HermesBDD: A Multi-Core and Multi-Platform Binary Decision Diagram Package

Systems Design Laboratory (2022/2023)

Computer Engineering for Robotics and Smart Industry

Luigi Capogrosso Luca Geretti Tiziano Villa

Outline

HermesBDD

- HermesBDD is a **parallel multi-core** and **multi-platform library** of Binary Decision Diagrams, written in C++ and fully **developed at the University of Verona**
- The source code of the library is available here:
  `https://github.com/luigicapogrosso/HermesBDD`
- The slides related to this lecture are available here:
  `https://github.com/luigicapogrosso/SDL`

Getting HermesBDD

- You can **freely download** the latest version **directly from the GitHub repository**, so:
  `$ git clone https://github.com/luigicapogrosso/HermesBDD.git`
- HermesBDD has the following dependencies:
  - **CMake**, for compiling
  - **Sphinx**, for documentation generation
- For further information about CMake and Sphinx, check the following 1 and 2 documentation pages, respectively
- For this lecture, Sphinx is not required. You can read the documentation online at:
  `https://luigicapogrosso.github.io/HermesBDD/`

Building HermesBDD

- The library is tested for compilation using **GCC** (minimum required: 10.2), **Clang** (minimum required: 11.0), and **MSVC** (minimum required: 19.20)
- To build the library from sources in a clean way, it is preferable that you set up a `build` subdirectory, say:
  ```
  $ mkdir build && cd build
  ```
- Then, you can prepare the environment, choosing a *Release* build for maximum performance:
  ```
  $ cmake ..  -DCMAKE_BUILD_TYPE=Release
  ```
- At this point, if no error arises, you can build with:
  ```
  $ cmake --build .
  ```

Building HermesBDD (cont'd)

In particular, the package provides the following options that can be set for the compilation step:

- `NO_CACHE`: Do not use cache
  - Possible values: `OFF` (default) or `ON`
- `NO_THREAD`: Do not use threads
  - Possible values: `OFF` (default) or `ON`
- `NO_DYNMEM`: Do not use dynamic memory allocation
  - Possible values: `OFF` (default) or `ON`
- `COVERAGE`: Enable coverage reporting (only for testing)
  - Possible values: `OFF` (default) or `ON`

Fundamentals

Why

There are many BDD libraries available and new ones are added. For an early survey see [1]

Code	*Name/Tool*	*Author(s)*	*Affiliation*
ABC	ABCD 0.3	Armin Biere	ETH Zurich, Switzerland
ALL	Alliance 5.0	Jacomme Ludovic	ASIM/LIP6 Paris, France
BUD	BuDDy 1.9	Jørn Lind-Nielsen	ITU, Denmark
CAL	CAL (VIS)	Rajeev Ranjan	UC Berkeley, USA
CMU	CMU (VIS)	David Long	CMU/ATT, USA
CUD	CUDD 2.3.1	Fabio Somezi	Boulder, CO, USA
EST	EST 1ed	Robert Meolic	Maribo, Slovenia
IBM		Geert Janssen	IBM Watson, USA
MON	MONA	Anders Møller	ATT/BRICS, USA
PDT		Cusinato/Corno	Politecnico di Torino, Italy
STA	StaticBdd 1.0	Stefan Edelkamp	Freiburg, Germany
TGR	TiGeR 3.0	Coudert/Madre/ Touati	Bull/DEC/ Xorix, USA
TUD	TUDD 0.8.3a	Stefan Höreth	Darmstadt, Germany

ABC http://i10www.ira.uka.de/armin/abcd/index.html
ALL http://www-asim.lip6.fr/alliance/
BUD http://www.itu.dk/research/buddy/
CAL http://www-cad.eecs.berkeley.edu/Respep/Research/bdd/cal_bdd/
CMU http://www-2.cs.cmu.edu/afs/cs/project/modck/pub/www/bdd.html
CUD http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html
EST http://www.el.feri.uni-mb.si/est/
IBM http://www.research.ibm.com/da/bdd.html
MON http://www.brics.dk/mona/
PDT ftp://ftp.polito.it/pub/people/panta/bdd.tar.Z
STA http://www.informatik.uni-freiburg.de/~edelkamp/StaticBdd/
TGR http://www-2.cs.cmu.edu/~bwolen/fmcad98/packages/tiger/
TUD http://www.rs.e-technik.th-darmstadt.de/~sth/download.html

[1] Janssen, Geert. *"A consumer report on BDD packages."* 16th Symposium on Integrated Circuits and Systems Design, 2003

Goal

- In particular, the goal of HermesBDD is to provide a BDD library that is:
  - **Highly parallel** (multi-core)
  - **Multi-platform** (Linux, Windows, and macOS)
  - Completely written in C++, **with no need to rely on external libraries**
  - **Written according to engineering principles** such as *Code Coverage* and *Continuous Integration* for reliability and easy maintenance over time
  - For **teaching and learning purposes**
  - Designed for **ease of use**
- HermesBDD is not yet feature-complete, and there are still many interesting things left for *you* to do. So, this project welcomes contributions and suggestions

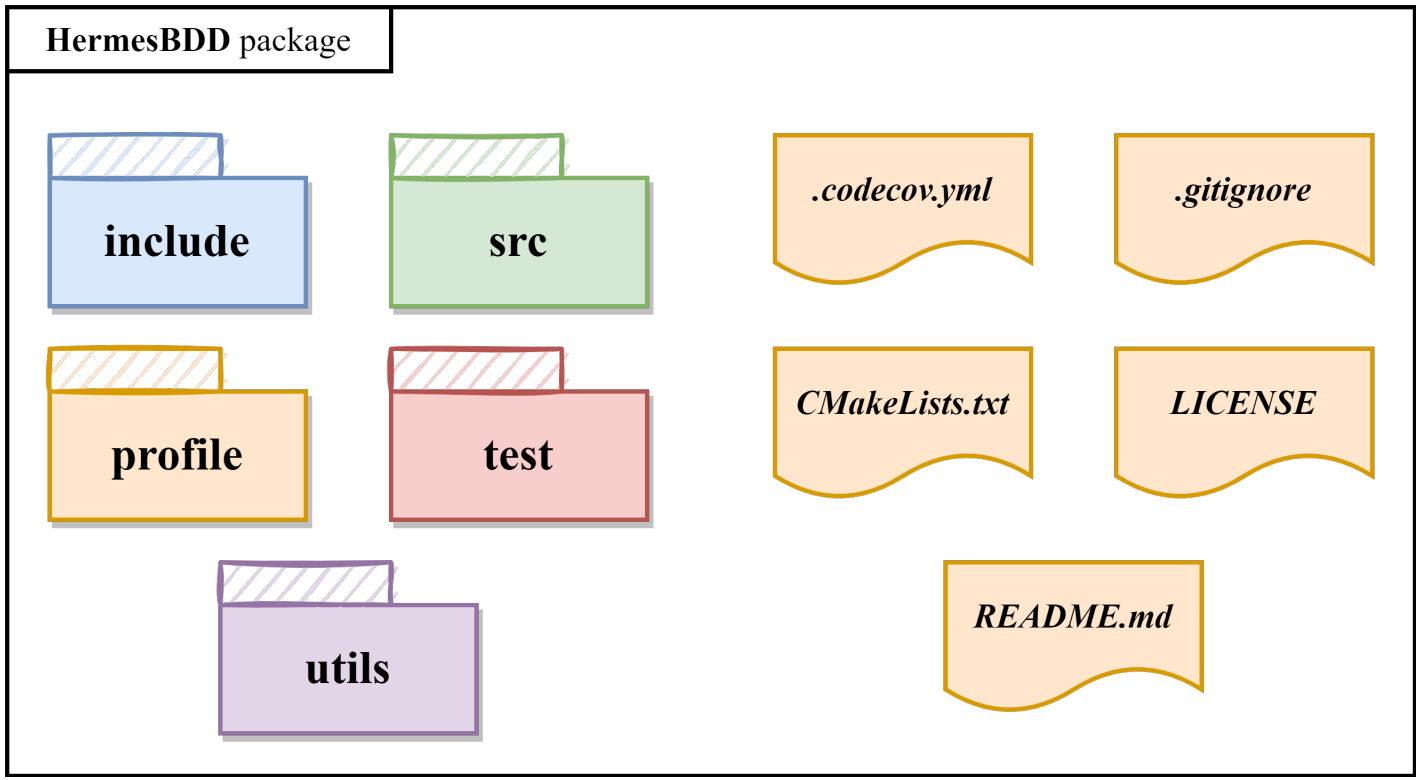
Goal (cont'd)

So, why did we develop HermesBDD?

- HermesBDD has a **well-documented source code**
- Also, you can **compile it on your laptop in a few seconds**, and it is **easy to understand** by disabling all advanced optimization techniques that are implemented in it
- If you are a beginner in BDD development, perhaps HermesBBD is the right library to start with

Package Structure

- The packages in this project are primarily organized based on the functionalities developed, including:



Package Structure (cont'd)

- `include/`: Contains the header files of the library
- `src/`: Contains the source code of the library
- `profile/`: Contains the code to profile the package
- `tests/`: Contains the following tests: *HermesBDD*, *ITE*, and *n*-Queens
- `utils/`: Contains all source code for the library utilities
- `.codeconv.yml`: The Yaml file for all Codecov settings
- `.gitignore`: A file for untracked files that Git should ignore
- `CMakeLists.txt`: The CMake configuration file
- `LICENSE`: The MIT license file
- `README.md`: The README file

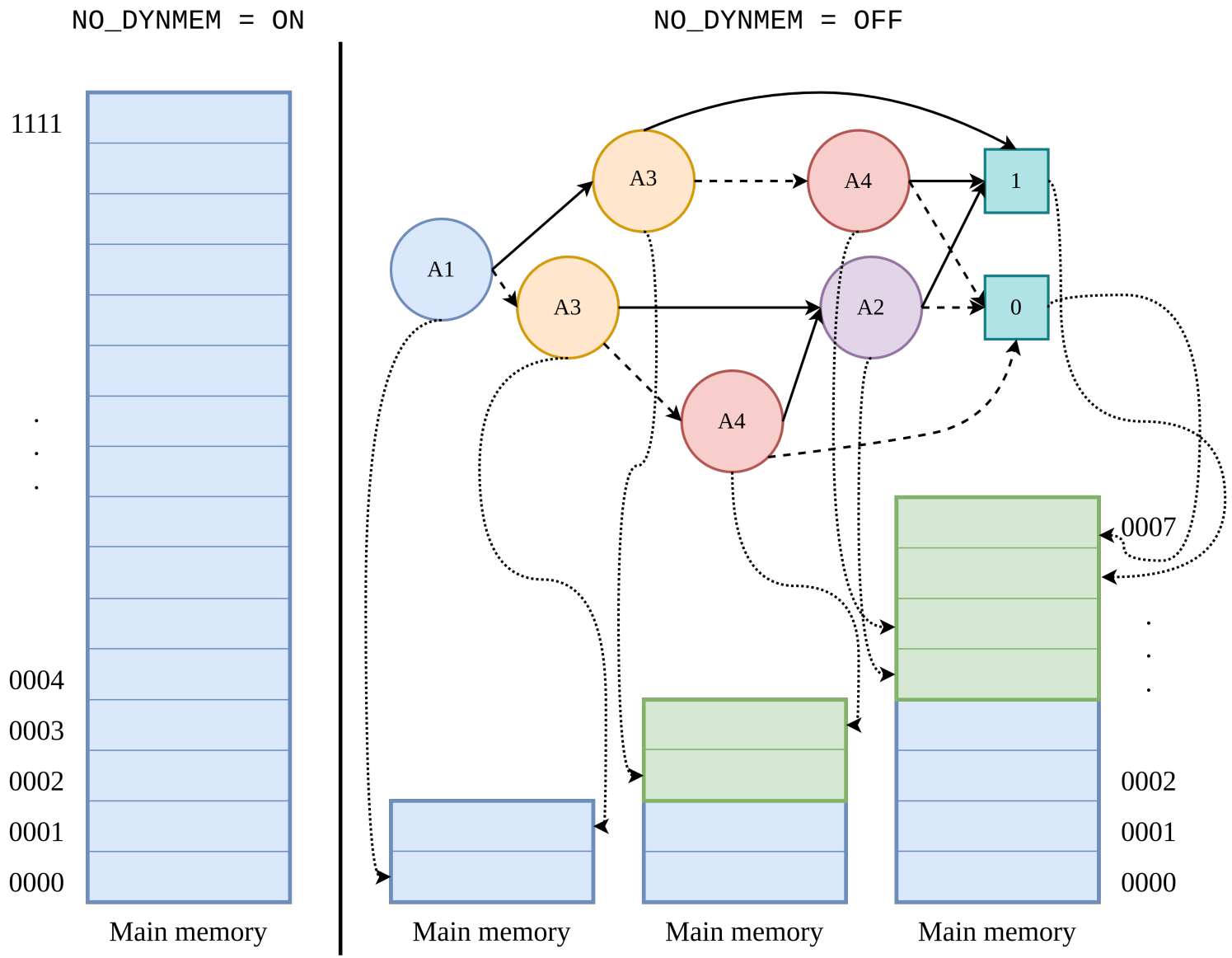
Outline

Memory Management

HermesBDD implements two different memory management mechanisms:

- **Static Allocation**: A contiguous slice of memory is reserved at the start of the process. This remains unchanged throughout the execution of the process. If the running program needs more memory, the process is killed by the OS
- **Dynamic Allocation**: At the beginning of the process a portion of memory is allocated to store N nodes. In case this space is not enough, a new space of size $N * 2$ will be allocated. Then, in case this space is no longer needed, the memory will be deallocated

Memory Management (cont'd)



Memory Management in HermesBDD

- The concepts explained above are coded in `HermesBDD/src/memory_manager.cpp`
  - In particular, at line 90 we can see the static allocation of the memory: `nodes.init(mem_size)`
  - On the other hand, at line 93, we can see the dynamic allocation of the memory: `nodes.init()`
- The value of `mem_size` is 2 GB. Unfortunately, this can currently only be changed in *hard-coded* mode. A solution for more efficient handling of this parameter will be implemented soon

The *ITE* Procedure

```
1  ITE(f, g, h)
2  {
3      if (terminal_case)
4      {
5          return (r = trivial_result)
6      }
7      else
8      {
9          if (table_has_entry {(f, g, h), r})
10         {
11             return r
12         }
13         else
14         {
15             x = top_var(f, g, h)
16             t = ITE(f_x, g_x, h_x)
17             e = ITE(f_x1, g_x1, h_x1)
18
19             if (t == e)
20             {
21                 return t
22             }
23             r = find_or_add_in_table(x, t, e)
24             update_table((f, g, h), r)
25
26             return r
27         }
28     }
29 }
```

- Just like:
  - A *If-Then-Else* in a programming language
  - A *MUX* in hardware
- As is shown in the algorithm on the left, **the `ITE` procedure evaluates the $ITE(f, g, h)$ operator recursively**
- $ITE(f, g, h)$:
  - If (f) then (g) else (h)
  - $fg + f'h$

The *Parallel ITE* Procedure

- **We parallelize this function whenever there are two recursive calls, and the final result is computed using a hash table**
- To this end, we use the C++ function `async()`, which is a higher-level wrapper for threads and futures, followed by the matching function `get()` to retrieve the results
- With this implementation, **the only synchronization between workers is that the results of suboperations are stored in a shared memoization table.** This table is shared globally

The *Parallel ITE* Procedure (cont'd)

```
1: def ITE(A, B, C):
2:     if (A == true_node)
3:         return B;
4:
5:     if (A == false_node)
6:         return C;
7:
8:     if (B == true_node && C == false_node)
9:         return A;
10:
11:    if (B == false_node && C == true_node)
12:        return complement(A);
13:
14:    if (B == C)
15:        return B;
16:
17:    [...]
18:
19:    x = get_root_var(A, B, C);
20:
21:    auto future = std::async(ITE, A_false, B_false, C_false);
22:    R_false = future.get();
23:
24:    auto future = std::async(ITE, A_true, B_true, C_true);
25:    R_true = future.get();
26:
27:    result = make(x, R_true, R_false);
28:    cache.insertITE(A, B, C, result);
29:
30:    return result
```

In particular, Fig. on the left shows the **pseudocode of the parallel `ITE` algorithm**

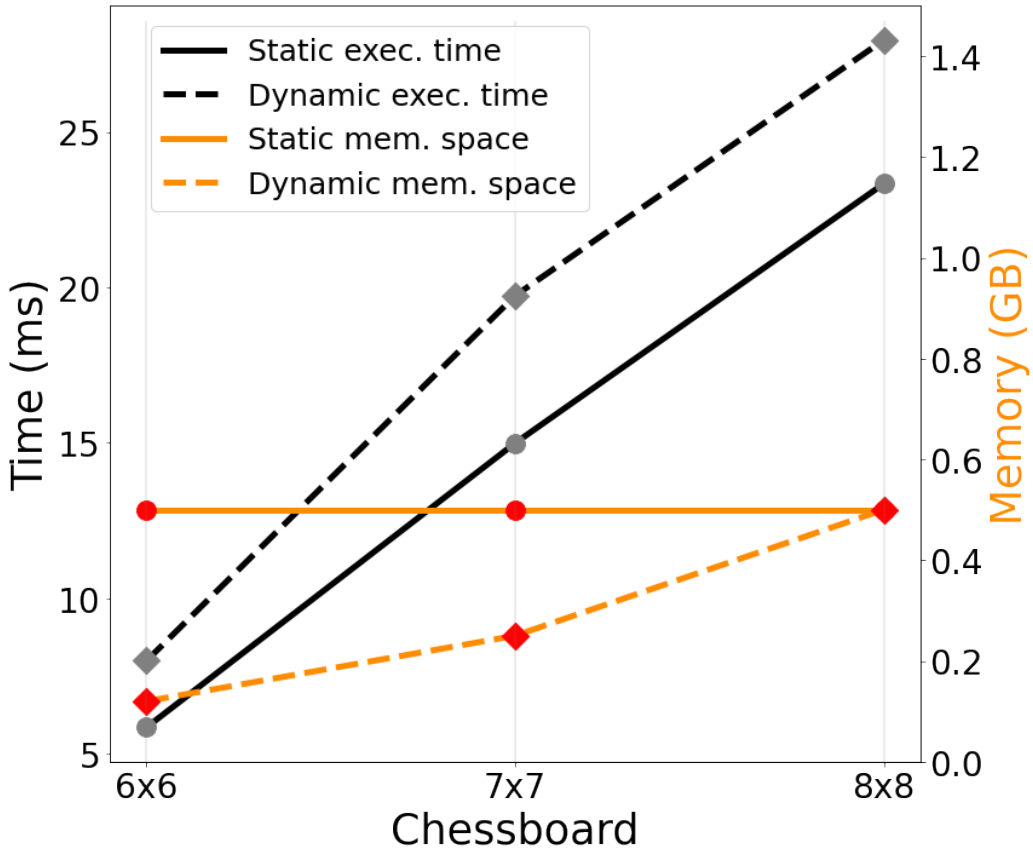
Therefore, what we did w.r.t the classical sequential implementation of the `ITE` algorithm was to redefine the rows from 21 to 25

Parallel ITE in HermesBDD

- The `ITE` routine is declared in `HermesBDD/src/node.hh`
  - In particular, at line 85 we can see `ITE(A, B, C)`
- And it is implemented in `HermesBDD/src/node.cpp`
  - Line 204 for the `ITE(A, B, C)`
- Exercise: Look at the code. Is everything clear? What does the section of code from line 255 to line 310 represent?

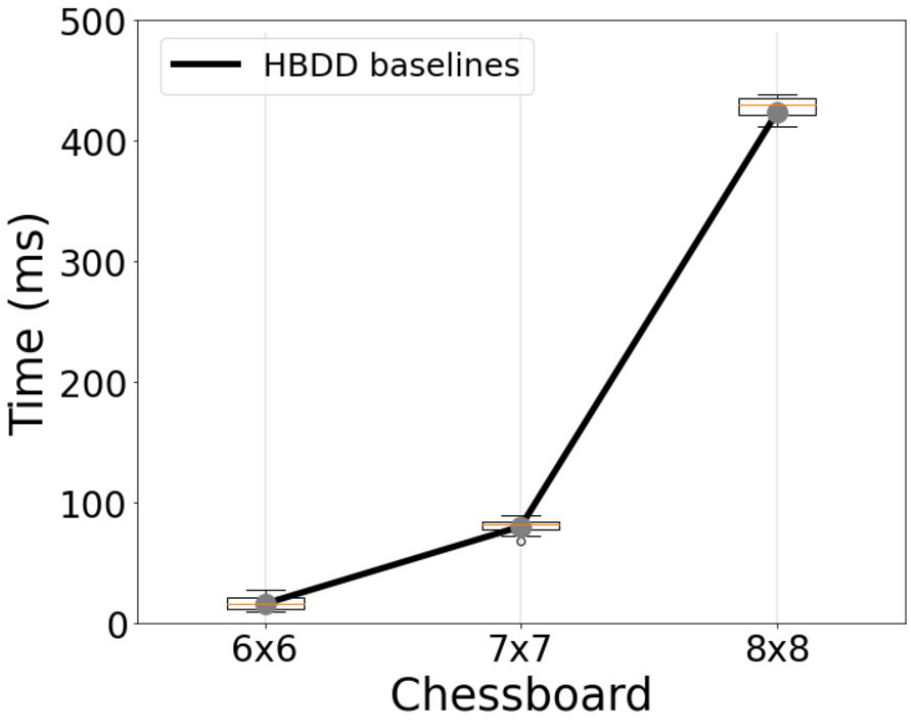
Our Results

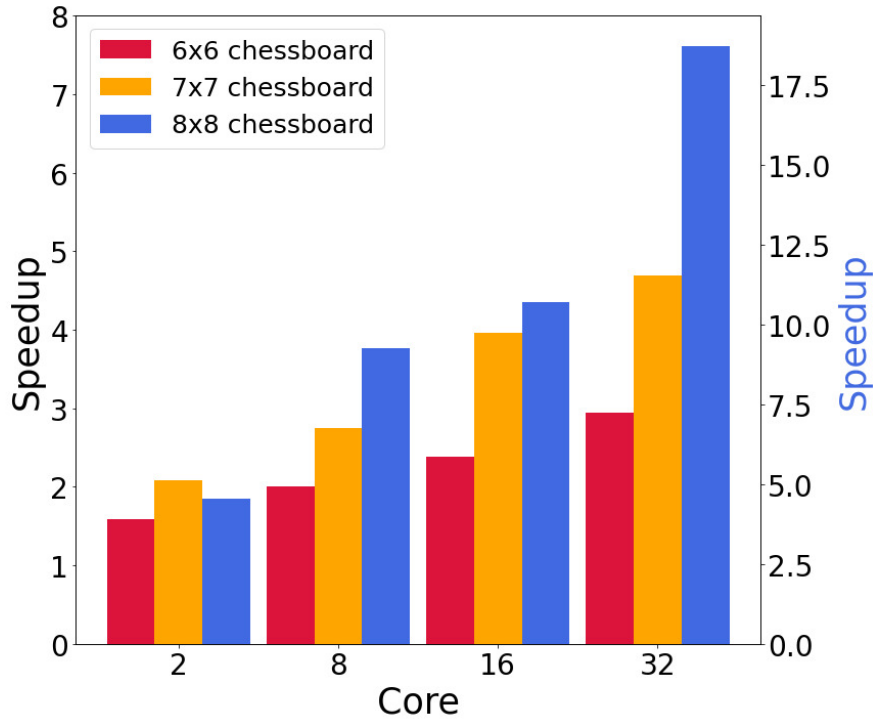
HermesBDD results on the impact of memory allocation in terms of time and memory space, on an average of 50 samples of the *n*-Queens problem on a 32-core machine



Our Results (cont'd)

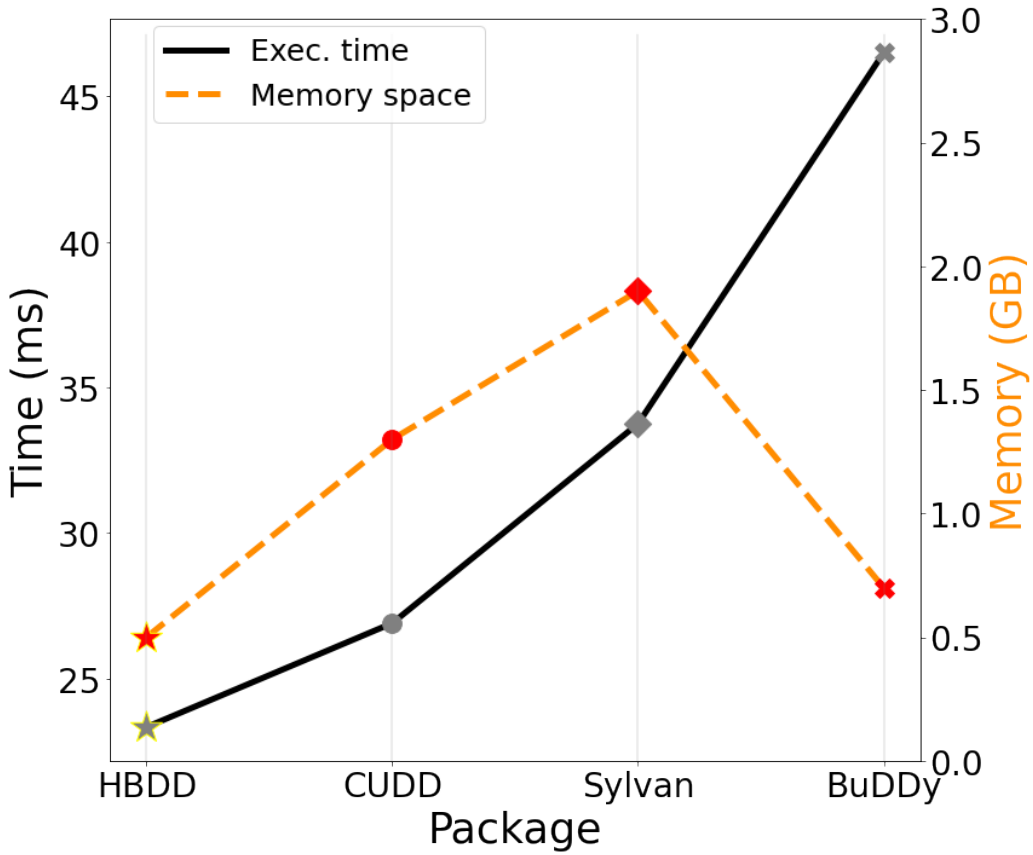
This plot shows our non-parallel baselines (left) and parallel speedup (right) on the *n*-Queens problems with the 6×6, 7×7 and 8×8 chessboard. Values on the average of 50 samples using the static memory allocation on a 32-core machine





Our Results (cont'd)

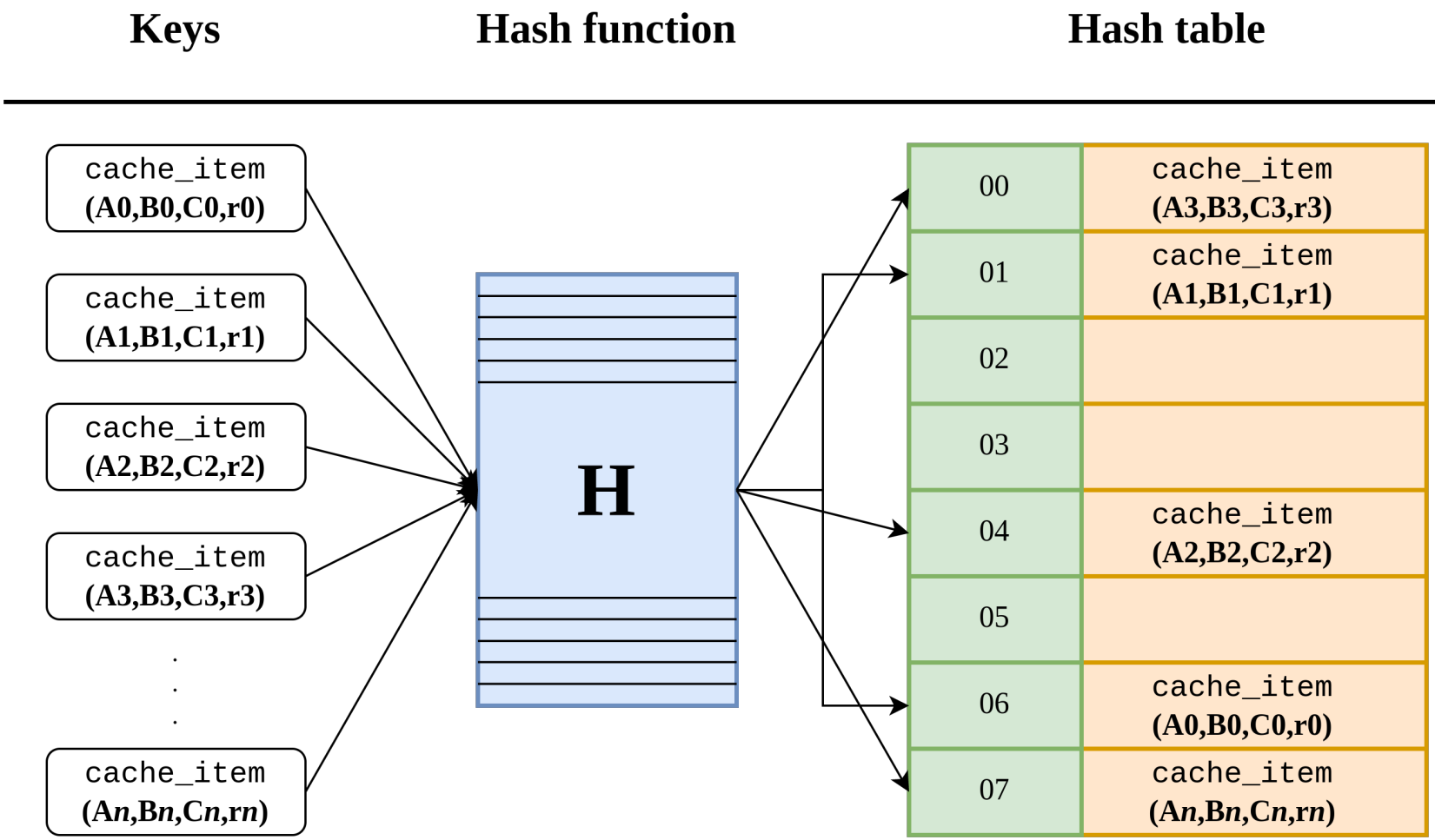
HermesBDD w.r.t. CUDD, Sylvan, and BuDDy execution time and memory space required. Results were obtained on average from 50 samples of the *n*-Queens problem, using a 32-core machine, and the static memory allocation



The Caching Mechanism

- The **biggest performance bottleneck** in BDD packages is the **long latency of the main memory**
- Thus, the main architectural decisions for the new library are motivated by the desire to be as cache-friendly as possible
- **So, in order to minimize the execution time, in HermesBDD we developed a dynamic cache management algorithm based on a hash table**

The Caching Mechanism (cont'd)



Caching Mechanism in HermesBDD

- The caching mechanism fot the `ITE` routine is declared in `HermesBDD/src/node.hh`
  - In particular, at line 94 we can see `ITE_without_cache(A, B, C)`
- And it is implemented in `HermesBDD/src/node.cpp`
  - Line 204 for the `ITE_without_cache(A, B, C)`
- In addition, the cache initialization is located in `HermesBDD/src/memory_manager.cpp`, where at line 93 we can find `cache.init(cache_size)`
- Exercise: Look at the code. Is everything clear? In lines 236 and 244 we find the function `cache.findITE()` and `cache.insertITE()`. Where are they declared? And implemented? How do they work?

Outline

How to create a new program

- In this lecture, we are going to do the assigned exercises in the form of **library tests**
- First, create a new file in the folder `test/`, *e.g.*, `es1.cpp`
- Then, add this file in the `set()` variable of the file `HermesBDD/tests/CMakeLists.txt`
- At this point, compile the library again. The executable of the test will be in the `HermesBDD/build/tests/` folder

Code!

- Exercise: look at the example `test_hermesbdd`. Is everything clear? Is there any functionality you didn't understand?
- Exercise: write the code to build the BDD for the function $f = \neg x_1$

HINTS

- Look inside the `HermesBDD/src/bdd.hpp`, there might be some functions that will help you out...

Code! (cont'd)

- Exercise: write the code to build the BDD for the function $f = x_1 \wedge x_2$
- Exercise: write the code to build the BDD for the function $f = x_1 \vee x_2$
- Exercise: write the code to build the BDD for the function $f = x1 \oplus x2$
- Exercise: write the code for testing the N-queen problem using a 3×3 chessboard