# Cenários de uso e aplicação de *Spin Locks*: uma mini revisão sistemática da literatura

Gert Müller[1], Larissa Falcão[1]

[1] Universidade Federal de Pernambuco, Centro de Informática, Recife, Brasil
{gumn,lctf}@cin.ufpe.br

**Resumo.** Na Programação Paralela (PP), o conceito de Locks é bastante sólido e difundido. No entanto, existem poucos estudos que evidenciam os cenários onde estas devem ser utilizadas. Dessa forma, se torna difícil a escolha de qual tipo utilizar quando não se conhece a fundo as implementações de cada uma. Logo, este trabalho tenta evidenciar, de forma rápida, simples e concisa, a influência dos contextos de hardware e software na escolha de Spin Locks, se utilizando para isso de uma mini revisão sistemática da literatura.

**Palavras-chave:** programação paralela; Spin Locks; mini revisão sistemática

## 1 Introdução

Com a criação e evolução dos microprocessadores multi-core percebeu-se a necessidade de utilizar múltiplos núcleos ou processadores no desenvolvimento de software. Isso aconteceu porque os computadores pessoais passaram a prover essa evolução em termos de hardware. Para isso é necessária a utilização da programação paralela.

Como afirma Freitas [7], os computadores pessoais, estações de trabalho e clusters de alto desempenho são baseados em arquiteturas paralelas, mas para explorá-los totalmente, é importante programar em paralelo. Segundo Diaz, Munoz-Caro e Nino [2], a computação paralela pode aumentar o desempenho de sistemas, executando-os em vários processadores.

Na programação paralela de é comum o uso de threads para executar tarefas de forma concorrente/paralela. Como afirmam Herlihy e Shavit [10], quando utiliza-se múltiplas threads existe a necessidade de utilizar uma zona de exclusão mútua: um bloco de código que pode ser executado por apenas uma thread por vez. A forma padrão para abordar a exclusão mútua é através de um objeto Lock implementado de forma correta. Isso acontece quando uma thread adquire (locks alternadamente) um lock quando executa chamada do método lock(), e libera (desbloqueia alternadamente) o lock quando se executa a chamada de método unlock(). Segundo Herlihy e Shavit [10], quando a thread não pode adquirir o lock ela pode continuar tentando adquirir o lock, nesse caso o lock é chamado de Spin Lock.

Dyba, Kitchenham e Jorgensen [4], afirmam que os engenheiros de software podem tomar decisões incorretas em relação a adoção de novas técnicas e tecnologias, se não considerar as evidências científicas sobre a eficácia das mesmas. Eles devem considerar

o uso de procedimentos semelhantes aos desenvolvidos para a medicina baseada na evidência. Esses procedimentos são os da Engenharia de Software Baseada em Evidências.

Apesar do conceito de Locks ser um conceito bastante sólido e difundido, é difícil a escolha de qual tipo utilizar quando não se conhece a fundo as implementações de cada uma. Este trabalho apresenta um Mini Mapeamento Sistemático que busca identificar de forma rápida, simples e concisa, a influência dos contextos de hardware e software na escolha de Spin Locks.

O restante deste trabalho está organizado da seguinte forma: na Seção 2 apresenta-se a fundamentação teórica necessária para o entendimento do trabalho, na Seção 3 é apresentada a metodologia utilizada. A Seção 4 descreve os resultados obtidos e discussão acerca dos mesmos, e a Seção 5 traz as considerações finais e trabalhos futuros.

## 2 Fundamentação Teórica

Nesta seção será apresentada a fundamentação teórica realizada através de uma revisão bibliográfica informal. Os tópicos abordados são programação paralela, locks, spin locks e engenharia de software baseada em evidências. As subseções a seguir explicam cada um dos tópicos citados anteriormente.

### 2.1 Programação Concorrente e Paralela

A indústria de computadores tem desfrutado de um grande avanço nos últimos 30 anos, onde a lei de Moore levou avanços aos processadores, que por sua vez levaram ciclos de atualização de capacidades cada vez maiores. Nesse contexto surgiram os processadores multi-core que podem, em princípio, ter uma performance melhor com a utilização de níveis mais baixos de energia. Mas, para que isso aconteça é necessário utilizar softwares concorrentes/paralelos [18].

Hoje em dia, podemos falar de duas abordagens de microprocessadores [12]. A primeira, abordagem, multi-core, integra alguns núcleos em um único microprocessador, laptops e desktops reais incorporaram este tipo de processador. A segunda abordagem, many-core, utiliza um grande número de núcleos (atualmente da ordem de várias centenas) e está especialmente orientada para a taxa de transferência de execução de programas paralelos. Esta abordagem é exemplificada pelas Unidades de processamento gráfico (GPUs) disponíveis hoje [2].

Nesse contexto, a computação paralela pode aumentar o desempenho das aplicações executando-os em vários processadores/núcleos [2]. Freitas [7] afirma que para reduzir o tempo de processamento, a carga tem de ser dividida em várias partes, para serem executados simultaneamente. Neste cenário, podemos encontrar questões relacionadas com algoritmos (simultaneidade e granularidade), sistemas operacionais (processo/sincronismo de threads, programação e mapeamento), memória e redes (arquitetura e comunicação gerais), e de avaliação de desempenho (aumento de velocidade, escalabilidade e eficiência).

Neste contexto, o paralelismo é integrado de várias maneiras, e sequencialidade não deve mais ser visto como o paradigma principal, mas apenas como uma das possíveis formas de cooperação entre as entidades individuais. Tem sido argumentado por um longo período de tempo que o paralelismo em uma linguagem de programação torna a tarefa de programação muito mais difícil, especialmente se a linguagem de programação a ser usada não tem recursos que permitam separar o problema a ser resolvido a partir da sua implementação [9].

A programação paralela é um desafio para uma série de razões. Além de todos os desafios da computação sequencial, programas paralelos também podem sofrer de condições de corrida e de impasse, e mesmo se correto pode ser não-determinístico, o que complica o teste. Atingir um alto desempenho com um programa paralelo também exige a minimização de contenção de recursos limitados, tais como comunicação e largura de banda de memória. A falha de justificar adequadamente esses fatores pode levar a programas que dramaticamente sejam abaixo de sua performance [21].

Linguagens de programação clássicas e bibliotecas têm dificuldades para lidar com esses vários níveis de paralelismo. A principal razão é que, sem um bom conhecimento do funcionamento paralelo do computador, é fácil introduzir atrasos, processo desequilibrado ou grandes comunicações. Linguagens de alto nível visam facilitar o desenvolvimento de aplicações paralelas e favorecer a reutilização de código, desde que o compilador é capaz de otimizar o código de acordo com o computador de destino. É claro que tal linguagem raramente produz o programa mais otimizado, portanto, um bom equilíbrio deve ser encontrado entre esses dois objetivos aparentemente contrários [16].

Muitas bibliotecas estão disponíveis em várias linguagens de programação para desenvolver softwares parelelos (por exemplo, C ++, Haskell). Como os padrões de design, esqueletos de algoritmos pode ser implementados por hardwares diferentes e alguns sistemas oferecem ferramentas de auto-ajuste para escolher a aplicação mais adequada para um determinado computador [16].

Em ambos os casos, as bibliotecas podem ser estendidas para acompanhar os avanços na tecnologia de hardware preservando a compatibilidade dos programas existentes. Este tipo de tecnologia de software é o futuro da programação paralela fornecendo ambas as ferramentas de programação de alto nível e implementações de adaptação eficientes [16].

Escrevendo um código paralelo em C com Pthreads exige do programador criar threads manualmente e de forma explícita, para atribuir tarefas para cada segmento e decidir quantas threads serão criadas. O OpenMP não requer do programador criar threads explicitamente, mas exige que ele decida quantas threads serão invocadas. Por outro lado, TBB cria threads implicitamente e determina automaticamente o número de threads que serão invocados [17].

Pthreads exige do programador cuidar da sincronização explicitamente entre threads usando travas. Entretanto, escrever código com Pthreads é mais complexo comparar a OpenMP e TBB, mas Pthreads é mais flexível, enquanto o poder expressivo de OpenMP e TBB é limitado [17].

Enquanto inúmeros modelos de programação, Application Programming Interfaces (APIs) e padrões de projeto paralelo existem para programação paralela e paralelização

de programas sequenciais, a complexidade da elaboração de programas paralelos corretos continuamente tem sido subestimada; modelos de programação existentes/APIs não possuem a estrutura necessária que permite que os desenvolvedores de raciocinar sobre os comportamentos dos sistemas que produzem [5].

Durante décadas, pesquisadores e desenvolvedores de linguagem têm vindo a explorar e propor bibliotecas e linguagens paralelas (PLL, muitas vezes integrados em conjunto em um ambiente de programação.), além de extensões para apoiar a computação paralela em larga escala [11].

### 2.2 Locks

Usar múltiplas threads de forma explícita continua a ser o caminho mais direto para programar sistemas paralelos. No mundo de programação multithread, a interferência entre as threads é uma questão importante e resulta em defeitos de difícil rastreamento, tais como condições de corrida ou impasses. Tradicionalmente os programadores têm coordenado threads usando padrões de programação baseadas em travas (locks) de exclusão mútua [26].

Problemas com locks incluem deadlock, livelock, desaceleração de desempenho, e as condições de corrida. Além disso, os locks ocultos, muitas vezes no sistema de bibliotecas devem ser considerados, assim como os definidos pela aplicação [22]. No entanto, o aumento de concorrência em um programa é frequentemente não-trivial, devido a vários possíveis gargalos de escalabilidade. Um gargalo comum é a contenção de locks, onde a escalabilidade é limitada com várias threads esperando para adquirir algum lock com segurança para acesso à memória compartilhada [24].

As seções críticas protegidas com locks limitam o desempenho de programas paralelos ou cuja execução tem de ser concorrente. Eles podem ser divididos em três fases: lock de transferência, carga/computação e de desbloqueio. A segunda fase é inata ao algoritmo paralelo, enquanto que a primeira e a terceira fases são dependentes da implementação de locks e apenas necessário para a execução correta. O mesmo padrão de acesso aplica-se a estruturas de dados protegidos por locks de leitura e escrita, o que permite um único escritor ou vários leitores simultâneos. Para proporcionar o máximo de desempenho, um sistema paralelo deve fornecer uma implementação com baixo overhead de locks [27].

A fim de reduzir a complexidade da programação concorrente, línguagens modernas de alto nível mais populares - Java e C # fornecem locks reentrantes, que de programação concorrente facilidade. No entanto, é difícil de usar locks reentrantes corretamente e que o uso incorreto pode resultar em erros simultâneos desagradáveis. Linguagens de alto nível existentes não oferecem quaisquer mecanismos eficazes para evitar tais erros, por isso, é importante desenvolver uma técnica de verificação para raciocinar sobre programas concorrentes com locks reentrantes. O mecanismo de reentrada permite que uma thread volte a adquirir um lock que já detém [8].

Como representar sintaticamente mecanismos correspondentes na linguagem pode, é claro, variar. Uma opção é a delimitação do âmbito lexical, por exemplo com base no métodos sincronizados para spin locks em Java, ou usando uma palavra-chave atômica a definição das regiões protegidas ou abordagens semelhantes [25].

O suporte embutido para o controle concorrente em Java é baseada em trava; cada objeto está equipado com uma trava (reentrante), que pode ser usado para especificar métodos sincronizados. O controle à base de trava em Java oferece escopo léxico de proteção com base em exclusão mútua. A classe ReentrantLock e a interface de Lock permitem mais liberdade ao programador [25].

Ao fazer travamento mais refinado pode-se aumentar a concorrência, mas potencialmente os riscos da introdução de condições de corrida sutis. No contexto de Java, a biblioteca padrão java.util.concurrent (abreviada como j.u.c.) fornece um número de estruturas de dados e construções de travamento que também podem ser úteis, com as suas próprias vantagens e desvantagens [24].

Entretanto, muitas dificuldades surgem quando se transforma manualmente um programa para usar as construções de travas de j.u.c., motivando um melhor suporte da ferramenta. Em primeiro lugar, essas construções não têm a sintaxe concisa e intuitiva dos blocos sincronizados associados com monitor integrado travas de Java. Em vez disso, os locks são modelados como objetos e operações de travas como chamadas de método, e o ônus é do programador para garantir que a aquisição e liberação de locks são devidamente combinados. Em segundo lugar, o desempenho relativo de diferentes tipos de locks depende fortemente do número de módulos e sua carga de trabalho, e na arquitetura e JVM que está sendo usada [24].

Por isso, os programadores podem ter de alternar entre diferentes tipos de locks para determinar o melhor lock para o programa. Além disso, todos os blocos de código usando a mesma trava devem ser transformados em um conjunto para garantir a preservação de comportamento, e descobrir todos esses blocos pode ser não trivial. Em alguns casos, a migração para travas avançadas é impossível quando o programa estende-se uma estrutura que conta com uma forma específica de sincronização [24].

### 2.3 Spin Locks

Spin lock é uma das primitivas de sincronização fundamentais para o acesso exclusivo a recursos compartilhados em multiprocessadores de memória compartilhada. Ele geralmente é executado com operações atômicas de leitura-modificação-escrita em uma única palavra (ou palavras contínuas alinhadas) de memória compartilhada, como test-and-set, fetch-and-store (swap), ou compare-and-swap. Vários algoritmos de Spin Lock usando essas operações têm sido propostos [1].

O uso de um algoritmo de spin-lock para a exclusão mútua foi sugerido pela primeira vez em um artigo escrito por Dijkstra [3], que propôs um algoritmo que resolveu conflitos em decorrência da concorrência ao atribuir o lock do processo que emitiu o último pedido de escrita executado [20].

A alocação de recursos tem um grande impacto no desempenho de microprocessadores com mais de um núcleo. Um processo pode solicitar o acesso exclusivo a um recurso usando um Spin Lock, que controla uma estrutura de dados compartilhada em um laço, "busy-waiting" ou "spinning", até que o estado da estrutura compartilhada indica que o lock é adquirido pelo processo solicitante. Spin-locks são geralmente usa-

dos para proteger as seções críticas curtas onde o tempo de suspender/reiniciar um processo de solicitação é maior do que o tempo gasto na checagem da variável compartilhada [20].

No entanto, um algoritmo de spin lock não é trivial de ser contruído, o projeto de uma estrutura de dados adequada para spin locks requer a consideração de ambos, desempenho escalável e justiça. Um rápido crescimento no número de pedidos de uma estrutura de dados compartilhada pode potencialmente saturar o sistema de memória, e um lock mal projetado pode causar starvation [20]. Em decorrência de o desempenho de spin locks serem fundamentais para as aplicações, muitos pesquisadores exploram modelos Spin Lock para compreender seu comportamento de desempenho [1].

A operação testAndSet() foi a principal instrução de sincronização fornecida por muitas arquiteturas de multiprocessadores iniciais. Esta instrução opera em uma única palavra de memória (ou byte). Essa palavra tem um valor binário, verdadeiro ou falso. A instrução testAndSet () atomicamente armazena verdade na palavra, e retorna o valor anterior dessa palavra, trocando o valor verdadeiro ou valor atual da palavra. À primeira vista, esta instrução parece ideal para a implementação de um Spin Lock. O lock é livre quando o valor da palavra é falsa, e ocupado quando ela é verdadeira. O método lock () repetidamente aplica testAndSet () para o local até que a instrução retorna falso (ou seja, até que o lock seja liberado). O método de unlock() simplesmente escreve o valor false para ele [10].

Todas as implementações de Spin Locks sofrem de problemas de desempenho em níveis elevados de contenção. Em baixa contenção a linha de cache correspondente é, provavelmente, ainda local e gravável pela thread que adquiriu o lock. Em contraste, com altos níveis de disputa, cada thread tentar adquirir o lock terá uma cópia somente leitura da linha de cache, e a thread que adquire o lock terá que invalidar todas as cópias antes que possa realizar a atualização que libera o lock. Em geral, quanto mais CPUs e threads existem, maior a sobrecarga incorrida ao liberar o lock em condições de alta contenção [19].

### 2.4 Engenharia de Software Baseada em Evidências

O sucesso da medicina baseada em evidências levou muitas outras áreas que fornecem serviços para, ou para, os membros do público para tentar adotar uma abordagem semelhante. Nesse sentido a engenharia de software tentou aproveitar desses avanços científicos para o benefício da sociedade em sua área [13].

O objetivo da medicina baseada em evidências (MBE) é "a integração da melhor evidência de pesquisa com experiência clínica e valores do paciente" [23]. Por analogia, Kitchenham, Dyba e Jorgensen [13] sugerem que o objetivo da engenharia de software baseada em evidências (EBSE) deve ser: para fornecer os meios pelos quais melhor evidência atual da pesquisa podem ser integrados com a experiência prática e os valores humanos no processo de tomada de decisão em relação ao desenvolvimento e manutenção de software.

Kitchenham, Dyba e Jorgensen [13] relata ainda que a engenharia de software baseada em evidências proporciona:

- Um objetivo comum para os pesquisadores individuais e grupos de pesquisa para garantir que sua pesquisa é direcionada para as necessidades da indústria e outros grupos interessados.
- Um meio pelo qual os profissionais do setor podem tomar decisões racionais sobre a adoção da tecnologia.
- Um meio para melhorar a confiabilidade de sistemas intensivos de software, como resultado de uma melhor escolha de tecnologias de desenvolvimento.
- Uma maneira de aumentar a aceitação de sistemas de software intensivos que interagem com os cidadãos individuais.
- Uma entrada para processos de certificação.

Assim, a EBSE visa melhorar a tomada de decisões relacionadas com o desenvolvimento e manutenção de software, integrando melhor evidência atual da pesquisa com a experiência prática e os valores humanos [13]. Segundo Dyba, Kitchenham e Jorgensen [4] a EBSE envolve cinco passos:

- Converter um problema relevante ou precisar de informações para uma questão a ser respondida.
- Pesquisar a literatura para a melhor evidência disponível para responder a pergunta.
- Avaliar criticamente as evidências de sua validade, impacto e aplicabilidade.
- Integrar as provas avaliadas com a experiência prática e os valores do cliente e as circunstâncias para tomar decisões sobre a prática.
- Avaliar o desempenho e buscar formas de melhorá-lo.

Na EBSE existem as Revisões Sistemáticas da Litetura, do inglês Sistematic Literature Reviews (SLRs), e são um meio de agregar conhecimento sobre um tema de engenharia de software ou questão de pesquisa [6, 14]. A metodologia SLR pretende ser o mais imparcial possível por ser auditável e repetível. SLRs são referidos como estudos secundários e os estudos que analisam são referidos estudos primários. Segundo Kitchenham, Pretorius e Budgen [15], existem dois tipos diferentes de SLRs:

- SLRs Convencionais: resultados agregados relacionados a uma questão de pesquisa específica. Se há estudos primários suficientes comparáveis com estimativas quantitativas da diferença entre os métodos, meta-análise pode ser usada para realizar uma agregação com base estatística formal.
- Mapping studies: Estes estudos destinam-se a encontrar e classificar os estudos primários em uma área tópico específico. Eles podem ser usados para identificar a literatura disponível antes de realizar SLRs convencionais. Eles usam os mesmos métodos para a busca e extração de dados como SLRs convencionais, mas confiar mais na tabulação dos estudos primários em categorias específicas.

Diante dos objetivos da pesquisa, foi decidido utilizar a engenharia de software baseada em evidências como metodologia do trabalho, baseado nas características e definições citadas nesta seção. Para isso foi utilizada uma forma reduzida e mais específica da Revisão Sistemática, a Mini Revisão Sistemática, diante do curto prazo, recursos e fontes limitadas. Segundo Griffiths [28] este tipo de Revisão Sistemática possui as seguintes características:

- Escopo bem delimitado;
- Execução mais rápida;
- Questões de pesquisa mais profundas e refinadas;
- Critérios de inclusão dos estudos bem mais rigorosos

# 3 Metodologia

Seguir uma metodologia científica é indispensável a qualquer trabalho acadêmico, pois esta torna o trabalho replicável, imparcial e auditável. Assim, esta seção apresentará a abordagem metodológica usada nesta pesquisa.

## 3.1 Classificação da Pesquisa

Esta pesquisa utiliza o método de abordagem indutivo baseado em dados de natureza qualitativa [29]. O método de procedimento utilizado foi o de mini revisão, que é um tipo de revisão sistemática [14].

De acordo com Marconi & Lakatos [30] o método de abordagem indutivo é um processo mental por intermédio do qual, a partir de dados específicos, suficientemente constatados, infere-se uma verdade não contida nas partes examinadas. Para toda indução devem ser considerados três elementos essenciais: (i) observação dos fenômenos com a finalidade de descobrir as causas de sua manifestação, (ii) comparação com a finalidade de descoberta da relação entre eles e (iii) generalização da relação entre os fenômenos semelhantes.

O método de procedimento é a etapa mais concreta no processo de investigação. Para esta pesquisa, foi definida uma mini revisão sistemática da literatura, que é uma forma de avaliar e interpretar as pesquisas disponíveis mais importantes referentes a uma questão de pesquisa em particular ou fenômeno de interesse. A escolha de uma mini revisão, ao invés de uma revisão sistemática da literatura tradicional, para esta pesquisa se deu pela limitação de escopo e tempo para sua realização.

Por fim, no que diz respeito ao uso de pesquisa qualitativa, Marconi & Lakatos [30] destacam que a escolha por esse tipo de método de pesquisa se dá por que o mesmo preocupa-se em analisar e interpretar aspectos mais profundos, descrevendo a complexidade do comportamento humano, fornecendo dessa forma, análises mais detalhadas sobre as investigações, hábitos, atitudes, tendências de comportamento, etc. Além disso, os métodos qualitativos são capazes de prover informações mais exploratórias e ajudam a refinar as proposições para que melhor se ajustem aos dados.

## 3.2 Ciclo da Pesquisa

A busca de meios para realizar revisões da literatura abrangentes e imparciais deu origem ao conceito de Revisões Sistemáticas da Literatura (RSL). Segundo Travassos [31], revisões sistemáticas fornecem meios para atingir estes objetivos, pois elas (i)

resumem as evidências sobre o assunto estudado, (ii) podem identificar lacunas no estado da arte, (iii) fornecem uma base para direcionar novas pesquisas e (iv) apoiam a geração de novas ideias e hipóteses sobre o fenômeno em questão.

Para isso, as revisões sistemáticas utilizam uma metodologia confiável, rigorosa e auditável [14]. O processo de revisão sistemática começa com a definição de um protocolo que especifica as questões de pesquisa e as estratégias que serão utilizadas para conduzir a revisão. Segundo Kitchenham [32], este processo é dividido em três fases principais: Planejando a Revisão, Conduzindo a Revisão e Reportando a Revisão. As etapas incluídas em cada uma destas fases são detalhadas na Tabela 1.

**Tabela 1.** Fases do processo de Revisão Sistemática

| Fase | Etapa |
|---|---|
| Planejando a Revisão | Identificação da necessidade de uma revisão |
| | Desenvolvimento do protocolo |
| Conduzindo a Revisão | Identificação da pesquisa |
| | Seleção dos estudo primários |
| | Avaliação da qualidade dos estudos |
| | Extração dos dados e monitoramento |
| | Síntese dos dados |
| Reportando a Revisão | Preparação do relatório |

No entanto, o escopo de uma revisão sistemática é muito amplo e responder a questões muito focadas e específicas torna-se uma tarefa muito difícil. Sob este aspecto, Griffiths [28] afirma que uma Mini Revisão Sistemática (MRS) é uma alternativa. Uma Mini Revisão Sistemática da Literatura é um método que fornece uma visão bem mais específica e aprofundada de um tema de pesquisa, além de uma execução bem mais rápida, apoiada pela estrutura das revisões sistemáticas tradicionais, com a definição adicional de limites arbitrários ao escopo. Nesse contexto, a proposta deste trabalho é realizar uma Mini Revisão Sistemática da Literatura para atingir o objetivo principal de agregar os últimos conhecimentos disponíveis na literatura acerca do tema estudado.

Inicialmente, foi realizada uma revisão *ad hoc* da bibliografia com os principais temas que envolvem a pesquisa. A partir disso, foi observada a importância do tema e evidenciada a dispersão de estudos recentes em relação à aplicação de s*pin locks* em diferentes e diversos cenários, tornando assim explícita a necessidade de ser realizado um estudo sistemático da literatura para reunir os dados e extrair informações importantes acerca destes. A partir deste ponto, o foco da pesquisa foi definido e expresso em uma questão de pesquisa que será apresentada posteriormente. As etapas da pesquisa podem ser visualizadas na Figura 1.

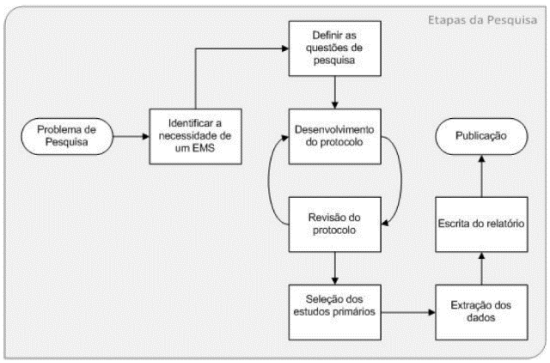


**Figura 1.** Etapas da Pesquisa

Foi confeccionado um protocolo para execução da revisão para coletar evidências com o intuito de responder as perguntas de pesquisa propostas. Serão descritas à frente, as etapas executadas no processo, que podem ser visualizadas na Tabela 1. Por fim, será construída uma discussão sobre dos dados extraídos, visando promover contribuição à comunidade científica através da agregação do conhecimento obtido acerca do tema abordado.

*3.2.1 Escopo e Questões de Pesquisa*

Com o intuito de delinear a abrangência da pesquisa e de explicitar os elementos que fizeram parte das questões de pesquisa, foi utilizado uma estrutura citada por Kitchenham [5], que recomenda considerar as questões de pesquisa a partir da seguinte estrutura denominada PICOC:

- **População** (*Population*): Pesquisas acerca de experimentos ou medições quem envolvam explicitamente *spin locks* realizadas nos últimos dois anos;
- **Intervenção** (*Intervention*): aplicação/uso de *spin locks*;
- **Resultado** (*Outcomes*): Um texto que resume as últimas descobertas acerca da performance e cenários de aplicação de *spin locks*;
- **Contexto** (*Context*): indústria e academia.

O terceiro item, Comparação (*Comparison*), não foi utilizado, uma vez que o estudo não realiza comparações entre os estudos ou as aplicações de *spin locks*. Pode-se notar que foi aplicado um limite arbitário à População, a fim de tangenciar o trabalho ao prazo disponível para execução do trabalho. Após a definição da estrutura PICOC, a pergunta que guiou o estudo foi definida:

- **(Q1)** Em que cenários o uso de *spin locks* é indicado?

*3.2.2 Estratégia de Busca*

Para se realizar a pesquisa dos estudos primários em uma revisão sistemática da literatura, é necessário seguir uma estratégia específica [14]. Diante disto, nesta seção são

especificados os passos necessários à construção desta estratégia. São eles: (i) definição dos termos principais da pesquisa, (ii) construção da *string* que será usada para a pesquisa e (iii) definição das fontes onde serão realizadas as pesquisas.

Depois da formulação das questões, foram definidas as principais palavras-chave da pesquisa. Estes termos foram traduzidos para o inglês por ser a língua utilizada nas fontes de pesquisa eletrônicas, além dos jornais e conferências dos temas de investigação. Também foram listados os sinônimos para cada um dos termos. Para evitar possíveis gargalos, alguns termos serão pesquisados no singular e plural, sendo usado um asterisco (*) para indicar a ocorrência da variação.

Segundo Kitcheham [14], as *strings* de busca são construídas a partir das estruturas das questões e, devido às necessidades de cada base de dados dentre as fontes de busca, é necessário fazer adaptações. Desse modo, a *string* foi gerada a partir da junção de alguns termos-chave, e qualquer necessidade de alteração da *string* por causa das bases de dados será reportada durante a execução da pesquisa. A *string* é apresentada a seguir:

```
(spin lock*) AND (application* OR use* OR experiment* OR
        experimentation* OR quasi-experiment*)
```

*3.2.3 Processo de Busca*

O processo que foi utilizado para pesquisar estudos primários incluiu buscas automáticas em engenhos de busca de uma biblioteca digital, onde foi usada a *string* de busca construída, e também com a execução de buscas manuais em anais de eventos e periódicos relevantes, com o objetivo de ampliar a abrangência do estudo. No caso da busca manual, o que determinou o período de tempo que limitou a busca foi a disponibilidade do material na Internet. A biblioteca digital utilizada na busca automatizada foi a ACM Digital Library[1]. Já a conferência utilizada na busca manual foi o ACM Symposium on Operating Systems Principles (SOSP).

*3.2.4 Critérios de Inclusão e Exclusão dos Estudos*

Para seleção dos estudos, foi adotado um processo com duas etapas. Na primeira, um pesquisador analisou os dados obtidos através das bibliotecas digitais, lendo o título, resumo e palavras-chave, selecionando os estudos publicados nos últimos 2 (dois) anos. A partir daí foi elencado um conjunto de potenciais estudos primários. Na segunda etapa, esse conjunto foi analisado por outro pesquisador para dar origem ao conjunto final de estudos primários.

Os critérios usados para inclusão de um dos trabalhos em potencial nesse conjunto final foram os seguintes:

- Estudos que se relacionem ao travamento e/ou sincronização de aplicações paralelas com *spin locks*;
- Estudos que realizam algum tipo de experimento ou que demonstrem alguma aplicação do objeto de estudo.

[1] Endereço na Internet: http://dl.acm.org

Os critérios usados para a exclusão dos trabalhos foram os seguintes:

- Estudos que não realizam nenhum experimento ou aplicação;
- Estudos repetidos;
- Estudos duplicados;
- Estudos com mais de 2 anos de realização.

*3.2.5 Estratégia de Extração e Síntese dos Dados*
A extração e síntese dos dados se utilizam de formulários de coleta em planilhas eletrônicas, identificando algum tipo de informação relevante à pergunta de pesquisa. Cada um dos trechos retirados dos estudos primários é associado a um número identificador, que categoriza o tipo de informação que cada um desses trechos provêem. Esses identificadores seguem um padrão, a fim de se conseguir uma melhor organização e rastreabilidade dos dados coletados.

# 4 Resultados

A revisão foi conduzida de acordo com os processos definidos no capítulo anterior, levando em consideração todos os limites arbitrários definidos para sua realização.

## 4.1 Extração dos Dados

A *string* de busca construída, quando executada na biblioteca digital definida, retornou um quantitativo de 1671 trabalhos. Além disso, a busca manual, empreendida com o intuito de aumentar o alcance do estudo, resultou num quantitativo de 5 trabalhos selecionados. O resultado final das buscas juntas resultou num espaço de estudos primários cujo montante é de 1676 trabalhos, onde aproximadamente 99% foram coletados na busca automatizada e o restante da busca manual.

Após a aplicação dos critérios de inclusão, o número alto de estudos foi drasticamente reduzido para 42 (uma redução de aproximadamente 97,5%). Depois disso, os estudos passaram pela aplicação dos critérios de exclusão, o que reduziu o número para 18. Com os potenciais estudos selecionados, estes foram lidos completamente pela dupla de revisores com o objetivo de selecionar os estudos primários que fazem parte deste trabalho. Chegou-se ao número de 5 estudos primários, dos quais apenas 1 faz parte dos resultados da busca manual. O restante foi selecionado a partir da busca automatizada. Os estudos selecionados foram os seguintes:

- **EP_1:** Alexander Wieder and Björn B. Brandenburg. 2013. On Spin Locks in AUTOSAR: Blocking Analysis of FIFO, Unordered, and Priority-Ordered Spin Locks. In Proceedings of the 2013 IEEE 34th Real-Time Systems Symposium (RTSS '13). IEEE Computer Society, Washington, DC, USA, 45-56.
- **EP_2:** David Dice, Virendra J. Marathe, and Nir Shavit. 2012. Lock cohorting: a general technique for designing NUMA locks. In Proceedings of the 17th ACM

SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP '12). ACM, New York, NY, USA, 247-256.

- **EP_3:** Tudor David, Rachid Guerraoui, and Vasileios Trigonakis. 2013. Everything you always wanted to know about synchronization but were afraid to ask. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13). ACM, New York, NY, USA, 33-48.
- **EP_4:** Sándor Juhász, Ákos Dudás, and Tamás Schrádi. 2012. Cost of mutual exclusion with spin locks on multi-core CPUs. In Proceedings of the 5th WSEAS congress on Applied Computing conference, and Proceedings of the 1st international conference on Biologically Inspired Computation (BICA'12), Nikos Mastorakis, Valeri Mladenov, and Zoran Bojkovic (Eds.). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 15-19.
- **EP_5:** Bryan C. Ward and James H. Anderson. 2013. Fine-grained multiprocessor real-time locking with improved blocking. In Proceedings of the 21st International conference on Real-Time Networks and Systems (RTNS '13). ACM, New York, NY, USA, 67-76.

### 4.2 Endereçamento das Evidências

O objetivo da questão de pesquisa deste trabalho é evidenciar quais os cenários é aconselhável o uso do spin locks em detrimento de outro tipo de trava ou método de sincronização. Foram encontradas evidências em todos os estudos primários que explicitam ou ajudam a decidir em que cenários é aconselhável usar spin locks, inclusive baseando-se nas características de comportamento dos objetos dos estudos em determinadas ocasiões.

A seguir são apresentados os trechos coletados como evidências:

**EP_1:** *"The choice of spin lock algorithm depends on many factors in practice, among them hardware-dependent considerations such as memory availability and support for atomic operations. Besides such engineering concerns, there are also analytical concerns that may force the use of a specific lock type." ... "The choice of spin lock algorithm depends on many factors in practice, among them hardware-dependent considerations such as memory availability and support for atomic operations. Besides such engineering concerns, there are also analytical concerns that may force the use of a specific lock type." ... "However, despite their many benefits, FIFO-ordered locks are fundamentally unsuitable for some workloads." ... "In general, if some tasks have less than (m − 1) · Lmax slack, where Lmax denotes the maximum (task-independent) critical section length, then FIFO-ordered locks are inappropriate and locking priorities are fundamentally required."*

**EP_2:** *"For read-heavy loads, the performance of all the locks except HBO and C-BO-BO is identical, with all locks enabling over 5X scaling. For loads with moderately high set ratios, we observe that all the spin locks except HBO and CBO-BO significantly outperform pthread locks, and are generally competitive with each other. For write-heavy loads, the NUMA-aware locks clearly outscale the NUMA-oblivious locks by at*

*least 20%. The untuned HBO and C-BO-BO locks scale poorly in all configurations. It appears that C-BO-BO suffers because of contention on the local BO locks, whereas HBO suffers with contention on the central lock. FC-MCS performs better than HBO and C-BO-BO, but worse than all other spinlocks."*

**EP_3:** *"Spin locks are generally considered to scale poorly because they involve high contention on a single cache line, an issue which is addressed by queue locks." ... "More precisely, while spin locks closely follow the cache-coherence latencies, more complex locks generally introduce some additional overhead." ... "Low Contention: Finally, the best performance in this scenario is achieved by simple spin locks." ... "We confirm that plain spin locks do not scale across sockets and present some optimizations that alleviate the issue." ... "Nevertheless, if we reduce the context of synchronization to a single socket (either one socket of a multi-socket, or a single-socket many-core), then our results indicate that spin locks should be preferred over more complex locks. Complex locks have a lower uncontested performance, a larger memory footprint, and only outperform spin locks under relatively high contention."*

**EP_4:** *"There are many alternatives for implementing simple spin locks. The correct type of the lock to use is not trivial. Spinning on some sort of a flag wastes CPU cycles and can cause cache contention for accessing the same cache line often, while more sophisticated locks can increase the mere overhead of acquiring a lock for which there is no contention." ... "The spin locks all feature basically the same performance characteristics: they scale really well with the number of threads and no significant difference can be observed in performance." ... "The difference between the locks lies in the cost of testing the lock. If the contention for a lock is high, spinning on the lock flag is carried out often. This spinning has considerably different costs for the different types of locks."*

**EP_5:** *"For both spin- and suspension-based protocols, progress mechanisms such as non-preemptive spinning or priority donation can cause priority inversions for non-resource-using tasks." ... "Depending upon the system (clustered, partitioned, or globally scheduled), as well as the type of analysis being conducted (spin-based, s-oblivious, or s-aware), different tokens locks, number of tokens T and RSMs can be combined to form an effcient locking protocol."*

### 4.3 Discussão sobre os Resultados

Apesar do pequeno número de trabalhos selecionados como estudos primários e dos limites arbitrados à execução da revisão, esta se mostrou satisfatória na tarefa de responder à questão de pesquisa.

A questão de pesquisa visa descobrir em que cenários o uso de *spin locks* é aconselhado, em detrimento de outros tipos de travas. Respondendo a esta pergunta, a revisão mostra nas evidências que a escolha por *spin locks* depende de vários fatores, como a

disponibilidade de memória para execução do programa, do suporte de software e hardware para execução de instruções atômicas, e número de threads que estarão em funcionamento durante a execução do programa.

Outro fator incluso no questionamento deste estudo que também é respondido, é que dependendo do tipo de *spin lock* (MCS, TTAS, CLH, …), existem performances diferentes. Por exemplo, travas TTAS sofrem com a alta contenção. Por sua vez, travas MCS sofrem menos que os outros tipos de *spin locks*, que no entanto são menos escaláveis.

Outro fator considerado é que outros tipos de travas adicionam considerável *overhead* de travamento, ao contrário de *spin locks*. Por exemplo, quando existe um espaço de memória disponível maior, é preferírel usar travas complexas. Mas, quando o espaço de memória é menor, e existe um nível relativamente alto de contenção, *spin locks* são mais aconselháveis.

Um comportamento de *spin locks* observado nos estudos é que a diferença entre estes reside no custo de teste durante o *spinning*. Se a disputa por um bloqueio for alta, então a trava irá ficar girando mais frequentemente. Esta rotação tem custos consideravelmente diferentes para os diferentes tipos de *spin locks*.

Outra afirmação que pode ser inferida após analisar os resultados apresentados é que há poucos estudos recentes que discutam novas formas de sicronização entre threads e as comparem com o objeto deste estudo.

# 5 Considerações Finais

## 5.1 Limitações e Trabalhos Futuros

Trabalhos futuros devem se concentrar numa análise que contemple um maior número de estudos, repositórios de dados, conferências e ampliando os anos das publicações, a fim de fornecer uma visão mais abrangente sobre o tema pesquisado.

A limitação de tempo e recursos foi um desafio para aplicação de uma metodologia utilizando a Engenharia de Software Baseada em Evidências. Entretanto, com um escopo reduzido e a utilização do conceito de Mini Revisão Sistemática foi possível alcançar os objetivos propostos.

## 5.2 Conclusão

Este trabalho teve como objetivo descobrir em que cenários o uso de *spin locks* é aconselhado, em detrimento de outros tipos de travas, analisando aspectos em termos hardware e software. Para isso foi feito o uso de uma Mini Revisão Sistemática como forma metodológica.

Foi possível constatar que a escolha por *spin locks* depende de vários fatores, como a disponibilidade de memória para execução do programa, do suporte de software e hardware para execução de instruções atômicas, e número de threads que estarão em funcionamento durante a execução do software. Para chegar a essa conclusão de 1676 estudos primários, apenas 5 resultaram e estes foram os analisados.

Os dados também apontam que se reduzirmos o contexto de sincronização para um único socket os resultados de EP_3 indicam que os Spin Locks deve ser preferido sobre travas mais complexas. O estudo realizado também permitiu identificar que em um cenário de baixa contenção o melhor desempenho é atingido por Spin Locks simples.

## Referências


1. Cai-Dong W.; Takada, H.; Sakamura, K., "Priority inheritance spin locks for multiprocessor real-time systems," Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second International Symposium on , vol., no., pp.70,76, 12-14 Jun 1996
2. Diaz, J.; Munoz-Caro, C.; Nino, A, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era," Parallel and Distributed Systems, IEEE Transactions on , vol.23, no.8, pp.1369,1386, Aug. 2012
3. Dijkstra, E. W. Solution of a problem in concurrent programming control. Communications of the ACM, 8(9):21–65, September 1965.
4. Dyba, T., Kitchenham, B. A., and Jorgensen, M. 2005. Evidence-Based Software Engineering for Practitioners. IEEE Softw. 22, 1 (January 2005), 58-65.
5. Ebnenasir, A; Beik, R., "Developing parallel programs: A design-oriented perspective," Multicore Software Engineering, 2009. IWMSE '09. ICSE Workshop on , vol., no., pp.1,8, 18-18 May 2009
6. Fink, A. Conducting Research Literature Reviews. From the Internet to Paper, Sage Publication, Inc., 2005.
7. Freitas, H.C., "Introducing parallel programming to traditional undergraduate courses," Frontiers in Education Conference (FIE), 2012 , vol., no., pp.1,6, 3-6 Oct. 2012
8. Fu, M.; Zhang, Y.; Li, Y. "Formal Reasoning about Concurrent Assembly Code with Reentrant Locks," Theoretical Aspects of Software Engineering, 2009. TASE 2009. Third IEEE International Symposium on , vol., no., pp.233,240, 29-31 July 2009
9. Ghanemi, S., "High-Level Abstract Parallel Programming Platform: Application to GIS Image Decomposition," Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on , vol., no., pp.1,5, 7-11 April 2008
10. Herlihy, M., Shavit, N. The Art of Multiprocessor Programming. 1a ed. (revised reprint). Morgan-Kaufmann. 2012.
11. Heroux, M.; Zhaofang Wen; Junfeng Wu; Yuesheng Xu, "Initial Experiences with the BEC Parallel Programming Environment," Parallel and Distributed Computing, 2008. ISPDC '08. International Symposium on , vol., no., pp.205,212, 1-5 July 2008
12. Hwu, W., Keutzer, K., and Mattson, T.G. “The concurrency challenge,” IEEE Design and Test of Computers, vol. 25, no. 4, pp. 312-320, July 2008.
13. Kitchenham, B. A., Dyba, T. and Jorgensen, M. 2004. Evidence-Based Software Engineering. In Proceedings of the 26th International Conference on Software Engineering (ICSE '04). IEEE Computer Society, Washington, DC, USA, 273-281.
14. Kitchenham, B.A., Charters, S. Guidelines for Performing Systematic Literature Reviews in Software Engineering Technical Report EBSE-2007-01, 2007.
15. Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O. P., Turner, M., Niazi, M., and Linkman, S.. 2010. Systematic literature reviews in software engineering - A tertiary study. Inf. Softw. Technol. 52, 8 (August 2010), 792-805.

16. Limet, S., "High level languages for efficient parallel programming," High Performance Computing and Simulation (HPCS), 2012 International Conference on , vol., no., pp.541,542, 2-6 July 2012
17. Marowka, A, "Towards High-Level Parallel Programming Models for Multicore Systems," Advanced Software Engineering and Its Applications, 2008. ASEA 2008 , vol., no., pp.226,229, 13-15 Dec. 2008
18. Mattson, T.; Wrinn, M., "Parallel programming: Can we PLEASE get it right this time?," Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE , vol., no., pp.7,11, 8-13 June 2008
19. McKenney, P. Is Parallel Programming Hard, And, If So, What Can You Do About It? 1a ed. Disponível em http://kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook-e1.pdf. 2014.
20. Meyer, J.C.; Elster, AC., "Latency Impact on Spin-Lock Algorithms for Modern Shared Memory Multiprocessors," Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on , vol., no., pp.786,791, 4-7 March 2008
21. McCool, M. D. 2010. Structured parallel programming with deterministic patterns. In Proceedings of the 2nd USENIX conference on Hot topics in parallelism (HotPar'10). USENIX Association, Berkeley, CA, USA, 5-5.
22. Reiss, S.P.; Tarvo, A, "Automatic categorization and visualization of lock behavior," Software Visualization (VISSOFT), 2013 First IEEE Working Conference on , vol., no., pp.1,10, 27-28 Sept. 2013
23. Sackett, D.L., Straus, S.E., Richardson, W.S., Rosenberg, W., And Haynes, R.B. Evidence-Based Medicine: How to Practice and Teach EBM, Second Edition, Churchill Livingstone: Edinburgh, 2000.
24. Schafer, M.; Sridharan, M.; Dolby, J.; Tip, F., "Refactoring Java programs for flexible locking," Software Engineering (ICSE), 2011 33rd International Conference on , vol., no., pp.71,80, 21-28 May 2011
25. Tran, T. M. T.; Owe, O.; Steffen, M., "Safe Typing for Transactional vs. Lock-Based Concurrency in Multi-threaded Java," Knowledge and Systems Engineering (KSE), 2010 Second International Conference on , vol., no., pp.188,193, 7-9 Oct. 2010
26. Usui T., Behrends R., Evans J., and Smaragdakis Y. 2009. Adaptive Locks: Combining Transactions and Locks for Efficient Concurrency. In Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques (PACT '09). IEEE Computer Society, Washington, DC, USA, 3-14.
27. Vallejo, E.; Beivide, R.; Cristal, A; Harris, T.; Vallejo, F.; Unsal, O.; Valero, M., "Architectural Support for Fair Reader-Writer Locking," Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on , vol., no., pp.275,286, 4-8 Dec. 2010
28. Griffiths, P. Evidence informing practice: introducing the mini-review. In: British Journal of Community Nursing, Vol. 7, No. 1, 01.2002, p. 38 - 39.
29. MORESI, E. (2003). Metodologia da Pesquisa. 1ª Edição. Brasília, Universidade Católica de Brasília.
30. MARCONI, M., LAKATOS, E. (1992). Metodologia do Trabalho Científico. 4ª Edição. São Paulo, Atlas.
31. TRAVASSOS, G., BIOLCHINI, J. (2007). "Revisões Sistemáticas aplicadas à Engenharia de Software". In Proceedings of the 21th Brazilian Symposium on Software Engineering (SBES '07). João Pessoa, Brasil.
32. KITCHENHAM, B. (2004). Procedures for Performing Systematic Reviews. In Joint Technical Report TR/SE-0401, Keele University. Keele, Reino Unido.