expansion)
        release lock(grid)
}
```

# Our turn – Towards Parallel Lee v2.0

```
Grid grid

for i in list of routes {
        expand (from source to destination)
        traceBack (from destination to origin)
        cleanup(expansion)
}
```

# Our turn – Parallel Lee v2.0

```
Grid grid
VectorOfLocks vector
SynchronizedQueueOfRoutes queue, queueForLongRoutes

while (thereAreMoreRoutes & IAmActive) {
        nextRoute (queue)
        determine to which grid partition route belongs // coordinates
        if route fits within partition{
                acquire lock(vector, coordinates for partition)
                expand (from source to destination)
                traceBack(from destination to origin)
                clenup (expansion)
                release lock(vector, coordinates for partition)
        }
        else {
                add route to queueForLongRoutes
        }
        // decide whether IAmActive still, grow partition & swap
        //  queue andqueueForLongRoutes
}
```

# Your turn – Towards Parallel Lee v3.0

```
Grid grid

for i in list of routes {
        expand (from source to destination)
        traceBack (from destination to origin)
        cleanup(expansionGrid)
}
```

# A pause for reflection

- Parallel programming -> easy/complex
- Deadlock/livelock
- Composing parallel libraries
- Message passing vs. shared memory
- Memory model (SC, relaxed)

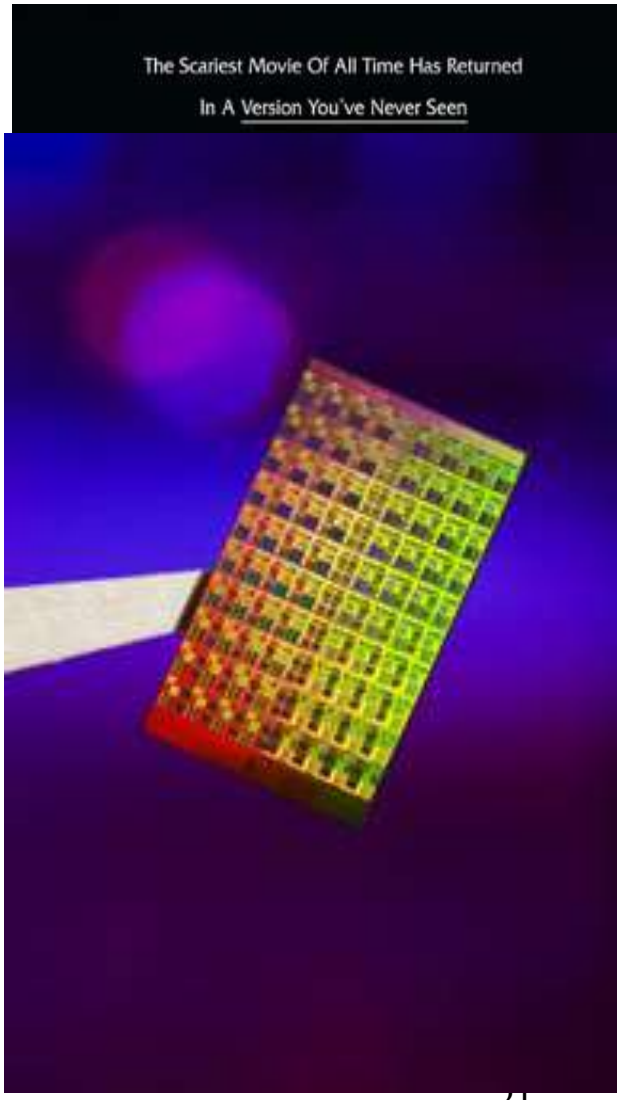
- Can we offer these abstractions to expert software developers? To high productivity ones?

# Roadmap for today

- APT Group Intro & Need for SW/HD co-design
- Lee's algorithm
  - Understand the problem
- Parallel implementations
  - Different choices
  - Lessons
- Transactional Memory
  - Basic concept
  - Lee with transactions
- Performance analysis
- Improving performance
- Teraflux

# Transactional Memory Hype – Big Promises

- Composition
- Easy to use as a single global lock
- As efficient as fine grain locking



# One transaction in databases?

- ACID
  - **Atomicity**: is the property which guarantees that every operation has been performed or none at all (never halfway)
  - **Consistency**: is the property which guarantees that read and written values are coherent
  - **Isolation**: is the property which guarantees that one transaction will not be affected by another transaction
  - Durability: is the prosperity which guarantees persistent data

# Transactional Memory - Syntax

```
synchronized(foo) {
  x++;
  y++;
  z++;
}
```

```
atomic {
  x++;
  y++;
  z++;
}
```

# Locks - Example

T1:

```
synchronized(foo) {
  x++;
  y++;
  z++;
}
```

T2:

```
synchronized(foo) {
  x++;
  y++;
  z++;
}
```

# Locks – Example two

```
T1:
synchronized(foo) {
  x++;
  y++;
  z++;
}
```

```
T2:
synchronized(foo) {
  a++;
  b++;
  c++;
}
```

# Transactional Memory – Example two

```
T1:
atomic {
  x++;
  y++;
  z++;
}
```

```
T2:
atomic {
  a++;
  b++;
  c++;
}
```

# Sets and conflict detection

```
Tx1:
atomic {
  x = y + z;
}
```

- {y, z} read set
- {x} write set

- Transaction Tx1 will have a conflict with another parallel executing transaction

  IFF the intersection of the sets is not empty

Which ones?

# Transactional Memory - Requirements

- To be able to store the read set and the write set
- To be able to computer the intersection of the sets

- When one Tx executes optimistically -> to be able of restore the state of the program and computer architecture to the state before the transaction started

# TM Implementations (landscape)

- Granularity
- Conflict detection (eager vs. lazy)
- Speculative state (write operations)

- Software (DSTM2, RSTM, tinySTM, TL2, DiSTM, etc.)
- Hardware (TCC, LogTM, Rock,…) & Haswell
- Hybrid (Rock, Intel Research, Microsoft Research)

# Roadmap for today

- Multi-core: ubiquitous and future trends
- Lee's algorithm
  - Understand the problem
- Parallel implementations
  - Different choices
  - Lessons
- Transactional Memory
  - Basic concept
  - Lee with transactions
- Performance analysis
- Improving performance
- Teraflux

# Lee's algorithm (pseudo code)

```
Grid grid

for i in list of routes {
        expand (from source to destination)
        traceBack (from destination to origin)
        cleanup(expansion)
}
```

# Transaccional Lee (pseudo code)

```
Grid grid

forall routes  { // work queue
        atomic{
                expand (from source to destination)
                traceBack (from destination to origin)
                cleanup(expansion)
        }
}
```

# Can we improve it?

- Privatization

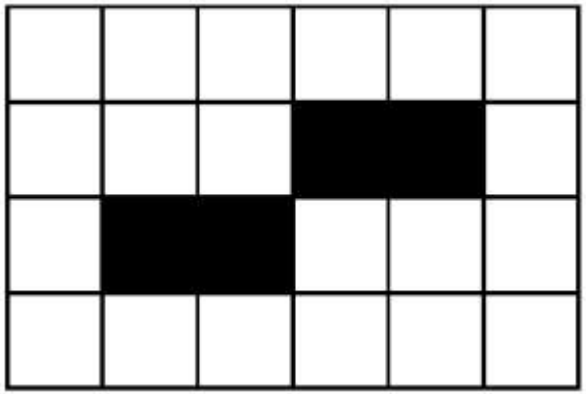
# Transactional Lee (privatization)

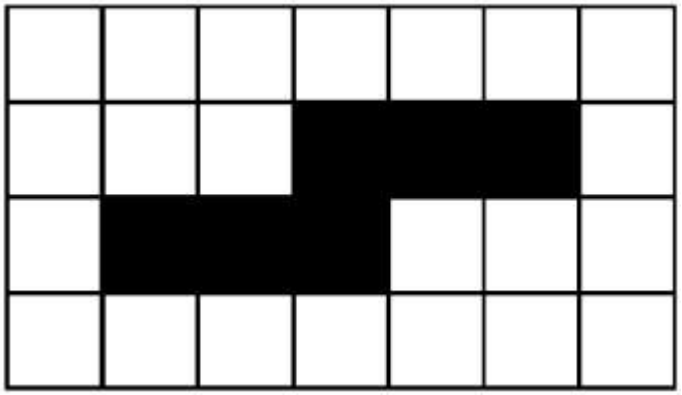
```
Grid grid

forall routes  { // work queue
        atomic{
                Grid local
                expansion (from source to destination)
                // read global & write local
                traceBack (from destination to origin)
                // read local & write global
                // NO: cleanup(expansion)
        }
}
```

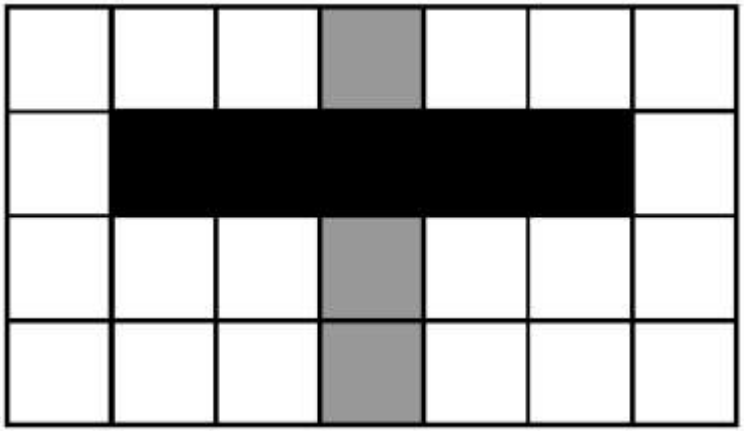
# We'll look at the performance later

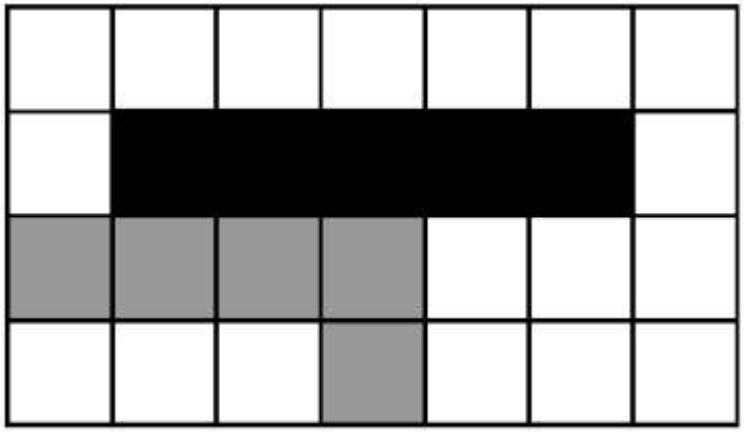
- But, have we reached the optimum?

# Routes: disallowed vs allowed









# Transactional Lee (privatization)

```
Grid grid

forall routes  { // work queue
        atomic{
                Grid local
                expansion (from source to destination)
                // read global & write local
                traceBack (from destination to origin)
                // read local & write global
                // NO: cleanup(expansion)
        }
}
```

# Transactional Lee (early release)

```
Grid grid

forall routes  { // work queue
        atomic{
                Grid local
                expansion (from source to destination)
                // ER: read global & write local
                traceBack (from destination to origin)
                // read local, compare with global &
                // write global
                // NO: cleanup(expansion)
        }
} // We are not advocating for early release
```

# Roadmap for today

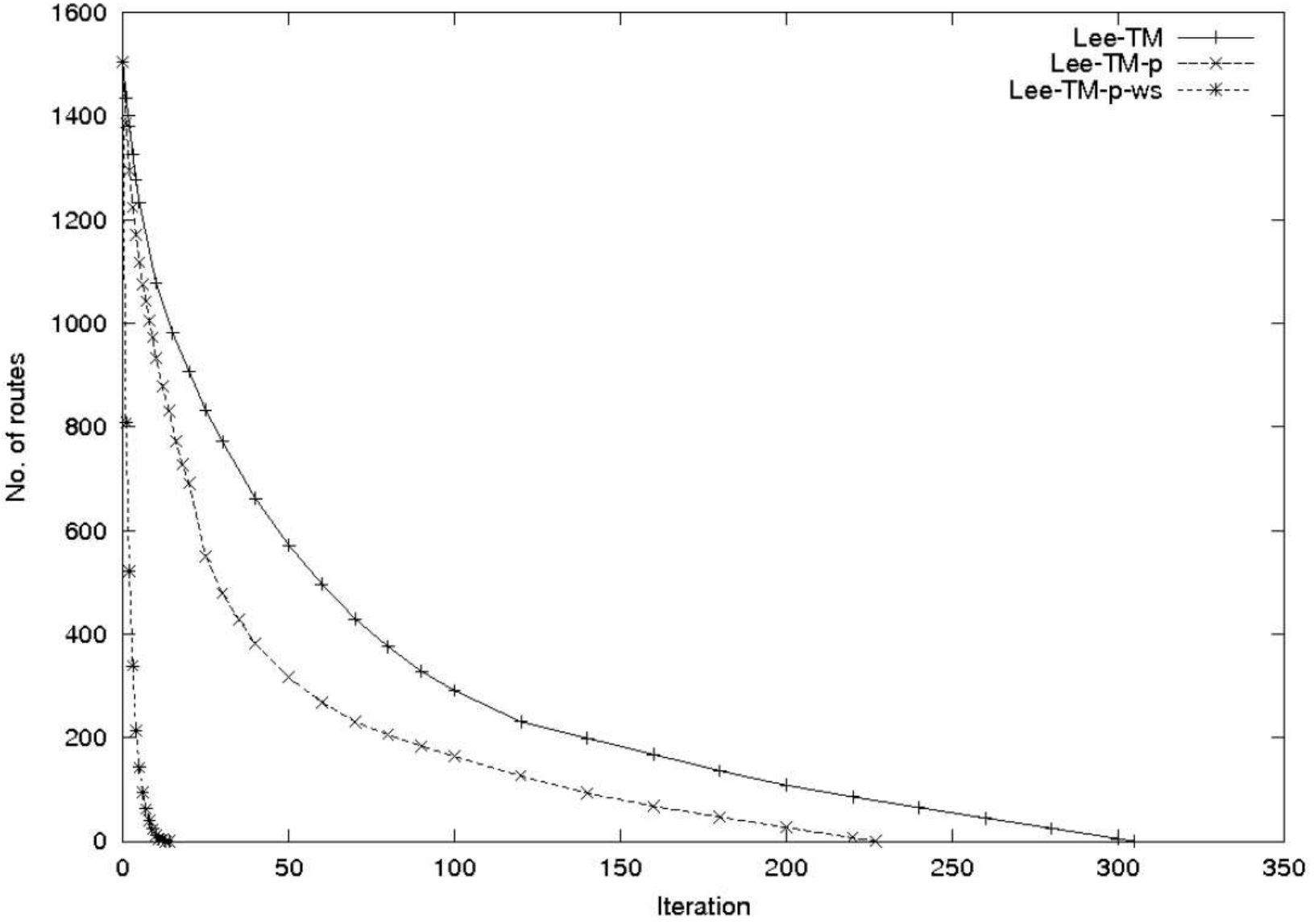
- Multi-core: ubiquitous and future trends
- Lee's algorithm
  - Understand the problem
- Parallel implementations
  - Different choices
  - Lessons
- Transactional Memory
  - Basic concept
  - Lee with transactions
- Performance analysis
- Improving performance
- Teraflux

# Experiment: abstract TM

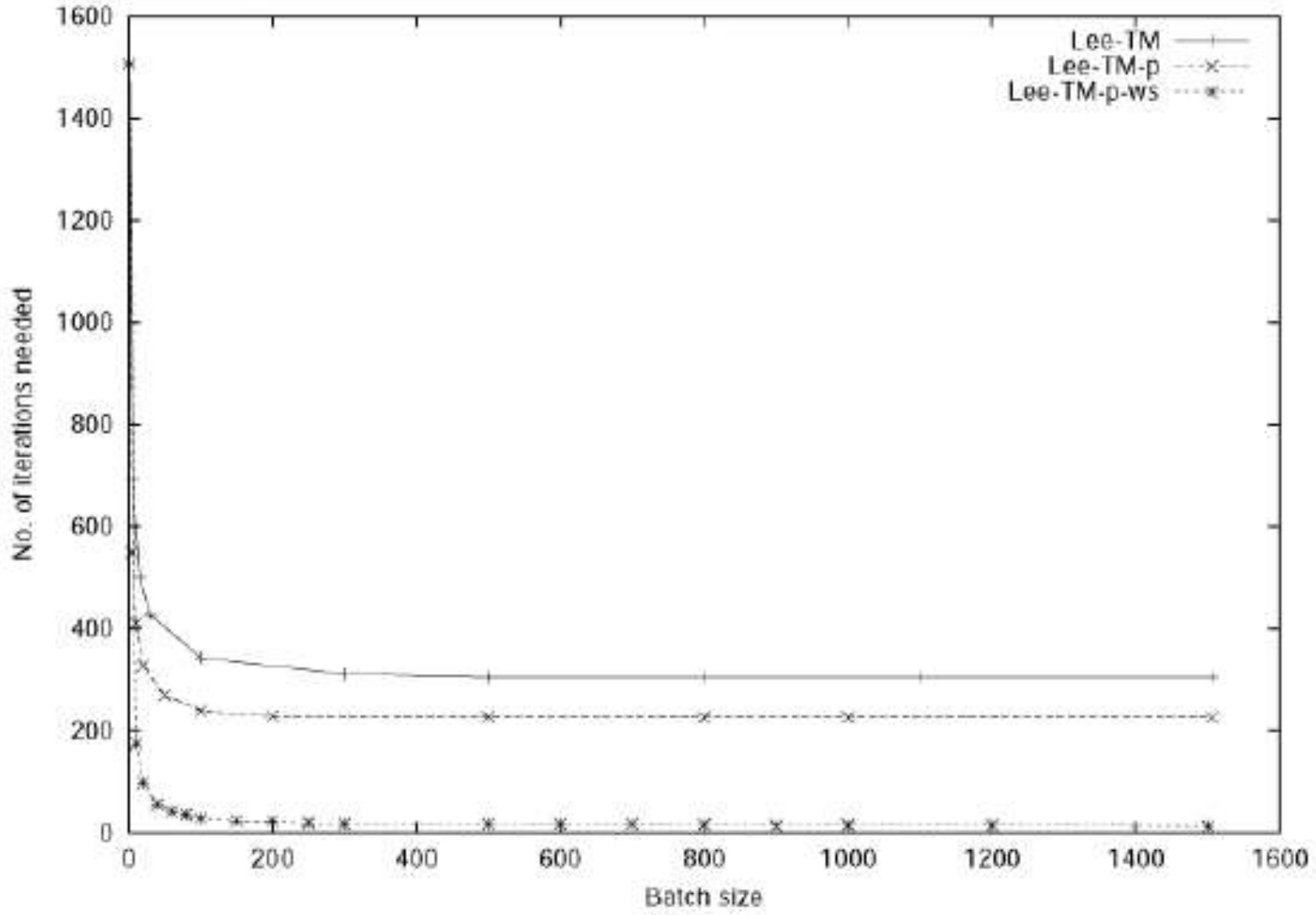
| #Iterations | 305 | 227 | 14 |
|---|---|---|---|
| 1st Iteration | 79 | 118 | 697 |
| Failed attempts | 89534 | 53838 | 374 |
| | Lee-TM | Lee-TM privatization | Lee-TM early release |

- 1506 routes
- Routes shorted in increasing order
- Algorithm tries to avoid "spaghetti" routes

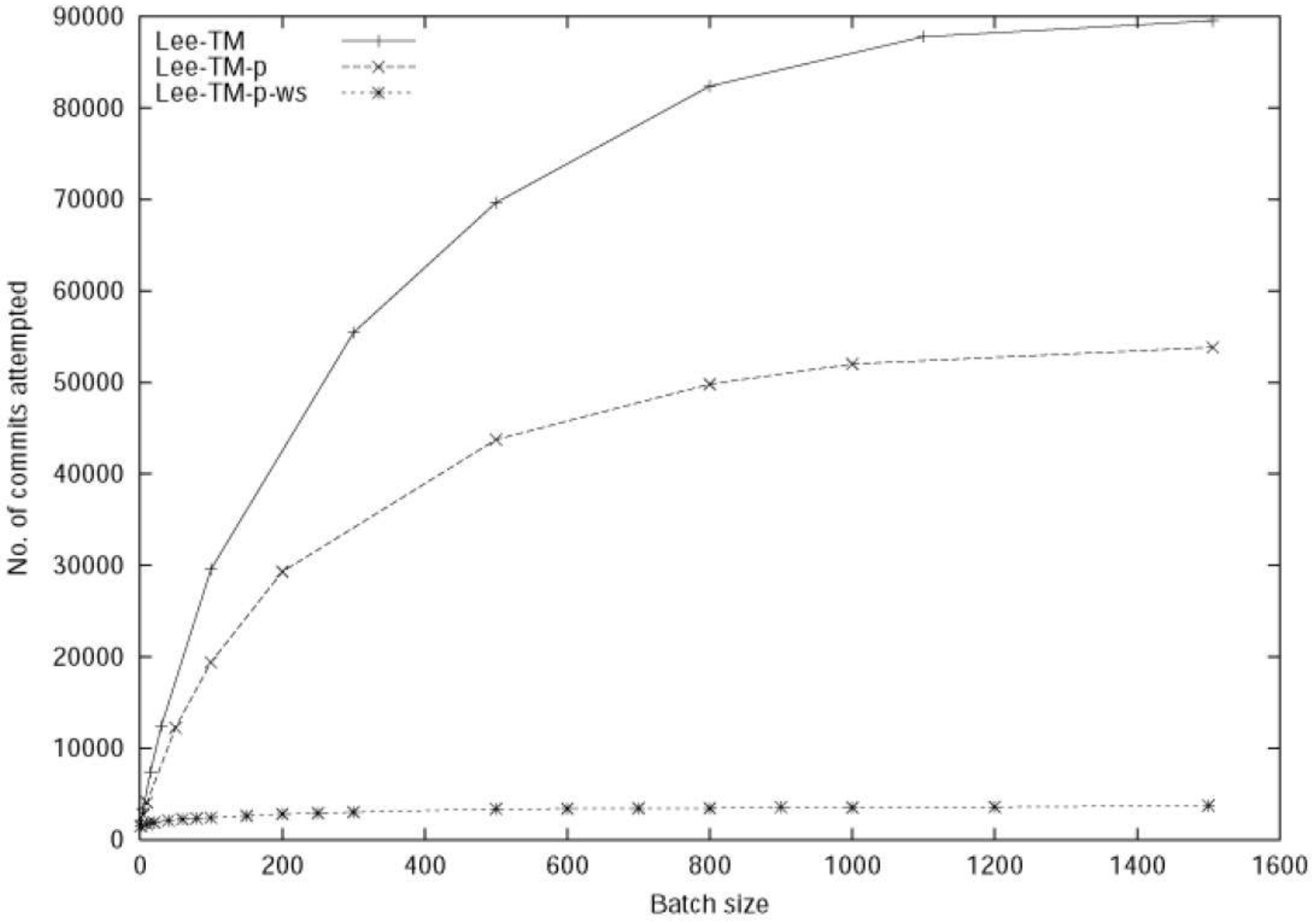
# Experiment: abs. TM (pending routes)



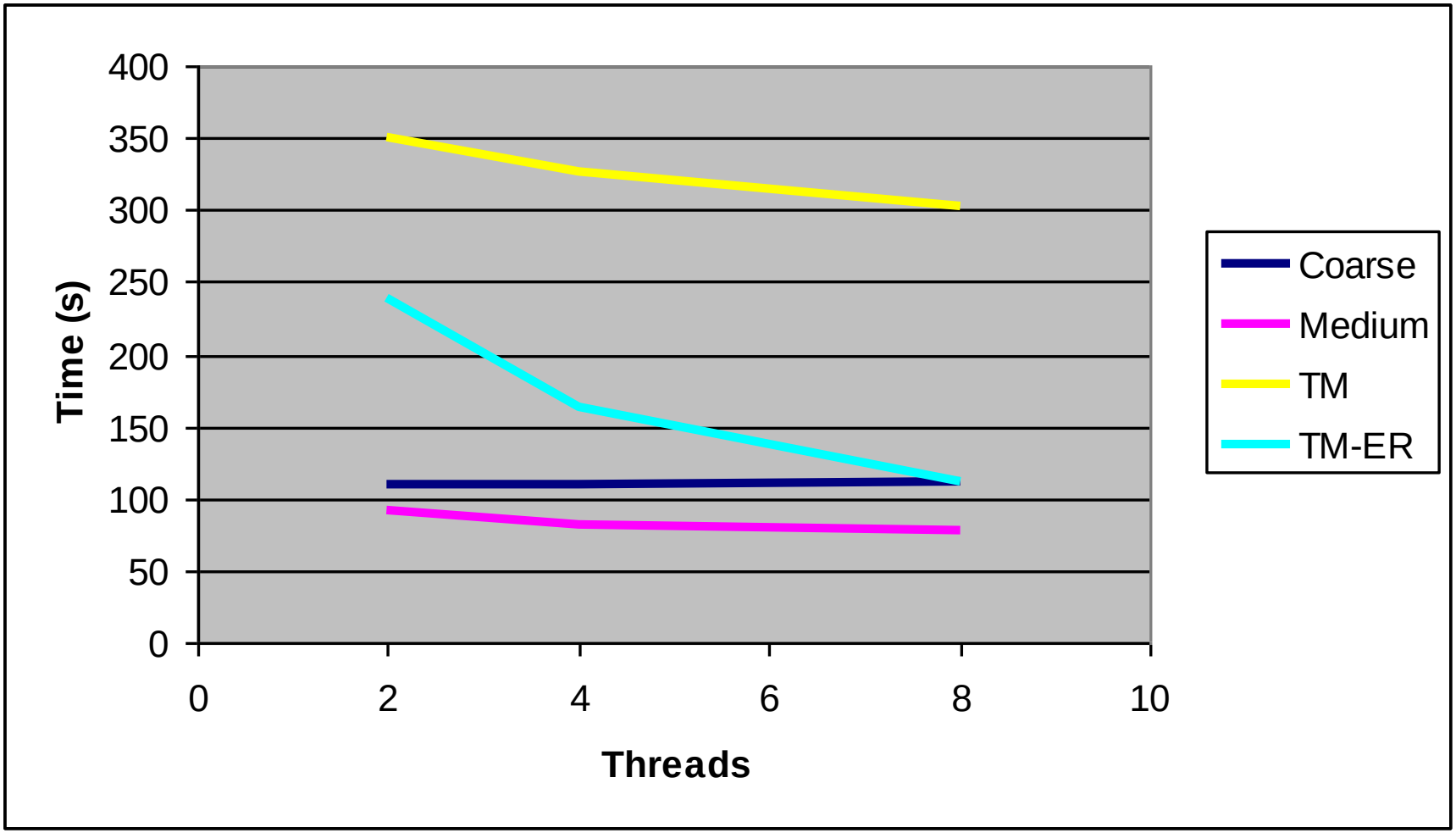
# Experiment: abs. TM (#iterations vs. #processors)



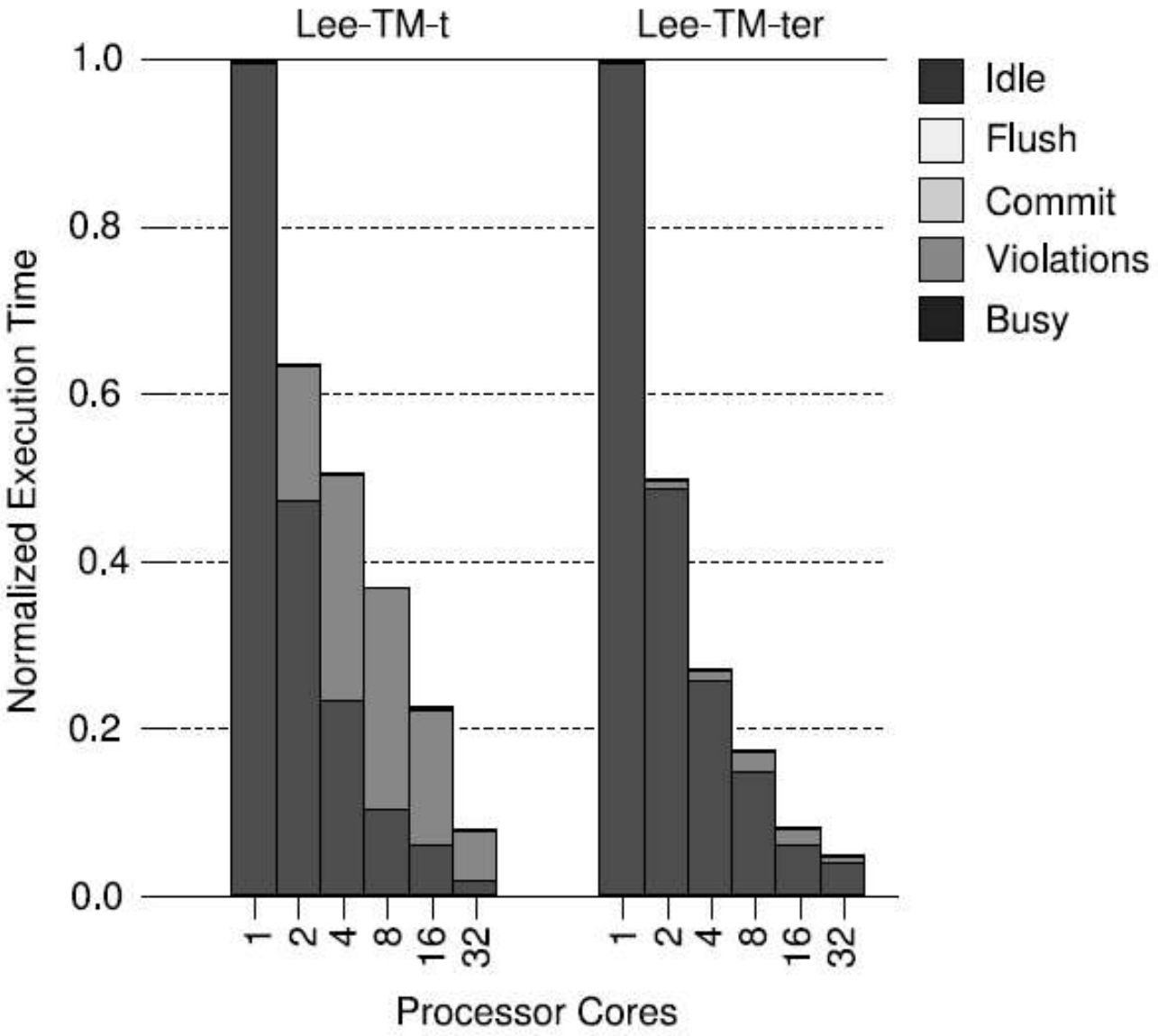
# Experiment abss TM (#executed transactions)



# Experiment with DSTM2 on 8-core AMD



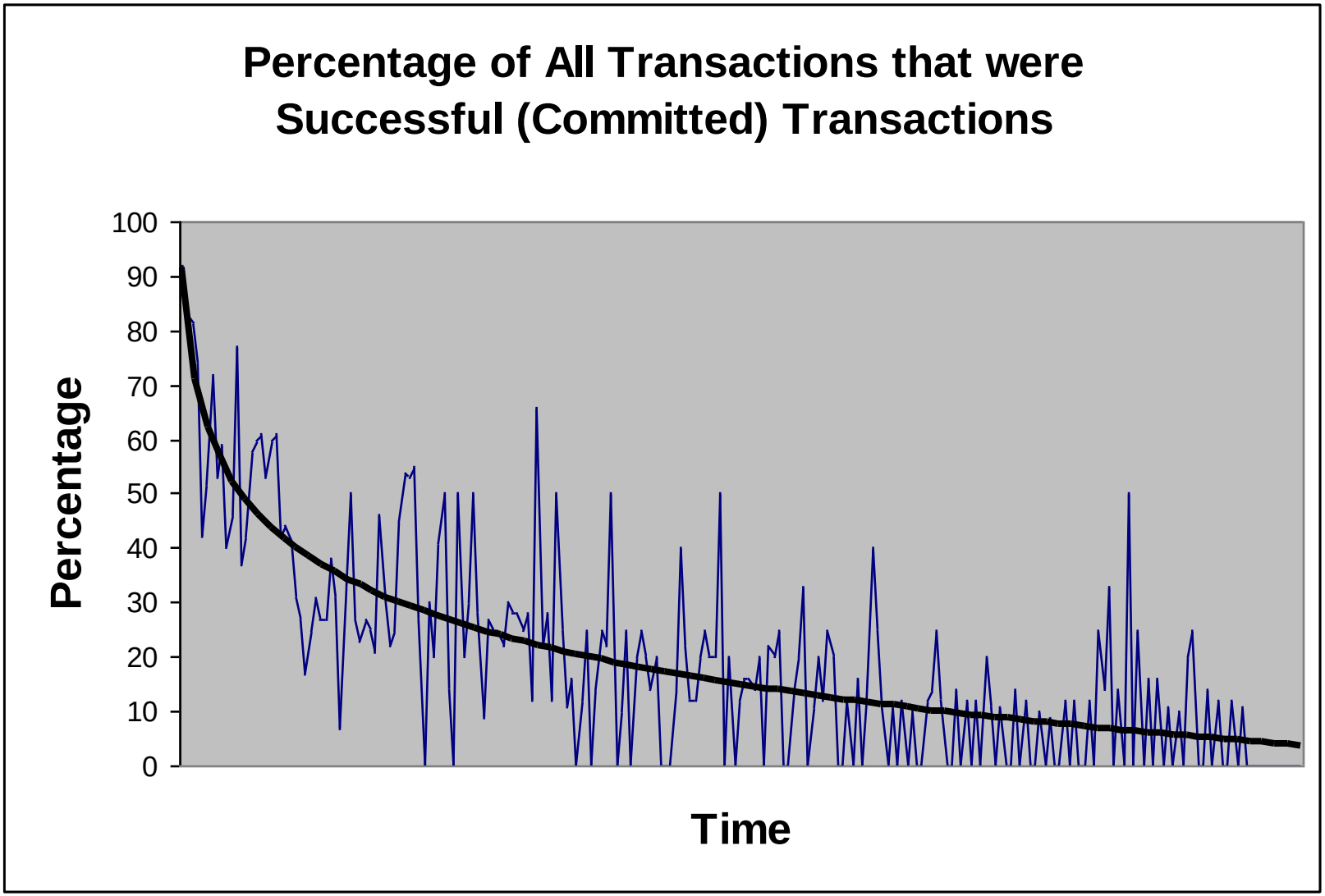
# Experiment with our HardwareTM



# Roadmap for today

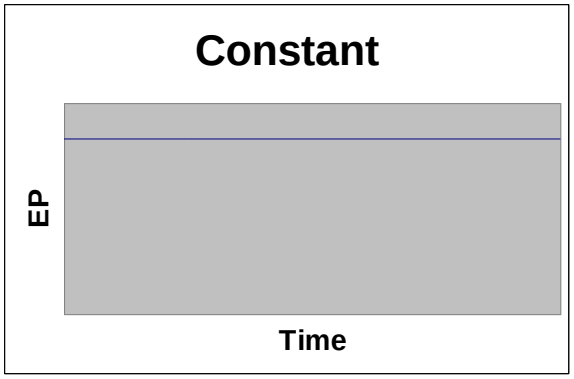
- APT Group Intro & Need for SW/HD co-design
- Lee's algorithm
  - Understand the problem
- Parallel implementations
  - Different choices
  - Lessons
- Transactional Memory
  - Basic concept
  - Lee with transactions
- Performance analysis
- Improving performance
- Teraflux

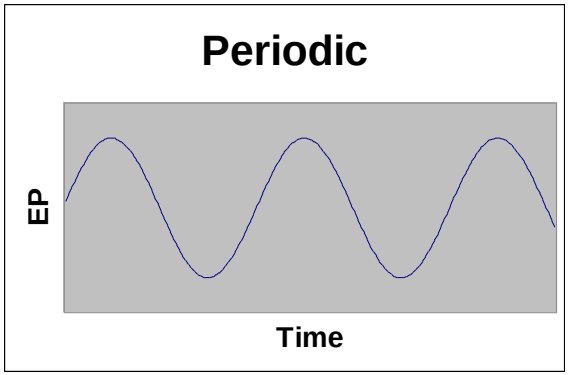
# Transactional Lee: a closer look (DSTM2)

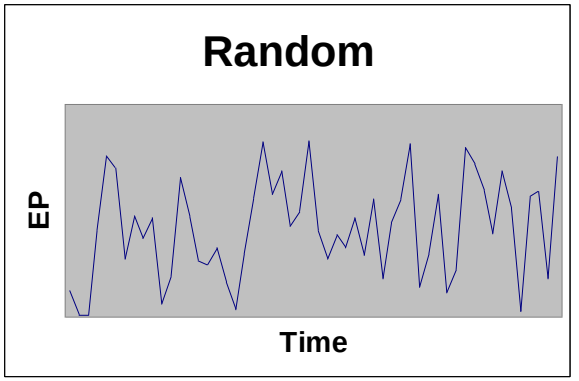


# Control (auto-tune) number of transaction

- TM applications can exhibit different phases with different levels of parallelism
- Relation between the number of transaction executing without conflicts and the amount of parallelism available in an application utilizable



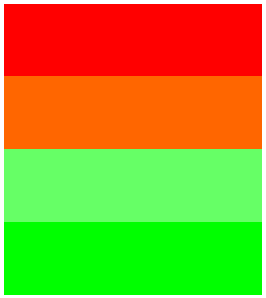



# How can we make it work?

- Use TCR as an approximation to the amount of parallelism available
- **Transaction Commit Rate (TCR)**
  - NumCommittedTx/NumTotalTx (in a give period of time)
  - If is high -> allow more parallel executing transactions
  - If is low -> allow fewer parallel executing transactions

# Results (execution time improvement)



< 0.9

0.9 - 1.0

1.0 - 1.1

> 1.1

| Contention Manager | Simple Adjust | | | | Exponential Interval | | | | Exponential Adjust | | | | Exponential Combined | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 1 | 2 | 4 | 8 | 1 | 2 | 4 | 8 | 1 | 2 | 4 | 8 | |
| **Aggressive** | 0.94 | 1.24 | 0.94 | 1.06 | 0.92 | 1.13 | 1.00 | 1.07 | 1.01 | 1.25 | 1.07 | 1.10 | 1.08 | 1.18 | 1.03 | 1.04 | **1.07** |
| **Backoff** | 0.82 | 0.74 | 1.63 | 2.47 | 0.84 | 0.87 | 1.39 | 2.73 | 0.76 | 0.90 | 1.41 | 3.00 | 0.89 | 0.91 | 1.41 | 2.47 | **1.45** |
| **Eruption** | 0.72 | 1.14 | 1.12 | 1.42 | 0.82 | 1.13 | 1.03 | 1.39 | 0.81 | 1.21 | 0.95 | 1.49 | 0.83 | 1.21 | 0.93 | 1.52 | **1.11** |
| **Greedy** | 1.20 | 1.08 | 1.00 | 1.34 | 0.99 | 0.98 | 1.00 | 1.26 | 1.14 | 1.04 | 1.00 | 1.36 | 1.08 | 0.99 | 0.94 | 1.33 | **1.11** |
| **Karma** | 1.12 | 1.04 | 1.05 | 1.31 | 1.02 | 1.21 | 1.05 | 1.30 | 1.18 | 1.13 | 1.04 | 1.41 | 1.05 | 1.13 | 1.03 | 1.41 | **1.16** |
| **Kindergarten** | 1.12 | 1.18 | 0.99 | 1.06 | 1.13 | 1.07 | 0.91 | 1.02 | 1.30 | 1.22 | 0.99 | 1.05 | 1.35 | 1.14 | 0.99 | 1.01 | **1.10** |
| **Polka** | 0.96 | 1.23 | 0.97 | 1.08 | 1.01 | 1.03 | 0.94 | 1.09 | 1.07 | 1.09 | 1.08 | 1.24 | 1.04 | 1.02 | 0.92 | 1.14 | **1.06** |
| **Priority** | **1.32** | **1.09** | **1.05** | **0.98** | **1.13** | **0.95** | **1.04** | **0.98** | **1.21** | **1.08** | **1.04** | **1.00** | **1.23** | **1.05** | **1.04** | **0.98** | **1.07** |

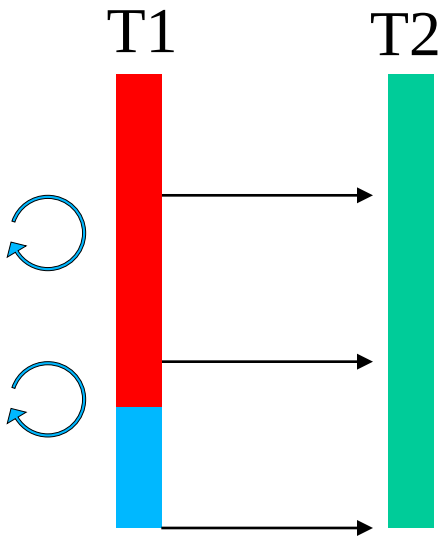
# Results (improvement #used cores)

- SimpleAdjust with 8 initial threads

| Contention Manager | Resource utilization (%) |
|---|---|
| Aggressive | 46 |
| Backoff | 82 |
| Eruption | 59 |
| Greedy | 57 |
| Karma | 53 |
| Kindergarten | 44 |
| Polka | 41 |
| Priority | 41 |

# Scheduling vs. Aborts: Example

- T1 and T2 execute concurrently
- T1 conflicts with T2
- T1 aborts
- T1 restarts (immediately)
- T1 conflicts with T2 ***again***
- T1 aborts ***again***
- T1 restarts (immediately)
- T1 conflicts with T2 ***again***
- …

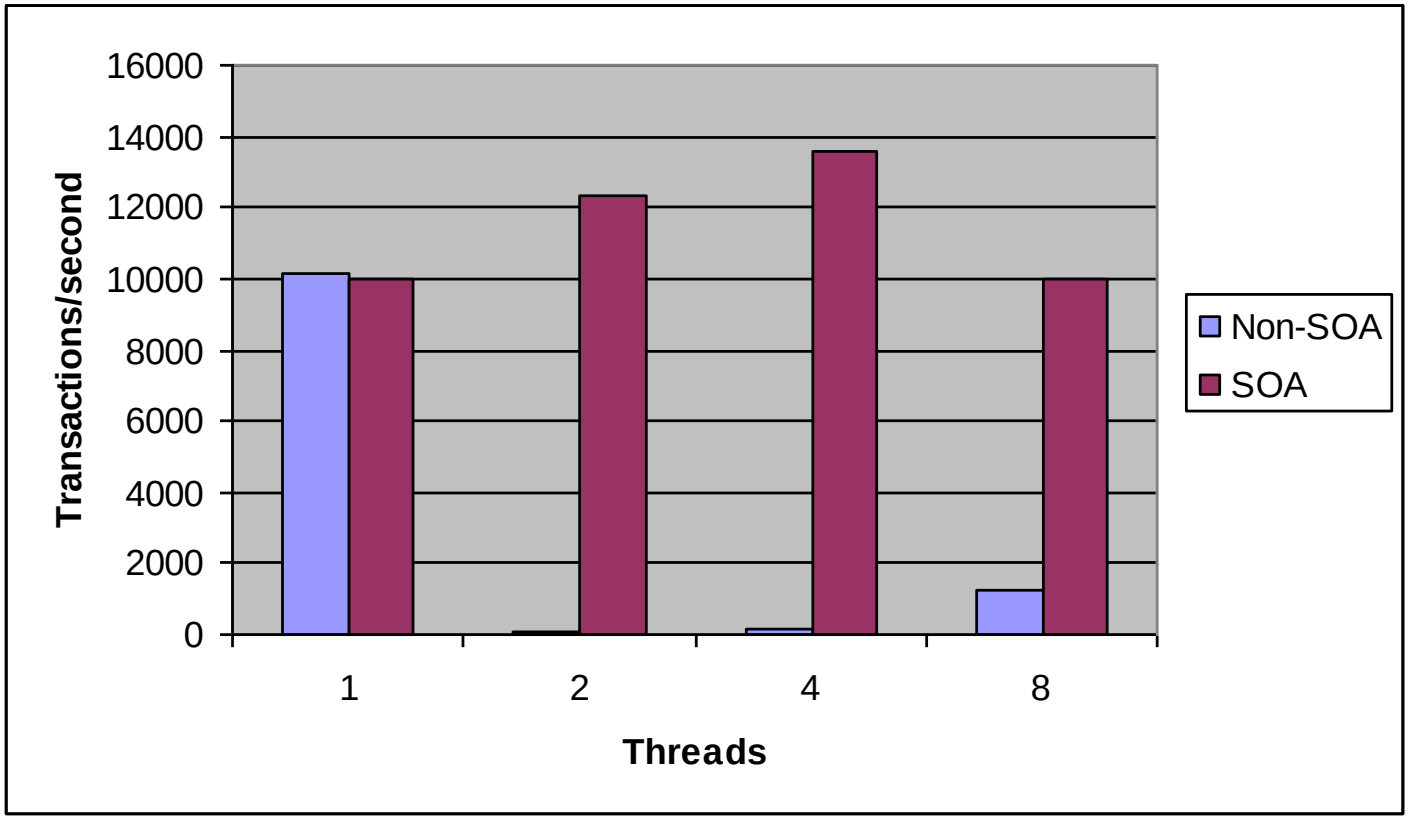



…

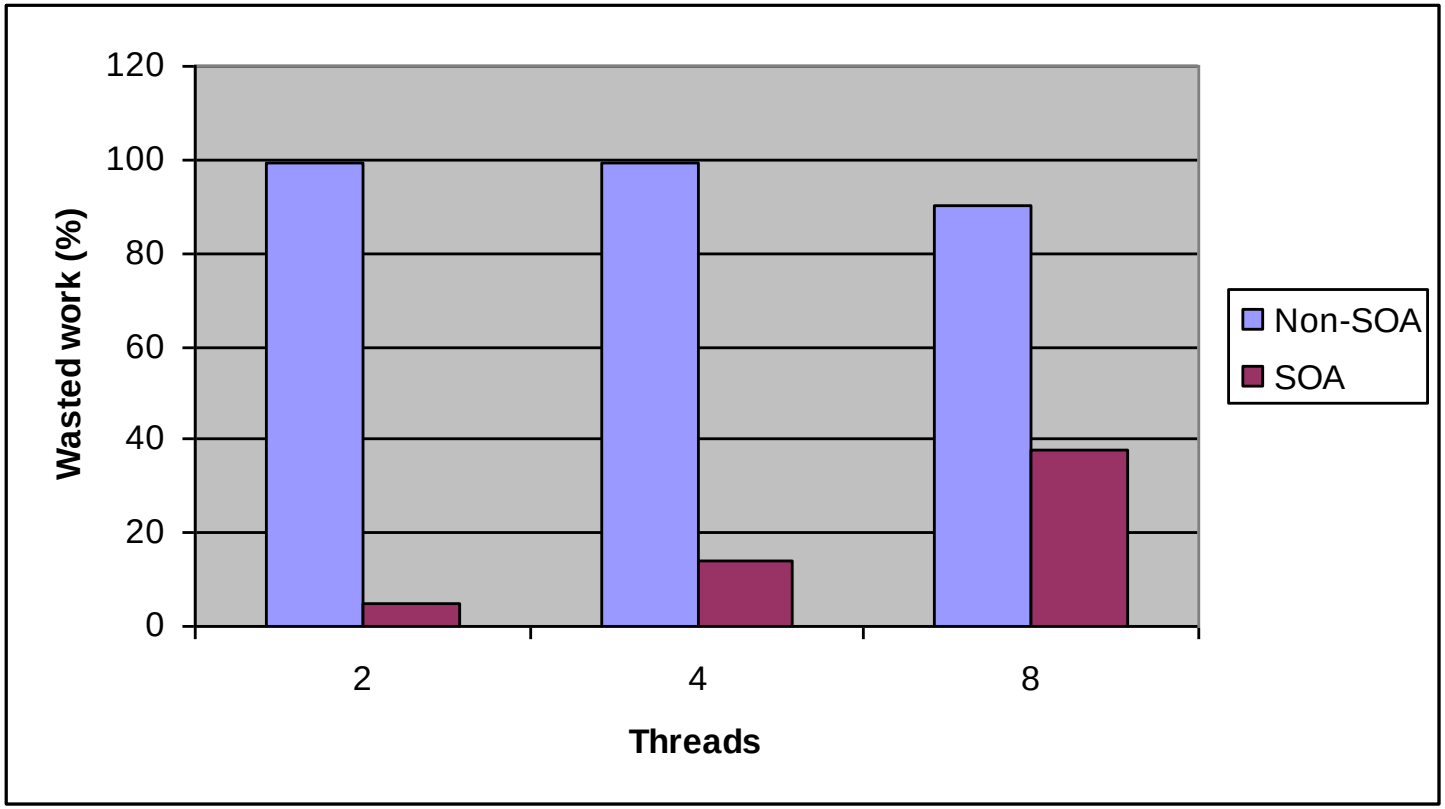
# Steal-on-Abort

- In general, difficult to predict first conflict/abort
- Once observed, simple to avoid next conflict/abort
  - Do not execute T1 & T2 concurrently
- Steal-on-abort design:
  - Automatically make scheduling decisions to avoid conflicts:
    - On abort, transaction **stolen** by aborter
    - Aborted transaction released after stealer commits
  - Additionally, attempt to improve performance:
    - Thread whose transaction is stolen obtains another transaction to execute. May commit, improving performance.

# Performance



# Wasted Resources



# Roadmap for today

- Multi-core: ubiquitous and future trends
- Lee's algorithm
  - Understand the problem
- Parallel implementations
  - Different choices
  - Lessons
- Transactional Memory
  - Basic concept
  - Lee with transactions
- Performance analysis
- Improving performance
- Teraflux

# Transactional Memory

- Transactional Memory is not a silver bullet.
- But, provides both a *concurrent programming abstraction* which is much simpler than traditional techniques; and
- A more relaxed coherence semantics. Program state must be coherent at the start and end of transaction.
- We are interested in Transactional Memory as a key component of a computational model

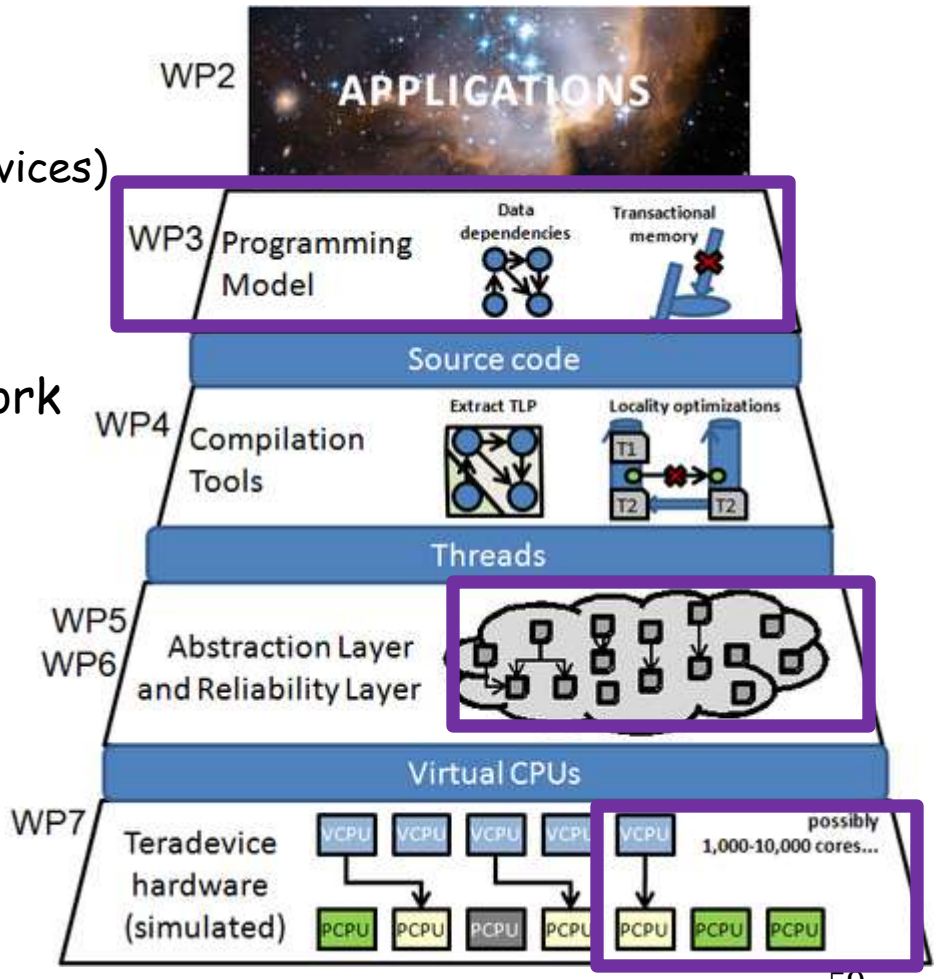
# My personal baggage with Parallel Systems

- Undergraduate: Shared Memory vs. Message Passing Programming
  - Equivalent, pain developing and debugging, performance (memory allocator, cache coherence)
- PhD: High Productivity for HPC
  - Java and OO for Numerical Linear Algebra
  - Recover lost performance with compilation techniques
  - Advisor: John Gurd
- Sun Microsystems DARPA High Productivity Computing System project
  - Runtime software for Petascale System (order of $10^6$ hardware threads)
  - PGAS, GUPS & Global address space vs. Cache Coherence
- Transactional Memory in Manchester
  - Software, Hardware, Distributed, Scheduling, Applications …
  - Work with Ian Watson & Chris Kirkham
- Teraflux

# The TERAFLUX Project

## Exploiting Dataflow Parallelism in Teradevice Computing

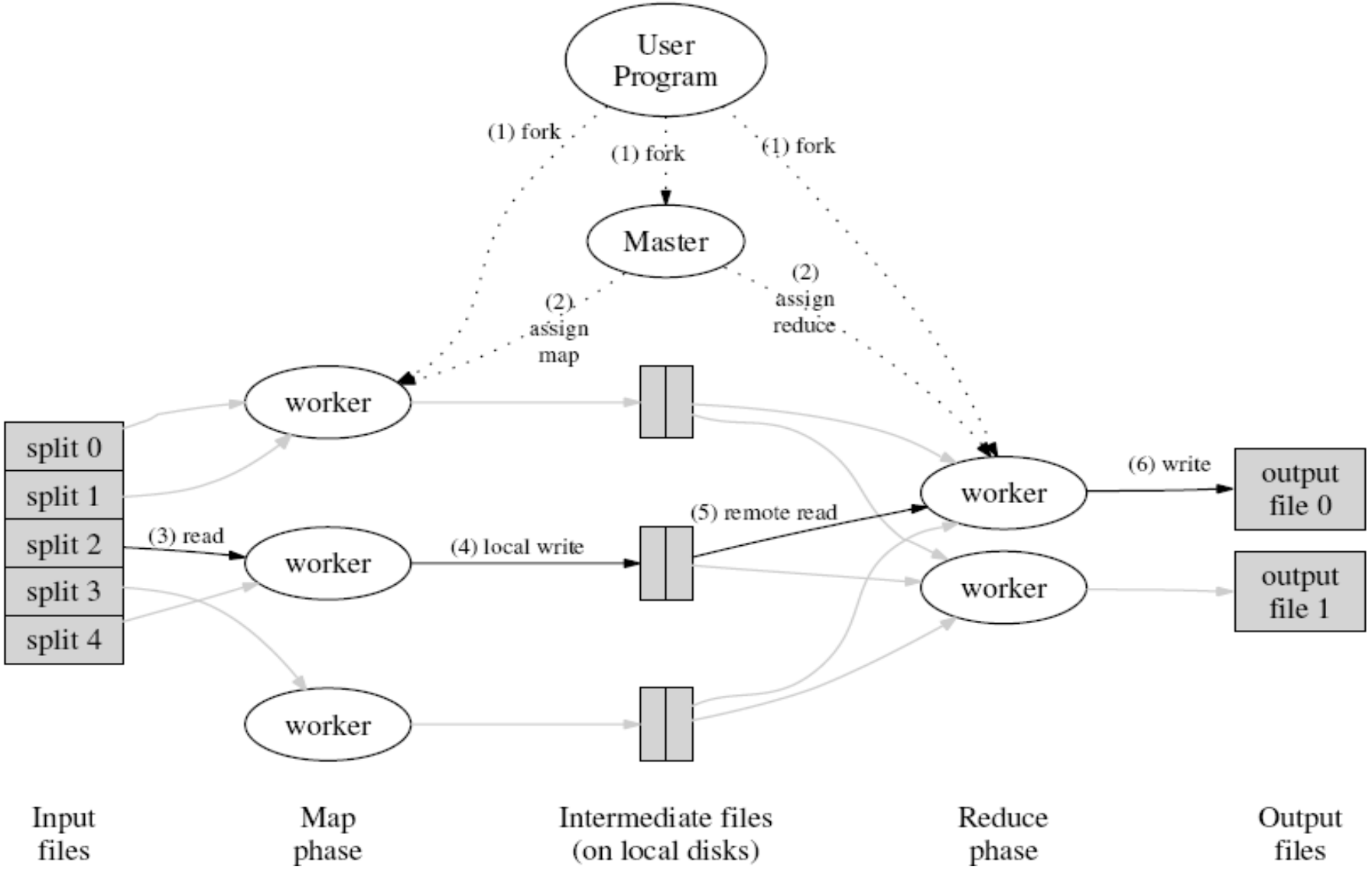
- What is it about?
  - Many-cores (1000+ cores or Teradevices)
  - General purpose computing
  - Dataflow (data driven execution)
  - Reliability
- Funded by the EU Seventh Framework
  - University of Siena (co-ordinator)
  - Barcelona Supercomputing Centre
  - CAPS Enterprise
  - Hewlett Packard
  - INRIA
  - Microsoft (Israel)
  - THALES
  - University of Cyprus
  - University of Augsburg
  - University of Manchester



# What is the fuss with Dataflow?

- Computation Model:
  - Computation is described as a graph
  - Edges describe unidirectional data dependencies
  - Nodes represent computation (side-effect free computation)
  - Execution follows data driven
    - A node is "fired" once all its input data is ready
    - Parallel execution is natural: multiple nodes can execute in parallel as long as their input data is available
- Relation with pure procedures (side effect free computation, nothing shared),…
- What was wrong with the Manchester Dataflow?

# Google MapReduce on data-centres
## OSDI'04



# Is Dataflow the silver bullet?

- A flexible and efficient way of exploiting parallelism
- Maybe its 'time has come' in the many core era
  - Consider MapReduce, NLA, GPUs, FPGAs
- But is it general purpose?
  - Is certainly good at irregular (i.e. general purpose) parallelism where other approaches fail
  - But a big weakness is (with its underlying side effect free connections) an inability to deal well with shared mutable state
  - Transactional memory provides a good mechanism for updating shared mutable state (Isolation and Atomicity)

# Dataflow plus Transactions

- A major aim of the TERAFLUX project is to investigate the introduction of Transactional Memory into Dataflow
  - Computational Model vs Programming Environment
  - Hardware Support
  - Fault-tolerance
  - Applications
- I'm just giving you
  - a high level overview & motivation
  - a description and perspective of work-in-progress in Manchester

# Prototyping in Scala

- Scala
  - High Productivity Developers
  - Combines functional programming with OO
- We have extended Scala with Transactional syntax and have provided a Software Transactional Memory
  - http://apt.cs.man.ac.uk/projects/TERAFLUX/MUTS
  - Manchester University Transactions for Scala (MUTS)
- We have implemented a new Dataflow library
- We are investigating means of generating automically dataflow execution. Developer does not create threads
  - Reimplementation of the Scala parallel collection using dataflow plus transactions
  - Analysis for Lee-TM of benefits of Dataflow plus transactions
- We are investigating how a subset of Scala and the "right" type system can simplify the software development

# Many-core Architecture in Manchester

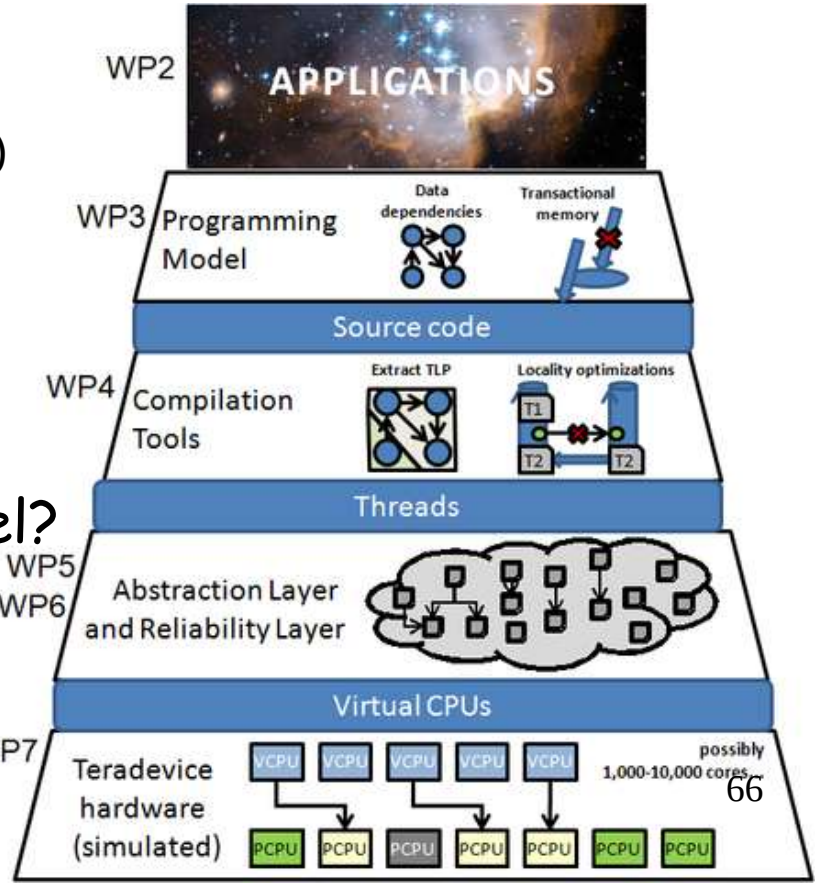
- Contributing to the memory model
- Investigating how to simplify coherency & consistency by using Dataflow and TM computational model
  - No "traditional" cache coherence across the chip, but globally accessible address space
- Investigating how to scale hardware TM
  - Can dataflow simplify the TM implementation?
- Investigating relation between hardware Dataflow scheduler and hardware TM
- How to simulate large many-cores? NoCs 2012
- How to make TM compatible with fault-tolerance mechanism proposed by our partners.
- MCTS for GO game and other applications

# Summary

- Dataflow plus Transactions seems to be a promising new approach to extend the power of the Dataflow model to include shared state

- What is it about?
  - Many-cores (1000+ cores or Teradevices)
  - General purpose computing
  - Dataflow (data driven execution)
  - Reliability

DF+TM = efficient general purpose parallel computational model?



# More Information

- http://www.teraflux.eu
- http://www.cs.manchester.ac.uk/apt/projects/TM
- http://www.cs.wisc.edu/trans-memory/

- Transactional Memory. Harris, Larus & Rajwar, 2010.