OS-Level Task-Based Mechanism for Lightweight Manycore Processors

João Vicente Souto
joao.vicente.souto@posgrad.ufsc.br

Dept. of Informatics and Statistics
Federal University of Santa Catarina - Florianópolis

November 10, 2020

LaPeSD

UFSC

Presentation Outline

Introduction

Introduction

- **Processing power vs Power consumption**
- Massive thread-level parallelism with low-power consumption
- Lightweight manycores exhibit distinct architectural characteristics

LaPeSD

Introduction

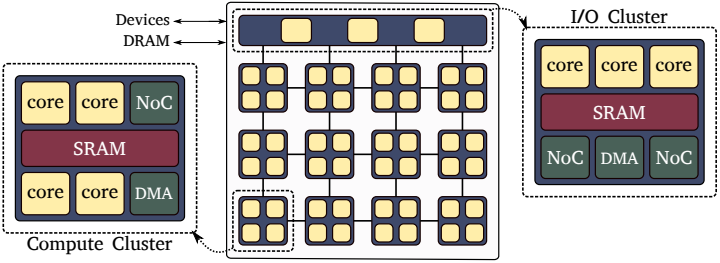
- Processing power vs Power consumption
- **Massive thread-level parallelism with low-power consumption**
- Lightweight manycores exhibit distinct architectural characteristics

LaPeSD

Introduction

- Processing power vs Power consumption
- Massive thread-level parallelism with low-power consumption
- **Lightweight manycores exhibit distinct architectural characteristics**

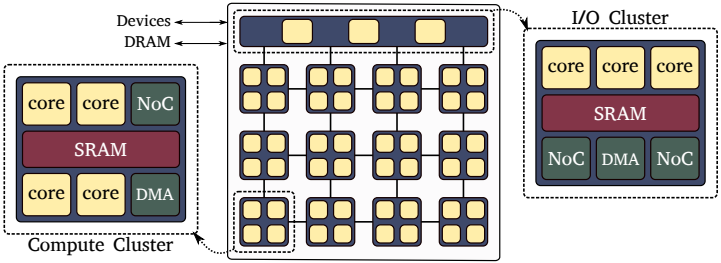
Lightweight Manycores Particularities



Overview of a Manycore

- **Hundreds of Lightweight Cores**
  - Expose Massive thread-level parallelism
  - Feature low-power consumption
  - Target MIMD workloads
- Distributed Memory Architecture
- On-Chip Heterogeneity

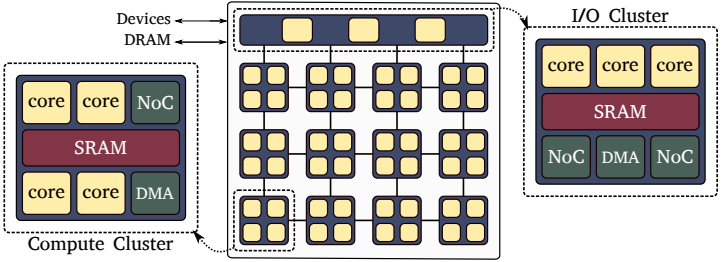
Lightweight Manycores Particularities



Overview of a Manycore

- Hundreds of Lightweight Cores
- **Distributed Memory Architecture**
  - Grants scalability
  - Relies on a Network-on-Chip (NoC)
  - Has constrained memory systems
- On-Chip Heterogeneity

Lightweight Manycores Particularities



Overview of a Manycore

- Hundreds of Lightweight Cores
- Distributed Memory Architecture
- **On-Chip Heterogeneity**
  - Features different components

Motivation

- *How to deal with* the **reduced amount of local memory** of a cluster?
- Local memory is **not exclusive** to the user
  - OS code and data
  - User code and data
  - Libraries
- Support for **multiple execution streams consume a considerable amount of memory**

LaPeSD

Goals and Contributions

- Design an **OS-level task-based mechanism**
- Enhancement **memory use**
- Improve **core utilization**
- Facilitate the *modeling of internal kernel functionalities*

LaPeSD

Problem

Problem Definition

- **Memory management** influences all abstraction levels
- **OSes must be lightweight** to make the most memory available to the application
- **Asymmetric microkernel** alleviate cache coherence problems

Problems

- **Memory utilization:** each new thread requires memory pages
- Data locality: shared-memory region competition and no hardware support for cache coherence
- Core utilization: asymmetric microkernel does not use master core between syscall resquests
- Asynchronous operations: *lightweight manycores* may not feature a DMA, thus, a thread must polling data into the NoC
- Periodic operations: receive external commands and data requires that a thread exists to request a communication check

Problems

- Memory utilization: each new thread requires memory pages
- **Data locality:** shared-memory region competition and no hardware support for cache coherence
- Core utilization: asymmetric microkernel does not use master core between syscall resquests
- Asynchronous operations: *lightweight manycores* may not feature a DMA, thus, a thread must polling data into the NoC
- Periodic operations: receive external commands and data requires that a thread exists to request a communication check

Problems

- Memory utilization: each new thread requires memory pages
- Data locality: shared-memory region competition and no hardware support for cache coherence
- **Core utilization:** asymmetric microkernel does not use master core between syscall resquests
- Asynchronous operations: *lightweight manycores* may not feature a DMA, thus, a thread must polling data into the NoC
- Periodic operations: receive external commands and data requires that a thread exists to request a communication check

Problems

- Memory utilization: each new thread requires memory pages
- Data locality: shared-memory region competition and no hardware support for cache coherence
- Core utilization: asymmetric microkernel does not use master core between syscall resquests
- **Asynchronous operations:** *lightweight manycores* may not feature a DMA, thus, a thread must polling data into the NoC
- Periodic operations: receive external commands and data requires that a thread exists to request a communication check

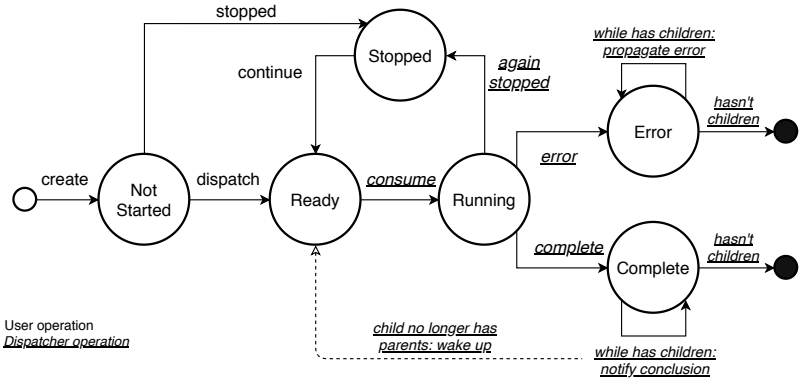
Problems

- Memory utilization: each new thread requires memory pages
- Data locality: shared-memory region competition and no hardware support for cache coherence
- Core utilization: asymmetric microkernel does not use master core between syscall resquests
- Asynchronous operations: *lightweight manycores* may not feature a DMA, thus, a thread must polling data into the NoC
- **Periodic operations:** receive external commands and data requires that a thread exists to request a communication check

Solution

- **Support for multiple simple execution streams** decoupled from threads
- *Task abstraction* is a special case of coroutines
- A **Task encapsulates a subroutine** that can be performed regardless of who created it
- Introduce a **dispatcher**, a generic task executor

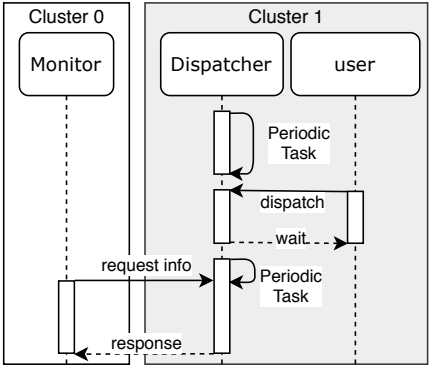
Task State


stopped
Stopped
continue
again
stopped
while has children:
propagate error
Error
hasn't
children
error
create
Not
Started
dispatch
Ready
consume
Running
complete
Complete
hasn't
children
User operation
Dispatcher operation
child no longer has
parents: wake up
while has children:
notify conclusion

Dispatcher Algorithm


```
while Not shutdown do
    Waits for a task;
    Execute task function;
    switch Task return do
        case TASK_RET_SUCCESS do
            Complete the task and schedule children;
        end
        case TASK_RET_AGAIN do
            Reschedule the task;
        end
        case TASK_RET_STOP do
            Insert the task into a waiting queue;
        end
        case TASK_RET_ERROR do
            Propagate the error and release all tasks;
        end
    end
end
```

Asynchronous/Periodic Tasks



Solutions

- **Memory utilization:** multiple execution streams define into task, reduced to a limited number of dispatchers, i.e., less memory used
- Data locality: dispatcher explore cache locality
- Core utilization: dispatcher can cooperate with the master thread and use the idle time of master core
- Asynchronous operations: dedicated task to perform asynchronous tasks instead of a thread
- Periodic operations: periodic tasks can be created to perform this kind of operations

Solutions

- Memory utilization: multiple execution streams define into task, reduced to a limited number of dispatchers, i.e., less memory used
- **Data locality:** dispatcher explore cache locality
- Core utilization: dispatcher can cooperate with the master thread and use the idle time of master core
- Asynchronous operations: dedicated task to perform asynchronous tasks instead of a thread
- Periodic operations: periodic tasks can be created to perform this kind of operations

Solutions

- Memory utilization: multiple execution streams define into task, reduced to a limited number of dispatchers, i.e., less memory used
- Data locality: dispatcher explore cache locality
- **Core utilization:** dispatcher can cooperate with the master thread and use the idle time of master core
- Asynchronous operations: dedicated task to perform asynchronous tasks instead of a thread
- Periodic operations: periodic tasks can be created to perform this kind of operations

Solutions

- Memory utilization: multiple execution streams define into task, reduced to a limited number of dispatchers, i.e., less memory used
- Data locality: dispatcher explore cache locality
- Core utilization: dispatcher can cooperate with the master thread and use the idle time of master core
- **Asynchronous operations:** dedicated task to perform asynchronous tasks instead of a thread
- Periodic operations: periodic tasks can be created to perform this kind of operations

Solutions

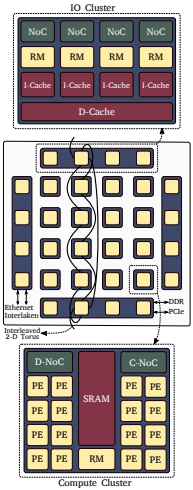
- Memory utilization: multiple execution streams define into task, reduced to a limited number of dispatchers, i.e., less memory used
- Data locality: dispatcher explore cache locality
- Core utilization: dispatcher can cooperate with the master thread and use the idle time of master core
- Asynchronous operations: dedicated task to perform asynchronous tasks instead of a thread
- **Periodic operations:** periodic tasks can be created to perform this kind of operations

Experiments

Experimental Environment

Lightweight Manycore Kalray MPPA-256

- **288 processing cores**
  - 16 Compute Cluster (CC)
  - 4 I/O Cluster (IO)
- **Local Memory**
  - CC: 2 MB of SRAM
  - IO: 4 MB of SRAM
- **Network-on-Chip (NoC)**
  - Data NoC (D-NoC)
  - Control NoC (C-NoC)



Experiments

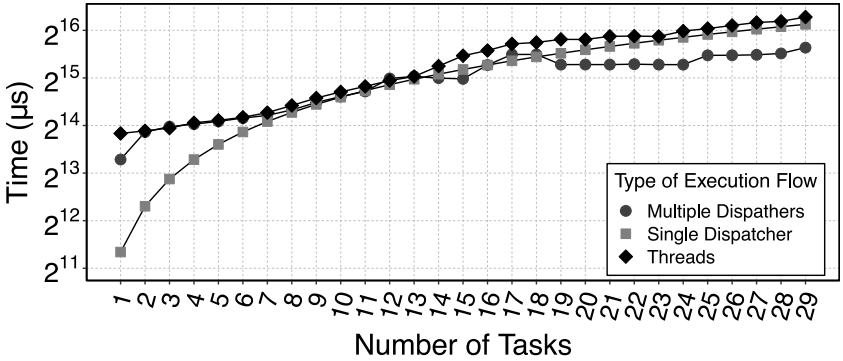
- **Synthetic benchmark:** write each byte of 16 memory pages (64 KB)
- Ensure **95% of confidence**
- 50 replications, discarding the first 10 (warm-up period)

Benchmark	#Tasks	#Threads	Memory
Single Dispatcher	[1, 29]	1	8 KB
Multiple Dispatchers	[1, 29]	14	112 KB
Threads	[1, 29]	[1, 29]	[8, 232] KB

Table 1: Benchmark parameters

Results

Results



Conclusions

Conclusions

- Lightweight manycores are promising candidates for computing systems to reach the *exascale* era
- Distinctive features introduce new challenges on software development
- **Memory system is a critical component**
- Proposed a **OS-level task-based mechanism to improve some types of memory use**
- Results showed that our proposed solution **has similar performance** to the thread-based one
- Future work:
  - Model communication system of Nanvix using tasks
  - Introduce well-known ordering scheduling algorithms

LaPeSD

Obrigado!
Perguntas?

João Vicente Souto
joao.vicente.souto@posgrad.ufsc.br

Dept. of Informatics and Statistics
Federal University of Santa Catarina - Florianópolis

November 10, 2020

LaPeSD

UFSC

References I

BARBALACE, A. et al. Popcorn: Bridging the Programmability Gap in Heterogeneous-ISA Platforms. In: *Proceedings of the 10th European Conference on Computer Systems*. Bordeaux, France: ACM, 2015. (EuroSys '15), p. 1–16. ISBN 978-1-4503-3238-5. Disponível em: http://dl.acm.org/citation.cfm?doid=2741948.2741962.

BAUMANN, A. et al. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*. Big Sky, Montana, USA: ACM, 2009. (SOSP '09), p. 29–44. ISBN 978-1-60558-752-3. Disponível em: https://dl.acm.org/citation.cfm?doid=1629575.1629579.

CASTRO, M. et al. Seismic wave propagation simulations on low-power and performance-centric manycores. *Parallel Computing*, v. 54, p. 108—-120, 2016. ISSN 01678191.

DINECHIN, B. D. de et al. A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor. In: *Procedia Computer Science*. Barcelona, Spain: Elsevier, 2013. (ICCS '13, v. 18), p. 1654–1663. ISBN 1877-0509. Disponível em: http://linkinghub.elsevier.com/retrieve/pii/S1877050913004766.

References II

FRANCESQUINI, E. et al. On the Energy Efficiency and Performance of Irregular Application Executions on Multicore, NUMA and Manycore Platforms. *Journal of Parallel and Distributed Computing (JPDC)*, v. 76, n. C, p. 32–48, fev. 2015. ISSN 0743-7315. Disponível em: http://linkinghub.elsevier.com/retrieve/pii/S0743731514002093.

FREITAS, H. C. de. *Arquitetura de NoC Programável Baseada em Múltiplos Clusters de Cores para Suporte e Padrões de Comunicação Coletiva*. Tese (Doutorado) — Programa de Pós-Graduação em Computação, UFRGS, Porto Alegre, 6 2009. An optional note.

KELLY, B.; GARDNER, W.; KYO, S. AutoPilot: Message Passing Parallel Programming for a Cache Incoherent Embedded Manycore Processor. In: *Proceedings of the 1st International Workshop on Many-core Embedded Systems*. Tel-Aviv, Israel: ACM, 2013. (MES '13), p. 62–65. ISBN 978-1-4503-2063-4. Disponível em: http://dl.acm.org/citation.cfm?doid=2489068.2491624.

KLUGE, F.; GERDES, M.; UNGERER, T. An Operating System for Safety-Critical Applications on Manycore Processors. In: *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. Reno, Nevada, USA: IEEE, 2014. (ISORC '14), p. 238–245. ISBN 978-1-4799-4430-9. Disponível em: http://ieeexplore.ieee.org/document/6899155/.

References III

KOGGE, P. et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Techinal Representative*, v. 15, 01 2008.

NIGHTINGALE, E. et al. Helios: Heterogeneous Multiprocessing with Satellite Kernels. In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. Big Sky, Montana, USA: ACM, 2009. (SOSP '09), p. 221–234. ISBN 978-1-60558-752-3. Disponível em: http://doi.acm.org/10.1145/1629575.1629597.

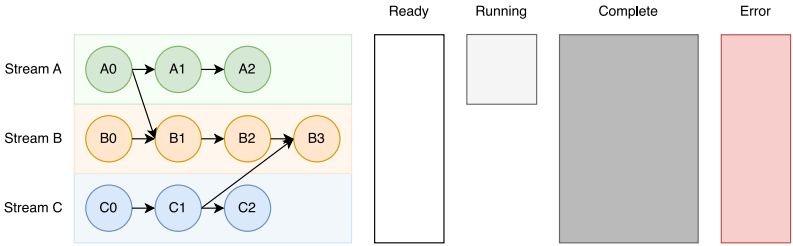
OLOFSSON, A.; NORDSTROM, T.; UL-ABDIN, Z. Kickstarting High-Performance Energy-Efficient Manycore Architectures with Epiphany. In: *2014 48th Asilomar Conference on Signals, Systems and Computers*. Pacific Grove, California, USA: IEEE, 2014. (ACSSC '14), p. 1719–1726. ISBN 978-1-4799-8297-4. Disponível em: http://ieeexplore.ieee.org/document/7094761/.

RHODEN, B. et al. Improving Per-Node Efficiency in the Datacenter with New OS Abstractions. In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. Cascais, Portugal: ACM, 2011. (SoCC '11), p. 1–8. ISBN 978-1-4503-0976-9. Disponível em: https://dl.acm.org/citation.cfm?id=2038941.

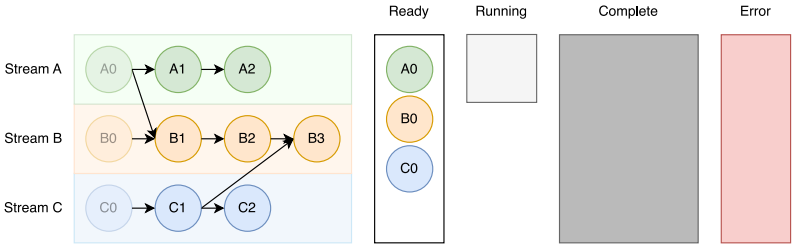
References IV

ZHENG, F. et al. Cooperative Computing Techniques for a Deeply Fused and Heterogeneous Many-Core Processor Architecture. *Journal of Computer Science and Technology (JCST)*, v. 30, n. 1, p. 145–162, jan. 2015. ISSN 1000-9000. Disponível em: https://link.springer.com/article/10.1007%2Fs11390-015-1510-9.

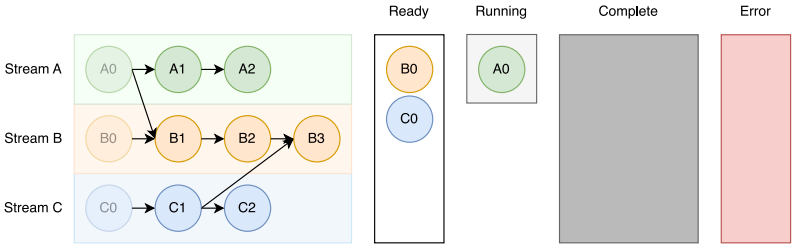
Task Example



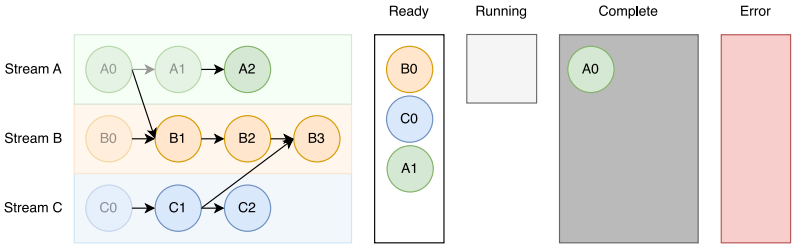
Task Example



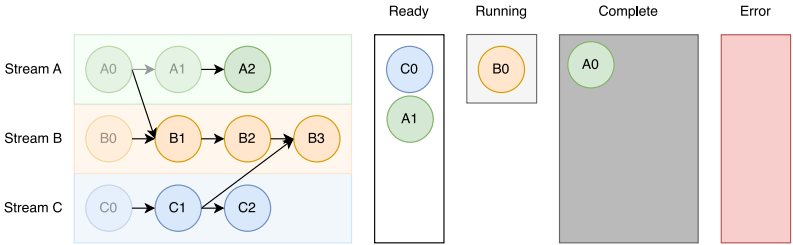
Task Example



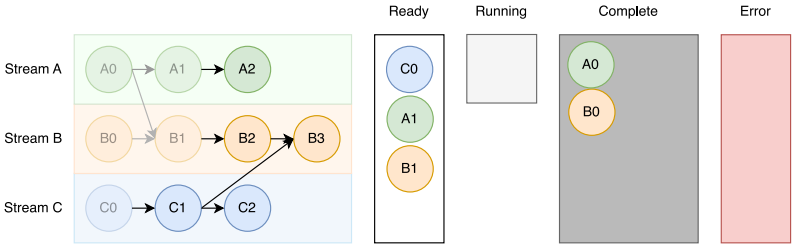
Task Example



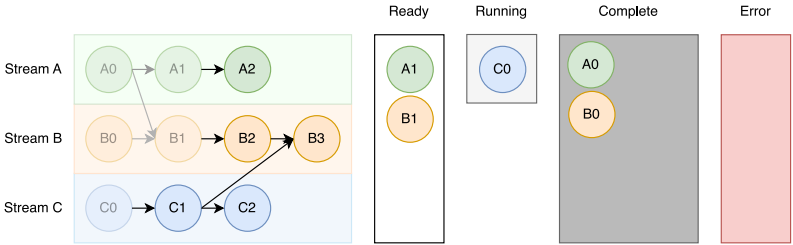
Task Example



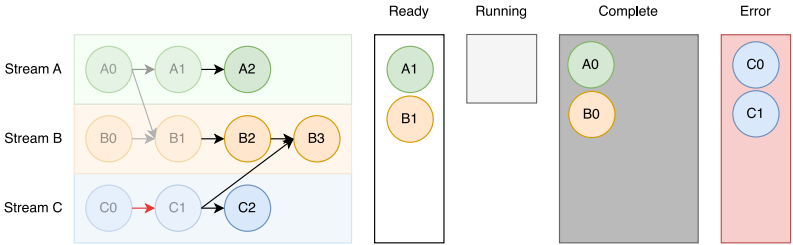
Task Example



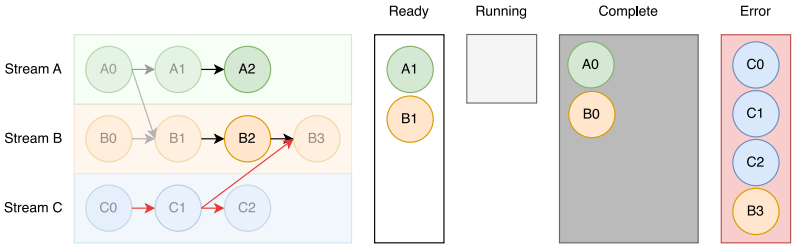
Task Example



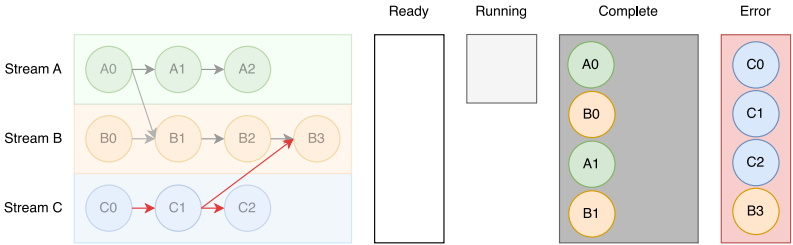
Task Example



Task Example



Task Example



Referências

(FREITAS, 2009). (KOGGE et al., 2008). (**??**). (**??**). (**??**). (DINECHIN et al., 2013). (OLOFSSON; NORDSTROM; UL-ABDIN, 2014). (ZHENG et al., 2015). (KELLY; GARDNER; KYO, 2013). (FRANCESQUINI et al., 2015). (CASTRO et al., 2016). (BARBALACE et al., 2015). (BAUMANN et al., 2009). (KLUGE; GERDES; UNGERER, 2014). (NIGHTINGALE et al., 2009). (RHODEN et al., 2011).