Transparent and *Adaptive* Computational Block Caching for Multi-Agent-based Simulation on a PDES Core

Yin Xiong, Maria Hybinette, Eileen Kraemer

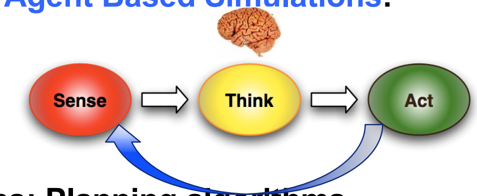
Computer Science Department

The University of Georgia

Target Applications: Variable Computation Period

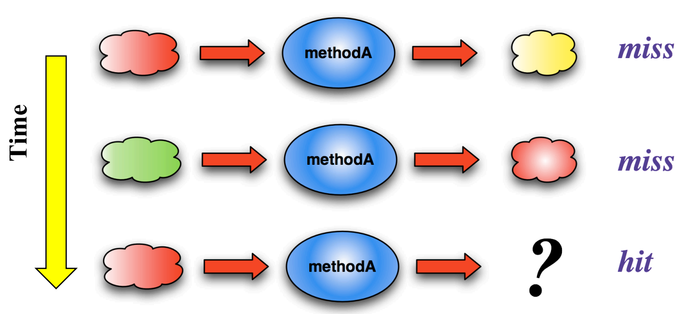
- **Problem**: Inefficiencies in Agent Based Simulations: Redundant computations
- **Observations**:
  - » Computations Repeat
    - – Cyclic systems
  - » Expensive computations: Planning algorithms (e.g., A* deliberates for 10 ms - 1,000 ms on a 2 GHz Pentium).
  - » Classic caching: Hide disk access cost: KNN
- **Main Goal:**

  Increase efficiency by reusing computation



General Approach

- Cache computations and re-use when they repeat instead of re-compute (Function Caching).



Input Transformation Output

Factors Affecting Benefit of Caching

- Cache size
- Cost of consulting & updating the cache
- Execution time of the computation
- Probability of a hit: Hit rate

$$E(Cost_{use\_cache}) = hit\_rate * Cost_{lookup\_hit} + (1 - hit\_rate) * (Cost_{lookup\_miss} + Cost_{computation} + Cost_{insert})$$

Effective Time

Caching is *Not Always a Good Idea*

$E(Cost_{use\_cache}) =$

$hit\_rate * Cost_{lookup\_hit}$

$+ (1 - hit\_rate) * (Cost_{lookup\_miss} + Cost_{computation} + Cost_{insert})$

- Low hit rate
- Very *fast* computations (e.g., many PDES computations).
- Only when $Cost_{use\_cache} < Cost_{computation}$ is caching worthwhile

How Much Speedup is Possible?

Neglecting cache warm up and fixed costs

Expected Speedup = $Cost_{computation} / Cost_{use\_cache}$

Upper bound (hit_rate = 1)

= $Cost_{computation} / Cost_{lookup}$

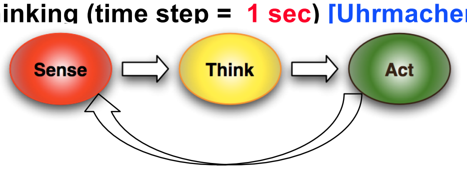
In our experiments $Cost_{computation} / Cost_{lookup}$ = ~1- ~10

1.68 ms (experimental threshold when it is worthwhile) - 16 ms

On the Computational *Granularity* of Agent Based Computations.

Observations:

- Many agent based systems assume a time step of 33 msec (video output frequency) e.g., Player/Stage [Gerkey et. al 2005].
- Typically 'thinking' time is computationally intensive
  - » Example: A* [Hart et. al. 1968], a classic (and well used) planning algorithm - overhead ranges between 10 msec – 1,000 ms on a 2 GHz Pentium) [Balch 2008].
  - » Lees et al. 2004 use a 10 msec deliberation delay in their experiments (to emulate a planning period).
- Other researchers report similar overhead, e.g., 80% of an agents time step was spent on thinking (time step = 1 sec) [Uhrmacher 2000].



Adaptive Caching

- **General Observation**: Independent of the particular deliberating / planning algorithm, the expected length of the computations in ABSs are *variable* in time.
- **Solution:** Use adaptive caching –
  - » for lengthy computation avoid re-computation by using a cache (e.g., deliberative agents).
  - » for short or finer computations (e.g., *fast* reactive agents) avoid caching computations.
  - » for medium computations: reactive 'schemas' that are amenable to caching –manipulating sensor information *across* 'motor schemas' e.g., an agent reacts to a stimulus in different schemes.
    - – Input may be 'nondeterministic' initially – but the same input is analyzed many times across motor schemas/

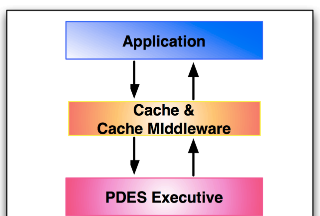
Summary of Key Ideas

- Only cache when it is worthwhile avoid caching when it is not.
  - » Avoid
  - » Cluster
- Use an on-line pre-processor to monitor computations.
- Maximize transparency.

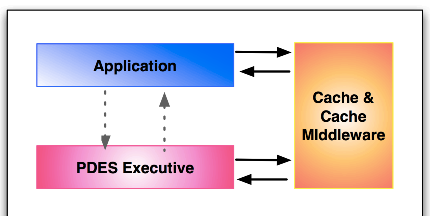
Problems:

- **Problems**: What about a large input space, random input variables, and time stamps (do not repeat often)?
- **Solution:** Enable breaking the computations into smaller units or blocks, we call it **Block Caching.**

Previous vs. New Approach



LP Caching



Block Caching

- **Earlier Approach**: Exploited the PDES paradigm (messages (intercepted) at the logical process level) [Chugh & Hybinette 2004]. Simulation dependent.
- **New Approach**: Simulation *Independency*.

Overview of Adaptive Caching

Execution time:

1. Warm-up execution phase, for each function:
   a) **Monitor**: hit rate, query time, function run time
   b) **Threshold**: Determine utility of using cache
2. Main execution phase, for each function:
   a) Use cache (or not) depending on results from 1
   b) Continue to randomly sample: hit rate, query time, function run time
      - » Revise decision if conditions change

Cacheability

- Methods (do not need to be annotated)
- Blocks (need to be annotated)

Example Block

```
int a;
int b;
methodA( a, b, c, d ) ;
if ( c > d )
  doSomething( c ) ;
else
  doSomethingElse( d );
```


Example

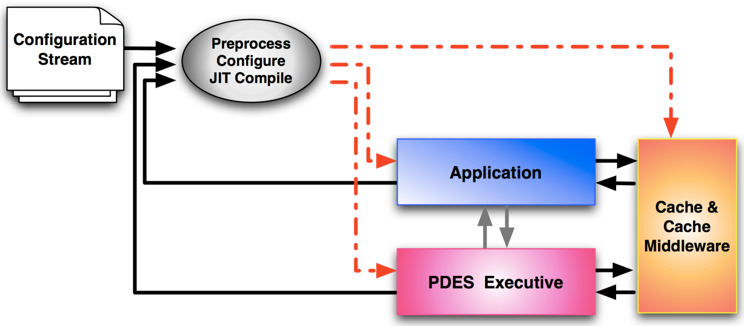
```
// beginComputationBlock dummy2 **
int a;
int b;
methodA( a, b, c, d ) ;
if ( c > d )
  doSomething( c ) ;
else
  doSomethingElse( d  );
// endComputationBlock dummy2
```

** Annotation only needed for *blocks* not methods

Example Configuration Stream

```
begin:dummy1
packageName: app
className: JPHold
return: length=double, point=int
Parameters: int a, double b
StateVariables: int height, int age
cachingFlag: on
end:dummy1
```

On-the-Fly Configuration



- Preprocessor reads configuration file (or stream snippet)
- Rewrites (re-generates) & recompiles effected code / objects *on-the-fly*
- Regenerated code enabled in middleware

Statistical Manager

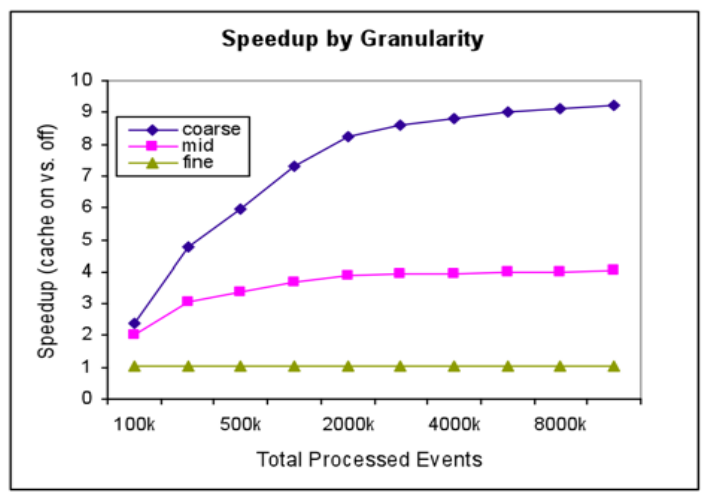
Preprocesses: Analyzes blocks:

- » runs each block with a range of input parameters.
- » Determines threshold when it is worthwhile to cache or not.
- » Whether blocks are adaptive or not (soft (adaptive), hard (not adaptive).

Experimental Platform

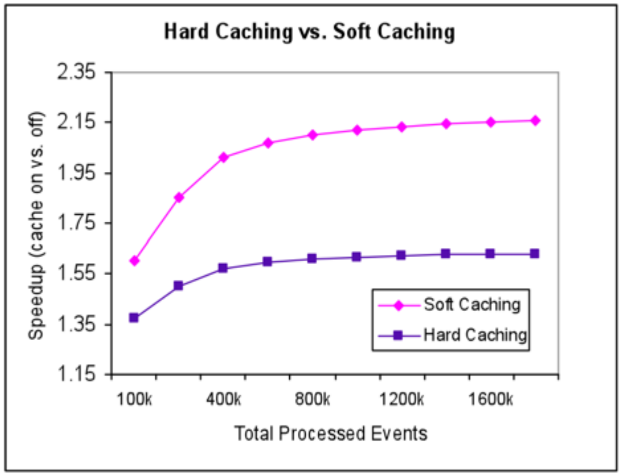
- Tested Caching in the SASSY framework– Scalable Agent Simulation SYstem – Java Based Optimistic Simulation System with an optional Agent Based API.
- Benchmark: JPHold (with added 'thinking' computation, fibonacci – to vary the computational load and assess the cost of accessing cache).
- 10 Machines
- 40 PEs
- 1000 LPs

Cache-on vs Cache-off (assess the threshold)



- 16 ms
- 6.5 ms
- 1.68 ms
- 1.5 ms

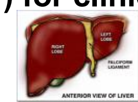
Adaptive (soft) Caching



- Max: 2.64 ms

Future Work

- Nested Functions or Blocks
- Various planning algorithm and input parameters.
- Mix reactive and deliberative agents.
- Tile World.
- Ant / Bee Models (motor schemas given a certain input share information).
- Lung Cancer & Liver Cell Cancer Models (Deisboeck & Wang, Harvard-MIT) for clinic predictions.




Related Work

- **Function Caching**: Replace application level function calls with cache queries:
  - » Introduced by: Bellman (1957); Michie (1968)
  - » Incremental computations:
    - – Pugh & Teitelbaum (1989), Liu & Teitelbaum (1995)
  - » Sequential discrete event simulation:
    - – Staged Simulation: Walsh & Sirer (2003) function caching + currying (break up computations), re-ordering and pre-computations) for network simulations (framework).
- **Simulation Cloning & Branching & Updateable Simulations:**
  - » Hybinette & Fujimoto (1998); Chen & Turner, et al (2005); Straβburger (2000), Peschlow, Martini & Liu (2008)

Updateable Simulations (Ferenci et al 2002)

- **Related Optimization Techniques**
  - » Lazy Re-Evaluation: West (1988)
- **LP Caching** (Chugh & Hybinette)

Summary of Key Ideas

- Only cache when it is worthwhile avoid caching when it is not.
- Use an on-line pre-processor to monitor computations.
- Maximize transparency.