Value Prediction in Parallel Architectures

Jean-Luc Gaudiot

PASCAL: PArallel Systems and Computer Architecture Lab.

University of California, Irvine
Agenda

- The Problems of Modern Parallel Processing
- Architectures and Models of Execution
- Parallelism in Modern (and Future) Applications
- A Theoretical Framework of Value Prediction in Parallel Systems
- Implementation of Value Prediction Techniques
- Conclusions and Future Work
New Architectures

- Multiple processors on a chip
  - SUN Niagara: chip-multi-threading machine, 8 cores, each supports 4-way SMT
  - IBM Cyclops: 160 thread units on a chip
  - Intel Teraflop: 80-core-on-a-chip

- How do we harness the computing power from these machines?
  - Do application programs contain enough parallelism for many-core designs?
  - Can we exploit more than the intrinsic parallelism?
An interesting approach …

- Problems
  - High communication latency (~100s cycles) in consumer-producer parallelism
  - The consumer thread often has to idle to wait for the producer to complete
  - Performance bottleneck

- Value Prediction
  - At specific synchronization points, the consumer runs ahead with a predicted value instead of waiting for the producer
  - With correct predictions, parallelism is increased
  - With incorrect predictions, the consumer thread needs to rollback and re-execute

- In this talk, we will discuss
  - The parallelism contained in modern (and future) applications
  - The potential of value prediction techniques to enhance the performance of parallel systems
  - The implementations of value prediction techniques on various architectures
Agenda

- The Problems of Modern Parallel Processing
- Architectures and Models of Execution
- Parallelism in Modern (and Future) Applications
- A Theoretical Framework of Value Prediction in Parallel Systems
- Implementation of Value Prediction Techniques
- Conclusions and Future Work
The Evolution of General-Purpose CPUs

- **Single-Core CPUs**
  - Exploits ILP through superscalar architecture
  - Supports limited TLP with SMT design
  - Easy to program

- **Multi-Core CPUs**

- **Many-Core CPUs**
The Evolution of General-Purpose CPUs

- **Single-Core CPUs**

- **Multi-Core CPUs**
  - Usually two to four independent cores on a chip
  - Share cache and memory
  - Suitable for multitasking and low degree of parallel processing
  - Still relatively easy to program

- **Many-Core CPUs**
The Evolution of General-Purpose CPUs

- Single-Core CPUs

- Multi-Core CPUs

- Many-Core CPUs
  - Tens to hundreds of independent cores
  - On-chip network for communications
  - Distributed shared memory
  - Difficult to program: parallelization, scheduling, memory management, communications…
Alternate Architectures

- **GPU and GPGPU**
  - Specialized for graphic and math functions
  - Massively data parallel
  - Limited to one kind of parallelism

- **FPGA**
  - Reconfigurable to accelerate different application programs
  - Extend the host processor’s ISA

- **Heterogeneous Multicore**
  - General-purpose CPU + special-purpose accelerators
  - IBM Cell Processor
    - One power processor: control and sequential processing
    - Eight special computation elements: parallel processing
Synchronization Techniques

- Ensure the correctness of parallel programs
  - But there may be a high overhead that seriously limits performance (remember Amdahl’s law)…

- Ideal synchronization model for many-core systems
  - *Fine-grain*: the degree of parallelism which can be exploited is limited by synchronization granularity
  - *Low-latency*: synchronization latency should not dominate execution time
  - *Contention-free*: efficiency in performance, resource utilization, and power consumption
  - *Scalable*: independent of number of cores in the system
Synchronization State Buffer

- **Observation**: only a small fraction of memory locations are actively participating in synchronization at any instance of parallel execution
- **SSB**: a buffer which records and manages the states of frequently synchronized data
- **Fine-grain synchronization**: one buffer line is allocated for each word under synchronization
- **Scalable**: each memory bank contains a synchronization buffer
- **Elimination of busy-waiting**: split-phase operations for producer-consumer type of synchronization
Existing Problems and one Potential Solution

- Problems:
  - Synchronization techniques help exploit intrinsic parallelism but they don’t “create” parallelism
  - Synchronization techniques are used to guarantee the correctness of parallel execution
  - Modern (and future) applications may not contain enough parallelism to run on many-core designs

- Our Solution: Value Prediction
  - Exploits data redundancy to “generate” extra parallelism
Value Prediction

- Observations
  - The current data has a high probability of being used again in the near future (data locality)
  - Simplest form of predicted sequence: loop counter, if current loop counter is $n$, then the next value is likely to be $n+1$
  - Short sequences of data tend to repeat during program execution, if there is a data sequence pattern seen before, then it is highly probable that the next data value can be predicted

- Basic Value Prediction Schemes:
  - Last Value Predictor (Sample data sequence: 1 2 3 3 3 3 3 4)
  - Increment Predictor (Sample data sequence: 1 2 3 4 5 6 7)
  - Finite Context Predictor (Sample data sequence: 1 0 -1 2 1 0 -1 2 1 0 -1 2)
Value Locality and Data Prediction
--Lipasti et al. ASPLOS 1996

- **Value Locality**: the likelihood of the recurrence of a previously-seen values.
- **Load Value Prediction**: full register values being loaded from memory are frequently predictable.
- **Implementation**: load value prediction table is used to predict the value being loaded from memory by associating the load instruction with the value previously loaded by that instruction.
- The table is indexed by the load instruction address
- Result: 3% ~ 21% improvements across various benchmarks:
  - 60% ~ 90% prediction accuracy observed
Two types of predictors:
- **Computational predictors** perform an operation on previous values to yield predicted next values.
- **Context Based Predictors** match recent history (context) with previous history and predict values based on observed patterns.

Simulations with unbounded prediction tables that are immediately updated using correct data values:
- No latency
- No storage overhead

Results on SPEC95 benchmarks:
- 56% ~ 91% prediction accuracy
- Context-based predictors typically have an accuracy about 20% higher than computational predictors.
Data speculation to facilitate automatic parallelization

--Steffan et al. HPCA 1998

- Motivation
  - Hard to parallelize programs with complex data dependencies
  - Compilers can’t do it yet….

- Thread-Level Data Speculation
  - Partition the programs into threads assuming NO data dependencies across threads,
  - Run the threads simultaneously,
  - If a data dependency is detected, rollback and re-execute with the correct live-in values

- Simulation of a 4-core platform with SPEC95
  - 15% ~ 290% speedup over the sequential version
Exploiting Speculative Thread-Level Parallelism

--Marcuello et al. IEEE Transactions on Computers, 2004

- Combination of Thread-Level Speculation and Value Prediction
  - For known data dependencies between a producer thread and a consumer thread, use value predictors to predict the live-in values
  - Value prediction: predict values produced by one thread and consumed by others

- Results:
  - Prediction accuracy: 73% ~ 84% on spectInt95 benchmarks
  - The 16-thread configuration provides a speedup of 2.7 on programs that are very hard to parallelize
Summary of Related Work

■ Previous work
  - Focus on the interaction between value prediction techniques and specific architectures, or
  - Assume a perfect architecture model to explore the predictability of the data values, or
  - Propose specific value prediction techniques

■ Our work
  - Focus on the values themselves (architecture independent)
  - Study the redundancy embedded in these values using information theory measures
  - From the redundancy measure, derive the theoretical upper bound for value predictability
Agenda

- The Problems of Modern Parallel Processing
- Architectures and Models of Execution
- Parallelism in Modern (and Future) Applications
- A Theoretical Framework of Value Prediction in Parallel Systems
- Implementation of Value Prediction Techniques
- Conclusions and Future Work
Experiments with Micro-Benchmarks

- **Benchmarks: Livermore Loops**
  - LL6: general linear recurrence equation, highly parallel
  - LL4: banded linear equation, medium parallel
  - LL20: discrete ordinates transport, embarrassingly sequential

- **Simulation Platform**
  - SESC multi-core architecture simulator
  - 256-core machine with on-chip mesh network
  - Distributed shared memory and scratch pad for each core
  - Synchronization State Buffer
Experiments with Micro-Benchmarks

- LL6: high scalable, demonstrates the efficiency of SSB
- LL4 - LL20: Speedup/scalability strictly restricted by data dependencies
Experiments with Micro-Benchmarks

- Communication latency: the number of cycles the consumer thread spends at the synchronization point waiting for the producer to reach completion.
Experiments with Micro-Benchmarks

As shown in the cases of LL4 and LL20, parallelism is limited by data dependencies

Implementation with a “virtual” value predictor:

- Make prediction at synchronization points where the consumers have to wait for the producers
- Prediction accuracy is a control parameter
- Assume no rollback/re-execution overhead
- Assume a 2-cycle overhead in making each prediction
Experiments with Micro-Benchmarks

- LL6 is highly parallel; using value prediction does not bring much benefit
- LL4 is less parallel; value prediction is beneficial only after saturation
- LL20 is sequential; value prediction is always beneficial
Applications for Many-Cores

- Scientific Computation
  - Well-structured and highly parallel
  - Divide and conquer methods: data parallelism
  - SPLASH-2 benchmarks: *fft, ocean, raytracing*…. 

- Recognition-Mining-Synthesis Applications
  - “Future” applications
  - Data mining, machine learning, and graphic workloads
  - Thread-level and data-level parallelism
  - PARSEC benchmarks: *blackscholes, facesim, fluidanimate*…
Experiments with Scientific and RMS Applications

- **Methodology**
  - **PIN**: Binary instrumentation tool developed by Intel
  - **PARSEC benchmarks**: Recognition-Mining-Synthesis (RMS) applications
  - **SPLASH-2 benchmarks**: Scientific applications

- **To characterize parallelism**
  - Binary instrumentation on benchmark programs
  - Use a large instruction window (4096 in our case) to scan the instructions of a program
  - Identify the data dependencies within the instruction window
  - Infer the number of independent instructions that can be executed in each cycle (instantaneous parallelism)
SPLASH-2

Average IPC=26.42

![Graphs showing performance comparison for different SPLASH-2 benchmarks](image-url)
PARSEC

Average IPC=13.09
Experiments with Scientific and RMS Applications

- **Results:**
  - the average IPC for SPLASH-2 is 26 and the average IPC for PARSEC is 13 → may not contain enough parallelism for many-core designs.

- **Observations:**
  - Most applications contain long regions of low parallelism
  - These produce the performance bottleneck referred to by Amdahl’s law

- **Our Research:**
  - Use value prediction techniques to reduce sequential regions
  - How much can be predicted?
  - What is the impact of value prediction on overall performance?
Agenda

- The Problems of Modern Parallel Processing
- Architectures and Models of Execution
- Parallelism in Modern (and Future) Applications
- A Theoretical Framework of Value Prediction in Parallel Systems
- Implementation of Value Prediction Techniques
- Conclusions and Future Work
A Theoretical Framework of Value Prediction in Parallel Systems

- Previous value prediction studies focused on the interaction between value prediction techniques and the target architecture.
- Now study the characteristics of data generated by programs in order to understand the potential of value prediction.
- To achieve this, we utilize information theory measures:
  - Information theory aims at the quantification of information (originally developed to identify the fundamental limits on compressing and communicating data in the presence of random noise).
  - Treat the data streams as random processes and quantify the redundancy of data values in each data stream.
  - From this information, derive the predictability of data values.
A Theoretical Framework of Value Prediction in Parallel Systems

■ Sources of speedup
  - **Intrinsic parallelism**: exploited by fine-grained synchronization mechanisms.
  - **Inherent data redundancy**: exploited by fine-grained value prediction schemes.

■ Amdahl’s formulation
  - The maximum speedup that can be achieved by exploiting the intrinsic parallelism
  - Amdahl’s formulation fails to consider the inherent data redundancy

$$SU_{\text{max}} = \frac{1}{f + \frac{1-f}{n}}$$

- \(f\): fraction of sequential section
- \(n\): number of processing elements available
Extending Amdahl’s formulation

- Value prediction “moves” part of the sequential portion into the parallel portion
- The prediction accuracy upper bound can be characterized by information theory measures

\[ \text{SU}_{\text{max}} = \frac{1}{f(1-p)+fp+(1-f)} \]

- But, how do we find the prediction accuracy \( P \) for different applications?

  - We can identify the lower bound \( mp \) of the misprediction rate using information theory measures…
Information Theory—quantification of information

- Prediction accuracy depends on the inherent redundancy of data values in the program and it can be characterized with information theory measures such as information entropy, information redundancy, and Fano’s inequality.
- **Entropy**: quantifies the uncertainty involved in a random process.

\[
H(X) = - \sum_{i=1}^{\left|S\right|} p(i) \log(p(i))
\]

- **\(H(X)\)**: entropy
- \(S\): set of all distinct symbols
- \(p(i)\): probability of the occurrence of the \(i^{th}\) symbol in the data sequence
Fano’s Inequality: defines the lower bound of prediction error probability

\[
mp \geq \frac{H(X) - h(mp)}{\log(|X|) - 1}
\]

- \(mp\): ratio of mis-prediction (prediction accuracy is \(p = 1 - mp\))
- \(H(X)\): entropy of data sequence \(X\)
- \(h(mp)\): \(h(mp) = -(mp \log(mp) + (1-mp) \log(1-mp))\)
- \(|X|\): number of values in data sequence \(X\)
Experiments

- Characterization of inherent data redundancy on PARSEC and SPLASH-2 benchmarks
  - Utilize the binary instrumentation tool on PARSEC and SPLASH-2 benchmarks to collect runtime statistics
  - Apply analysis tools (based on information theory) to the runtime statistic data to extract measures such as information entropy, redundancy, and minimum misprediction rate

- Verification of the Potential of Fine-Grained Value Prediction
  - LVP: Last Value Predictor
  - FCP: Finite Context Predictor
  - SVP: Stride Value Predictor
Results

- Even naïve value prediction techniques (LVP, FCP) are able to significantly improve program speedup…
- But we are still far from the theoretical upper bound (SU-new)
Impact of available information on prediction accuracy

- 0\textsuperscript{th} order: no previous information available
- 1\textsuperscript{st} order: immediate previous value known
- 2\textsuperscript{nd} order: two previous values known

\textit{the more previous information we use, the more accurately we can predict}
Thread-level parallelism

- Routines that contain tens of instructions and basic blocks that contain five to ten instructions are good candidates for fine-grained threads.
- Characterize the predictability of routine return values and basic-block live-in values with the theoretical model that we developed.

There exists a strong data redundancy in routine return values and basic block live-in values, implying the feasibility of utilizing value prediction to enhance TLP.
### Thread-level parallelism

- Routines that contain tens of instructions and basic blocks that contain five to ten instructions are good candidates of fine-grained threads.
- Characterize the predictability of routine return values and basic-block live-in values with the theoretical model that we developed.

*There exists a strong data redundancy in routine return values and basic block live-in values, implying the feasibility of utilizing value prediction to enhance TLP.*

<table>
<thead>
<tr>
<th>Routine</th>
<th>BBL</th>
<th>Routine</th>
<th>BBL</th>
<th>Routine</th>
<th>BBL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barnes</td>
<td>0.95</td>
<td>Radix</td>
<td>0.97</td>
<td>Canneal</td>
<td>0.63</td>
</tr>
<tr>
<td>Radiosity</td>
<td>0.91</td>
<td>Raytrace</td>
<td>0.98</td>
<td>Dedup</td>
<td>0.73</td>
</tr>
<tr>
<td>FFT</td>
<td>0.88</td>
<td>Volrend</td>
<td>0.84</td>
<td>Facesim</td>
<td>0.68</td>
</tr>
<tr>
<td>FMM</td>
<td>0.88</td>
<td>Bodytrack</td>
<td>0.82</td>
<td>Fluidanimate</td>
<td>0.83</td>
</tr>
<tr>
<td>LU</td>
<td>0.96</td>
<td>Swaptions</td>
<td>0.73</td>
<td>Freqmine</td>
<td>0.70</td>
</tr>
<tr>
<td>Ocean</td>
<td>0.96</td>
<td>Blackscholes</td>
<td>0.89</td>
<td>Streamcluster</td>
<td>0.29</td>
</tr>
</tbody>
</table>
### Thread-level parallelism

- Routines that contain tens of instructions and basic blocks that contain five to ten instructions are good candidates of fine-grained threads
- Characterize the predictability of routine return values and basic-block live-in values with the theoretical model that we developed

*There exists a strong data redundancy in routine return values and basic block live-in values, implying the feasibility of utilizing value prediction to enhance TLP*

<table>
<thead>
<tr>
<th></th>
<th>Routine</th>
<th>BBL</th>
<th></th>
<th>Routine</th>
<th>BBL</th>
<th></th>
<th>Routine</th>
<th>BBL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Barnes</strong></td>
<td>0.95</td>
<td>0.76</td>
<td><strong>Radix</strong></td>
<td>0.97</td>
<td>0.90</td>
<td><strong>Canneal</strong></td>
<td>0.63</td>
<td>0.79</td>
</tr>
<tr>
<td><strong>Radiosity</strong></td>
<td>0.91</td>
<td>0.47</td>
<td><strong>Raytrace</strong></td>
<td>0.98</td>
<td>0.79</td>
<td><strong>Dedup</strong></td>
<td>0.73</td>
<td>0.95</td>
</tr>
<tr>
<td><strong>FFT</strong></td>
<td>0.88</td>
<td>0.61</td>
<td><strong>Volrend</strong></td>
<td>0.84</td>
<td>0.86</td>
<td><strong>Facesim</strong></td>
<td>0.68</td>
<td>0.69</td>
</tr>
<tr>
<td><strong>FMM</strong></td>
<td>0.88</td>
<td>0.77</td>
<td><strong>Bodytrack</strong></td>
<td>0.82</td>
<td>0.83</td>
<td><strong>Fluidanimate</strong></td>
<td>0.83</td>
<td>0.85</td>
</tr>
<tr>
<td><strong>LU</strong></td>
<td>0.96</td>
<td>0.69</td>
<td><strong>Swaptions</strong></td>
<td>0.73</td>
<td>0.84</td>
<td><strong>Freqmine</strong></td>
<td>0.70</td>
<td>0.78</td>
</tr>
<tr>
<td><strong>Ocean</strong></td>
<td>0.96</td>
<td>0.72</td>
<td><strong>Blackscholes</strong></td>
<td>0.89</td>
<td>0.88</td>
<td><strong>Streamcluster</strong></td>
<td>0.29</td>
<td>0.86</td>
</tr>
</tbody>
</table>
Agenda

- The Problems of Modern Parallel Processing
- Architectures and Models of Execution
- Parallelism in Modern (and Future) Applications
- A Theoretical Framework of Value Prediction in Parallel Systems
- Implementation of Value Prediction Techniques
- Conclusions and Future Work
A Sophisticated Generic Value Predictor

- A combination of multiple component predictors:
  - LVP: last value predictor
  - SVP: stride value predictor
  - FCP: finite context predictor
  - Each predictor covers a fraction of the predictor space

- A confidence estimator for each component predictor:
  - To keep track of the performance of each component predictor

- A decision unit to decide which predictor to use
  - Only one predictor is used for each prediction
  - Pick one with high probability of making a correct prediction
A Generic Hybrid Value Predictor

- Each component predictor produces a prediction
- The fusion logic picks one predicted value to use based on the performance history of each predictor
Performance Results of the Hybrid Predictor

- **Pred**: prediction is correct; **est**: estimation unit picks the correct predictor to use
- Performance gain is generated only when both the prediction is correct and the estimation unit takes the correct predictor to use (**pred & est**)
- Rollback/re-execution (performance degradation) happens only when the prediction is incorrect but the estimation unit thinks it is correct (**!pred & est**)

_Y-axis: fraction of all predictions_
Value Prediction and Speculation on GPU?

- Value prediction and speculation on CMP
  - Communication and synchronization between different cores incur a high overhead (~100 cycles)
  - To amortize this overhead, the granularity of the thread needs to be at least in the 100s of instructions
  - If the thread granularity becomes too high, it requires excessive storage of the speculative states

- What about GPU?
  - GPU architecture provides an abundant amount of closely coupled hardware threads
  - Large number of registers for each thread
  - Fast synchronization operations
Value Prediction and Speculation on GPU!

- Strong *data redundancy* in application programs.
- GPU is good for SIMD/data parallel applications
  - Many datapaths: simultaneously handle many speculative streams
  - No need for a decision circuit, map one predictor to each speculative stream, if one is correct, the execution can move on
  - Aggressive techniques: speculate a few steps ahead
  - Low synchronization overhead: low rollback cost

- Example:
  ```c
  for( int i=0; i<N; i++)
  {
    k[i] = Op(k[i-1]);
  }
  ```
A Hierarchy of threads

- Each block contains 32 threads, and there is shared memory in each block
- Threads in each block can cooperate and communicate with each other
- Independent blocks do not communicate through shared memory, but have to communicate through the global memory
- GeForce 8800 contains 16 streaming multiprocessors (SM), each SM contains 8 streaming processors (SP). With low thread switching overhead, it can support 768 active threads
- CUDA automatically maps the thread hierarchy onto GeForce 8800 hardware
Value Prediction on GPU

Iteration n

Speculation: Iteration n+1

Speculation: Iteration n+2

If (verification 1)
if ( verification 2)
iteration n+3
else
iteration n+2
// both speculation operations succeed
else
iteration n+1
// speculation 1 succeeds and 2 fails
else
// both speculation operations fail

Non-speculative thread

speculative threads
Value Prediction on GPU

- **Stage 1**: generates live-in values for speculative (slave) and non-speculative (master) threads
- **Stage 2**: both speculative and non-speculative threads start simultaneously
- **Stage 3**: update value predictors and commit results to memory
Value Prediction on GPU

Outer loop iterations are independent and can be run in parallel:
- Map each of these independent outer iterations to a thread block
- Within each thread block, start one non-speculative thread and multiple speculative threads to calculate the inner loop iterations

```c
1:   for (l=1 ; l<=m ; l++) {
2:       for( k=0 ; k<n ; k++ ) {
3:           xx[l][k+1] = ( x[l][k] - xx[l][k] )* dn + xx[l][k];
4:       }
5:   }
```
GPU with and without value prediction

- **Work**: the amount of work in stage 2 relative to the total work
  - Stages 1 and 3 are strictly overhead
- **Accuracy**: prediction accuracy as a control parameter

![Design Space Exploration](image-url)
Value Prediction on GPU

- HFY_pred: taken from Swaption of PARSEC benchmark
- About 30% prediction accuracy
- But overhead of stages 1 and 3 is 60% of the execution time
Overhead Analysis

- Stages 1 and 3: commit and table update operations
- In the current pure software approach, it takes many instructions to finish these operations:
  - Commit: lines 1~5, lines 9~16
  - Table update: lines 17~20
- These result in long sequential sections
- If we moved these operations into hardware, the performance would be much improved

```assembly
1:  ld.param.u32   $r75, [__cudaparm_xx];
2:  ld.shared.s32  $r76, [__cuda_spec8+0];
3:  mul.lo.u32     $r77, $r18, 4;
4:  add.u32        $r78, $r75, $r77;
5:  st.global.s32  [$r78+0], $r76;
6:  ld.shared.s32  $r81, [step];
7:  add.s32        $r82, $r81, 1;
8:  st.shared.s32  [step], $r82;
9:  ld.shared.s32  $r79, [hit];
10: mul.lo.u32     $r83, $r79, 4;
11: add.u32        $r84, $r83, $r20;
12: ld.shared.s32  $r85, [$r84+0];
13: add.s32        $r86, $r18, $r16;
14: mul.lo.u32     $r87, $r86, 4;
15: add.u32        $r88, $r75, $r87;
16: st.global.s32  [$r88+0], $r85;
17: add.s32        $r89, $r85, $r85;
18: ld.shared.s32  $r76, [__cuda_spec8+0];
19: sub.s32        $r90, $r89, $r76;
20: st.shared.s32  [__cuda_pred16+4], $r90;
```
Agenda

- The Problems of Modern Parallel Processing
- Architectures and Models of Execution
- Parallelism in Modern (and Future) Applications
- A Theoretical Framework of Value Prediction in Parallel Systems
- Implementation of Value Prediction Techniques
- Conclusions and Future Work
Conclusions

- In the many-core era, application programs may not contain enough parallelism
- Our theoretical model and experiments have shown that value prediction can greatly enhance the performance of parallel systems
- Value prediction implementation:
  - Hybrid predictor: capable of achieving high prediction accuracy
  - On GPU: rollback and speculative execution can be efficiently handled, but a high overhead due to long sequential regions is introduced
Future Work

- Value prediction on GPU:
  - Move *commit* and *table_update* operations into hardware
  - Instead of 5~10 software instructions, we can use one instruction to take care of these operations
  - Shrink the sequential region (bottleneck) by a factor of 5 to 10

- Reconfigurable value prediction unit
  - Implement a processor and its associated value prediction unit on an FPGA
  - Study the power and performance tradeoff of value prediction unit
  - Turn on the value prediction unit to improve performance
  - Turn off the value prediction unit to save power
Bibliography

1. **RHE: A Lightweight JVM Instructional Tool**  
   Shaoshan Liu, Chengrui Deng, Xiao-Feng Li, and Jean-Luc Gaudiot  
   *COMPSAC 2009, July 2009*

2. **Potential Impact of Value Prediction on Communication in Many-Core Architectures**  
   Shaoshan Liu and Jean-Luc Gaudiot  
   *IEEE Transactions on Computers, June 2009*

3. **Space-and-Time Efficient Garbage Collectors for Parallel Systems**  
   Shaoshan Liu, Ligang Wang, Xiao-Feng Li, and Jean-Luc Gaudiot  
   *Computing Frontiers, Ischia, Italy, May 2009*

4. **Value Prediction in Modern Many-Core Systems**  
   Shaoshan Liu (Ph.D. Student) and Jean-Luc Gaudiot (Advisor)  
   *IPDPS 2009, TCPP-Ph.D. Forum, Rome, Italy, May 2009*

5. **Packer: an Innovative Space-Time-Efficient Parallel Garbage Collection Algorithm Based on Virtual Spaces**  
   Shaoshan Liu, Ligang Wang, Xiao-Feng Li, and Jean-Luc Gaudiot  
   *IPDPS 2009, Rome, Italy, May 2009*

6. **The Potential of Fine-Grained Value Prediction in Enhancing the Performance of Modern Parallel Machines** - Best Paper Award  
   Shaoshan Liu and Jean-Luc Gaudiot  
   *ACSAC '08, Hsinchu, Taiwan, August 2008*

7. **Synchronization Mechanisms on Modern Multi-Core Architectures**  
   Shaoshan Liu and Jean-Luc Gaudiot  
   *ACSAC '07, Seoul, Korea, August 2007*
Questions?