Multiprocessor Scheduling

What we know, what we know we don't know, and the rest

Scheduling

- A scheduling talk with no equations!
- Some reflections on open issues and implications for programming languages

Applications

- Application is comprised of threads/tasks, with
  - Periods, T
    - Periodic and sporadic treads
  - Deadlines, D
  - Computation times, C
- A platform consists of a number of cores

Number of cores

- How many cores are you considering?

Number of cores

- How many cores are you considering?

- Not enough!

Number of cores

Number of cores

- Burns’ Classification

Number of cores

- Burns' Classification
  - 1

Number of cores

- Burns' Classification
  - 1
  - A few (homogeneous)

Number of cores

- Burns’ Classification
  - 1
  - A few (homogeneous)
  - Lots (and heterogeneous)

Number of cores

- Burns' Classification
  - 1
  - A few (homogeneous)
  - Lots (and heterogeneous)

  - Too many

Single Processor

- Lots of well known results
- EDF is an optimal scheme
  - 100% usage if period=deadline
- Fixed priority is a very efficient scheme
- Response-Time Analysis (RTA) can cope with most application models
- Optimal priority assignment available

Single Processor

- Processor Demand Analysis (PDA) can cope with most application models for EDF
- Shared objects implemented effectively and efficiently by priority ceiling protocols (FP and EDF)

Main Problem

- Safe but accurate computation times are very difficult to obtain on modern hardware
    - Worst-case rare and >> average
- Models are too complex to use
- Measurement is intrusive and difficult to undertake

One Approach

- Try and obtain predictability as an emergent property
- Randomise aspects of the (temporal) behaviour of the hardware
  - For example a random cache replacement policy

A few cores (n)

- Many more natural application threads than cores
- So first concern is allocation

- Partitioned and global approaches to thread allocation
  - Affinity of a thread

Partitioned Systems

- First we allocate, then we have n single core systems
  - Assumes a fixed, static program
- Results from single processor systems can be then be applied
- But allocation is a NP-hard problem

Allocation

- An effective scheme is first fit based on utilisation or density
  - Largest T/C first (if D=T)
  - Largest D/C first if D<T
- But utilisation bound is n/2
  - Consider a system that only has threads with utilisation .50001
- For systems with small threads FF-EDF bound is approx 82%

Dynamic Schemes

- Influential Dhall paper in 1978 showed bound is $1 + \varepsilon$
  - Killed research until 1990s
- Then research was able to show that more intelligent allocations can give high utilisation, close to n

What we know

- EDF is not optimal
- EDF is not always better than FP
- Optimal scheduling of periodic threads requires excessive migrations (Pfair)
- Optimal scheduling of sporadic threads requires clairvoyance

What we know

- Many scheduling results are not sustainable
  - A schedulable system becomes unschedulable when things get better
  - ie C decreases, or
  - T increases
- Critical instance (worst-case arrival pattern) is NOT when all threads arrive together

What we know

- For fixed priority schemes
  - Effective scheduling tests do not give rise to optimal priority orderings
  - Can be better to use a sufficient test that can utilise Audsley's optimal priority assignment scheme

What we know

- Effective schemes deal with large threads (high utilisation) separately from small threads
- A typical scheme is to statically allocate large threads, global EDF for the rest, switching to non-preemption when a thread hits zero laxity

What we know

- A general strategy for determining schedulability is to
  - Define a problem window
  - Derive a necessary condition for non-schedulability
  - Invert to produce a sufficient test for schedulability

What is now understood

- Dynamic allocation is not producing significantly better results than partitioned
- Tests are very complex and run-time behaviour is non trivial
- Empirical studies highlight the cost of thread migration

Hybrid Schemes

- Clustering
  - Migration only over a small set of cores, perhaps 4 (with coherent cache)
- Semi-partitioned
  - Most threads statically allocated
  - At most n-1 thread migrations
  - From statically fixed source and destination cores

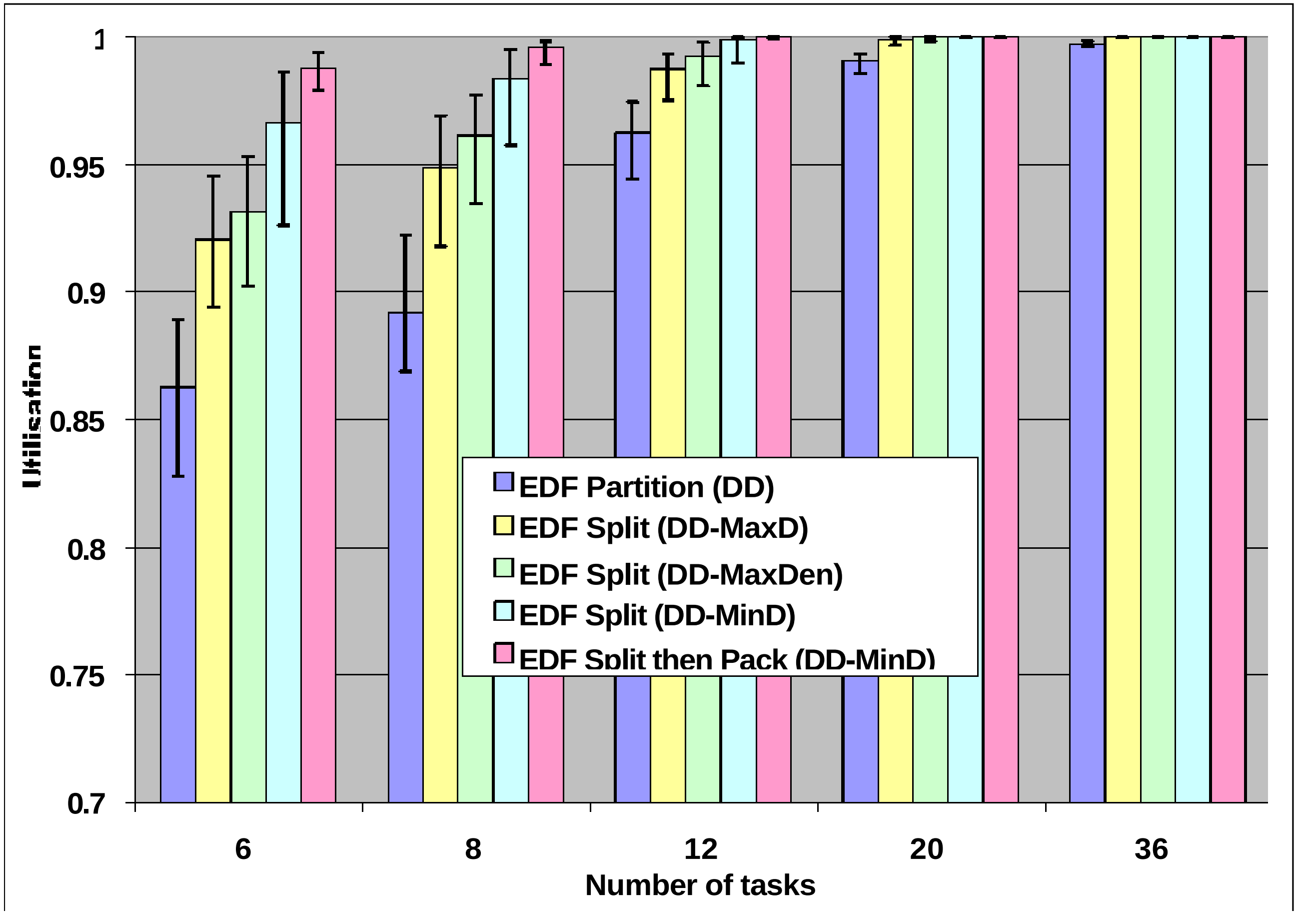
C=D Thread Splitting

- Cores split into domains
- Most threads fixed on domain and core
- EDF scheduling on each core
- One task per core migrates after a time of non-preemptive execution to another core in the same domain

Evaluation

- Using analysis the optimal point to split a thread is obtained
- But still a number of different heuristic are possible for deciding which thread to split
- Experiments undertaken for evaluation
- Results are average utilisation of all but last processor

Thread Splitting Performance



Problems

- Resource locking protocols are not well defined for multiprocessor platforms
- Estimations of execution times for a multi-core gets even more difficult
  - Shared busses (non-deterministic interference)
  - NoC – another resource to schedulable

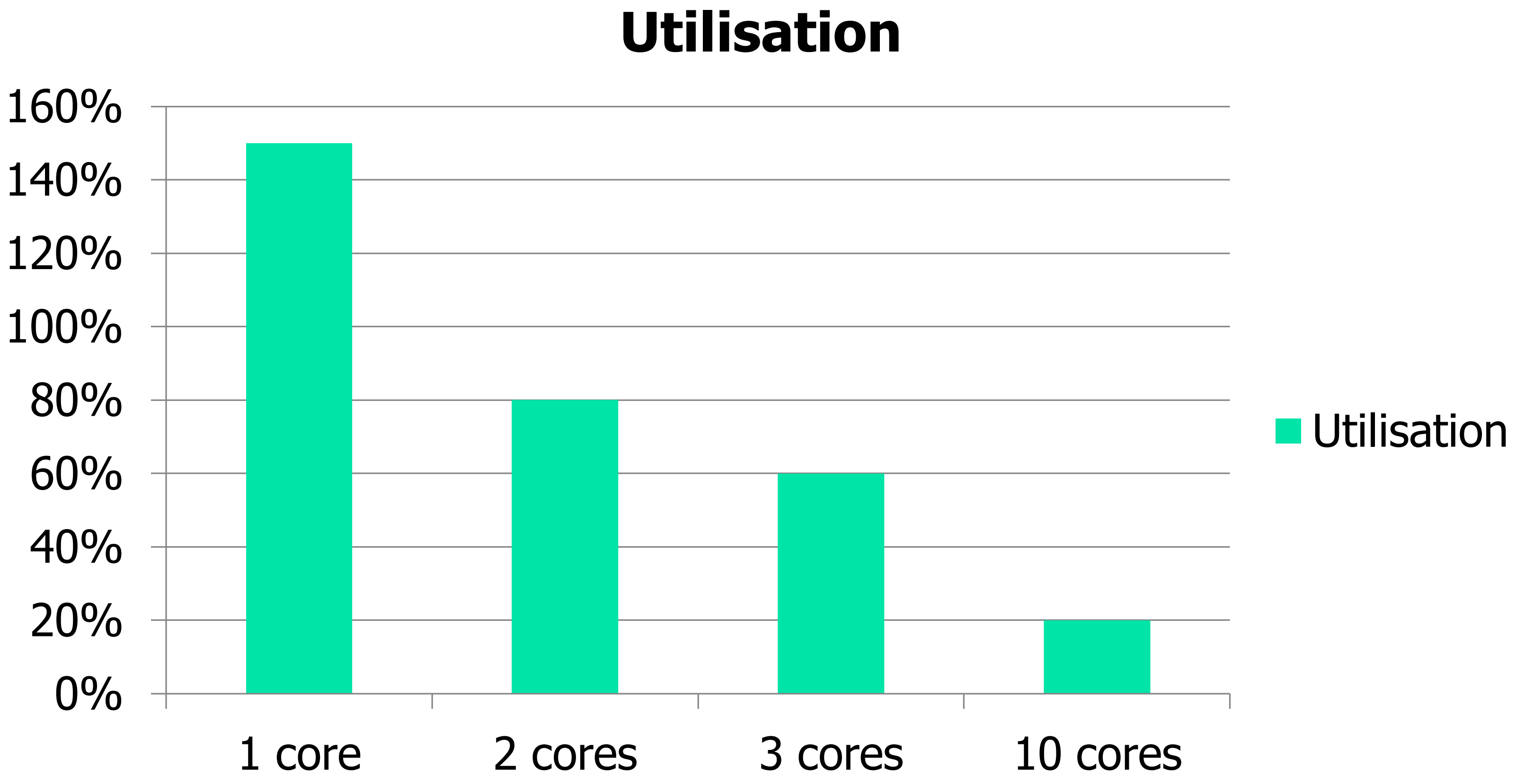
Language Support

- Deadlines and EDF (or fixed priority)
- Affinity control: domains, cores; program a move of an active task
- Timing events: trigger migration
- Volatile variables: Non-locking algorithms
- Fifo queues, ceiling control, monitors
- Atomic code: for transactional memory

and the rest – lots of cores

- The task is the right abstraction for real-time applications
- But if n >> m, compilers and hardware must help
- Languages must free up code from inappropriate sequencing
- Every application task is implemented by a number of platform threads

Profile of a task



Composability

- We then need to be able to schedule a set of tasks by composing their profiles
- Are the profiles composable?
- Perhaps if the hardware is more random

Randomising the hardware

- Predictability as an emergent property
  - At the time scale relevant to the application
  - Gases are predictable, molecules aren’t
  - Tasks can be predictable even if instructions aren’t (in time)

Contrived example

- Basic hardware instruction is iid with cost
  - 1 90% of the time
  - 10 10% of the time
- A program consists of 100,000 instructions
  - Worst-case: 1,000,000
  - Average: 190,000
  - WCET, P(A>E)<10^-9?

Contrived example

- Basic hardware instruction is iid with cost
  - 1 90% of the time
  - 10 10% of the time
- A program consists of 100,000 instructions
  - Worst-case: 1,000,000
  - Average: 190,000
  - WCET, P(A>E)<10^-9: 195,122

Summary

- We know how to schedule single processors
- We know many results for multiprocessors
- We know things that we will never know
- We know massively parallel hardware in on the way
- But still so many unknown unknowns

Sources

- A Survey of Hard Real-Time Scheduling for Multiprocessor Systems, Davis and Burns, ACM Computer Surveys, 2011.
- Partitioned EDF scheduling for Multiprocessors using a C=D task splitting scheme, Burns, Davis, Wang and Zhang, Real-Time Systems Journal, 2011.
- Predictability as an Emergent Behaviour, Burns and Griffin 4th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)