Evaluating the capabilities of the Xeon Phi platform in the context of software-only, thread-level speculation

Alvaro Estebanez, **Diego R. Llanos**,
Arturo Gonzalez-Escribano
Trasgo Computing Research Group
University of Valladolid, Spain

HLPP 2015, Pisa, Italy, July 2nd, 2015

Grupo Trasgo
Universidad de Valladolid

Departamento de Informática
Universidad de Valladolid

Universidad de Valladolid

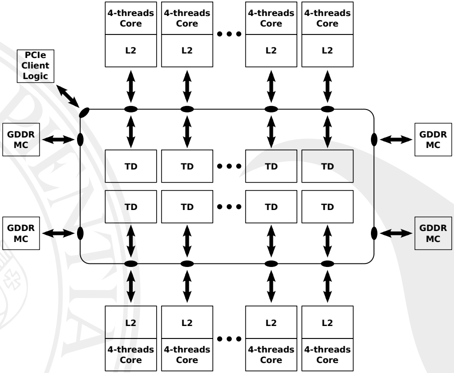
Summary

In this talk, we will. . .

- Briefly review the capabilities of the Xeon Phi coprocessor.
- Present a not-so-common case study: Thread-Level Speculation (TLS).
- Describe our software-based TLS solution (ATLaS).
- Show what happens when running ATLaS in the Xeon Phi.
- Enumerate conclusions and discuss future work.

Intel Xeon Phi in a nutshell

- Coprocessor launched in 2012.
- Can run an OS by itself, but still needs a host computer.
- Composed of up to 61 four-way SMT cores.
- Interconnected by a high-speed bidirectional ring.



Intel Xeon Phi: Pros and cons

- Pros:
  - It acts as a shared-memory multiprocessor.
  - Standard parallel programming models (OpenMP, MPI, OpenCL) can be used.
  - Uses Intel 64-bits architecture with 512-bits-wide FP SIMD instructions.
  - Excellent memory bandwidth (240 GB/s vs. 51.2 GB/s of our AMD Opteron SM system).
- Cons:
  - In-order execution.
  - Modest clock speed.
  - If SIMD instructions are not heavily used, no so many computational units to overcome these limitations.

How to use the Intel Xeon Phi

Two ways:

- **Native execution** Once an OS is installed, log into the system, compile and run the parallel application natively.
- **Offloading from the host** Compile the code in the host system, using software extensions in the source code to offload tasks from the host to the Xeon Phi.
  - OpenMP example: `#pragma offload target{mic}`
  - Variables exchange should be declared explicitly, with `in()`, `out()` or `inout()` clauses.

How to use the Intel Xeon Phi

Two ways:

- **Native execution** Once an OS is installed, log into the system, compile and run the parallel application natively.
- **Offloading from the host** Compile the code in the host system, using software extensions in the source code to offload tasks from the host to the Xeon Phi.
  - OpenMP example: `#pragma offload target{mic}`
  - Variables exchange should be declared explicitly, with `in()`, `out()` or `inout()` clauses.

Our goal: to evaluate the Xeon Phi capabilities when running a software-based Thread-Level Speculation (TLS) system.

Thread-Level Speculation (TLS)

Parallelization with OpenMP

```
for (i=0; i<MAX; i++) {
   b = func(i);
   v[i] = b * a[i];
}
```

(a) Original loop

```
#pragma omp parallel for \
 private (i,b) shared (a,v)
for (i=0; i<MAX; i++) {
   b = func(i);
   v[i] = b * a[i];
}
```

(b) Loop parallelized with OpenMP directives

Thread-Level Speculation (TLS)

What happens if the loop may present dependence violations?

```
for (i=0; i<MAX; i++) {
   b = func(i);
   if (b==k)
      v[i] = v[i-b];
   else
      v[i] = b * a[i];
}
```

Loop not safely parallelizable

Thread-Level Speculation (TLS)

- Aims to execute in parallel fragments of code without requiring compile-time analysis.
- Iterations are divided in blocks and optimistically executed in parallel, hoping that no dependence violations will appear.
- In software-based TLS, original code is augmented with function calls that monitors the parallel execution at runtime.
- *Offending* threads (that have consumed a value before being produced by a predecessor thread) are dynamically stopped and re-started with correct values.
- Consistency with sequential semantics is ensured by the runtime system.
- Suitable for shared-memory systems.

The ATLaS framework

- ATLaS[1] is a software-based, TLS framework that extends OpenMP functionalities to allow the parallelization of loops that may present dependences between iterations.
- ATLaS allow the speculative management of scalar variables and data structures, and can handle accesses through pointer arithmetic.
- Documentation and software download:

atlas.infor.uva.es

[1] **An OpenMP Extension that Supports Thread-Level Speculation**, Aldea, Estebanez, Llanos, Gonzalez-Escribano, IEEE TPDS, 2015.

Speculative parallelization using ATLaS

(a) Loop not parallelizable safely	(b) Loop parallelizable using ATLaS

<pre>for (i=0; i<MAX; i++) {
   b = func(i);
   if (b==k)
      v[i] = v[i-b];
   else
      v[i] = b * a[i];
}</pre>
	<pre>#pragma omp parallel for \
	 private (i,b) shared (a,k) \
	 speculative(v)
	for (i=0; i<MAX; i++) {
	   b = func(i);
	   if (b==k)
	      v[i] = v[i-b];
	   else
	      v[i] = b * a[i];
	}</pre>

The ATLaS compile phase

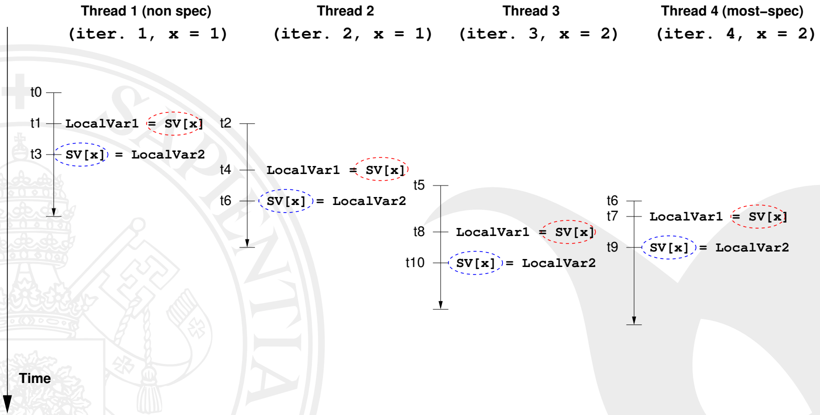
- At compile time, the ATLaS GCC compiler plug-in replaces accesses to speculative variables with calls to speculative functions.
- Example: Let a, b and c be three variables labeled as speculative:

Original code	**Augmented code**

`#pragma omp...`
` speculative (a,b,c)`
`for (...) {`
`   a = 9;`
`   b = 11.7*2;`
`   lhs = c;`
`}`
	`#pragma omp...`
	
	`for (...) {`
	`   specstore(&a,sizeof(a),9);`
	`   specstore(&b,sizeof(b),23.4);`
	`   specload(&c,sizeof(c),&lhs);`
	`}`

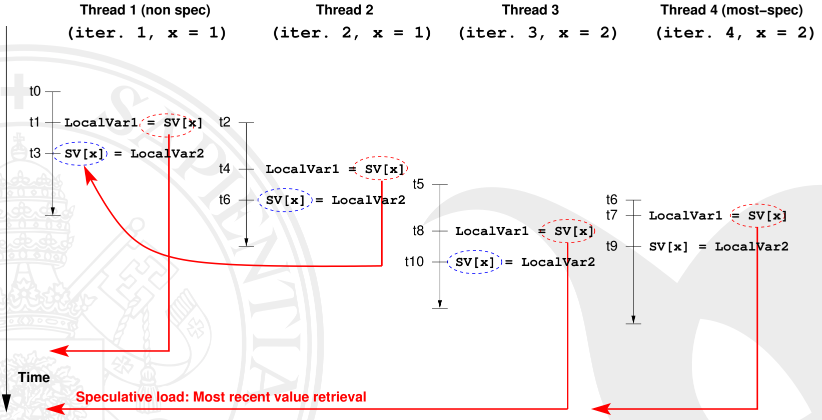
TLS: An example

- Suppose that the SV vector was labeled as speculative.
- Index x is not known at compile time.



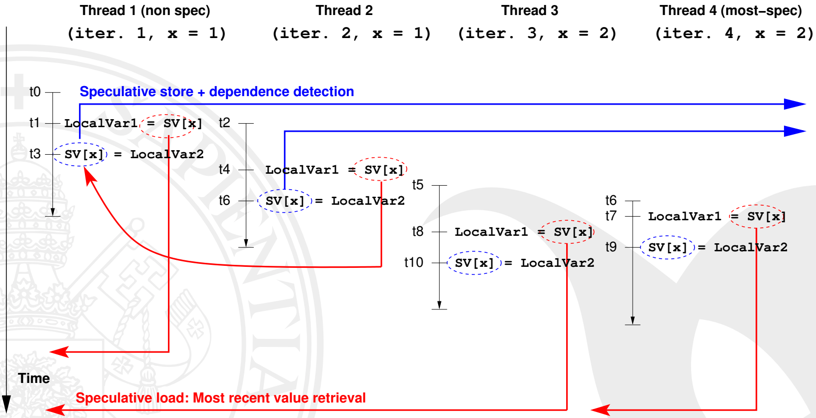
TLS: Speculative loads

- Suppose that the `SV` vector was labeled as `speculative`.
- Index `x` is not known at compile time.



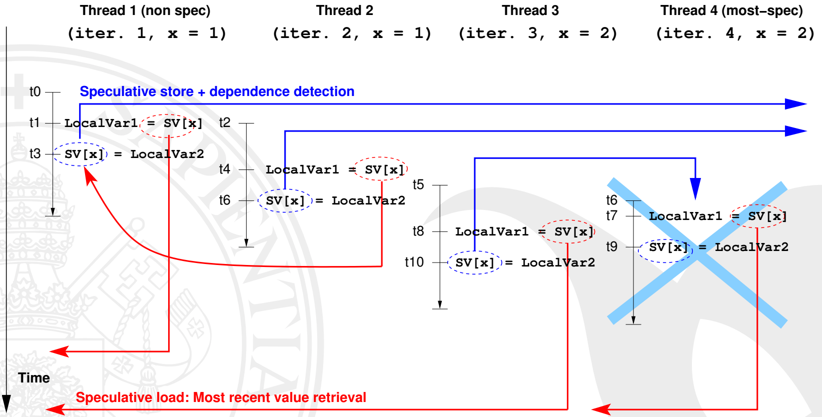
TLS: Speculative stores (no violations)

- Suppose that the SV vector was labeled as speculative.
- Index x is not known at compile time.



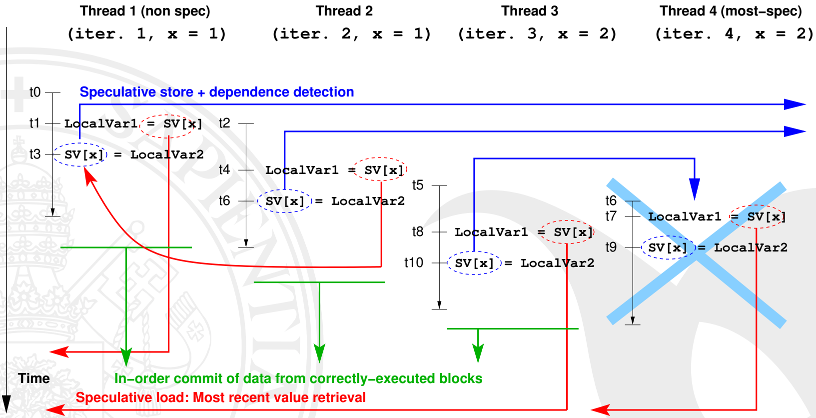
TLS: Speculative stores (squash)

- Suppose that the `SV` vector was labeled as `speculative`.
- Index `x` is not known at compile time.

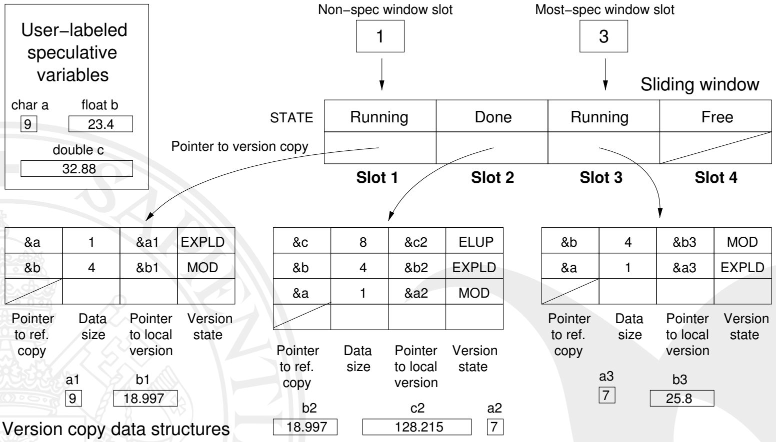


TLS: Commitment of shared variables

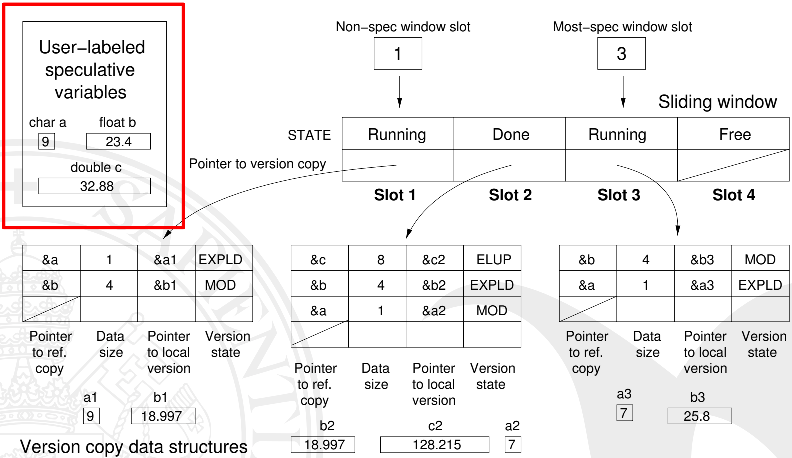
- Suppose that the SV vector was labeled as speculative.
- Index x is not known at compile time.



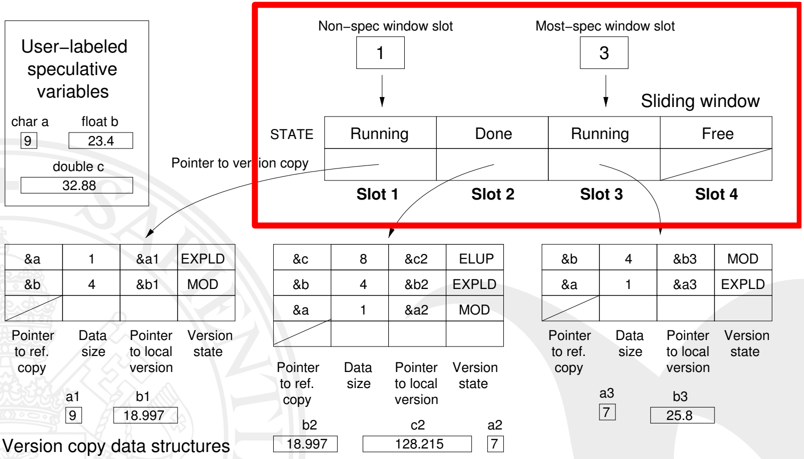
The ATLaS runtime system



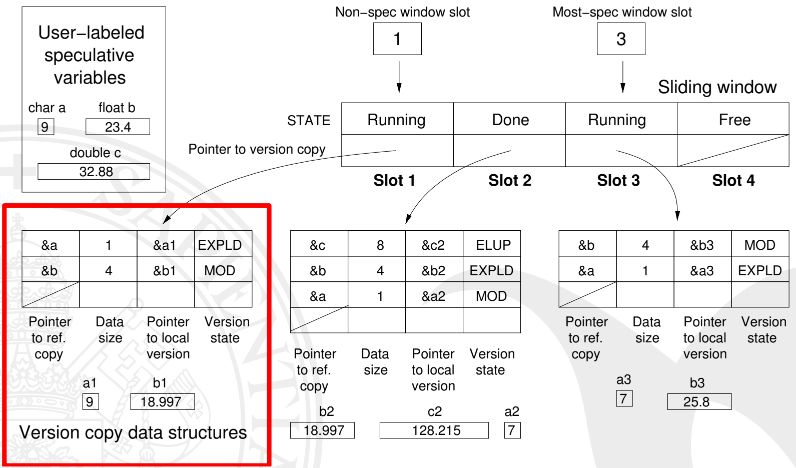
The ATLaS runtime system



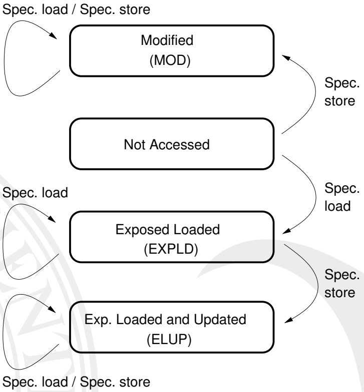
The ATLaS runtime system



The ATLaS runtime system



ATLaS transition state diagram

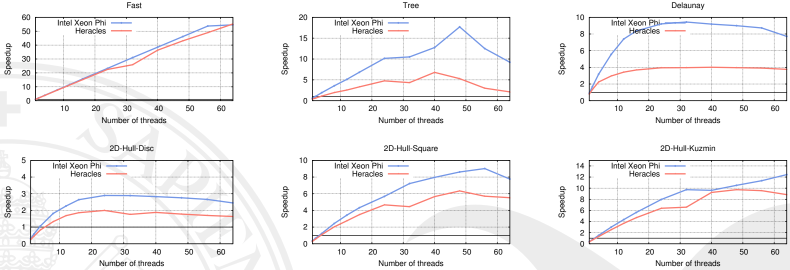


Evaluating ATLaS running on Xeon Phi

Benchmarks considered

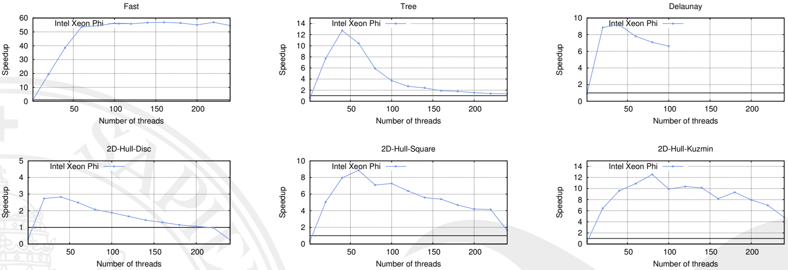
- Three benchmarks representative of real-world problems:
  - 2D-DT (Delaunay Triangulation)
  - 2D-Hull (Convex Hull) with three different input sets.
  - TREE
- They present a significant squash-and-restart rate due to dependences (up to 15%), challenging STLS systems.
- One additional synthetic benchmark, FAST, to measure TLS overheads.

Results #1: Scalability



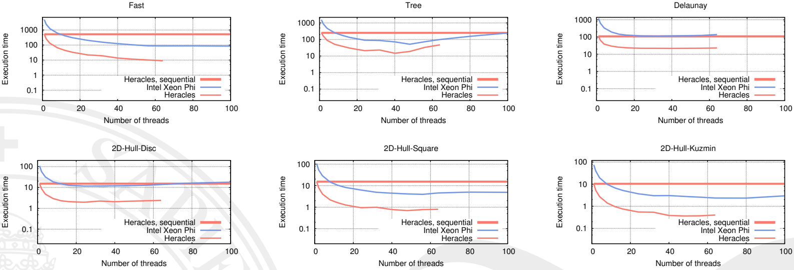
- **Scalability** is better for the Xeon Phi, thanks to its superior bandwidth.

Results #2: Using all threads of the Xeon Phi



- In general, performance degrades when launching more threads than processors: the 4-way SMT offered by the Xeon Phi is not useful in this case.

Results #3: Absolute performance



- True, our benchmarks do not benefit from vectorization.
- The Xeon Phi platform is consistently five times slower than our Intel Xeon shared-memory system (OOO execution, higher clock speed).

Conclusions

- The Xeon Phi was too good to be true: Hundreds of CPU-like threads for just € 1 500!
- However, absolute performance for ATLaS (in the end, an OpenMP application) is five times poorer than when using a good shared-memory system.
- The scalability is really good: The problem are the processors.
- We are aware that our parallel code does not need vectorization.

Conclusions

- The Xeon Phi was too good to be true: Hundreds of CPU-like threads for just € 1 500!
- However, absolute performance for ATLaS (in the end, an OpenMP application) is five times poorer than when using a good shared-memory system.
- The scalability is really good: The problem are the processors.
- We are aware that our parallel code does not need vectorization.
- Our "Wish List" for future releases of the Xeon Phi regarding to TLS:
  - Faster computational units.
  - Out-of-order execution.
  - Some hardware support for TLS would help a lot.

Conclusions

- The Xeon Phi was too good to be true: Hundreds of CPU-like threads for just € 1 500!
- However, absolute performance for ATLaS (in the end, an OpenMP application) is five times poorer than when using a good shared-memory system.
- The scalability is really good: The problem are the processors.
- We are aware that our parallel code does not need vectorization.
- Our "Wish List" for future releases of the Xeon Phi regarding to TLS:
  - Faster computational units.
  - Out-of-order execution.
  - Some hardware support for TLS would help a lot.
- Future work: to use the remaining threads to help in the execution of the threads in charge of each chunk of speculative iterations.

Conclusions

- The Xeon Phi was too good to be true: Hundreds of CPU-like threads for just € 1 500!
- However, absolute performance for ATLaS (in the end, an OpenMP application) is five times poorer than when using a good shared-memory system.
- The scalability is really good: The problem are the processors.
- We are aware that our parallel code does not need vectorization.
- Our "Wish List" for future releases of the Xeon Phi regarding to TLS:
  - Faster computational units.
  - Out-of-order execution.
  - Some hardware support for TLS would help a lot.
- Future work: to use the remaining threads to help in the execution of the threads in charge of each chunk of speculative iterations.
- You are invited to try ATLaS **atlas.infor.uva.es**

Evaluating the capabilities of the Xeon Phi platform in the context of software-only, thread-level speculation

Alvaro Estebanez, **Diego R. Llanos**,
Arturo Gonzalez-Escribano
Trasgo Computing Research Group
University of Valladolid, Spain

HLPP 2015, Pisa, Italy, July 2nd, 2015

Grupo Trasgo
Universidad de Valladolid

Departamento de Informática
Universidad de Valladolid

Universidad de Valladolid