Redução do tempo de processamento do Cálculo Atuarial das Forças Armadas: uma aplicação da Computação Paralela

Carlos Francisco Simões Gomes
cfsg1@bol.com.br
UFF

Marcos dos Santos
marcosdossantos_doutorado_uff@yahoo.com.br
CASNAV - UFF

Ernesto Rademaker Martins
radmart@yahoo.com.br
CASNAV

Thierry Faria da Silva Gregório
thierrygregorio@ime.uerj.br
CASNAV - UERJ

Ronaldo Cesar Evangelista dos Santos
ronaldo@casnav.mar.mil.br
CASNAV

Resumo: As Forças Armadas (FFAA) possuem uma considerável massa de dados financeiros e biométricos dos seus pensionistas que precisa ser submetida a um modelo matemático atuarial próprio, para a obtenção do resultado da projeção atuarial de 75 anos, sendo esta uma obrigatoriedade legal que deve ser atendida anualmente. O objetivo do estudo é reduzir os tempos computacionais necessários à obtenção dos resultados da referida projeção, por meio da aplicação de técnicas de Computação Paralela e de Engenharia de Software. É sabido que o Cálculo Atuarial é um método matemático que utiliza conceitos financeiros, econômicos e probabilísticos para dimensionar o montante de recursos e de contribuições necessárias ao pagamento de benefícios futuros dos segurados. A pesquisa ora apresentada alcançou resultados significativos, na medida em que reduziu em mais de 90% os tempos de processamento dos cálculos referentes às projeções atuariais dos pensionistas das FFAA. No caso específico do Exército Brasileiro, a partir de uma massa de dados de 500.000 registros, foi possível executar um complexo modelo matemático de alta recursividade em cerca de 5 minutos. Atualmente, o mundo passa por inúmeras transformações, gerando novas demandas pela sociedade. Os problemas crescem em quantidade e em complexidade, num contexto cada vez mais dinâmico e volátil, exigindo rápidas decisões dos gestores, principalmente em nível estratégico. Daí a importância da celeridade do cálculo atuarial em estudo, pois, assim, pode-se tempestivamente traçar inúmeros cenários de maneira a apoiar a decisão da Alta Administração das FFAA.

Palavras Chave: Computação Paralela - Redução do Tempo - Cálculo Atuarial - Forças Armadas - Projeção Atuarial

1. INTRODUÇÃO

O Cálculo Atuarial é um método matemático que utiliza conceitos financeiros, econômicos e probabilísticos para dimensionar o montante de recursos e de contribuições necessárias ao pagamento de benefícios futuros dos segurados dos Fundos e Institutos de Previdência Social, também chamados de Regimes Próprios de Previdência Social, que busca o equilíbrio entre os resultados financeiros e a projeção atuarial. A manutenção deste equilíbrio é de suma importância, principalmente em tempos de crise econômica.

Não se trata de uma disciplina nova, que tenha surgido nos últimos séculos, mas de um desdobramento de procedimentos realizados desde a Antiguidade, os quais naturalmente assumiram nova feição com os trabalhos de Fermat e de Pascal na França, De Witt na Holanda, Grauns e Halley na Inglaterra, que avançaram nos estudos de probabilidade e da demografia relacionada à longevidade humana (DIAS e SANTOS, 2010).

As projeções atuariais de receitas e despesas para uma entidade de previdência e pensões têm por objetivo quantificar os custos futuros estimados com pagamento de benefícios e as receitas futuras estimadas de contribuições de participantes. Estas projeções são importantes para que as entidades e governos provisionem tais cifras monetárias para os anos subsequentes, garantindo o equilíbrio atuarial e diminuindo o risco de falta de liquidez. Segundo Conde (2005) quando as contribuições são definidas adequadamente, o plano tende a ter um equilíbrio financeiro-atuarial.

As entidades previdenciárias e de pensões projetam os custos e receitas futuras a um horizonte temporal de 75 anos, entretanto, nada impossibilita que a projeção seja calculada a 80 ou 100 anos. As principais premissas que impactam neste cálculo atuarial são: tábua de mortalidade de válidos, tábua de mortalidade de inválidos, tábua de entrada em invalidez, taxa de rotatividade, crescimento salarial, taxa de inflação e reposição de ativos.

No Brasil, a obrigatoriedade da realização anual da projeção atuarial de 75 anos obriga as empresas estatais e/ou seus fundos de pensão a investir no desenvolvimento das tecnologias necessárias à obtenção dos resultados ou a contratar serviços de terceiros. Em ambas as situações, contudo, o uso de técnicas tradicionais de programação, ou seja, sem o uso de computação paralela, ainda que utilizadas as melhores práticas de engenharia de software, leva a produção de resultados que, embora de boa acurácia, ainda podem ser melhorados em termos de desempenho computacional.

2. METODOLOGIA

Ainda que não possuam sistema previdenciário, por não haver previsão legal na legislação brasileira, os Ministério da Defesa (MD) vêm realizando há alguns anos a avaliação atuarial dos compromissos da União com os benefícios pagos pelo Sistema de Pensões dos Militares das Forças Armadas. Esta avaliação é realizada anualmente visando apresenta-la aos órgãos fiscalizadores e ao Ministério do Planejamento, Orçamento e Gestão (MPOG) os resultados da mesma.

A equação (1) é usada para calcular o valor presente de benefício futuro de reversão em pensão de um militar reformado por invalidez, é um exemplo da complexidade do modelo matemático formulado.

$$13.\sum_{t=0}^{k-1} \left({}_tp_x^{aa}\cdot{}_{(CSA)}v^t\cdot i_{x+t}\cdot\sum_{j=0}^{z-(x+t)}\left({}_jp_{x+t}^{i}\cdot v^j\cdot q_{x+t+j}^{i}\cdot{}_{(CBA)}valor_{x+j}^{s}\cdot\left({}_{(CBA)}HPE_{x+t+j}^{i(12)}-{}_{(CBA)}H_{x+t+j}^{i(12)}\right)\right)\right) \quad (1)$$

Este modelo utiliza o conceito de cálculo individual, ou seja, os cálculos são realizados para cada militar ativo, inativo ou pensionista existente no banco de dados. Este conceito, aliado à complexidade dos cálculos, resultou na necessidade do desenvolvimento de

um *software* atuarial específico que, se desenvolvido com técnicas padrão de Engenharia de Software, demandaria a necessidade de um considerável esforço computacional para a produção dos resultados. Neste contexto, o uso de técnicas e recursos de computação paralela mostrou-se como solução para reduzir o esforço computacional, permitindo, por exemplo, a utilização de toda a capacidade de processamento extra, disponível na máquina.

Para dar suporte ao *software* do cálculo atuarial, buscou-se o conhecimento em áreas ligadas a computação massivamente paralela, tendo como foco a criação de um ambiente computacional que propiciaria a solução desejada ao problema apresentado. Assim, o software utilizado na implementação do cálculo atuarial foi desenvolvido em C#. O C# é uma linguagem de programação criada para o desenvolvimento de uma variedade de aplicações que executam sobre o .Net Framework. A linguagem C# é simples, fortemente tipada e orientada a objeto. As várias inovações no C# permitem o desenvolvimento rápido de aplicativos mantendo a expressividade das linguagens de estilo C.

Assim, a adoção de modelo que utilize recursos de computação paralela, no desenvolvimento de um software para a solução de problemas de cálculo atuarial, pode, extrapolando-se os resultados alcançados no âmbito das Forças Armadas (FFAA), além de reduzir consideravelmente os tempos necessários à obtenção dos resultados, reduzir os gastos das empresas estatais com a contratação dos serviços de terceiros ou com a alocação de material humano e recursos computacionais, por exemplo.

2.1 PROGRAMAÇÃO PARALELA COM C#

Os processadores com múltiplos núcleos (cores) estão no mercado há muitos anos e, atualmente, estão disponíveis na maioria dos dispositivos. Entretanto, muitos desenvolvedores continuam a fazer o que sempre fizeram: criar programas que usam um único fluxo de controle sequencial isolado dentro de um programa denominado *thread*. Isto faz com que não seja usado todo poder de processamento disponível na máquina. A maioria dos computadores comercializados atualmente conta com um processador de dois núcleos e quatro *threads*. Ao adquirirem um computador com tais características, os usuários pagam pela capacidade de processamento extra. Contudo, ao disponibilizar um programa que usa um único *thread*, o desenvolvedor, por meio do software, não permite a utilização desta capacidade extra de processamento.

Kirk e Hwu (2010) apontam que é uma tarefa simples conseguir um speedup de 10x quando um aplicativo lança mão do paralelismo de dados. Uma computação diz-se paralela quando um programa é executado sobre uma máquina multiprocessador em que todos os processadores partilham o acesso à memória disponível, ou seja, o mesmo endereço em processadores diferentes corresponde à mesma posição de memória.

Processamento em múltiplos *threads* não é novidade para desenvolvedores C# experientes, mas nem sempre é fácil desenvolver programas que usem toda a capacidade de processamento disponível. Além disso, a evolução das linguagens de programação tem facilitado o trabalho dos desenvolvedores ao simplificar a implementação da programação paralela.

Para que se possa avançar no uso da programação paralela, é importante fixar dois importantes conceitos: execução síncrona e execução assíncrona. Um bom discernimento destes dois modos de execução é um conhecimento basilar para aprimorar o desempenho de suas aplicações. Quando se executa uma operação em modo síncrono, o programa executa todas as tarefas em sequência, como mostrado na Figura 1. Ao disparar a execução, cada tarefa só será executada após o término da tarefa anterior.

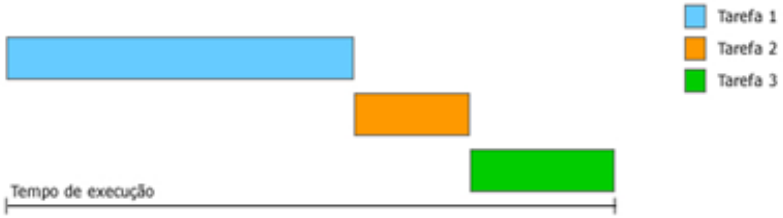

Figura 1: Execução síncrona
Fonte: Autores (2017)

Quando se executa uma operação em modo assíncrono, o programa não mais executa as tarefas em sequência: ele dispara as tarefas, quando necessário, e as mesmas são iniciadas e encerradas concomitantemente à execução de outras tarefas, como mostra a Figura 2.

Figura 2: Execução assíncrona
Fonte: Autores (2017)

Como induz a análise das imagens, as mesmas tarefas sendo executadas de modo assíncrono, demandarão um menor tempo de execução do que quando executadas de modo síncrono, pelo simples fato de que não há a necessidade de que uma tarefa "espere" outra acabar para ser iniciada. A espera, neste caso, só seria aceitável se houvesse relação de dependência entre as tarefas, ou seja, se uma tarefa dependesse, por exemplo, de um cálculo que outra tarefa, ainda em execução, está produzindo.

Do parágrafo anterior é possível depreender que o uso, ou não, da programação paralela deve ser avaliado pelo desenvolvedor do *software*. Em algumas situações, o seu uso pode ser bastante benéfico ao diminuir os tempos de execução e produção dos resultados. Em outras situações, contudo, o seu uso desbalanceado pode, até mesmo, prejudicar o desempenho do software. Como a implementação da programação paralela com o C# exige a inserção explicita no código de instruções específicas de paralelismos, cabe ao desenvolvedor decidir quando e onde usar tais instruções.

Pode surgir o questionamento do porque muitos desenvolvedores ainda optam pela execução síncrona, se a execução assíncrona leva menos tempo. A resposta a tal pergunta não é simples. O que pode ser dito de imediato é que com a programação assíncrona, o desenvolvedor tem alguns novos desafios:

- Sincronizar tarefas. Supondo-se que na Figura 2 seja necessário iniciar uma tarefa somente após o término das outras duas. Seria preciso, neste caso, criar um mecanismo de espera para aguardar todas as tarefas terminarem, antes de executar a nova tarefa;
- Resolver problemas de concorrência. Se existe um recurso compartilhado, como uma lista que é escrita em uma tarefa e lida em outra tarefa, seria preciso criar um mecanismo para garantir que a lista seja mantida em um estado conhecido;

- Adaptar-se a uma nova lógica de programação, já que não há uma sequência lógica. As tarefas podem terminar a qualquer momento e não se ter mais controle de qual termina primeiro.

A programação assíncrona exige uma mudança de paradigma por parte do desenvolvedor. A sua adoção, entretanto, traz algumas vantagens. Uma das mais significativas é o não travamento da interface do usuário (UI), em função das tarefas poderem ser executadas em segundo plano. Outra vantagem, é a capacidade de poder usar todos os núcleos da máquina, fazendo melhor uso dos seus recursos.

Em relação às desvantagens da programação síncrona, ressalta-se a incapacidade da mesma de usar a arquitetura de múltiplos núcleos dos novos processadores. Desta forma, não importa se o programa será executado em um processador com 1 (um) núcleo ou com 8 (oito) núcleos, pois ele será executado com a mesma velocidade em ambos.

2.2. PROGRAMAÇÃO ASSÍNCRONA COM ASYNC E AWAIT

É possível evitar gargalos de desempenho e melhorar a resposta geral de um *software* usando a programação assíncrona. No entanto, técnicas tradicionais para escrever aplicativos assíncronos podem ser complicadas, tornando difícil o desenvolvimento, a depuração e a manutenção.

A assincronia é essencial para atividades expostas a um potencial bloqueio, como por exemplo, quando o aplicativo acessa a web. O acesso a um recurso da web às vezes é lento ou sujeito a atrasos. Se tal atividade for bloqueada dentro de um processo síncrono, todo o aplicativo ficará em espera. Em um processo assíncrono, o aplicativo pode continuar com a execução de outra tarefa, que não dependa do recurso da web, até que a tarefa exposta a um potencial bloqueio termine.

A assincronia é especialmente útil para aplicativos que acessam o *thread* da UI, porque todas as atividades relacionadas à interface do usuário normalmente compartilham um único thread. Se um processo for bloqueado em um aplicativo síncrono, todos serão bloqueados. Seu aplicativo para de responder, levando à conclusão equivocada de que o aplicativo falhou, quando em vez disso, o aplicativo está apenas aguardando.

Async e **await**, palavras-chave em c#, são o *core* da programação assíncrona. Usando essas duas palavras, pode-se usar os recursos do .NET Framework ou o tempo de execução do Windows para criar um método assíncrono quase tão facilmente quanto criar um método síncrono.

A Figura 3, mostra um método assíncrono. Quando se usa a palavra-chave async, pode-se escrever seu código da mesma maneira que se escreve o código síncrono, pois o compilador se encarrega de toda a complexidade e libera o programador para escrever a lógica do programa.

```
async Task<int> AccessTheWebAsync()
{
    HttpClient client = new HttpClient();
    Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com");
    DoIndependentWork();
    string urlContents = await getStringTask;
    return urlContents.Length;
}
```

Figura 3: Método Assíncrono
Fonte: Autores (2017)

Para escrever um método async deve-se seguir a seguinte sintaxe:

- A assinatura do método deve ter a palavra-chave async;
- O nome de um método assíncrono, por convenção, deve terminar com o sufixo "Async";
- O método deve retornar Task, Task<T> ou void.

Para usar este método, deve-se esperar seu retorno usando a palavra-chave await, como no exemplo: “string urlContents = await client.GetStringAsync()”. Seguindo estas diretrizes, quando o compilador encontra um método com a expressão await, que marca o ponto em que o método não pode continuar até que a operação assíncrona aguardada seja concluída, ele inicia sua execução em segundo plano e continua a execução de outras tarefas. Quando o método está completo, a execução retorna na instrução seguinte à chamada do método.

Os aprimoramentos da linguagem C# com as palavras-chaves async e await restauram a ordem sequencial do código, enquanto usam os recursos do sistema de maneira eficiente. Ainda há alguns aspectos relevantes a serem observados, como concorrência ou sincronização de tarefas, mas estes são menores, comparado ao trabalho necessário para a criação de um bom programa que usa processamento paralelo. A utilização das técnicas aqui descritas ajuda sobremaneira na criação de programas que fazem uso da computação paralela e que, portanto, melhor utilizam os recursos do sistema.

2.3 PREMISSAS ATUARIAIS

Nas últimas décadas houve um crescimento significativo na quantidade de informação armazenada em formatos eletrônicos. Szalay et al (2000) estimam que a quantidade de informação no mundo dobra a cada 20 meses. Isso foi proporcionado por diversos fatores, como por exemplo, a queda de preços dos equipamentos de armazenamento e processamento, e os avanços nos mecanismos de captura e geração de dados, tais como, leitores de código de barras, sensores remotos e satélites espaciais.

Segundo Piatetsky-Shapiro (1991), volumes de dados, produzidos e armazenados em larga escala, são inviáveis de serem lidos ou analisados por especialistas, usando-se métodos tradicionais, tais como, planilhas eletrônicas e relatórios informativos operacionais. Por outro lado, sabe-se que grandes quantidades de dados equivalem a um maior potencial de informação.

As planilhas eletrônicas constituem um ferramental utilizado em larga escala pelos profissionais de Ciências Atuariais. Durante o trabalho de definição das premissas atuariais e financeiras, por vezes, o atuário tem a necessidade de manipular grandes volumes de dados para produzir o conhecimento basilar a respeito da população estudada.

As premissas atuariais e financeiras representam um conjunto formal de estimativas para eventos: biométricos, financeiros, econômicos, demográficos, sociais etc. A escolha e o uso de premissas atuariais descomprometidas com a realidade à qual os participantes, patrocinadores e entidade estão submetidos podem levar a custos incorretos, provocando déficit ou superávit técnico, bem como a superexposição a riscos ou subexposição a eles.

O uso de premissas mais conservadoras pode conduzir a custos iniciais mais elevados, embora com riscos menores de custos crescentes. No entanto, a adoção de premissas menos conservadoras deve ser feita com o conhecimento do risco de que elas podem não se confirmar, possibilitando que haja problemas críticos de solvência em data futura. (RODRIGUES, 2008)

As premissas atuariais serão sempre critérios preferentemente permeados pelo bom senso, lembrando que excessos de margens de segurança são tão gravosos quanto os excessos de riscos que se pretende assumir. Ambos conduzem à incapacidade de pagamento, ora de participante e/ou patrocinadores, ora da própria entidade de previdência ou Plano de Benefício.

Em função do exposto acima, surge o indicativo de que as planilhas eletrônicas talvez não sejam, em alguns casos, o ferramental ideal a ser usado por um atuário para a definição das premissas atuariais e financeiras.

3. IMPLENTAÇÃO DA CARGA PARALELA DAS BASES DE DADOS

No estudo atuarial das Forças Armadas Brasileira, realizado pelo Ministério da Defesa (MD), as premissas atuariais, são levantadas observando-se uma população de, aproximadamente, dois milhões e trezentos mil registros. Tal volume de dados deixa claras evidências de que a planilha eletrônica mais usada no mercado, neste caso notadamente o Microsoft Excel, não poderia ser utilizada. Esta planilha eletrônica na sua versão de 32 bits seria capaz de manipular, no máximo, 65.535 linhas ou registros, sem considerar o número de colunas. Já na sua versão de 64 bits seria capaz de manipular, no máximo, 1.048.576 linhas ou registros, sem considerar o número de colunas.

Em um cenário com um volume de dados dessa magnitude, um Sistema de Gerenciamento de Banco de Dados (SGBD) mostra-se como ferramental ideal para a manipulação e armazenamento dos dados envolvidos, não só na definição das premissas atuariais, como também para a realização do próprio cálculo atuarial. Daí, o estudo ora apresentado tem uma forte característica dual, na medida que pode ser utilizado no cálculo atuarial de qualquer outra instituição.

Coronel e Rob (2010) afirmam que um sistema de gerenciamento de banco de dados (SGDB) é um conjunto de programas que gerenciam a estrutura do banco de dados e controlam o acesso aos dados armazenados. Até certo ponto, o banco de dados se assemelha a um arquivo eletrônico com conteúdo muito bem organizado com a ajuda de um software robusto, conhecido como sistema de gerenciamento de banco de dados.

Apesar de definido formalmente o SGDB não tem, no âmbito deste trabalho, o papel de ator principal. O objetivo do estudo é mostrar que, independentemente das características técnicas do SGDB utilizado, a utilização do C# e as técnicas de Computação Paralela já são suficientes para reduzir sobremaneira o esforço computacional envolvido no cálculo atuarial, seja das Forças Armadas Brasileiras ou de qualquer outra instituição.

Tomando como base o conceito de que, em muitos casos, as premissas atuariais, são levantadas observando-se uma grande massa de dados e que um SGBD mostra-se como ferramental ideal para a manipulação e armazenamento dos mesmos, apresenta-se uma aplicação prática do uso da computação paralela para inserir registros em um banco de dados.

O primeiro passo é criar a tabela no banco onde os dados serão inseridos. Deve-se observar que, a sintaxe SQL é compatível com o Microsoft SQL Server 2008, podendo,

contudo, ser facilmente adaptada para o padrão utilizado por outros bancos de dados, conforme pode ser observado na Figura 4.

```
USE [DB.TESTEBASE]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TesteBase](
        [CAMPO01] [nvarchar](6) NOT NULL,
        [CAMPO02] [nvarchar](20) NOT NULL,
        [CAMPO03] [nvarchar](1) NOT NULL,
        [CAMPO04] [int] NULL,
        [CAMPO05] [nvarchar](1) NULL,
        [CAMPO06] [date] NULL,
        [CAMPO07] [int] NULL,
        [CAMPO08] [nvarchar](1) NULL,
        [CAMPO09] [int] NULL,
        [CAMPO10] [nvarchar](1) NULL,
        [CAMPO11] [int] NULL,
        [CAMPO12] [date] NULL,
        [CAMPO13] [int] NULL,
        [CAMPO14] [int] NULL,
        [CAMPO15] [numeric](15, 2) NULL,
CONSTRAINT [PK_TesteBase] PRIMARY KEY CLUSTERED
(
        [CAMPO01] ASC,
        [CAMPO02] ASC,
        [CAMPO03] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Figura 4: Criação da tabela no Banco de Dados
Fonte: Autores (2017)

No exemplo da Figura 5, a aplicação implementa uma função chamada InserirMilitar, que recebe o parâmetro pMilitar do tipo TMilitarATIVO.

```
public bool InserirMilitar(TMilitarATIVO pMilitar)
{
    bool vRetorno = false;
    TAplicacao vAplicacao = TAplicacao.Instancia;
    SqlConnection vConexao = new SqlConnection(vAplicacao.DataSourceConnectionString);
    vConexao.Open();
    try
    {
        SqlCommand vComandoSQL = new
        SqlCommand("INSERT INTO [TesteBase] " +

"([CAMPO01],[CAMPO02],[CAMPO03],[CAMPO04],[CAMPO05],[CAMPO06],[CAMPO07],[CAMPO08],[CAMPO09],[CAM
PO10],[CAMPO11],[CAMPO12],[CAMPO13],[CAMPO14],[CAMPO15]) " +
                "VALUES
(@CAMPO01,@CAMPO02,@CAMPO03,@CAMPO04,@CAMPO05,@CAMPO06,@CAMPO07,@CAMPO08,@CAMPO09,@CAMPO10,@CAMP
O11,@CAMPO12,@CAMPO13,@CAMPO14,@CAMPO15);", vConexao);
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO01", pMilitar.ANO_MES));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO02", pMilitar.ID_MILITAR));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO03", pMilitar.FORCA));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO04", pMilitar.ANO_MES));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO05", pMilitar.SEXO));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO06", pMilitar.DATA_DE_NASCIMENTO));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO07", pMilitar.POSTO_GRADUACAO));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO08", pMilitar.TIPO_DE_ATIVIDADE));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO09", pMilitar.CORPO_ARMA));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO10", pMilitar.CIRCULO_HIERARQUICO));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO11", pMilitar.REGIAO_DE_PAGAMENTO));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO12", pMilitar.DATA_DE_INGRESSO));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO13", pMilitar.TEMPO_DE_SERVICO));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO14", pMilitar.TEMPO_TOTAL));
        vComandoSQL.Parameters.Add(new SqlParameter("@CAMPO15", pMilitar.SALARIO_CONTRIBUICAO));
        vComandoSQL.ExecuteNonQuery();
        vComandoSQL = null;
        vRetorno = true;
    }
    catch
    {
        vRetorno = false;
    }
    vConexao.Dispose();
    return vRetorno;
}
```

Figura 5: Implementação da função *InserirMilitar*
Fonte: Autores (2017)

O que, por vezes, passa despercebido a muitos programadores é que essa função pode ser utilizada em uma abordagem sequencial ou paralela. Na abordagem sequencial, a função seria chamada dentro de um loop onde, um a um, todos os militares seriam inseridos no banco de dados. Na abordagem paralela, a função seria chamada dentro de um *loop* paralelo onde, também um a um, todos os militares seriam inseridos no banco de dados.

A biblioteca Task Parallel Library (TPL) suporta paralelismo de dados por meio da classe *System.Threading.Tasks.Parallel*. Essa classe fornece implementações paralelas baseadas em métodos de loops *for* e *foreach*. Escreve-se a lógica de um loop paralelo *for* ou *foreach* da mesma maneira que se escreve um loop sequencial. Não é preciso criar *threads* ou filas de trabalho. Em loops básicos, não há de se preocupar com bloqueios. A TPL manipula todo o trabalho de baixo nível.

Quando um *loop* paralelo é executado, a TPL particiona a fonte de dados para que as partes sejam operadas em loops simultâneos. Nos bastidores, o agendador de tarefas particiona a tarefa com base na carga de trabalho e os recursos do sistema. Quando possível, o agendador redistribui o trabalho entre vários threads e processadores se a carga de trabalho torna-se desequilibrada.

Na construção deste exemplo, utilizou-se uma pequena base de dados contendo 81.000 (oitenta e um mil) registros, com cada registro, contendo 15 (quinze) campos. Para fins, inicialmente, meramente ilustrativos, a aplicação foi dotada de uma interface extremamente simples, contendo apenas dois botões: *Teste sequencial* e *Teste paralelo*, conforme apresentado na Figura 6.

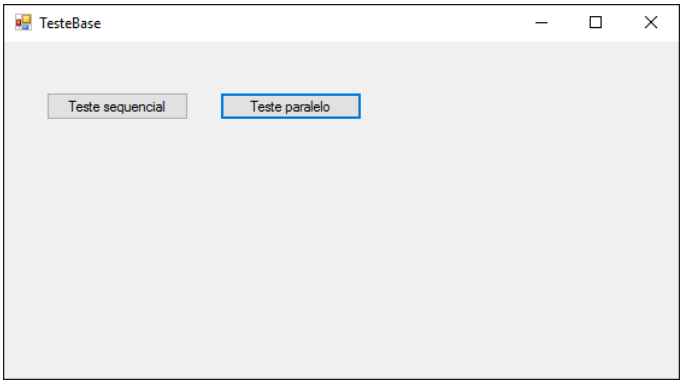

Figura 6: Interface do teste
Fonte: Autores (2017)

Por meio das codificações acopladas a cada um destes botões, pode-se verificar e comparar o esforço computacional quando da utilização da abordagem sequencial e quando da utilização da abordagem paralela. Salienta-se para leitor deste artigo que a codificação contém outros trechos de código que não serão explicados de forma detalhada por não fazerem parte do núcleo contextual, que é o uso da abordagem paralela. A seguir, na Figura 7 apresenta a codificação do botão *Teste sequencial*.

```
private void button1_Click(object sender, EventArgs e)
{
    t_inicio = DateTime.Now;
    THilitarAtivoDAO vHilitarAtivoDAO = new THilitarAtivoDAO();
    TListaHilitaresATIVOS vLista = new TListaHilitaresATIVOS();
    vLista = vHilitarAtivoDAO.ListarHilitaresAtivos();
    // Versão sequencial
    foreach (THilitarATIVO vHilitarATIVO in vLista)
    {
        InserirHilitar(vHilitarATIVO);
    }
    t_fim = DateTime.Now;
    t_diferenca = t_fim.Subtract(t_inicio);
    MessageBox.Show(t_diferenca.TotalSeconds.ToString("0.000000") + " segundos");
}
```

Figura 7: Codificação do botão *teste sequencial*
Fonte: Autores (2017)

A Figura 8, apresenta a codificação do botão *Teste paralelo*.

```
private void button2_Click(object sender, EventArgs e)
{
    t_inicio = DateTime.Now;
    THilitarAtivoDAO vHilitarAtivoDAO = new THilitarAtivoDAO();
    TListaHilitaresATIVOS vLista = new TListaHilitaresATIVOS();
    vLista = vHilitarAtivoDAO.ListarHilitaresAtivos();
    // Versão Paralela
    Parallel.ForEach(vLista, vHilitarATIVO => InserirHilitar(vHilitarATIVO));
    t_fim = DateTime.Now;
    t_diferenca = t_fim.Subtract(t_inicio);
    MessageBox.Show(t_diferenca.TotalSeconds.ToString("0.000000") + " segundos");
}
```

Figura 8: Codificação do botão *teste paralelo*
Fonte: Autores (2017)

Ao clicar em cada um dos botões, produz-se os resultados apresentados na Figura 9.

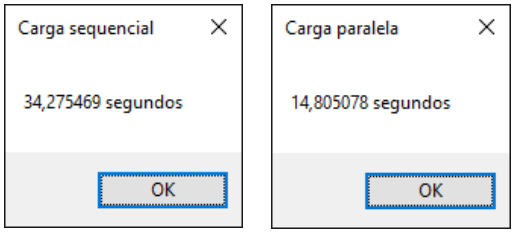

Figura 9: Tempo de processamento em cada carga
Fonte: Autores (2017)

A simples observação dos resultados deixa evidenciado que a mudança feita na programação, com a utilização de programação paralela, foi o suficiente para diminuir o tempo necessário para inserir os 81.000 registros do exemplo em questão no banco de dados, não importando qual seja este banco de dados.

A aplicação prática foi desenvolvida e executada em uma máquina relativamente simples, com apenas 8 GB de memória RAM e um processador Intel® Core™ i3-4150 Processor (3M Cache, 3.50 GHz), com 2 núcleos e 4 threads.

4. RESULTADOS ALCANÇADOS

Todo o trabalho descrito neste artigo teve como foco dotar o software do cálculo atuarial da capacidade de demandar um baixo esforço computacional para a produção dos

resultados da projeção atuarial dos pensionistas das Forças Armadas. Neste sentido, além do forte uso de orientação a objetos e da adoção de padrões de projeto de Engenharia de *Software*, o *software* do cálculo atuarial eliminou todo e qualquer acesso ao disco rígido (HD) durante a execução dos cálculos.

Para conseguir tal feito, desenvolveu-se uma técnica que carrega para a memória do computador, assim que a aplicação é iniciada, todos os dados necessários aos cálculos. Durante a carga da aplicação os dados são lidos das respectivas tabelas do banco de dados, ou gerados a partir de consultas SQL, e persistidos em estruturas (*arrays*) na memória. Uma vez carregados, os dados somente são lidos e persistidos nessas estruturas até que a aplicação seja encerrada, o que torna bastante rápido os processos de leitura e gravação dos dados.

Tal técnica mostrou-se eficiente, em função de ser o HD muito mais lento que a memória RAM. Enquanto um módulo DDR4-2400MHZ (1606R) comunica-se com o processador a uma velocidade teórica de 4200 MB/s, a velocidade de leitura sequencial dos HDs atuais dificilmente ultrapassa a marca dos 233 MB/s. Soma-se a isso o fato de que, o tempo de acesso do HD, ou seja, o tempo necessário para localizar a informação e iniciar a transferência, é consideravelmente mais alto que o da memória RAM.

Enquanto na memória fala-se em tempos de acesso inferiores a 10 nanosegundos, a maioria dos HDs trabalha com tempos de acesso superiores a 10 milissegundos. Isso faz com que o desempenho do HD seja muito mais baixo ao ler pequenos arquivos espalhados pelo disco, como é o caso da memória virtual. Em muitas situações, o HD chega ao ponto de não ser capaz de atender a mais do que duas ou três centenas de requisições por segundo. Pelo exposto, pode-se depreender que as estruturas (*arrays)* na memória, uma vez carregadas, por si só já permitem que seus dados sejam acessados a velocidades elevadas, quando comparadas com as velocidades de acesso ao disco rígido. E, quando os métodos de acesso aos dados de tais estruturas são desenvolvidos usando-se técnicas de programação paralela, o tempo necessário para a produção dos resultados é diminuído consideravelmente.

Neste exemplo da Figura 10, vLista é uma estrutura que contém todos os militares ativos carregados do banco de dados quando a aplicação foi iniciada, sendo usada como fonte de dados e percorrido pelo loop paralelo. Tem-se, portanto, uma implementação que acessa os dados de estruturas, neste caso um repositório de dados, em memória usando técnicas de programação paralela.

```
// Abordagem Paralela
Parallel.ForEach(vLista, vMilitarATIVO => InserirMilitar(vMilitarATIVO));
```

Figura 9: Tempo de processamento em cada carga
Fonte: Autores (2017)

A pesquisa apresentada desenvolveu-se a partir de técnicas de computação paralela, da utilização de padrões de Engenharia de *Software* e do uso da orientação a objetos, com vistas a demandar um baixo esforço computacional para a produção dos resultados das projeções atuariais dos pensionistas das Forças Armadas. A Tabela 1 apresenta o tempo necessário para a produção de resultados pelo software do cálculo atuarial em duas versões. A primeira versão, “Versão Anterior” não contempla nenhuma das técnicas, padrões ou recursos apresentados neste artigo. A segunda versão, denominada “Versão Refatorada”, é o produto final do trabalho aqui apresentado.

Os dados são apresentados em três grupos distintos, grupos estes que são notadamente aqueles que demandam maior tempo de processamento e, cujos resultados são de interesse do estudo em lide.

Tabela 1: Comparação dos tempos de processamento

PROCESSO	VERSÃO ANTERIOR [Aproximado] HH:MM:SS	VERSÃO REFATORADA [Aproximado] HH:MM:SS	REDUÇÃO DE TEMPO (em %)
Importação das bases de dados.	04:00:00	00:08:00	96,7%
Cálculo atuarial do Valor Presente.	23:00:00	00:07:00	99,5%
Projeção atuarial com prazo de 75 anos.	16:00:00	00:05:00	99,5%
TEMPO TOTAL	**43:00:00**	**00:20:00**	**99,2%**

Fonte: Autores (2017)

A tabela permite observar que em todos os grupos observados a diminuição do tempo necessário à produção dos resultados do cálculo atuarial foi reduzido de forma muito expressiva, notadamente no Cálculo Atuarial do Valor Presente.

5. CONSIDERAÇÕES FINAIS

A pesquisa apresentada alcançou o objetivo desejado na medida que reduziu em mais de 90% os tempos de processamento dos cálculos referentes às projeções atuariais dos pensionistas das FFAA. No caso específico do Exército Brasileiro, a partir de uma massa de dados de 500.000 registros, foi possível executar um complexo modelo matemático de alta recursividade em cerca de 5 minutos.

O mundo passa por inúmeras transformações, gerando novas demandas pela sociedade. Os problemas crescem em quantidade e em complexidade, num contexto cada vez mais dinâmico e volátil, exigindo rápidas decisões dos gestores, principalmente em nível estratégico. Daí a importância da celeridade do cálculo atuarial dos pensionistas das FFAA, pois, assim, pode-se rapidamente traçar inúmeros cenários de maneira a apoiar a Alta Administração das Forças Armadas.

6. REFERÊNCIAS

CONDE, CEZAR NEWTON. Tábua de mortalidade destinada a entidades fechadas de previdência privada. 1991. Tese (Mestrado)–Ciências Atuariais, Pontifícia Universidade Católica de São Paulo, São Paulo, 1991.

CONDE, NEWTON CEZAR. Os parâmetros atuariais dos planos de previdência complementar. Revista Fundos de Pensão. São Paulo, a. 24, n. 309, p. 79-83, out. 2005.

CORONEL, CARLOS; ROB, PETER. Sistemas de Banco de Dados - Projeto, Implementação e Administração Ed. Cengage Learning, 2010.

DIAS, CÍCERO RAFAEL BARROS; SANTOS, JOSENILDO DOS. Gestão atuarial dos regimes próprios de previdência social. In: APOSTILA de Curso de Pós-Graduação em RPPS. Recife: Centro Brasileiro de Estudos Previdenciários, 2010.

KIRK, DAVID B; HWU, WEN-MEI. Programming Massively Parallel Processors: A Hands-on Approach. Elsevier, 2010.

PIATETSKY-SHAPIRO, G. KNOWLEDGE. Discovery in real databases: A report on the IJCAI-89 Workshop. AI Magazine, Vol. 11, No. 5, Jan. 1991, Special issue, 68-70.

RODRIGUES, JOSÉ A. Gestão de risco atuarial**.** São Paulo: Ed. Saraiva, 2008.

SZALAY, A.; KUNSZT, P. Z.; THAKAR, A.; GRAY, J.; SLUT, D. R. Designing and mining multi-terabyte astronomy archives: The sloan digital sky survey. Proceedings of the ACM SIGMOD, pages 451-462. ACM Press, 2000.