Speculative Parallelization

Devarshi Ghoshal

Agenda

- Speculative Parallelization
- FastForward-A Speculation using Checkpoint/Restart
- System Design
- Software-based Speculation Systems
- Analysis
- Performance Benchmarks
- Current Status
- Future Work
- References

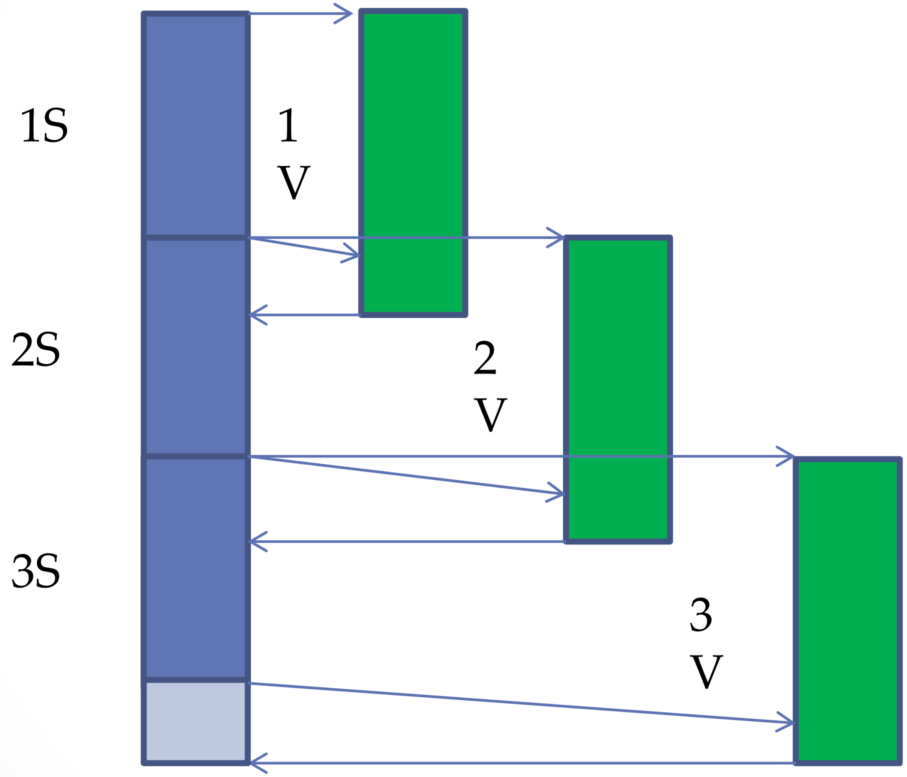
Speculative Parallelization

- A technique to execute loops, which cannot be classified as ‘parallel at compile time’, in parallel

- Writing with sequential semantics & letting the system figure out whether a region can really be parallelized safely

- In case of any dependency, the involved iterations are stopped and re-executed ‘in order’

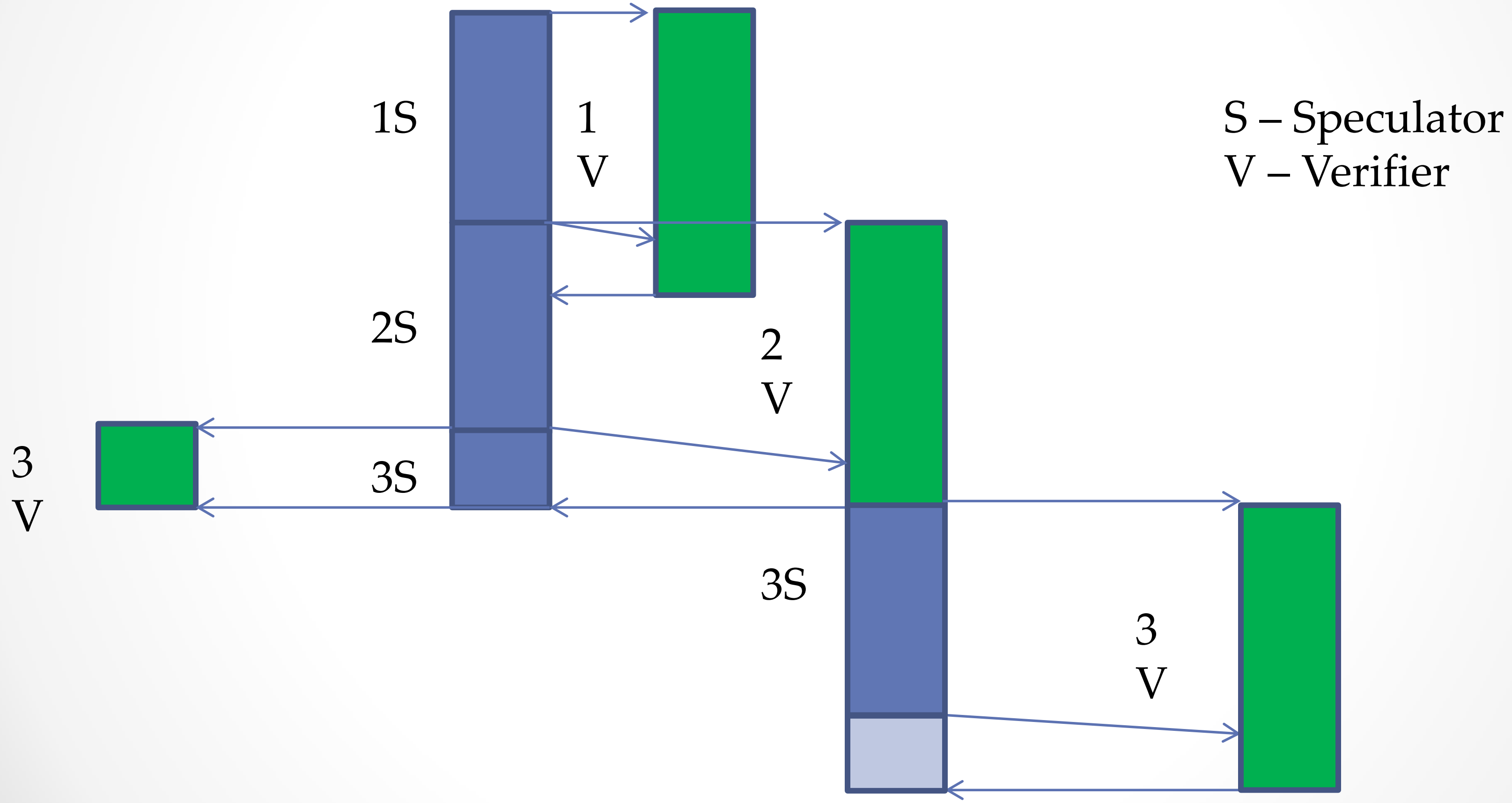
Execution Semantics-
Case 1: Correct Speculation



S – Speculator
V – Verifier

Case 1: All speculations are correct

Case 2: Incorrect Speculation



Case 2: Result of 2S is wrong

FastForward- Speculation using Checkpoint/Restart

- Checkpoint/Restart
  - Duplicate and unroll processes dynamically
- No intervention by the kernel
  - Everything occurs in user-space
  - Low overhead, maximum portability
- Distributed speculation over clusters
  - Using DMTCP
  - High interconnection networks for migrating and exchanging data

Example

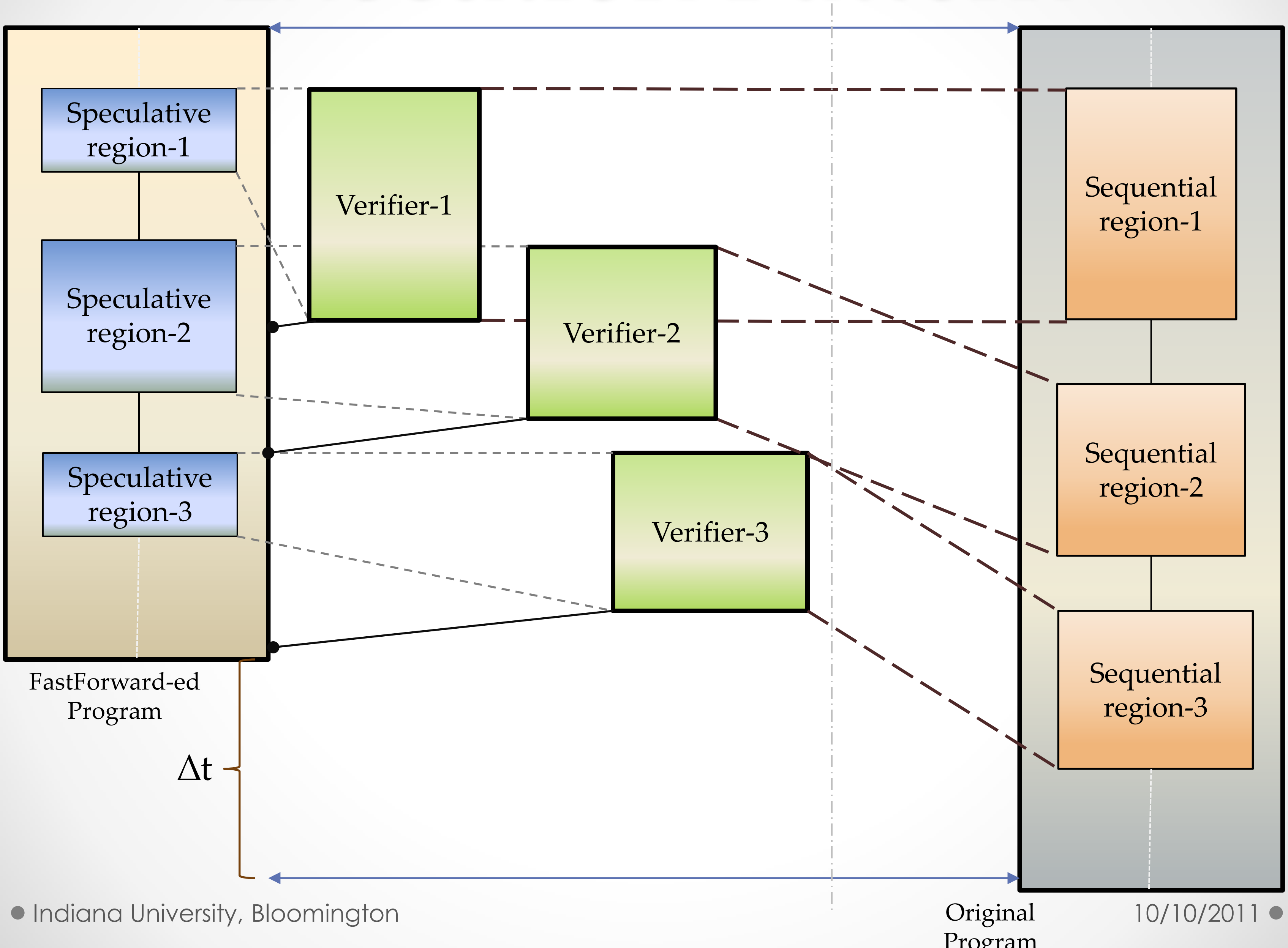
```
void foo(){
        double a[NUM_ELEMENTS];
        double r[NUM_ELEMENTS];
        int p[NUM_ELEMENTS];
        :
        for(i = 0; i < NUM_ELEMENTS; i++){
                a[p[i]] = compute_some_value();
        }
        :
        for(i = 0; i < NUM_ELEMENTS; i++){
                r[i] = use_value(a);
        }
        :
}
```

FastForward Transformed Program

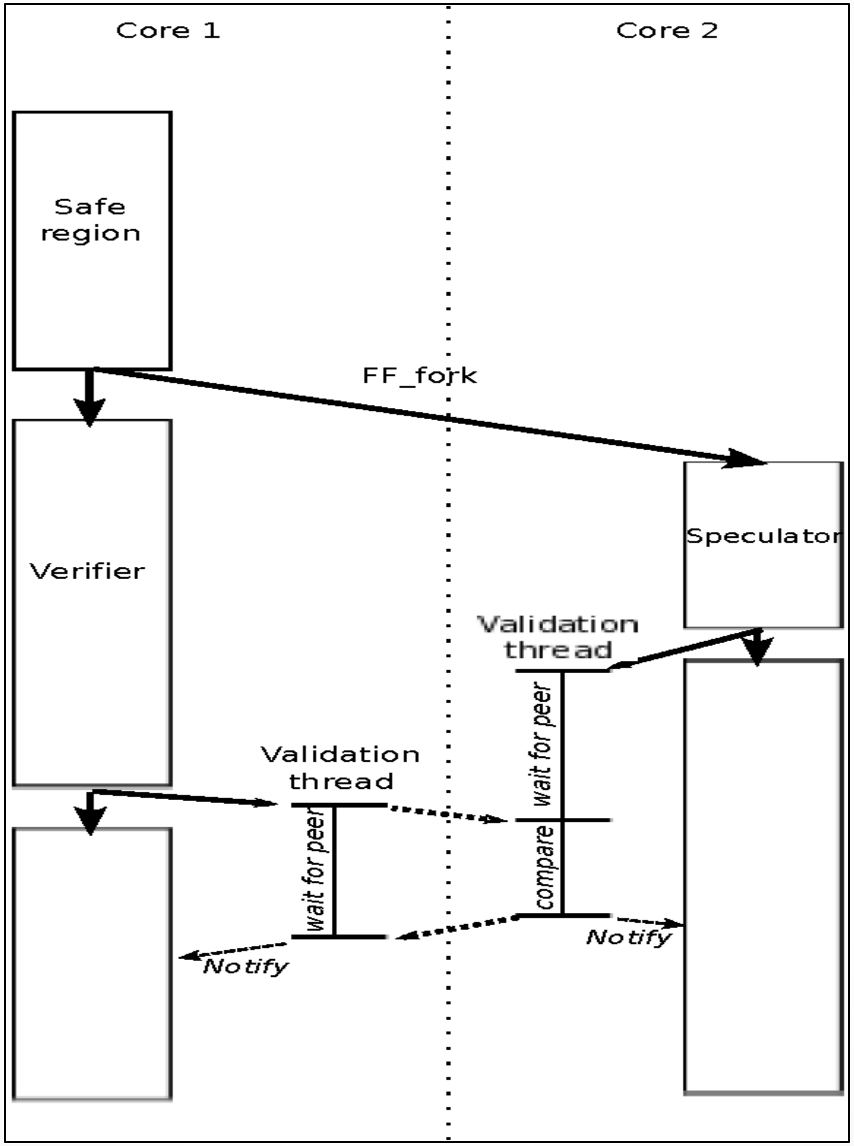
```
void foo(){
   double a[NUM_ELEMENTS];
   double r[NUM_ELEMENTS];
    int p[NUM_ELEMENTS];
    :
   for(i = 0; i < NUM_ELEMENTS; i++){
      a[p[i]] = compute_some_value();
   }
   :
   for(i = 0; i < NUM_ELEMENTS; i++){
      r[i] = use_value(a);
   }
   :
}
```

```
void foo(){
    :
    if((type=dmtcpCheckpoint()) == VERIFIER){
       for(i = 0; i < NUM_ELEMENTS; i++){
          a[p[i]] = compute_some_value();
       }
    }
    else{ // SPECULATOR
       #pragma omp parallel for
       for(i = 0; i < NUM_ELEMENTS; i++){
          a[p[i]] = compute_some_value();
       }
    }
    :
}
```

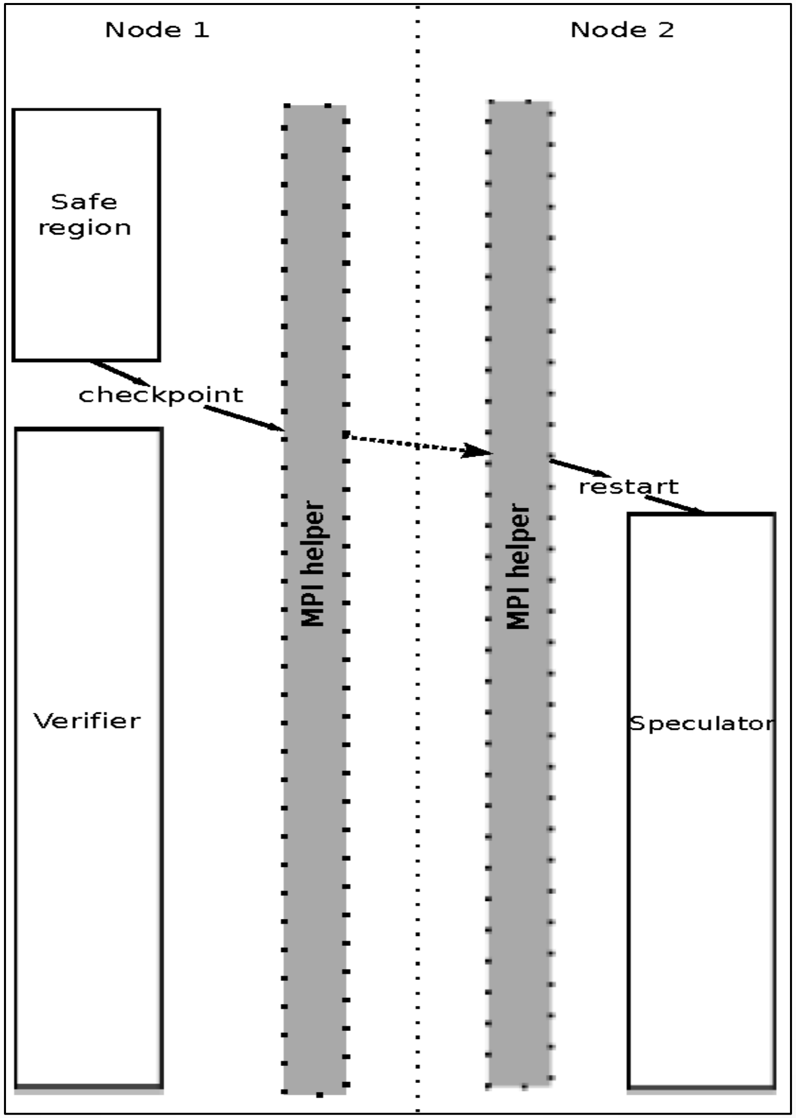
Execution Pattern



Intra-node FastForward

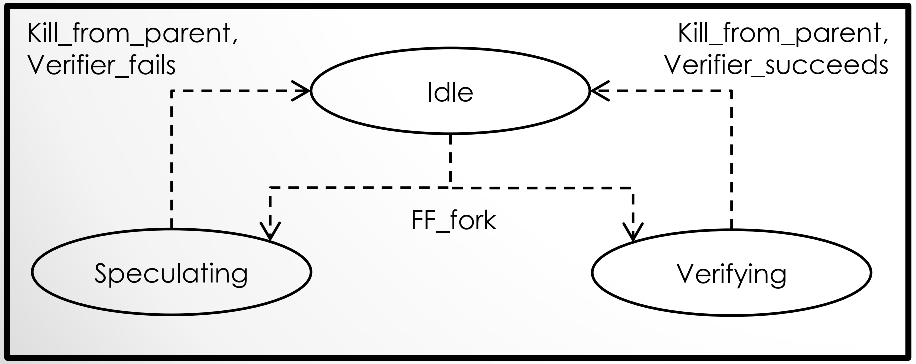


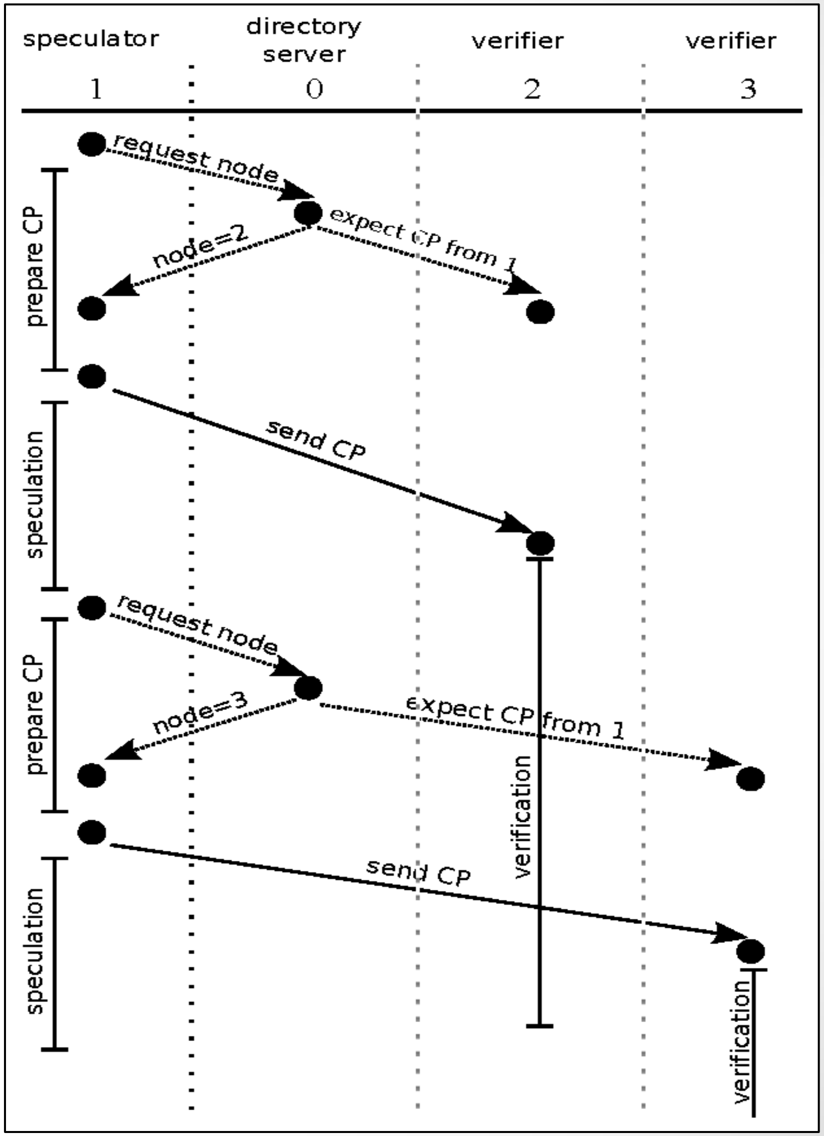
Inter-node FastForward



Implementation

- Directory-Service
- MPI-Helper threads:
  - Remote-communication Thread
  - Local-communication Thread
- Multi-level speculation
- One speculator, many verifiers model





Software-based Speculation Systems

- FastTrack
- Software Behavior Oriented Parallelization
- Transaction Memory

Fast Track

- Creates dual track regions which involves code that can be run speculatively
- Runs unoptimized code parallely (against sequential version) on multiple processors
- Checks correctness after sequential version is executed
- Proceeds with speculative version if results are correct / sequential version otherwise

Loop Semantics

```
while (...) {
  ...
  if (FastTrack ()){
   /* unsafely */
   /* optimized */
   fast_fortuitous();
  }
  else {
   /* safe code */
   safe_sequential();
  }
  EndDualTrack();
  ...
}
```

Function Semantics

```
...
if (FastTrack ())
    /* optimized */
    fast_step_1();
else
    /* safe code */
    step_1();
...
if (FastTrack ())
    /* optimized */
    fast_step_2();
else
    /* safe code */
    step_2();
```

System Design

- Compiler support
  - Records changes made by both dual track regions
  - Compiler's inherent support for stack variables
  - Copy on write + access map for global & heap
- Run time support
  - Transfer pages of modified data using shared pipe
  - Compare memory state at the end of dual track region

System Limitations

- No Fault Tolerance
- Kernel Patching
- Limited Use of Resources
- Special care for all Program Termination Points inside Speculative Region

Software Behavior Oriented Parallelization

- Programmable software speculation
  - Program parallelized based on "partial" information about program behavior
  - User or analysis tool marks "possibly" parallel regions
  - Runtime system executes these regions speculatively
- Critical-path minimization
- Value-based correctness checking
- No change to the underlying hardware or operating system

System Design

- Possibly Parallel Regions (PPR)
  - Marking the start and end of the region with matching markers: BeginPPR(p) and EndPPR(p)
- Protects the entire address space by dividing it into possible shared and privatizable subsets
- The execution starts as the “lead” process
- Uses concurrent executions to hide the speculation overhead off the critical path
- At a (pre-specified) speculation depth k, up to k processes are used to execute the next k PPR instances

State Isolation

- Thread-based systems
  - Weak isolation
  - The updates of one thread are visible to other threads
- BOP
  - Strong isolation
  - The intermediate results of the lead process are not made visible to speculation processes until the lead process finishes the first PPR
  - Strong isolation comes naturally with process-based protection

System Limitations

- No Fault Tolerance
- Limited Use of Resources
- Extra care for handling of “lead” process
- Validation Overhead
  - Three types of data protection
    - Page-based protection of shared data
    - Value-based checking
    - Likely private data

Amdahl's Law

- Used to find Speedup for some enhancement

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \dfrac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

- Fraction(enhanced) - The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement
- Speedup(enhanced) - The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program

Mathematical Analysis

The maximum speedup, S, is given by:

$$S = \frac{t}{T_s}$$

Speculation-enabled computation time,

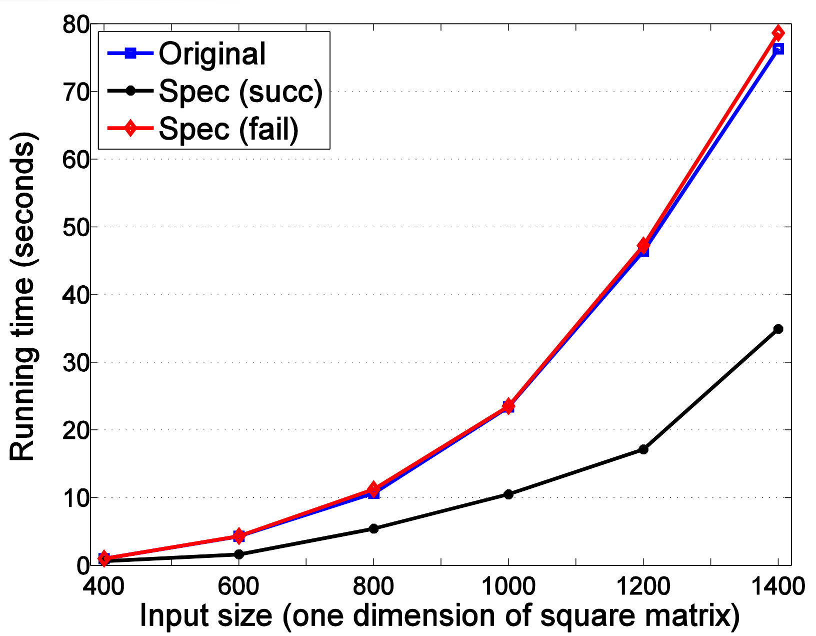
$$T_s = T + \frac{pkT}{s} + (1-p)kT$$

Total running time of the original code, $t = T(k+1)$

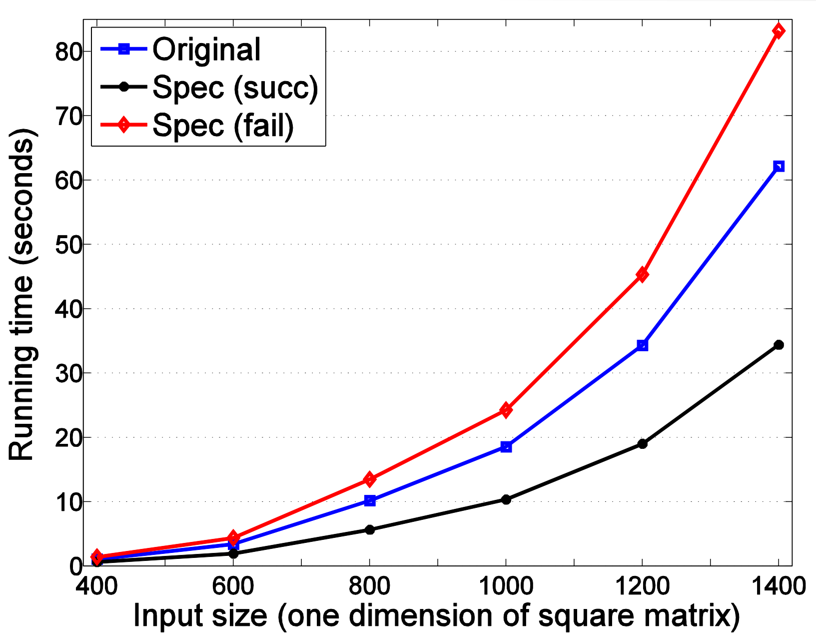
where,
$k: number\ of\ speculative\ regions$
T: time to execute each region
s: speedup of each speculative region
p: probability of each successful speculation
and, there is 1 non-speculative region

Performance Benchmarks

- Intra-node FastForward



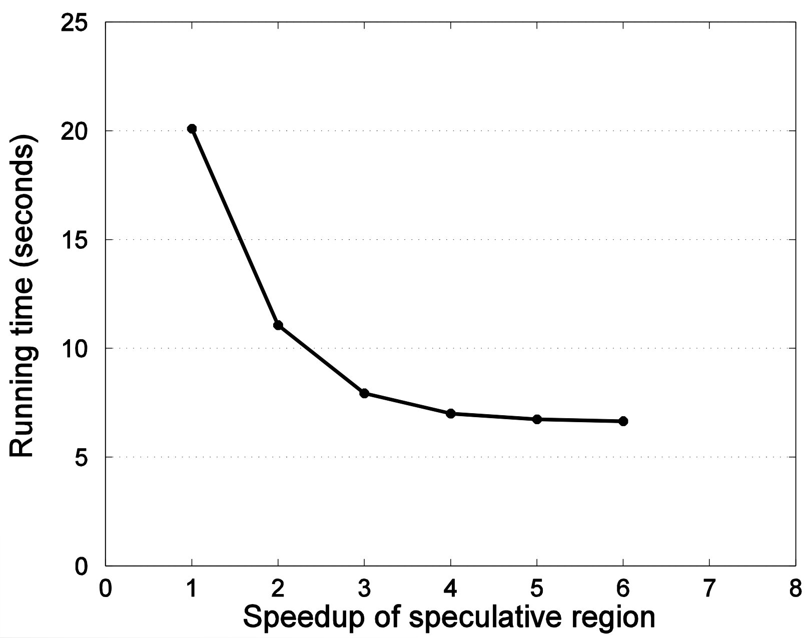
Using Shared-memory



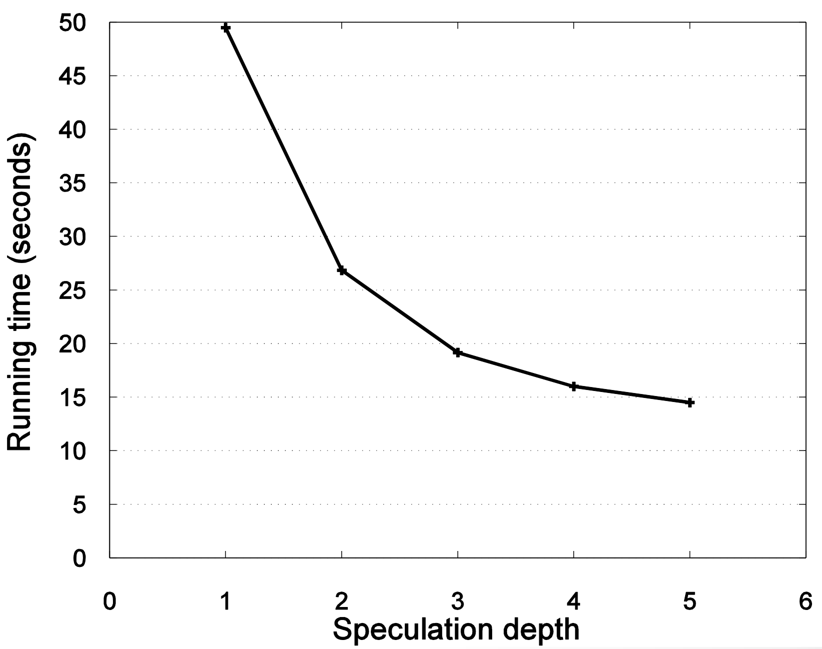
Using Named-pipes

Performance Benchmarks (cont….)

- Inter-node FastForward

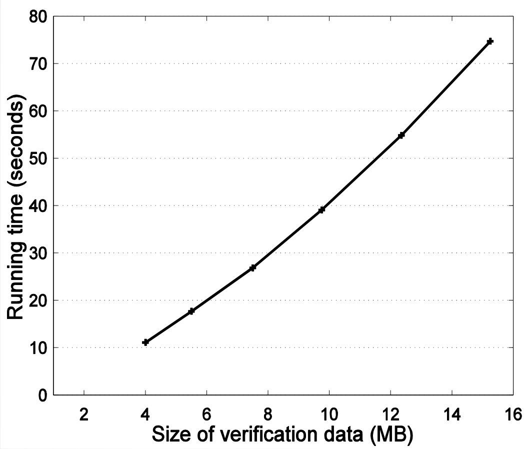


Varying the speedup of speculative version over non-speculative

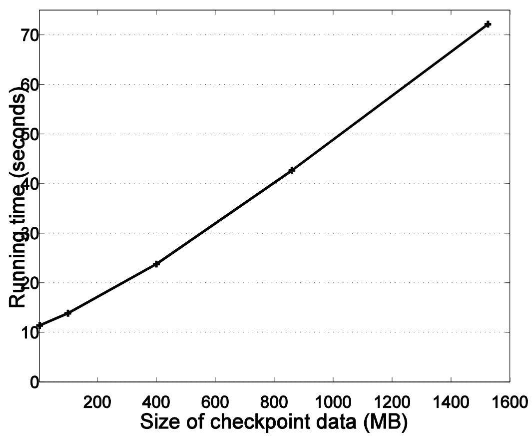


Varying the available depth of speculation

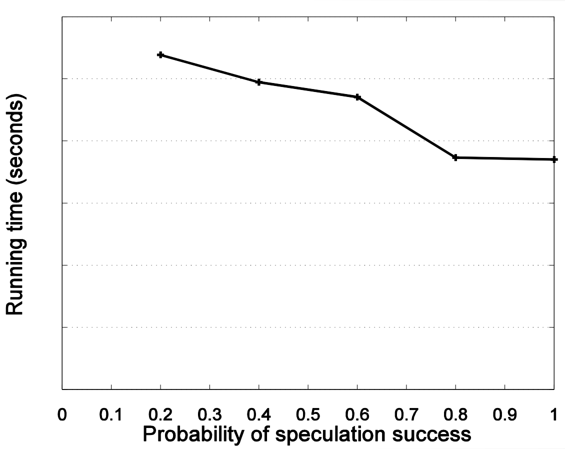
Performance Benchmarks (cont….)



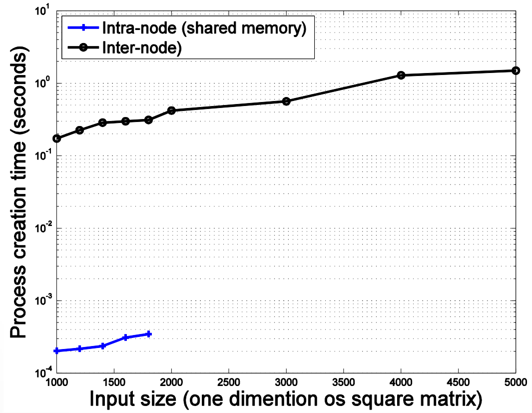
Varying the size of data to be compared for verification



Varying the size of checkpoint data



Varying the probability of speculation success



Advantages over Existing Systems

- No Parent-Child Process Relationship
- Inherent Fault Tolerance
- Efficient Use of resources in a Cluster

Current Limitations

- Local communication using pipes
- Reading check-pointed data through NFS
- Only supports basic data-types
- High Energy Usage

Future Work

- Extending the system to support recursive data-structures and memory references

- Optimizing Implementation
  - Shared memory implementation for inter-node FastForward
  - Incremental checkpointing
  - Checkpointing into memory

- Extending support to higher-level scripting languages

References

- Devarshi Ghoshal, Sreesudhan R Ramkumar, and Arun Chauhan. Distributed Speculative Parallelization using Checkpoint Restart. In Proceedings of the International Conference on Computational Science (ICCS), 2011.
- L.-L. Chen, Y. Wu, Aggressive compiler optimization and parallelization with thread-level speculation, in: International Conference on Parallel Processing (ICPP), 2003
- K. Kelsey, T. Bai, C. Ding, C. Zhang, Fast Track: A software system for speculative program optimization, in: Proceedings of the International Symposium on Code Generation and Optimization, 2009, pp. 157–168
- N. Shavit, D. Touitou, Software transactional memory, in: Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing, 1995, pp. 204–213
- J. Ansel, K. Arya, G. Cooperman, DMTCP: Transparent checkpointing for cluster computations and the desktop, in: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2009
- C. Tian, M. Feng, V. Nagarajan, R. Gupta, Speculative parallelization of sequential loops on multicores, International Journal of Parallel Programming 37 (5) (2009) 508–535

Questions & Clarifications