Autonomic Thread Parallelism and Mapping Control for Software Transactional Memory

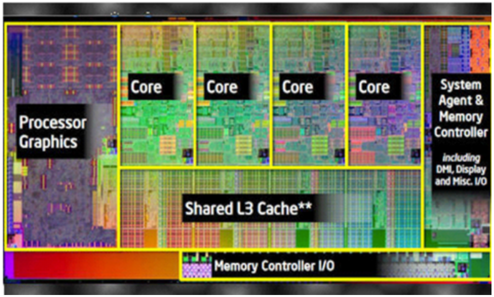
Presented by Naweiluo Zhou

PhD Supervisors:
Dr. Gwenaël Delaval
Dr. Éric Rutten
Prof. Jean-François Méhaut

PhD Defence

October 21, 2016

The era of multi-core processors



Single-core processors used to be dominant...

Single-core processors

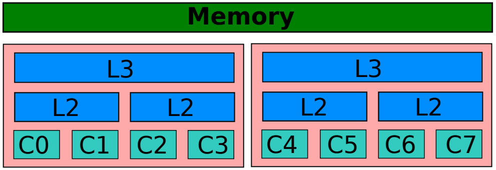
- Increment of clock frequency is approaching a physical end

Multi-core processors

1. Include multiple cores on a single chip
2. Give high performance with more cores

Parallel applications enables high performance

- Tasks of an application are processed by multiple cores
- Execution accelerates as tasks are processed in parallel
- Transactional memory
  - provides a good platform to deal with tasks in parallel
  - therefore, utilised in this thesis



An overview of multi-core processor

More cores, higher performance, but ...

issues of parallel programs

Concept: parallelism degree is the number of active threads (tn)

Synchronisation time, computing time, data access time

- A high parallelism degree may decline computing time, but increase synchronisation time.
  More threads maybe higher synchronisation

The trade-off between synchronisation and computation?

- Multi-core processors hold complicated memory architecture

How to reduce data access time to memory?

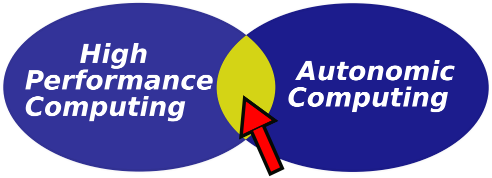
- The behaviour of an application can vary during its execution

How to control a system at runtime?

Contributions

This thesis contributes to:

- High performance computing
- Autonomic computing:
  techniques that can manage systems automatically



Overview

Outline

How to address synchronisation issues

threads need to synchronise

Locks

A conventional way for synchronisation. But:

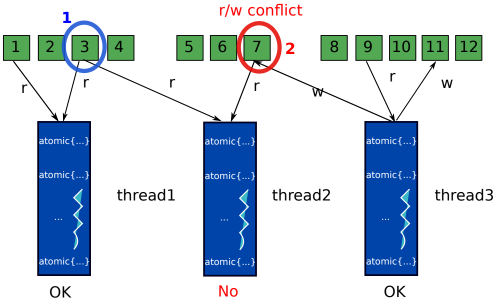
- Deadlocks, vulnerability to failures, faults...
- Difficult to detect deadlocks (*e.g.* not always reproducible)
- Hard to analyse interaction among concurrent operations

Transactional Memory

"Lock-free", easy to implement concurrent operations

Concepts of transactional memory

- Shared data access is wrapped into **transactions**
- A transaction is an **atomic block**
- Concurrent access is performed inside transactions
- Transactions are executed speculatively
- Can be implemented in STM (*e.g.* TinySTM, SwissTM...), HTM and HyTM

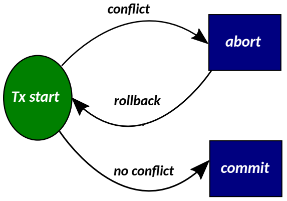


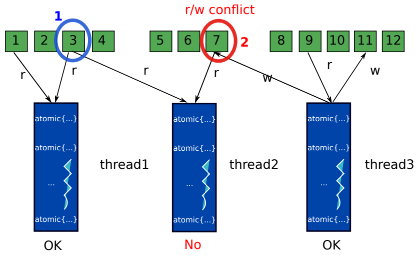
Transactional Memory

a transaction can either commit or abort, **no intermediate status**

Three concepts

1. Commit: a transaction succeeds — changes are made to memory
2. Abort: a transaction has a conflict — changes are discarded
3. Rollback: re-execute the aborted transactions





Transactional memory

Example

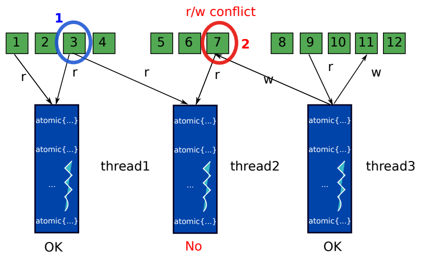
- Three threads access data from different memory locations
- Threads execute transactions

Case1

thread1 reads object3
thread2 reads object3

Case2

thread2 reads object7
thread3 writes object7



How to deal with conflicts in TM

When conflicts happen

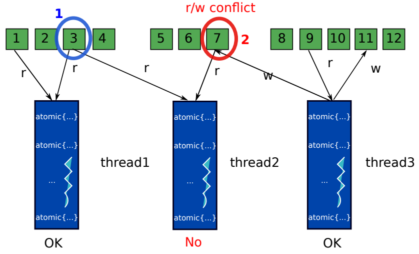
Many designs to solve conflicts: abort others, abort itself...

Case1

Thread1 Thread2 progress

Case2: resolving conflicts

one transaction aborts
one transaction continues

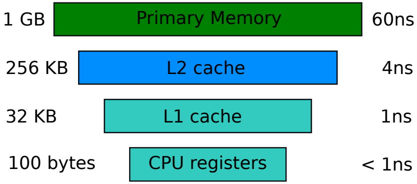


Multi-core processors also bring complexity

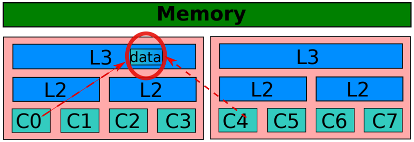
Data access latency differs

- Data access latency to different levels of memory differs
- Data access latency differs from one core to another

How to place threads on cores to better use resources?



Example of access latency



Example of multi-core processor

Thread mapping

Concept: assign threads to specific cores

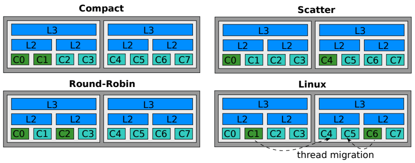
1. **Better use** of interconnects: using intra-chip interconnects
2. **Reduce** invalidation misses:
   reduce misses caused by two private caches:
   - holding the same data
   - and continuously invalidating each other
3. **Reduce** compulsory misses:
   caused by competition for the same cache.

Thread mapping strategies

Thread mapping: assign threads to specific cores

We map threads to different cores to gain better performance

1. **Compact** threads are physically placed on sibling cores
2. **Scatter** threads are distributed across different processors
3. **Round-Robin** threads share higher level of cache (*e.g.* L3) but not the lower ones
4. **Linux** default thread scheduling based on priorities



thread mapping strategies[1]
[1]: courtesy of *Castro*

Restrictions of HPC systems:

- Restrictions from TM systems
  - TM sytems are complicated and incorporate numerous tunable parameters (*e.g.* parallelism degrees)
  - these parameters of TM are usually set offline
  - few actions can be made to adapt them to runtime behaviour
- Hardware complexity brings more tunable parameters

Autonomic computing techniques are capable of:

- Monitoring behaviour dynamically
- Tuning parameters accordingly to ameliorate performance

Autonomic computing

autonomic computing techniques enable systems to be self-adaptive

This concept is first proposed by IBM in 2003

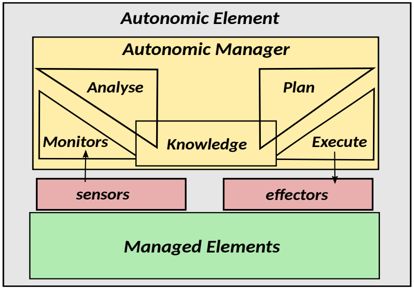
Autonomic control systems have at least one of the properties:

- **Self-optimisation:** seek to improve performance & efficiency
- **Self-configuration:** a new component learns the system configurations
- **Self-healing:** recover from failures
- **Self-protection:** defend against attacks

Autonomic computing

Elements of a feedback control loop:

1. Managed element: any software or hardware resource
2. Autonomic manager — a software component: monitor, plan, knowledge
3. Sensor: collect information
4. Effector: carry out changes

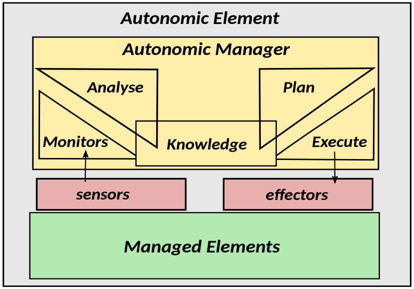


A feedback control loop

Autonomic computing

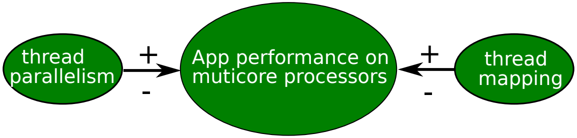
Components of the autonomic manager:

1. Monitor: sampling
2. Analyser
3. Knowledge
4. Plan: use the knowledge of the system to do computation
5. Execute: make changes



A feedback control loop

What we want to address



Our work considers two factors that can impact on performance

Alas...

- Difficult to set a parallelism degree offline
- Difficult to decide thread locations offline

Especially for applications with online behaviour variations

Autonomic computing techniques can tackle the above issues

Outline

Problems of control both parallelism and thread mapping

1. Is thread mapping necessary?
   e.g. when tn<core number, mapping can be performed
2. The order of decisions: parallelism or thread mapping first?
3. If parallelism changes, its mapping strategy differs?
4. Frequency of changing thread mapping strategy?
   Performance lost by thread migration

The solutions

1. Predict parallelism degree first
2. Predict the thread mapping strategy
3. Mapping is re-profiled when
   - parallelism degree varies
   - parallelism degree $\leq$ half of core number

The reasons

1. The maximum parallelism degree requires no thread mapping (by *Castro2012*)
2. High parallelism degree may give unstable behaviour regardless of mapping (based on experimental observation)
3. Influence of parallelism is more significant (based on experimental observation)

Metrics

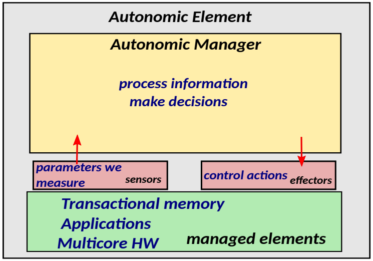
What we measure

1. parameters: commits, aborts, time
2. **commit ratio** (CR)= commits/(commits+aborts), **throughput**=commits/time

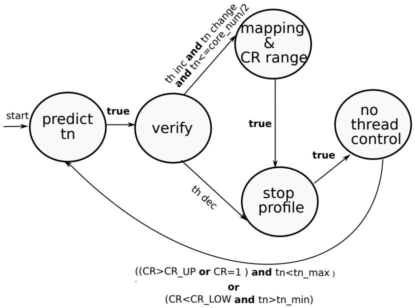
CR range : detect phase change

- CR fluctuates within the range during the same phase
- High CR low contention, high throughput fast execution

The feedback control loop



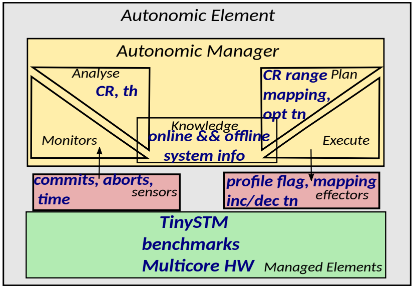
The feedback control loop
–overview of system architecture


start
predict tn
true
verify
th inc and tn change and tn<=core_num/2
mapping & CR range
true
th dec
stop profile
true
no thread control
((CR>CR_UP or CR=1) and tn<tn_max)
or
(CR<CR_LOW and tn>tn_min)

The structure of autonomic manager in an automaton shape

The feedback control loop

1. Control objectives: maximise throughput and reduce application execution time
2. Inputs: commits, aborts, time
3. Outputs: inc/dec parallelism degree, set thread mapping strategy

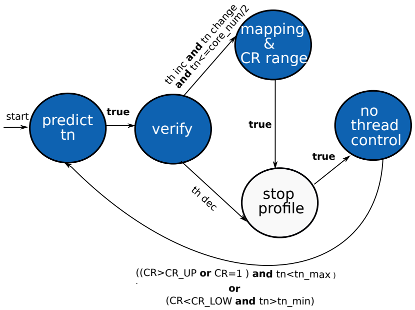


The feedback control loop in MAPE-K shape

Four groups of decision functions

Decision functions of:

1. Parallelism
2. Thread mapping strategy
3. CR range (phase detection)
4. Thread profile


start
predict tn
true
verify
th inc and tn change and tn<=core_num/2
mapping & CR range
true
th dec
stop profile
true
no thread control
((CR>CR_UP or CR=1) and tn<tn_max)
or
(CR<CR_LOW and tn>tn_min)

The structure of autonomic manager

Parallelism decision functions: two models

Simple model

- No action unless a phase change is detected
- Keep inc/dec tn by one until throughput decreases
- Inc tn when $CR > cr\_up$; dec tn when $CR < cr\_low$

Probabilistic model

Directly compute the near-optimum parallelism degree

Parallelism decision function: probabilistic model

The number of transactions N executed during L_0:

$$N = \frac{L_0}{L} \cdot n = \alpha \cdot n \tag{1}$$

- within a fixed period L_0 (N=aborts+commits)
- assuming the average length of transactions is L

The probability of a commit:

$$Pr = (1-p)^{(N-\frac{N}{n})} = q^{(N-\frac{N}{n})} \tag{2}$$

p (independent of n): probability of a conflict **between two txs**;
n is parallelism degree.

Parallelism decision function: probabilistic model

$$Pr = (1-p)^{(N-\frac{N}{n})} = q^{(N-\frac{N}{n})} \tag{2}$$

Equation (2) relies on two assumptions

- The same amount of transactions (txs) are executed in each active threads during a fixed period
- Enough amount of tx are executed making the probability of a conflict between two txs approaches a constant

Parallelism decision function: probabilistic model

$$P(X_i = 1) = q^{(N-\frac{N}{n})} = q^{\alpha(n-1)} \tag{3}$$

- $X_i = 1$ commit, $X_i = 0$ aborts
- T is a random variable, $T = \sum X_i$
- T follows a binomial distribution: $B(N, q^{(N-\frac{N}{n})})$

Hence the expected value of T is:

$$E[T] = N.q^{(N-\frac{N}{n})} = \alpha n q^{\alpha(n-1)} \tag{4}$$

$$T(n) = \alpha \cdot n \cdot q^{\alpha(n-1)} \tag{5}$$

- T is throughput, $\alpha = \frac{L_0}{L}$
- CR is the commit ratio

Parallelism decision function: probabilistic model

$$T'(n) = \alpha q^{\alpha(n-1)} + \alpha^2 n q^{\alpha(n-1)} \ln(q) = 0 \tag{6}$$

$$q = CR^{\frac{1}{\alpha(n-1)}} \tag{7}$$

Hence the optimum parallelism degree:

$$n_{opt} = -\frac{n-1}{\ln CR} \tag{8}$$

- n_{opt} is **optimum parallelism**
- n is the active thread number
- CR is the commit ratio

Two CR range decision functions

Simple phase detection algorithm

1. Obtain the optimum tn
2. Obtain CR when inc/dec optimum tn by one

Advanced phase detection algorithm:

1. $cr\_up = cr\_opt + cr\_opt * \delta$
2. $cr\_low = cr\_opt - cr\_opt * \delta$

cr_opt : CR generated by the optimum tn and mapping
δ inc 1% when a false phase change happens

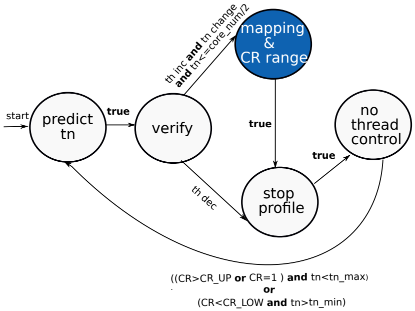
Thread mapping decision function

Conditions for mapping strategies

- tn $\leq$ half the core number
- tn has changed

Obtain the optimum mapping

- Profile each strategy
- Optimum strategy: generates the highest throughput


start
predict tn
true
verify
th inc and tn change and tn<=core_num/2
mapping & CR range
true
th dec
stop profile
true
no thread control
((CR>CR_UP or CR=1) and tn<tn_max)
or
(CR<CR_LOW and tn>tn_min)

The structure of autonomic manager

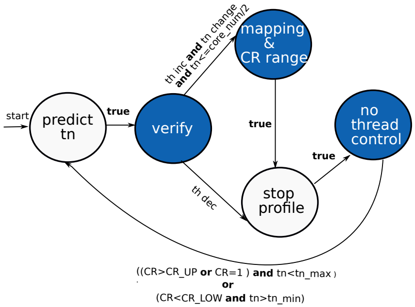
Thread profile decision function

stop thread profile

- Current throughput ≤previous throughput? set back old tn
- After thread mapping and CR range decision

start thread profile

CR falls out the CR range ?


start
predict tn
true
verify
th inc and tn change and tn<=core_num/2
mapping & CR range
true
th dec
stop profile
true
no thread control
((CR>CR_UP or CR=1) and tn<tn_max)
or
(CR<CR_LOW and tn>tn_min)

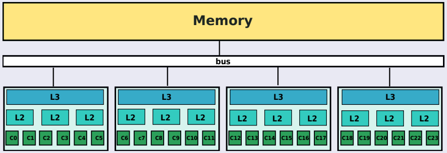
The structure of autonomic manager

Outline

Experimental platforms

Configurations for the UMA

number of cores	24
number of sockets	4
clock (GHz)	2.66
L2 cache capacity (KB)	3072 (each)
L3 cache capacity(MB)	16 (each)
DRAM capacity (GB)	64

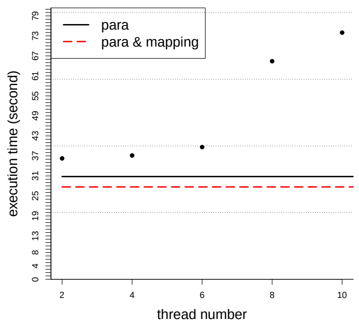


STM and benchmarks for evaluation

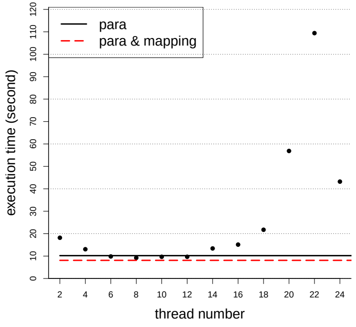
- STM: TinySTM
- Benchmarks:
  - **EigenBench**: artificial but highly configurable
  - **STAMP**: generic but less configurable (include: **yada** *etc.*)

Time comparison with static parallelism

average of 10-time executions
dot is execution time for static parallelism
our approaches outperform the static parallelism



EigenBench

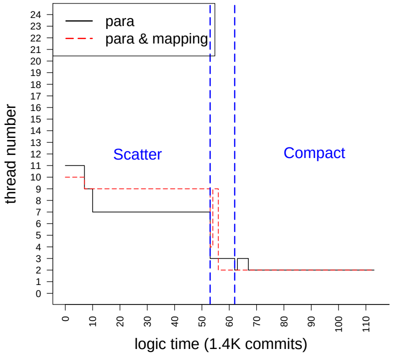


yada (from **STAMP**)

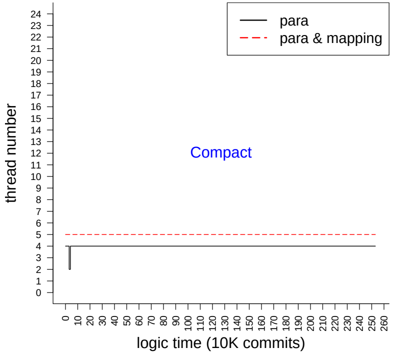
Runtime thread number and mapping strategy variation

runtime behaviour of one execution

applications with multiple phases show necessity of runtime para & mapping adaptation



EigenBench



yada (from **STAMP**)

How to verify the adaptive approaches

Throughputs indicate program progress

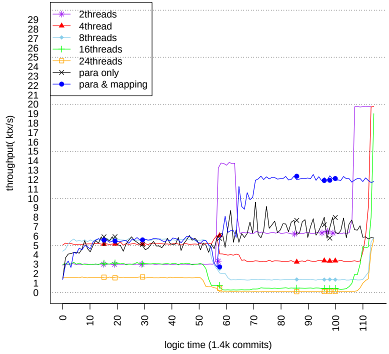
The throughputs by the autonomic control approaches:

- compare with max throughputs at each phase
- approach or exceed the max throughputs

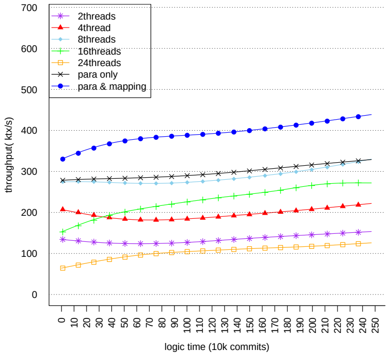
Online throughput variation

runtime behaviour of one execution

the throughputs from our methods **approach** or **exceed** those of the static parallelism



EigenBench



yada (from **STAMP**)

Discussion

Limitations

1. The probabilistic parallelism predictor is based on ideal situations.
   The prediction can be sub-optimum *de facto*.
2. The optimum mapping strategy can differ over several executions:
   - throughputs are affected by thread migration
   - some mapping strategies show similar performance

Outline

Related work on parallelism and mapping
—static and dynamic adaptation

1. Parallelism adaptation using control techniques (*Ansari2008*, *Rughetti2014*)
   - requires offline training procedure to obtain predictor
   - less sensitive to online behaviour change
2. Thread mapping adaptation (*Castro2012*, *Diener2010*)
   - dynamic mapping adaptation using machine learning on STM
   - offline searching for optimum mapping on non-TM platforms
3. Coordination of parallelism and adaptation
   - no previous work on adapting both parallelism & mapping dynamically for TM systems
   - some studies show insights into offline adaptation for non-TM platforms (*Wang2009*, *Corbalan2000*)

Outline

Conclusion

1. Introduce feedback control loops to dynamically control:
   - parallelism degree
   - thread mapping
   - coordination of the above
2. Propose:
   - two models for sub-optimum parallelism detection
   - two phase detection algorithms
3. Compare the performance of static parallelism with:
   - adjusting parallelism only
   - adjusting parallelism and thread mapping together

Future Work

1. Thread mapping
   - new mapping strategies
   - prediction of mapping strategies with assistance of compilers
2. From STM to HTM
3. Coordination of multiple feedback control loops
4. Feedback control loops for other parallel platforms (*e.g.* OpenMP)

Publications

1. N. Zhou, G. Delaval, B. Robu, É. Rutten, and J.-F. Méhaut, *Autonomic Parallelism and Thread Mapping Control on Software Transactional Memory*, in ICAC 2016 - 13th IEEE International Conference on Autonomic Computing, (Wursburg, Germany), July 2016.
2. N. Zhou, G. Delaval, B. Robu, É. Rutten, and J.-F. Méhaut, *Control of Autonomic Parallelism Adaptation on Software Transactional Memory*, in International Conference on High Performance Computing & Simulation (HPCS), (Innsbruck, Austria), July 2016. (**Outstanding Paper Nomination**)
3. N. Zhou, G. Delaval, B. Robu, É. Rutten, and J.-F. Méhaut, *Autonomic Parallelism Adaptation for Software Transactional Memory*, in Conférence d'informatique en Parallélisme, Architecture et Système (COMPAS), (Lorient, France), July 2016.
4. N. Zhou, G. Delaval, B. Robu, É. Rutten, and J.-F. Méhaut, *Autonomic Parallelism Adaptation on Software Transactional Memory*, Research Report RR-8887, Univ. Grenoble Alpes ; INRIA Grenoble, Mar. 2016.