Universidade Federal do Pampa
Gabriel Escobar Vasques

Estudo sobre Impacto da Criação Dinâmica de Processos do MPI-2 no Desempenho de Duas Aplicações Paralelas

Alegrete
2016

Gabriel Escobar Vasques

Estudo sobre Impacto da Criação Dinâmica de Processos do MPI-2 no Desempenho de Duas Aplicações Paralelas

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Márcia Cristina Cera

Universidade Federal do Pampa

Alegrete

2016

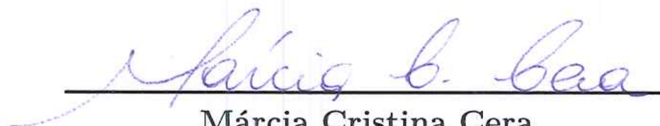
Gabriel Escobar Vasques

Estudo sobre Impacto da Criação Dinâmica de Processos do MPI-2 no Desempenho de Duas Aplicações Paralelas

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

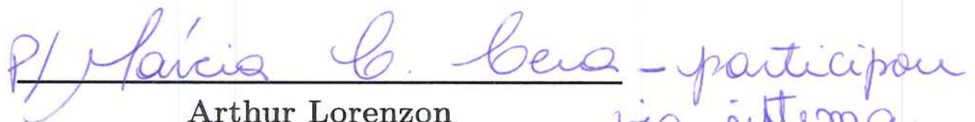
Trabalho de Conclusão de Curso defendido e aprovado em 28 de nov de 2016

Banca examinadora:



Márcia Cristina Cera
Unipampa

Claudio Schepke
Unipampa



Arthur Lorenzon
UFRGS

Este trabalho é dedicado a Deus, minha família, minha orientadora, professores, amigos e colegas.

Agradecimentos

Primeiramente agradeço a Deus pelo dom da vida, e por mais esta conquista.

Agradeço de forma especial, à minha orientadora Professora Márcia Cristina Cera pelos conselhos e atividades relacionadas à orientação, o que permitiu que este trabalho fosse realizado com sucesso, além da amizade e confiança, fundamentais para relação entre mestre e aluno e na vida acadêmica.

Agradeço a Universidade Federal do Pampa (UNIPAMPA), ao corpo docente do curso de Ciência da Computação, em especial àqueles que contribuiram de alguma forma para minha formação. Gostaria também de agradecer aos técnicos, vigilantes e demais funcionários do Campus Alegrete pela amizade e a boa vontade no auxílio a atividades relacionadas ao cotidiano e a vida acadêmica.

Agradeço aos meu colegas, em especial ao "pessoal da sala 303"(LAPIA, LEA...) pelo apoio nas horas de dificuldade, e pelas conquistas obtidas no decorrer do curso.

Agradeço aos meus amigos que conheci no decorrer destes anos em que vivi em Alegrete pelo auxílio em todos os momentos. Por fim, estendo meu agradecimento a todos meus amigos que contribuíram para a minha formação.

Por último, agradeço a minha família (Mariela, Imery, Susana, Rubens, etc) pelo apoio prestado durante esta etapa, o qual foi extremamente importante, fazendo com que nos momentos de dificuldade fosse encontrado conforto.

The future belongs to those who believe in the beauty of their dreams. - Eleanor Roosevelt

Resumo

MPI (*Message-Passing Interface*) é o padrão para o paradigma de troca de mensagens em sistemas de computação paralela distribuída. A norma MPI-2 além da possibilidade de criar processos estaticamente ao início da execução, como na norma MPI-1, adiciona a possibilidade de criar processos dinamicamente em tempo de execução. Este trabalho tem como objetivo comparar implementações com criação estática e dinâmica de processos acerca de desempenho e utilização de recursos computacionais para investigar o sobrecusto (*overhead*) causado pela criação dinâmica de processos em duas aplicações com carga de trabalho regular e irregular. Para isso foram utilizadas implementações MPI-1 e MPI-2 dos problemas do Jogo da Vida e *Skyline Matrix Solver*. Na primeira análise através do Jogo da Vida foi possível verificar o impacto da criação dinâmica de processos em aplicações com carga de trabalho regular e cujo tempo de computação é similar entre os processos. Nossos resultados mostram que o sobrecusto em aplicações MPI-2 foi atenuado pelo modelo de comunicação adotado. O consumo de memória, as trocas de contexto, interrupções tem comportamento muito semelhante exceto as etapas iniciais das execuções para ambas versões. O uso de *CPU* foi de 100% para todos os casos durante toda execução, com exceção para as execuções com 32 processos que tiveram oscilação durante toda execução. Já a segunda análise realizada sobre o *Skyline Matrix Solver* foi possível verificar o impacto da criação de processos em tempo de execução em aplicações com carga de trabalho irregular. Nossos resultados mostram que para esta aplicação a inserção da criação dinâmica de processos teve impacto negativo no desempenho da aplicação quando comparada a criação estática. O consumo de memória foi inferior para todas execuções na versão com criação dinâmica de processos exceto pela execução com 16 processos onde essa situação se inverte. Porém a versão com criação de processos em tempo de execução teve um número maior de trocas de contexto e interrupções devido a comunicação irregular da aplicação.

Palavras-chave: MPI, Message-Passing Interface. Criação Dinâmica de Processos, Programação Paralela, dmon.

Abstract

MPI (*Message-Passing Interface*) is the standard for messaging-passing paradigm in distributed parallel computing systems. The MPI-2 standard, besides the ability to create processes statically at the beginning of execution, as in the MPI-1 standard, adds the possibility of dynamically creating processes at run time. This work aims to compare implementations with static and dynamic creation of processes about the performance and use of computational resources to investigate the overhead caused by the dynamic creation of processes in two applications with regular and irregular workload. Para isso foram utilizadas implementações MPI-1 e MPI-2 dos problemas do Jogo da Vida e *Skyline Matrix Solver*. In the first analysis through the Game of Life it was possible to verify the impact of the dynamic creation of processes in applications with regular workload and whose computation time is similar between the processes. Our results show that the overhead in applications MPI-2 was attenuated by the communication model adopted. Memory consumption, context switches, interrupts have very similar behavior except the initial stages of the runs for both versions. *CPU* usage was 100% for all cases during execution, except for executions with 32 processes that oscillated during execution. The second analysis of the *Skyline Matrix Solver* was able to verify the impact of the creation of processes at run time in applications with irregular workload. Our results show that for this application the insertion of dynamic process creation had a negative impact on the performance of the application when compared to static creation. The memory consumption was lower for all executions in the version with dynamic process creation except for execution with 16 processes where this situation is reversed. However, the version with creation of processes at run time had a greater number of context changes and interruptions due to irregular communication of the application.

Key-words: MPI, Message-Passing Interface. Creating Dynamic Processes, Parallel Programming, dmon.

Lista de ilustrações

Figura 1 – Exemplos de arquiteturas segundo classificação de Flynn 26
Figura 2 – Jogo da Vida: Aplicação das Leis Genéticas 34
Figura 3 – Modelos de Comunicação Mestre/Trabalhador 35
Figura 4 – Pseudocódigo das versões do Jogo da Vida MPI-2 36
Figura 5 – Exemplo de matriz *Skyline* 37
Figura 6 – Matriz *Lower* 38
Figura 7 – Matriz *Upper* 39
Figura 8 – Pseudocódigo da função CalculaDiagonal 40
Figura 9 – Pseudocódigo da função CalculaLU 40
Figura 10 – Pseudocódigo da função CalculaSomatorioLU 41
Figura 11 – Jogo da Vida: Comparativo Sequencial/MPI - Matriz de ordem 8192 . 46
Figura 12 – Jogo da Vida: Comparativo Sequencial/MPI - Matriz de ordem 12288 . 46
Figura 13 – Jogo da Vida: Comparativo Sequencial/MPI - Matriz de ordem 16384 . 47
Figura 14 – Jogo da Vida - Consumo de Memória Sequencial 48
Figura 15 – Jogo da Vida - Consumo de Memória 4 Processos 49
Figura 16 – Jogo da Vida - Consumo de Memória 8 Processos 50
Figura 17 – Jogo da Vida - Consumo de Memória 16 Processos 52
Figura 18 – Jogo da Vida - Consumo de Memória 32 Processos 53
Figura 19 – Jogo da Vida - Trocas de Contexto e Interrupções Sequencial 54
Figura 20 – Jogo da Vida - Trocas de Contexto e Interrupções 4 processos 55
Figura 21 – Jogo da Vida - Trocas de Contexto e Interrupções 8 processos 56
Figura 22 – Jogo da Vida - Trocas de Contexto e Interrupções 16 processos 57
Figura 23 – Jogo da Vida - Trocas de Contexto e Interrupções 32 processos 58
Figura 24 – *Skyline Matrix Solver*: Comparativo Sequencial/MPI - Matriz de ordem 4096 59
Figura 25 – *Skyline Matrix Solver:* Comparativo Sequencial/MPI - Matriz de ordem 5120 60
Figura 26 – *Skyline Matrix Solver* - Consumo de Memória Sequencial 60
Figura 27 – *Skyline Matrix Solver* - Consumo de Memória 4 Processos 62
Figura 28 – *Skyline Matrix Solver* - Consumo de Memória 8 Processos 63
Figura 29 – *Skyline Matrix Solver* - Consumo de Memória 16 Processos 64
Figura 30 – *Skyline Matrix Solver* - Consumo de Memória 32 Processos 65
Figura 31 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções Sequencial . 66
Figura 32 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 4 Processos 67
Figura 33 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 8 Processos 68
Figura 34 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 16 Processos 69

Figura 35 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 32 Processos 70
Figura 36 – Jogo da Vida - Consumo de Memória Sequencial - Matriz 8192 77
Figura 37 – Jogo da Vida - Consumo de Memória 4 Processos 8192 78
Figura 38 – Jogo da Vida - Consumo de Memória 8 Processos 8192 79
Figura 39 – Jogo da Vida - Consumo de Memória 16 Processos 8192 80
Figura 40 – Jogo da Vida - Consumo de Memória 32 Processos 8192 81
Figura 41 – Jogo da Vida - Trocas de Contexto e Interrupções Sequencial 8192 . . 81
Figura 42 – Jogo da Vida - Trocas de Contexto e Interrupções 4 Processos 8192 . . 82
Figura 43 – Jogo da Vida - Trocas de Contexto e Interrupções 8 Processos 8192 . . 83
Figura 44 – Jogo da Vida - Trocas de Contexto e Interrupções 16 Processos 8192 . 84
Figura 45 – Jogo da Vida - Trocas de Contexto e Interrupções 32 Processos 8192 . 85
Figura 46 – Jogo da Vida - Consumo de Memória Sequencial - Matriz 12288 85
Figura 47 – Jogo da Vida - Consumo de Memória 4 Processos 12288 86
Figura 48 – Jogo da Vida - Consumo de Memória 8 Processos 12288 87
Figura 49 – Jogo da Vida - Consumo de Memória 16 Processos 12288 88
Figura 50 – Jogo da Vida - Consumo de Memória 32 Processos 12288 89
Figura 51 – Jogo da Vida - Trocas de Contexto e Interrupções Sequencial 12288 . . 89
Figura 52 – Jogo da Vida - Trocas de Contexto e Interrupções 4 Processos 12288 . 90
Figura 53 – Jogo da Vida - Trocas de Contexto e Interrupções 8 Processos 12288 . 91
Figura 54 – Jogo da Vida - Trocas de Contexto e Interrupções 16 Processos 12288 . 92
Figura 55 – Jogo da Vida - Trocas de Contexto e Interrupções 32 Processos 12288 . 93
Figura 56 – *Skyline Matrix Solver* - Consumo de Memória Sequencial 5120 93
Figura 57 – *Skyline Matrix Solver* - Consumo de Memória 4 Processos 5120 94
Figura 58 – *Skyline Matrix Solver* - Consumo de Memória 8 Processos 5120 95
Figura 59 – *Skyline Matrix Solver* - Consumo de Memória 16 Processos 5120 96
Figura 60 – *Skyline Matrix Solver* - Consumo de Memória 32 Processos 5120 97
Figura 61 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções Sequencial 5120 97
Figura 62 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 4 Processos 5120 98
Figura 63 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 8 Processos 5120 99
Figura 64 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 16 Processos 5120 100
Figura 65 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 32 Processos 5120 101

Lista de siglas

CPU *Central Processing Unit* - Unidade Central de Processamento

HPC *High Performance Computer* - Computação de Alto Desempenho

LU *Lower-Upper* (Inferor-Superior)

MIMD *Multiple Instruction Multiple Data* - Múltiplo Fluxo de Instruções

MISD *Multiple Instruction Single Data* - Múltiplos Fluxos de Instruções

MPI *Message-Passing Interface* - Interface para Passagem de Mensagens

MPI-1 *Message-Passing Interface*

MPI-2 *Message-Passing Interface*

MPMD *Multiple Program Multiple Data* - Múltiplos Programas Múltiplos Dados

SIMD *Single Instruction Multiple Data* - Único Fluxo de Instruções

SISD *Single Instruction Single Data* - Único Fluxo de Instruções

SPMD *Single Program Multiple Data* - Único Programa Múltiplos Dados

Sumário

1	**INTRODUÇÃO**	**21**
1.1	**Objetivo**	**22**
1.2	**Estrutura do Texto**	**22**
2	**REFERENCIAL TEÓRICO**	**25**
2.1	**Programação Paralela**	**25**
2.1.1	Programação Paralela e Distribuída	27
2.1.2	Uso de Recursos Computacionais pelo Paralelismo	28
2.2	**MPI - Massage-Passing Interface**	**30**
2.2.1	MPI-1	30
2.2.2	MPI-2	31
2.3	**Conclusões do Capítulo**	**32**
3	**PROBLEMAS ALVO**	**33**
3.1	**Jogo da Vida**	**33**
3.1.1	Principais características	33
3.1.2	Paralelização Jogo da Vida	34
3.1.2.1	Detalhes da Implementação MPI-1	35
3.1.2.2	Detalhes da Implementação MPI-2	35
3.2	**Skyline Matrix Solver**	**36**
3.2.1	Principais características	37
3.2.2	Paralelização *Skyline Matrix Solver*	38
3.2.2.1	Detalhes da Implementação MPI-1	39
3.2.2.2	Detalhes da Implementação MPI-2	39
3.3	**Conclusões do Capítulo**	**40**
4	**METODOLOGIA**	**43**
4.1	**Ambiente de Testes**	**43**
4.2	**Métodos e Métricas**	**43**
4.2.1	Condução dos testes	44
5	**ANÁLISE DOS RESULTADOS**	**45**
5.1	**Jogo da Vida**	**45**
5.1.1	Análise de Desempenho	45
5.1.2	Análise de Consumo de Memória	47
5.1.3	Análise de Trocas de Contexto e Interrupções	53

5.1.4 Análise de Utilização de *CPU* 57
5.2 Skyline Matrix Solver 58
5.2.1 Análise de Desempenho 59
5.2.2 Análise de Consumo de memória 59
5.2.3 Análise de Trocas de Contexto e Interrupções 66
5.2.4 Análise de Utilização de *CPU* 69
5.3 Conclusões do Capítulo 70

6 CONCLUSÕES 73

REFERÊNCIAS 75

APÊNDICE A – GRÁFICOS DE RESULTADOS ANÁLISES REALIZADAS 77

1 Introdução

A Lei de Moore (MOORE, 1965) diz que entre 18 e 24 meses o número de transistores dos processadores dobrariam não elevando o custo de produção e dessa forma o poder de processamento seria dobrado. Ela é uma tendência que motivou com que diversas formas de se atenuar seus impactos fossem propostas. Questões como dissipação de calor e consumo de energia motivaram a criação de diferentes arquiteturas. Dentre elas estão as arquiteturas *Multi-Core* e *Many-Core*. Os sistemas *Multi-Core* podem apresentar de 2 a 12 *cores* de processamento como os processadores *Core I3*, *Core I5* e *Core I7* por exemplo. Já os sistemas *Many-core*, podem apresentar de 10 a centenas de *cores*, como os co-processadores *Xeon Phi* da Intel. A grande capacidade dos computadores atuais, e a crescente quantidade de núcleos de processamento agregados nas máquinas, tornam a programação paralela uma alternativa atraente para o desenvolvimento de aplicações de alto desempenho, e assim a consolidando no desenvolvimento de aplicativos eficientes. Com o avanço nas pesquisas e desenvolvimento de máquinas com maior poder computacional, cada vez mais problemas complexos estão sendo estudados e resolvidos, visto que a abordagem sequencial apresenta diversas limitações, fazendo com que o tempo de execução seja muito alto e impraticável para problemas de larga escala. A programação paralela nada mais é que a divisão de tarefas de uma determinada aplicação, onde o objetivo é a redução do tempo total de execução (STALLINGS, 2002).

O MPI (*Message-Passing Interface*) é uma interface de programação paralela que se tornou padrão para a comunicação de dados em computação paralela e de alto desempenho, em arquiteturas de memória distribuída. Sua especificação é mantida pelo *MPI-Forum*. MPI utiliza o paradigma de troca de mensagens para prover rotinas de sincronização e controle entre processos, e ainda distribuir e consolidar dados entre os computadores interconectados (COELHO, 2012). Também tem como característica o uso de técnicas avançadas como comunicações não bloqueantes, definições de tipos de dados, manipulação de comunicadores, criação dinâmica de processos, dentre outras, com o objetivo de obter ganhos de desempenho (CERA M. C.; MAILLARD, 2010).

A norma MPI-1 é caracterizada por criar processos de forma estática, ocorrendo quando a execução da aplicação é iniciada. Já na norma MPI-2 os processos podem ser criados de forma dinâmica, ocorrendo em tempo de execução, mas um custo extra é gerado para a aplicação, que pode ter o impacto reduzido por meio de técnicas eficientes de programação, além de características avançadas oriundas do próprio MPI (LORENZON, 2013).

A criação dinâmica de processos torna possível o desenvolvimento de aplicações

MPI-2 flexíveis e de fácil adaptação à mudanças da arquitetura em tempo de execução, como por exemplo inclusão e exclusão de computadores em *clusters*. Estas aplicações podem auxiliar na melhor utilização dos recursos de *clusters*, dessa forma podemos executar aplicações científicas em nós computacionais que estão ociosos. A criação de processos em tempo de execução acarreta em custos extras à aplicação, pois a criação dinâmica se dá de forma síncrona, e dessa forma é constituída uma hierarquia no modelo de comunicação entre processos criados e processos criadores (LORENZON, 2013).

Este trabalho tem como objetivo realizar um estudo sobre o o impacto de aplicações com criação dinâmica de processos em MPI, observando os efeitos gerados pela criação dinâmica de processos em relação ao desempenho, consumo de memória, trocas de contexto, interrupções e utilização de CPU em ambiente de memória compartilhada.

Dessa forma, este trabalho utilizará os algoritmos sequenciais, e paralelos de melhor desempenho implementados por Lorenzon (2013) no estudo do impacto da criação dinâmica de processos, onde os dois problemas pertencem a suíte de problemas *Cowichan Problems*. Os problemas dessa suíte são largamente utilizados para análise de diferentes características em sistemas de programação paralela (WILSON, 1994). Utilizaremos o problema do Jogo da Vida, que é considerado um problema com carga de trabalho regular, onde os processos possuem tempo de execução semelhante. Também será utilizado o problema *Skyline Matrix Solver*, que é considerado um problema com carga de trabalho irregular, onde os processos possuem tempo de execução distintos.

1.1 Objetivo

O presente trabalho tem como objetivo principal estudar o impacto da criação dinâmica de processos através da norma MPI-2 no desempenho de duas aplicações paralelas. Com a finalidade de alcançar o objetivo principal. Definimos objetivos específicos:

1. Analisar o desempenho e consumo de recursos computacionais das implementações Sequencial, MPI-1 e MPI-2;

2. Identificar a sobrecarga (*overhead*) da criação dinâmica de processos no consumo de memória, trocas de cotexto, interrupções e uso de *CPU* durante a execução das aplicações.

1.2 Estrutura do Texto

O trabalho está estruturado da seguinte forma: O Capítulo 2 apresenta o padrão MPI, ressaltando algumas diferenças e as principais característias das normas MPI-1 e MPI-2. Também são apresentados conceitos sobre comunicação e modelo de memória dis-

tribuída. Seguido por uma breve descrição sobre a suíte de problemas *Cowichan Problems*. Por fim são apresentadas as conclusões referentes ao capítulo.

O Capítulo 3 apresenta as descrições dos problemas do Jogo da Vida, e *Skyline Matrix Solver*, além das principais características referentes aos dois problemas. As paralelizações dos algoritmos, questões de implementação, assim como as características do ambiente de testes utilizado pelo autor do trabalho utilizado como base também são apresentadas. Além de destacadas as versões dos códigos implementados por (LORENZON, 2013) de melhor desempenho, e que serão utilizadas nas análises deste trabalho, além das conclusões referentes ao capítulo.

Já o Capítulo 4 apresenta a abordagem utilizada, assim como o ambiente de testes, as ferramentas e o modo de condução dos mesmo, além dos métodos o métricas que foram utilizados nas análises. Por fim, as conclusões do capítulo também são apresentadas.

O Capítulo 5 apresenta os resultados obtidos nas análise acerca do desempenho, consumo de memória, trocas de contexto e interrupções apresentados pelas aplicações. Por fim, são apresentadas as conclusões do capítulo.

O Capítulo 6 discute os resultados obtidos nas análises realizadas nos problemas alvo. Por fim referências bibliográficas seguidas do Apêndice A onde os gráficos de resultados são apresentados.

2 Referencial Teórico

Este capítulo tem como objetivo contextualizar as principais características que envolvem a programação paralela e a programação com a biblioteca de comunicação MPI (*Message-Passing Interface*). A subseção 2.1.1 apresenta alguns detalhes sobre programação paralela distribuída, assim como alguns modelos de comunicação adotados em sistemas paralelos. Os recursos utilizados na programação paralela, indicando quais recursos tem seu uso aumentado por conta do paralelismo são aprensentados na subseção 2.1.2. A seção 2.2 apresenta o padrão MPI e as principais características das normas MPI-1 e MPI-2. Por fim, a seção 2.3 apresenta as conclusões do capítulo.

2.1 Programação Paralela

Na área de Ciência da Computação existem diversos desafios a serem vencidos, um deles é criar soluções computacionais eficientes, com tempo de processamento reduzido e resultados coerentes. Nesse contexto a programação paralela e de alto desempenho - HPC (*High Performance Computing*) surge como uma atraente alternativa para o desenvolvimento de diversas soluções. Por exemplo, cálculos de previsão do tempo (SILVA R.; DIAS; SOUZA, 2009), aplicações com processamento de grandes volumes de dados (NAVAUX, 1989), etc.

A programação paralela, ou concorrente, é um paradigma de programação que faz uso de arquiteturas *multicore* ou multiprocessadores para realizar tarefas concorrentemente (NAVARRO; HITSCHFELD-KAHLER; MATEU, 2014). Este molde de programação executa partes diferentes de um programa simultaneamente com objetivo principal de reduzir o *throughput* (latência) do processamento, aumentar o *speedup* (aceleração) e manter o processador ocupado o maior tempo possível (eficiência) (EAGER; ZAHORJAN; LAZOWSKA, 1989). Cabe ao programador identificar quais são as áreas do código que demandam de maior processamento e adequá-la para executar em paralelo.

Partindo de dois conceitos básicos, sendo estes a **sequência de instruções** e a **sequência de dados**, as arquiteturas paralelas foram classificadas em 4 grupos por (FLYNN, 1972). A sequência de instruções é a ordem na qual os programas depois de decompostos em instruções são executados as quais são executados pelo processador. Neste processo o compilador depois de realizar as análises semânticas e sintáticas gera um código com pequenas instruções, como por exemplo um código *Assembly* ou *MIPS*. As instruções mais demoradas e impactantes são as de acesso a memória para escrita. Existem também instruções de desvio condicionais e "pulos", o que pode alterar a ordem inicial das instruções envolvidas na execução. Já a sequência de dados trata-se da organização

dos dados envolvidos na execução e que estão carregados na memória. Os dados são carregados do disco para os níveis superiores de memória como memória principal, cache e registradores.

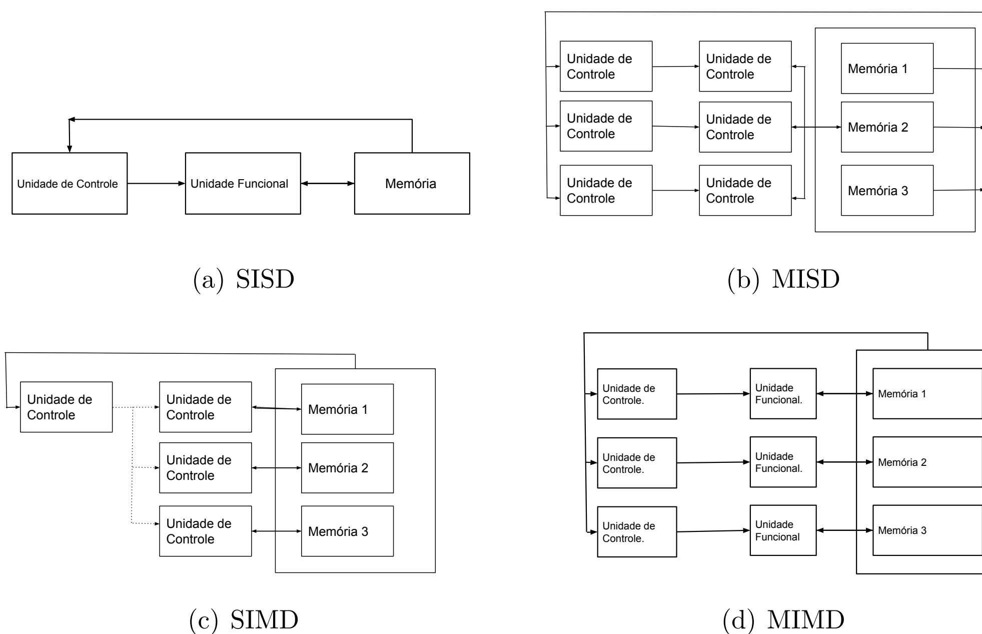
Single Instruction Single Data (SISD) - Neste modelo cada instrução opera sobre um único dado. Exemplo: Arquitetura de Von Neumann. A sub-figura *a* da Figura 1 ilustra esse modelo.

Multiple Instruction Single Data (MISD) - Neste modelo várias instruções operam sobre o mesmo dado. Exemplo: Não existem implementações. A sub-figura *b* da Figura 1 ilustra esse modelo.

Single Instruction Multiple Data (SIMD) - Neste modelo uma única instrução é executada sobre múltiplos dados de forma paralela. A sub-figura *c* da Figura 1 ilustra esse modelo.

Multiple Instruction Multiple Data (MIMD) - Neste modelo múltiplas instruções são executadas sobre múltiplos dados. Exemplos: Multiprocessadores ou multicomputadores. A sub-figura *d* da Figura 1 ilustra esse modelo.

Figura 1 – Exemplos de arquiteturas segundo classificação de Flynn



(a) SISD (b) MISD (c) SIMD (d) MIMD

Fonte: Lorenzon (2013)

Os testes foram realizados em uma máquina equipada com processador *Intel Xeon E5*, esta classificada como MIMD (*Multiple Instruction Multiple Data*) a sub-figura *d* da Figura 1 ilustra o modelo. Este modelo tem como característica a execução simultânea de múltiplos fluxos de instruções, onde cada um opera sobre seus dados de forma síncrona.

As arquiteturas paralelas apresentadas, podem ser organizadas como ambientes de

memória distribuído ou compartilhado. Em ambientes de memória distribuída, o fluxo do programa é dividido entre vários processadores que possuem recursos computacionais individuais, possibilitando que o trabalho computacional seja dividido para cada computador na forma de processos (BEZERRA H. P. NETO; COSTA, 2010). Já em ambientes de memória compartilhada, o fluxo de execuções do programa é dividido entre vários processadores que compartilham uma única memória, e o trabalho é dividido em *threads* (COSTA, 2008).

2.1.1 Programação Paralela e Distribuída

Em ambientes cujo espaço de endereçamento é distribuído, a execução de programas paralelos se dá através do uso cooperativo de processadores, que precisam trocar informações entre si utilizando troca de mensagens, possibilitado pelo uso de primitivas de envio do tipo (`send`), e de recebimento dados do tipo (`receive`). Arquiteturas como os *Clusters* e os *Grids* (*Grades*), apresentam diferenças em alguns detalhes de infra-estrutura de rede, mas basicamente são compostas por um conjunto de máquinas ou nós, que cooperam entre si na execução de aplicações utilizando troca de mensagens. A operação de troca de mensagens é facilitada pelo uso de padrões, onde o objetivo é que estes forneçam portabilidade e alto nível de abstração, tornando a tarefa de comunicação entre os processos mais simples. O MPI é um padrão altamente utilizado em computadores paralelos, pois fornece nível de segurança e confiabilidade em rotinas de trocas de mensagens (GEIST G. A.; KOHLA, 1996).

Exitem diferentes modelos que podem ser utilizados no desenvolvimento de aplicações paralelas. O modelo de programação Paralela **Mestre/Trabalhador** (KUMAR, 2002) possui um processo responsável (Mestre) por comandar a execução dos demais processos, os quais são denominados Trabalhadores. O Mestre é responsável pela distribuição das cargas de trabalho aos Trabalhadores, e pelo recebimento dos resultados computados pelos Trabalhadores. No modelo **Divisão/Conquista** (PEZZI et al., 2006) basicamente o problema é dividido em subproblemas, para os quais são calculadas soluções parciais, que ao final são combinadas gerando a solução total do problema, como por exemplo nos algoritmos clássicos de ordenação *Merge Sort* e *Quick Sort*. Já o modelo de **Fases Paralelas** (SCHEPKE; LIMA, 2015) é utilizado em aplicações onde diversas tarefas podem ser executadas em paralelo, e existem duas etapas específicas: a computação e sincronização.

O programador deve evitar problemas relacionados ao balanceamento de carga e comunicação, os quais podem degradar o desempenho das aplicações paralelas, fazendo com que o *speedup* alcançado seja abaixo do que realmente poderia ser obtido, se comparadas com implementações que fazem uso de distribuição de cargas de trabalho adequada entre os processos, ou com comunicações de maneira eficiente. As aplicações alvo utilizadas neste trabalho, foram implementadas por Lorenzon (2013) utilizando o modelo

Mestre/Trabalhador.

2.1.2 Uso de Recursos Computacionais pelo Paralelismo

A paralelização ocasiona o uso mais efetivo de recursos computacionais como os núcleos de processamento (*cores*), memória *RAM*, memória *cache*, *buffers*, quando comparada a abordagem sequencial (STALLINGS, 2002), porém alguns cuidados são exigidos para se obter maior desempenho. Os núcleos de processamento disponíveis em arquiteturas *multi-core* e *many-core* ociosos em execuções sequenciais, podem ser melhor aproveitados por aplicações paralelas. Dessa forma podemos dividir um determinado problema em subproblemas. Estes subproblemas podem ser representados por *threads* ou processos, que são computados em núcleos distintos ao mesmo tempo. Além disso alguns processadores permitem competição entre dois processos ou *threads* pelo mesmo núcleo através de tecnologias como *Hyperthreading* por exemplo.

Os processadores *multicore* e *many-core* possuem sistema de memória organizado em diferentes níveis, o que se denomina Hierarquia de Memória. Ela é fundamental para que haja ganho de desempenho em aplicações paralelas através dos *cores*. Em arquiteturas *multicore* a comunicação entre os *cores*, ou núcleos de processamente, se dá através de acessos a regiões ou níveis de memória compartilhadas entre os mesmos. Dessa forma o sistema de memória determina a capacidade de armazenamento de instruções e dados, juntamente com a velocidade com que estes dados e instruções são entregues aos *cores* (MELLO, 2014).

A hierarquia de memória possui algumas características como o princípio da inclusão, onde um dado carregado nos registradores por exemplo, também está carregado em todos os níveis abaixo (memória *cache*, memória principal e memória secundária). O princípio da localidade também é uma técnica presente no esquema de hierarquia de memória, ela pode ser analisado de duas diferentes formas: *Localidade Temporal*, onde um bloco de memória acessado recentemente tem grande probabilidade de ser acessado novamente. E *Localidade Especial* onde o processador ao acessar um dado, posteriormente tentará acessar outro dado na memória no nível abaixo do anteriormente acessado. Assim se um dado foi acessado recentemente há grande probabilidade que o próximo acesso acesso se dê em blocos subjacentes da memória no nível abaixo. Um exemplo do uso destas técnicas são as memórias *cache*, onde sua função principal é servir como um banco de palavras que de forma comum são acessadas pelo processador durante a execução de uma aplicação. Assim é reduzida a quantidade de acessos a memória principal e obtido ganho de desempenho (STALLINGS, 2002).

A Hierarquia de Memória possui multi-níveis de memória, que possuem diferentes tamanhos e velocidades. As mais rápidas são as que possuem o custo mais elevado de armazenamento por bit, e consequentemente são as menores. As memórias de menor

tamanho e mais rápidas são localizadas mais perto do processador. Os níveis da hierarquia são os seguintes:

1. **Registradores:** nível mais alto da hierarquia, corresponde ao banco de registradores localizados no núcleo do processador;

2. **Memória *cache*:** nível abaixo dos registradores, onde blocos de dados são armazenados;

3. **Memória principal:** nível abaixo da memória cache, constituída por célular de memória dinâmica (*DRAM*);

4. **Memória secundária:** último nível da hierarquia, composta por dispositivos de armazenamento de massa, como discos rígidos por exemplo.

O uso de memórias *cache* tem como objetivo obter um ganho de velocidade de acesso à memória próximo das memórias mais rápidas, e tornar uma memória de grande capacidade disponível ao sistema, com custo razoável. Memórias do tipo *cache* menores e mais rápidas, são combinadas com uma memória principal grande e lenta. A memória *cache* possui uma cópia de partes da memória principal. Dessa forma quando processador necessita ler uma palavra da memória, é verificado se a mesma não se encontra na memória *cache*, caso positivo é fornecida ao processador, caso negativo um bloco de dados da memória principal é lido para a memória *cache* e posteriormente a palavra requerida é entregue ao processador. Devido ao fenômeno da localidade de referências, quando um bloco é trazido para a memória *cache* para satisfazer alguma referência à memória, possivelmente sejam feitas referências para outras palavras desse bloco (STALLINGS, 2002).

Ao executar as aplicações nos núcleos de processamento, quando necessárias são realizadas trocas de contexto. Essas trocas de contexto são operações realizadas pelo sistema operacional. Nelas um ou mais processos que estão sendo executados são interrompidas para que outro processo com maior prioridade seja executado. É necessário que o sistema operacional salve os dados do processo para que seja mantida a coerência. Assim o estado do processo interrompido é salvo para que quando volte a ser executado se encontre no mesmo estado que se encontrava antes da troca de contexto. Uma interrupção faz com que o processador pare a execução da aplicação que está sendo executada e desvie a execução para um bloco de código, este chamado de rotina de interrupção. Ao terminar o tratamento da interrupção o controle retorna o programa que foi interrompido no mesmo estado em que estava quando ocorreu a interrupção, gerando assim uma trocas de contexto (TANENBAUM, 2003).

2.2 MPI - Massage-Passing Interface

Nos anos 90, um grupo de aproximadamente 60 pessoas de diferentes organizações dos Estados Unidos e Europa, participantes do MPI Forum (FORUM, 1994) definiram o padrão **MPI**. Posteriormente surgiram as especificações: MPI-1 onde a criação de processos é estática, e MPI-2 onde a criação de processos é dinâmica. O MPI é uma interface de programação paralela que se tornou padrão para comunicação de dados na computação de alto desempenho, através da troca de mensagens (GROPP et al., 1998), ela possui um elevado número de funções que podem ser utilizadas tanto em implementações paralelas, como em implementações distribuídas (SCHEPKE; LIMA, 2015).

Um processo MPI é que um programa o qual pode ser escrito em linguagem *C* ou *Fortran 77*, o qual realiza a comunicação com outros processos MPI realizando a chamada de rotinas MPI.

2.2.1 MPI-1

Publicada no ano de 1994, a norma MPI-1 (GROPP; LUSK; SKJELLUM, 1994) tem como principal característica a criação estática de processos, esta ocorrendo no início da execução do programa, dessa forma o número de processos não sofre qualquer alteração durante a execução. Os processos são criados durante a chamada da função `MPI_init()` e finalizados durante a chamada da função `MPI_Finalize()`.

Os processos recebem identificadores, os quais são denominados *ranks* ao inicializar o ambiente de execução através da função `MPI_Comm_rank()`. Um comunicador é responsável por transmitir os dados entre os processos MPI e identifica um grupo de processos. O comunicador MPI padrão pré-definido é denominado `MPI_COMM_WORLD`. O número total de processos de um comunicador é retornado pela função `MPI_Comm_size()`. Existem diferentes tipos de comunicadores, os **intracomunicadores** que realizam a comunicação interna entre um grupo de processos de mesmo comunicador. E também os *intercomunicadores* responsáveis por realizar a comunicação entre processos de diferentes intracomunicadores (LORENZON, 2013).

As comunicações são realizadas através das funções, `MPI_Send()` responsável pelo envio dos dados, e `MPI_Recv()` responsável pelo recebimento dos dados. Essas funções realizam operações do tipo bloqueante e exigem sincronismo entre os processos envolvidos. Assim, até que todos os dados sejam enviados a função fica bloqueada, e após completa cada processo segue seu fluxo execução. As aplicações podem ter gargalos quando implementadas utilizando operações bloqueantes, pois mantém os processos bloqueados até a conclusão da comunicação. Existem operações que quando utilizadas de maneira adequada podem atenuar essa limitação, como o uso de operações não-bloqueantes. As operações `MPI_Isend()` e `MPI_Irecv()` são exemplos de operações não-bloqueantes. A

operação `MPI_Isend()` retorna após a cópia da mensagem para o *buffer* de envio ter sido finalizada, o que torna possível a computação do processo enquanto a mensagem está sendo transmitida. Da mesma forma a operação `MPI_Irecv()` começa a operação sem a completar. E assim a função estará completa quando estiver armazenada no *buffer* de recebimento, e enquanto esta etapa ocorre o processo pode computar sobre outros dados. Para que seja verificado se as operações foram completadas ou não existem as funções `MPI_Wait()` e `MPI_Waitany()`. Esta indicam que o processo seguirá seu fluxo de execução somente quando as comunicações iniciadas forem concluídas. Uma alternativa interessante para desenvolvimento de aplicações MPI com comunicação eficiente, pode ser a combinação de operações bloqueantes e não-bloqueantes, como por exemplo a função de envio `MPI_Send()` juntamente com a função de recebimento `MPI_Irecv()`.

2.2.2 MPI-2

Publicada no ano de 1997, a norma MPI-2 (GROPP et al., 1998) define novos tópicos ao padrão MPI, como a criação dinâmica de processos, operações de entrada e saída paralela, assim como comunicações assimétricas e coletivas estendidas, onde daremos maior destaque a criação dinâmica de processos que será investigada neste trabalho. A criação dinâmica de processos possibilita que os processos sejam criados ou destruídos em tempo de execução. Geralmente a execução é iniciada com um único processo (*pai*), e através da primitiva `MPI_Comm_Spawn` os demais processos são criados de forma dinâmica (*filhos*). Esta função é invocada pelo processo pai, que cria um processo chamado filho, que pode ser diferente do pai. Processo Pai e processos filho pertencem a intercomunicadores diferentes. No processo filho o intercomunicador que referencia o pai é retornado pela função `MPI_Comm_get_parent()`, Já no processo pai o intercomunicador que referencia o filho é retornado pelo função `MPI_Comm_spawn()`, esta função pode criar um ou mais processos à cada chamada. Esta função por ser bloqueante faz com que o processo pai fique bloqueado até que a criação dos processos filhos seja concluída.

A comunicação entre pai e o(s) filhos(s) é feita através de um intercomunicador (`InterComm`). O ambiente após inicializado executa a função `MPI_Comm_get_parent()`, a qual retornará o intercomunicador `InterComm` que ligará com o intracomunicador do processo filho ao do pai. Cada processo está interno ao seu intracomunicador e interligado através do intercomunicador. Além dessas comunicações ponto a ponto através do uso de inter e intracomunicadores e coletivas através de intracomunicadores, o MPI-2 possibilita ainda que sejam realizadas comunicações coletivas através de intercomunicadores, dessa forma um processo de um determinado intracomunicador pode comunicar-se com N processos filhos pertencentes a outro intracomunicador por meio do intercomunicador que os liga.

A criação dinâmica de processos oriunda do MPI-2 auxilia na solução de problemas

através do padrão MPI, já que torna possível o desenvolvimento não só de problemas que seguem o modelo SPMD, mas também oferece suporte para programas que seguem o modelo MPMD. Todavia um custo extra para aplicação é gerado pela criação de processos em tempo de execução que pode ser compensado utilizando caraceterísticas avançadas do MPI.

2.3 Conclusões do Capítulo

Esse capítulo contextualizou a programação paralela. Assim, inicialmente a seção 2.1 destacou conceitos de programação paralela, apresentando exemplos do uso de computação de alto desempenho, assim como os objetivos. Também foram apresentadas, de forma breve, as classificações de arquiteturas paralelas destacando a classe da arquitetura utilizada nos testes. A subseção 2.1.1 apresenta conceitos sobre programação paralela em sistemas de memória distribuída, destacando as características comuns em arquiteturas como *Grids* e *Clusters*. Os modelos utilizados no desenvolvimento também são apresentados, salientando o modelo utilizado nas implementações utilizadas neste trabalho. Na subseção 2.1.2 a utilização de recursos computacionais no paralelismo foi apresentada, destacando métodos e técnicas que tornaram possível a execução de programas paralelos. Na seção 2.2 foram apresentadas características do MPI, assim como detalhes sobre as normas MPI-1, onde é a criação de processos ocorre de forma estática, e MPI-2 onde a criação de processos ocorre dinamicamente.

O Capítulo 3 apresenta os dois problemas alvo utilizados neste trabalho. Nele são detalhados cada um dos problemas através de suas principais características. Também são apresentados detalhes sobre as implementações sequenciais e paralelas implementadas com a criação estática de processos MPI-1, e com a criação dinâmica de processos MPI-2.

3 Problemas Alvo

Este capítulo apresenta as descrições e as principais características dos problemas do Jogo da Vida e do *Skyline Matrix Solver*, os quais serão utilizados neste trabalho para avaliar o desempenho e a utilização de recursos computacionais. Estes problemas fazem parte da suíde de problemas *Cowichan Problems*, e são utilizados por diversos autores para avaliar diferentes características de sistemas de computação paralela (WILSON, 1994). Baseado em Lorenzon (2013), descrevemos a implementação sequencial de cada aplicação, bem como os detalhes de implementação das versões paralelas MPI-1 e MPI-2. Por fim são apresentadas as conclusões do capítulo.

3.1 Jogo da Vida

O Jogo da vida (CONWAY, 1970) é um problema proposto pelo matemático Britânico John Horton Conway, além de ser um dos autômatos celulares mais conhecidos. É utilizado para avaliações em sistemas de programação com distribuição de dados e comunicação constante entre os processos, possui carga de trabalho regular, logo o tempo de computação dos processos é semelhante. Seu objetivo é simular a evolução da vida de uma determinada população, e o principal fator se dá ao fato da evolução não ocorrer em razão da ação de jogadores externos, mas por meio de regras determinísticas.

3.1.1 Principais características

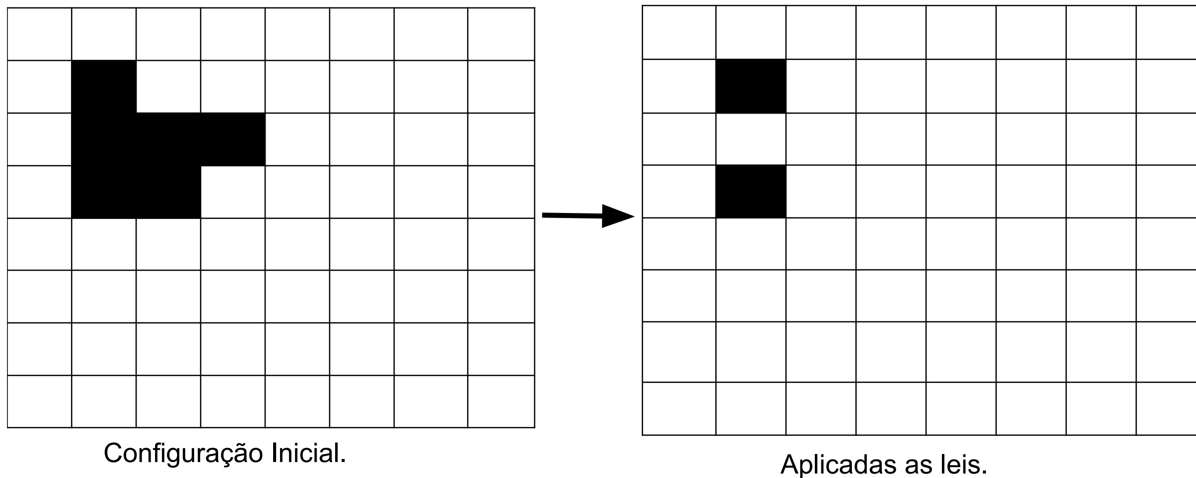
De maneira geral, a sociedade é organizada como um conjunto formado por células que podem assumir dois estados: vivas ou mortas.

As leis genéticas responsáveis pela evolução das células são as seguintes:

1. Uma célula morta somente se torna viva quando possuir exatamente três vizinhos vivos;
2. Células vivas morrem quando possuírem mais de três vizinhos (Superpopulação);
3. Células vivas morrem quando possuírem menos de dois vizinhos (Solidão);
4. Qualquer situação diferente das regras acima as células seguem inalteradas.

A versão sequencial do Jogo da Vida tem seu funcionamento da seguinte forma: Tendo como entrada uma matriz M de tamanho N x N e com a sociedade previamente alocada, é executado um laço de repetição responsável pelo controle da quantidade de

Figura 2 – Jogo da Vida: Aplicação das Leis Genéticas



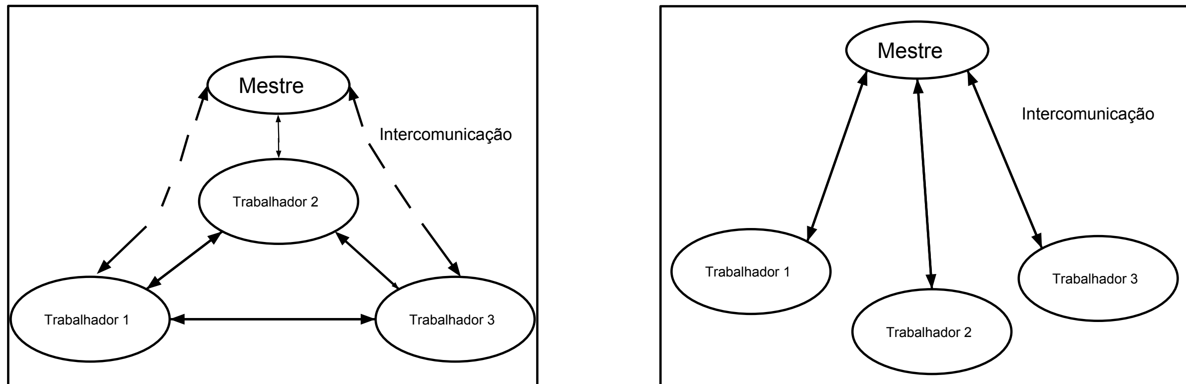
evoluções da sociedade, logo após dois laços aninhados percorrem a matriz aplicando as leis genéticas. A definição do estado de cada célula (viva ou morta) no ciclo seguinte da sociedade, se dá por meio da aplicação das leis genéticas à soma das células presentes em sua vizinhança. A Figura 2 apresenta a evolução da sociedade, onde a esquerda temos uma configuração inicial e a direita o estado da sociedade após a aplicação das leis genéticas. As leis são aplicadas analisando todos os vizinhos de cada célula, ou seja, em todos os sentidos, esquerda, direita, cima, baixo, e diagonais. As células situadas nas extremidades da matriz são a chamada *borda*, onde estas células fazem vizinhança com outras células da borda de lados opostos.

3.1.2 Paralelização Jogo da Vida

A implementação da paralelização do Jogo da Vida foi realizada utilizando o particionamento de dados (LORENZON; CERA; ROSSI, 2012), onde a borda foi incorporada no particionamento inicial dos dados. A versão do Jogo da Vida implementada com MPI-1 segue o modelo SPMD (*Single Program Multiple Data*), onde os processos executam partes distintas do mesmo problema, as quais são selecionadas por meio de identificadores únicos dos processos. Também foi utilizado na implementação o modelo *mestre/trabalhador* que é o mais utilizado por iniciantes em programação paralela, neste modelo há um processo denominado de Mestre e os demais processos são denominados *Trabalhadores*.

O Jogo da Vida conta com dois algoritmos para distribuição de dados: ADIL (Algoritmo de Divisão Igualitária de Linhas), no qual a carga é distribuída de forma igualitária entre os processos ao início da execução, e ADCL (Algoritmo de Divisão por *Chunk* de linhas), no qual é realizada a distribuição do *chunk* de dados sob demanda durante a execução. Foram utilizados dois modos de comunicação entre os processos, no primeiro o processo Mestre se comunica com todos os preocessos Trabalhadores, e

Figura 3 – Modelos de Comunicação Mestre/Trabalhador



os Trabalhadores também se comunicam entre si para que as bordas sejam atualizadas e mantida a coerência dos dados. Já no segundo modo cada processo Trabalhador se comunica apenas com o processo Mestre.

3.1.2.1 Detalhes da Implementação MPI-1

A figura Figura 3 à esquerda ilustra o modelo de comunicação que se mostrou mais eficiente entre os processos MPI-1 (LORENZON, 2013). Neste modelo todos os processos pertencem ao mesmo intracomunicador `MPI_COMM_WORLD`, os processos Trabalhadores comunicam-se com o Mestre e entre si para que a borda seja atualizada.

3.1.2.2 Detalhes da Implementação MPI-2

A versão MPI-2 do Jogo da Vida foi implementada seguindo o modelo MPMD (*Multiple Program Multiple Data*) no qual dois programas diferentes são executados, um executa o processo Mestre, e o outro executa os processos Trabalhadores. Foram implementadas duas versões. Na primeira versão, o processo mestre é lançado ao inicio da aplicação, através do comando `mpiexec`, realiza os ajustes iniciais como particionamento da matriz de entrada e definição dos vizinhos, e após esta etapa cria todos os processos Trabalhadores (filhos) em uma única chamada da função `MPI_Comm_Spawn()`. Já na segunda versão o processo Mestre cria apenas um processo Trabalhador a cada chamada `MPI_Comm_Spawn()`, o Trabalhador após criado, recebe sua carga de trabalho e elimina a necessidade de espera pela criação dos demais Trabalhadores. O modelo de comunicação que se mostrou mais eficiente entre os processos é apresentado na Figura 3 à direita, onde cada trabalhador possui um intercomunicador com o mestre. A segunda versão possui melhor desempenho quando comparada com a primeira em (LORENZON, 2013) para a maioria dos casos, e portanto será utilizada nos casos de teste deste trabalho. A Figura 4 apresenta os pseudocódigos das versões com criação dinâmica de processos implementadas

Figura 4 – Pseudocódigo das versões do Jogo da Vida MPI-2

Versão 1

```
Programa Mestre

1.  MPI_Init(...);
2.  Particiona Matriz;
3.  Define Vizinhos;
4.  MPI_Comm_spawn(NprocFIlhos);
5.  MPI_Send(dados);
6.  MPI_Send(mapeamento processos);
7.  MPI_Recv(dados computados);
8.  MPI_Finalize();

...................................................................

Programa Trabalhador

1.  MPI_Init(...);
2.  MPI_Recv(dados);
3.  MPI_Recv(mapeamento processos)
4.  for(i=1;i<=totalEvolucao;i++)
5.  Computação sob seus dados.
6.  MPI_IRecv(mapeamento processos);
7.  MPI_Sendbordas);
8.  }
9.  MPI_Send(dados coputados);
10. MPI_Finalize();
```

Versão 2

```
Programa Mestre

1.  MPI_Init...);
2.  Particiona Matriz;
3.  Define Vizinhos;
4.  for(p=0;p<NúmeroTrabalhadores;p++){
5.  MPI_Comm_spawn(1);
6.  MPI_Send(dados)
7.  for(i=1;i<=totalEvolucao;i++);
8.  MPI_Irecv(bordas);;
9.  MPI_Send(bordas);
10. }
11. MṔI_Recv(dados computados)
12. MPI_Finalize();
...................................................................
Programa Trabalhador

1.  MPI_Init(...);
2.  MPI_Recv(dados);
3.  for(i=1,i<=totalEvolucao;i++){
4.  computaçao sob seus dados
5.  MPI_IRecv(mapeamento processos);
6.  MPI_Send(bordas) };
7.  MPI_Send(dados computados);
8.  MPI_Finalize();
```

Fonte: Lorenzon (2013)

em Lorenzon (2013)

3.2 Skyline Matrix Solver

Skyline é uma matriz N x N que tem como característica um elevado número de elementos nulos (zeros) localizados em suas extremidades superior e esquerda, além disso não admite elementos nulos em sua diagonal principal (PRESS, 2002). A matriz *Skyline* utilizada neste trabalho possui índices para armazenar a posição onde se encontra o primeiro elemento não nulo, encontrado na linha ou coluna correspondente, melhorando o acesso aos elementos, e dessa forma, não realizando acessos em posições onde existam elementos nulos. A Figura 5 exemplifica este tipo de matriz e os vetores de índice de linha e coluna de elementos não nulos.

Este problema é utilizado para verificar a facilidade de um sistema de programação em representar estruturas matriciais irregulares. Logo possui carga de trabalho irregular e sua implementação requerer comunicação não-local.

Figura 5 – Exemplo de matriz *Skyline*

Índices de Coluna

1	2	3	2	5	4	3

Índices de linha

1
1
2
3
5
5
5

5	0	0	0	0	0	0
4	2	0	3	0	0	0
0	10	6	7	0	0	15
0	0	8	12	0	14	13
0	0	0	0	4	17	18
0	0	0	0	5	3	8
0	0	0	0	6	19	12

3.2.1 Principais características

A aplicação visa a resolução do sistema realizando a decomposição LU *(Lower-Upper)* de uma determinada matriz do tipo *Skyline*. A decomposição LU consiste na divisão de uma matriz de coeficientes M, gerando duas matrizes L *(Lower)* e U *(Upper)*. A Figura 6 ilustra a matriz *Lower* gerada, a qual armazena todos os valores situados abaixo da diagonal principal, inserindo o valor 1 em todos elementos da diagonal principal, que está evidenciado por meio da cor cinza, o restante da matriz é preenchida com o valor 0. Já a matriz *Upper* gerada, a qual é mostrada na Figura 7, armazena os elementos situados acima da diagonal principal, incluindo a mesma que está evidenciada pela cor cinza, o restante da matriz é preenchida com o valor 0. O resultado obtido através do produto entre as duas matrizes (*Lower* e *Upper*) resulta novamente na matriz de coefientes M.

O *Skyline Matrix Solver* caracteriza-se pela grande quantidade de dependências entre os dados, que podem ser dividida em três tipos: Dependência Diagonal, a qual envolve operações com os elementos da diagonal principal; Dependência de Linha, onde envolve operações dos elementos que estão situados acima da diaogonal principal; e Dependência de Coluna, que representa operações envolvendo os elementos situados abaixo da diagonal principal.

O *Skyline Matrix Solver* é resolvido através do método de Doolittle (PRESS, 2002) para realizar a decomposição LU (*Upper*) em matrizes *Skyline*, este método utiliza constantes que representam os índices de linha e coluna de elementos não nulos, dessa forma as posições que contém elementos nulos não acessadas.

3.2.2 Paralelização *Skyline Matrix Solver*

Na paralelização com MPI-1 e MPI-2, foi realizado o particionamento da matriz de entrada M, fazendo uso do modelo de divisão de dados em blocos, o que faz com que a matriz seja mapeada para uma matriz de blocos, o ocasiona em menor quantidade de comunicações, devido às suas características. O modelo de comunicação utilizado foi o modelo Mestre/Trabalhador. Os processos executam duas diferentes funções, onde a função **calculaDiagonal** apresentada através do pseudocódigo na Figura 8, é computada pelo Mestre e contém as operações relacionadas ao cálculo dos blocos diagonais, e a função **calculaLU** apresentada através do pseudocódigo na Figura 9, que é computada pelos Trabalhadores, e contém as operações relacionadas ao cálculo dos blocos situados acima e abaixo dos blocos da diagonal principal. Uma outra função é utilizada para realizar o cálculo do somatório LU apresentada através do pseudocódigo na Figura 10, com a finalidade de reduzir o tempo de computação do bloco diagonal, ela vai realizando o somatório preliminar dos dados utilizados para computar o bloco diagonal. Dessa forma os processos foram classificados como Processos principais, que são aqueles que computaram as operações das funções calculaDiagonal e calculaLU. Um destes processos sempre representará o Mestre, e os demais os Trabalhadores; Já os Processos auxiliares, que correspondem aos processos Trabalhadores, executam a função **calculaSomatórioLU**

Lorenzon (2013) utilizando sistemas de memória distribuída, encontrou alguns desafios como realizar o balanceamento de carga de trabalho e ajustar a comunicação de forma eficiente para reduzir ao máximo o impacto no desempenho da aplicação. O balanceamento de carga de trabalho em uma matriz do tipo *Skyline*, tem impacto direto no tempo de computação, como por exemplo uma das Matrizes (*Upper ou Lower*) possuir maior número de elementos nulos do que a outra, será logicamente computada mais rapidamente. Foi utilizada uma matriz com distribuição uniforme nos casos de teste, ou seja,

Figura 6 – Matriz *Lower*

1	0	0	0	0	0	0
4	1	0	0	0	0	0
0	10	1	0	0	0	0
0	0	8	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	5	1	0
0	0	0	0	6	19	1

Figura 7 – Matriz *Upper*

5	0	0	0	0	0	0
0	2	0	3	0	0	0
0	0	6	7	0	0	15
0	0	0	12	0	14	13
0	0	0	0	4	17	18
0	0	0	0	0	3	8
0	0	0	0	0	0	12

com uma distribuição uniforme de elementos nulos nas matrizes *lower* e *upper*. E assim mantendo de carga entre as matrizes *Lower* e *Upper* equilibrada.

3.2.2.1 Detalhes da Implementação MPI-1

Utilizando MPI-1 a implementação seguiu o modelo SPMD *Single Program Multiple Data*, onde a aplicação em sua inicialização cria todos processos (auxiliares e principais). Todos os processos executarão a função **decomposicaoLU**, que direcionará cada processo à sua função, onde o fluxo é definido através do seu identificador *rank* dentro do comunicador `MPI_COMM_WORLD`. Dessa forma, o processo mestre será representado pelo identificador 0 e executará a função **calculaDiagonal**. Processos trabalhadores executarão a função **calculaLU**, e a função **calculaSomatórioLU** é computada por processsos auxiliares, caso eles existam. Além disso Segue o modelo de comunicação ilustrado pela Figura 3 à direita, onde todas as comunicações passam pelo mestre.

3.2.2.2 Detalhes da Implementação MPI-2

A criação dinâmica de processos foi realizada utilizando MPI-2, por meio de quatro versões, todas elas seguindo o contexto MPMD. A versão onde a criação dinâmica de processos se dá através de um único `MPI_Comm_Spawn()`, e tem seu modelo de comunicação ilustrado pela Figura 3 à direita, onde cada um dos processos possui um intercomunicador com o Mestre, e não há comunicação direta entre processos Trabalhadores foi a que obteve melhor desempenho, e portanto foi utilizada neste trabalho. Nesta versão os blocos possuem tamanho "b", correspondente a ordem da matriz de entrada "m", dividida pelo número de processos filhos (n) + 1, dessa forma obtemos a fórmula para calculo do tamanho do bloco da seguinte forma: $b = (m / (n+1))$

Figura 8 – Pseudocódigo da função CalculaDiagonal

```
1.  void calculaDiagonal (int nProcsum, int nProcLU){{
2.  Carrega matriz e mapeia para blocos;
3.  computa blocoDiagonal;
4.  for(p=0;p<nProcLU;p++){
5.  Envio de blocos para calculaLU;
6.  Recebimento assíncrono dos blocos de calculaLU;
7.  }
8.  do{
9.  Aguarda blocos para computar a diagonal;
10. if (existe dados do somatorio)
11. Recebe o somatorio de calculaSomatorioLU;
12. Computa blocoDiagonal;
13. for(i=0;i<qtdRecvLinha;i++){
14.         Aguarda blocos linha de calculaLU;
15.         Envio do bloco linha para calculaSomatorioLU;
16.         if(existe computacao){
17.               Envios de blocos para calculaLU;
18.               Recebimento assíncrono de calculaLU;
19.        }
20. }
21. for(i=0,i<qtdRecvColuna;i++){
22.        Aguarda blocos coluna de calculaLU;
23.        Envio de bloco coluna para calculaSomatorioLU;
24.        if(existe computacao){
25.             Envios de blocos para calculaLU;
26.             Recebimento assíncrono de calculaLU;
27.         }
28. }
29. }while(houver computacao);
```

Fonte: Lorenzon (2013)

Figura 9 – Pseudocódigo da função CalculaLU

```
1.  void calculaLU(){
2.  Recebimento síncrono dos blocos de calculaDiagonal;
3.  do{
4.          Computa blocoLinha e blocoColuna;
5.          Envios dos blocos linha e coluna atualizados para calculaDiagonal;
6.          if(houver computacao)
7.                 Recebimento assíncrono dos blocos de calculaDiagonal;
8.    }while (houver computacao);
9.  }
```

Fonte: Lorenzon (2013)

3.3 Conclusões do Capítulo

Este capítulo contextualizou os problemas alvo, pertencentes a suíte de problemas *Cowichan Problems* (WILSON, 1994). Inicialmente, na seção 3.1 foram destadas as principais características do Jogo da Vida, alguns detalhes sobre a versões sequencial e paralelas, além de questões de implementação para tornar mais eficiente as versões com MPI-1 MPI-2 desenvolvidas por Lorenzon (2013). A seção 3.2 destacou as principais

Figura 10 – Pseudocódigo da função CalculaSomatorioLU

```
1.  void calculaSomatorioLU(){
2.  do{
3.      Recebimento assíncrono dos blocos de calculaDiagonal;
4.      computa somatorio dos blocos;
5.  }while(houver computacao)
6.  Envio do somatorio dos blocos para calculaDiagonal;
7.  }
```

Fonte: Lorenzon (2013)

características e desafios encontradas no problema *Skyline Matrix Solver*. Também foram apresentados detalhes sobre a paralelização, questões de implementação com MPI-1 e MPI-2, o modo como a matriz de entrada foi particionada, os modelos de programa utilizados.

Após estudarmos as características dos problemas alvo, o Capítulo 4 apresenta a metodologia de trabalho que será utilizada para coletar e analisar as informações de consumo de recursos computacionais na execução destes programas. Adicionalmente, detalharemos o ambiente de testes, as métricas e o método de coleta utilizado.

4 Metodologia

Este capítulo tem como objetivo apresentar a metodologia adotada neste trabalho. A seção 4.1 apresenta as características do ambiente de testes utilizado nas execuções das versões sequenciais e paralelas com MPI-2 e MPI-2. Os métodos e métricas utilizados nas avaliações e análises deste trabalho são apresentadas na seção 4.2 . Já a subseção 4.2.1 apresenta a forma como os testes foram conduzidos.

4.1 Ambiente de Testes

Para realização dos testes e execuções dos problemas alvo o ambiente utilizado possui as seguintes características:

Dois processadores Intel *Xeon* E5-2650 com frequência de 2.80 Ghz compostos por 8 núcleos físicos e 8 núcleos lógicos através do suporte a tecnologia *Hyperthreading*. Cada núcleo possui *cache* L1 de 32 KB, *cache* L2 de 2 MB e *cache* L3 compartilhada de 20 MB. 128 GB de memória RAM DDR3 com frequência de 1.600 MHZ. Sistema Operacional Ubuntu 16.04 LTS, tendo instaladas as versões do compilador *gcc* 4.8.4 e a versão *Open MPI* 1.10.0. Além destes softwares foi utilizado um *script* (NEVES, 2015) baseado na ferramenta *Dstat* que fornece estatísticas de utilização dos recursos computacionais em sistemas Linux, e desta forma os dados de utilização de recursos serão coletados.

4.2 Métodos e Métricas

Para realizar avaliações e análises referentes ao desempenho e ao uso de recursos computacionais das aplicações utilizadas neste trabalho são utilizadas as seguintes métricas:

Média de tempos de execução: Média aritmética dos tempos de execução, cada caso foi executado 10 vezes. Desvio Padrão, variabilidade entre os valores obtidos em relação a média;;

Speedup: A média de tempo de execução sequencial (ms) é dividida pela média de tempo de execução paralela (mp), sendo ambas expressas em segundos e assim resultando no ganho de desempenho obtido na paralelização: $Speedup = \frac{ms}{mp}$;

Uso de Memória RAM: Indica o valor expresso em *MegaBytes* da utilização de memória RAM em cada caso teste;

Interrupções: Indica o valor inteiro correspondente a quantidade de interrupções ocorridas durante a execução de cada caso de teste;

Trocas de contexto: Indica o valor inteiro correspondente a quantidade de trocas de contexto realizadas durante a execução de cada caso de teste;

Uso de *CPU*: Indica o percentual de uso de cada núcleo de processamento utilizado em cada caso de teste.

4.2.1 Condução dos testes

Os testes foram realizados no ambiente apresentado na seção 4.1 por meio de *Scripts Bash*. Estes *scripts* contém os comandos necessários para a compilação dos códigos sequenciais e paralelos. Além disso nele são especificados o número de processos, e outros parâmetros. Estes comandos e parâmetros são necessários para configuração de cada um dos casos de teste dos problemas. Esta abordagem tornou possível a geração das médias de tempo de execução das versões sequenciais e paralelas, além do cálculo dos desvios padrão dos casos de teste utilizados.

O Capítulo 5 apresenta uma análise sobre os dados obtidos nas execuções dos problemas alvo. Nele serão especificados detalhes da configuração de cada um dos casos de teste, analisando e destacando os melhores casos.

5 Análise dos Resultados

Este capítulo apresenta os resultados obtidos nas análises de desempenho e utilização dos recursos do problema do Jogo da Vida na seção 5.1 e do *Skyline Matrix Solver* na seção 5.2. Nele serão apresentadas análises de consumo de memória, trocas de contexto realizadas e interrupções geradas nas execuções. Também é apresentada uma análise de utilização de *CPU*. Ao final são apresentadas as conclusões do capítulo.

5.1 Jogo da Vida

Para o problema do Jogo da Vida foram utilizadas como entrada matrizes de dimensões 8192 x 8192, 12288 x 12288, 16384 x 16384. Foram realizadas 10 execuções para cada uma das configurações de teste com diferentes números de processos, sendo 4, 8, 16 e 32 processos respectivamente. Estes números de processos foram adotados baseados nas características do ambiente de testes. Foram geradas as médias de tempo de execução e desvio padrão para cada caso de teste, destacando que para as amostras de tempo coletadas o desvio padrão está abaixo de 2%.

5.1.1 Análise de Desempenho

Analisando os resultados relativos ao tempo total de execução da aplicação podemos notar que o tempo médio de execução foi reduzido e com isso obtido ganho de desempenho. As figuras Figura 11, Figura 12 e Figura 13 apresentam no eixo x os números de processos utilizados, onde um único processo corresponde a versão sequencial e quatro ou mais processos as versões paralelas. No eixo y principal são apresentadas as médias de tempo de execução em segundos. Já o eixo y secundário mostram-se os *speedups* ou aceleração obtidos pelas versões paralelas.

As paralelizações com MPI-1 e MPI-2 utilizando todos os tamanhos de matrizes de entrada reduziram o tempo médio de execução da aplicação. O melhor desempenho foi obtido pela versão MPI-2 com 32 processos, alcançando *speedups* superiores a 14. Já versão MPI-1 obteve seus maiores *speedups* com 16 processos chegando próximo a 8.

A Tabela 1 apresenta uma tabela com todos os *speedups* obtidos para melhor compreensão, os maiores valores estão destacados. Nela é possível notar que conforme o número de processos aumenta a aplicação obtém melhor desempenho. Com 4 processos para todos tamanhos de entrada o desempenho foi muito próximo para ambas as versões paralelas. Já com os demais números de processos a versão MPI-2 apresenta *speedups*

Figura 11 – Jogo da Vida: Comparativo Sequencial/MPI - Matriz de ordem 8192

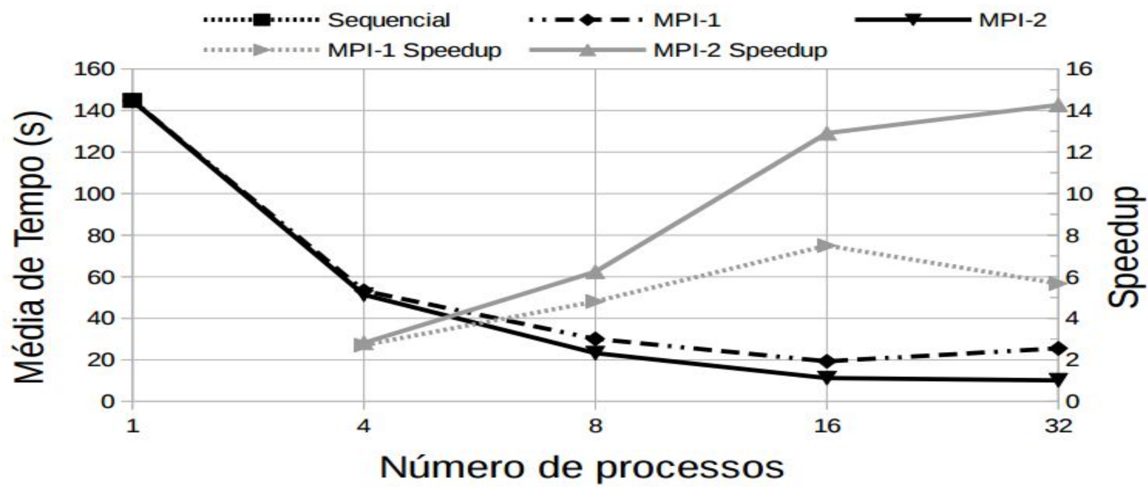


Figura 12 – Jogo da Vida: Comparativo Sequencial/MPI - Matriz de ordem 12288

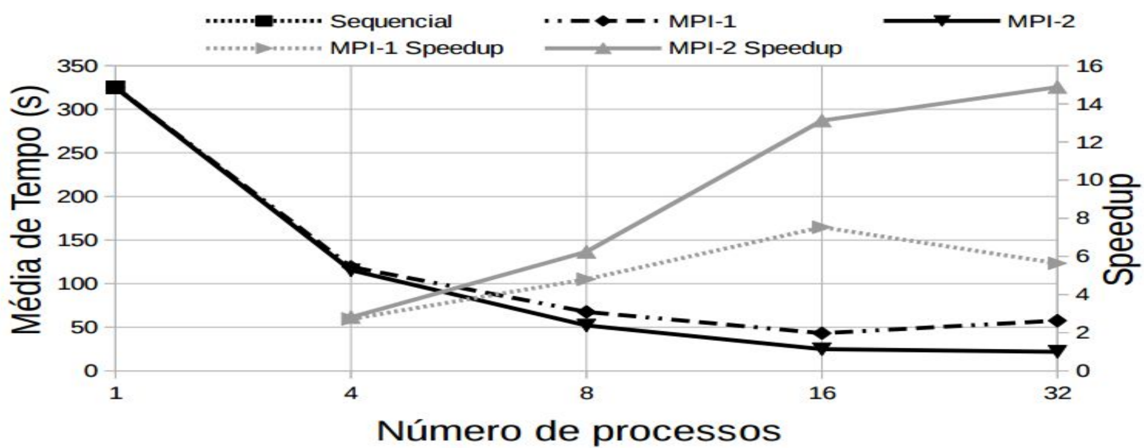
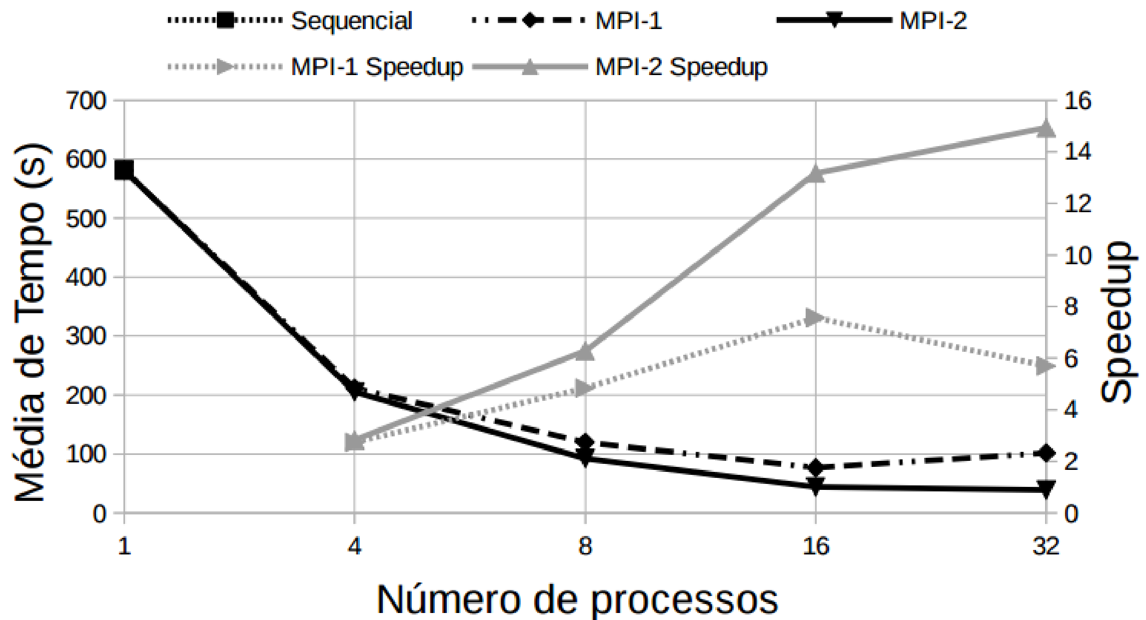


Tabela 1 – Jogo da Vida: *Speedups* das versões paralelas

Processos	MPI-1 8192	MPI-2 8192	MPI-1 12288	MPI-2 12288	MPI-1 16384	MPI-2 16384
4	2,71	2,82	2,72	2,81	2,74	2,83
8	4,81	6,24	4,8	6,24	4,84	6,28
16	**7,5**	12,91	**7,53**	13,12	**7,57**	13,17
32	5,67	**14,27**	5,64	**14,88**	5,7	**14,93**

superiores. Isto se dá ao modelo de comunicação adotado entre os processos que atenuou o impacto da criação dinâmica de processos.

Figura 13 – Jogo da Vida: Comparativo Sequencial/MPI - Matriz de ordem 16384



5.1.2 Análise de Consumo de Memória

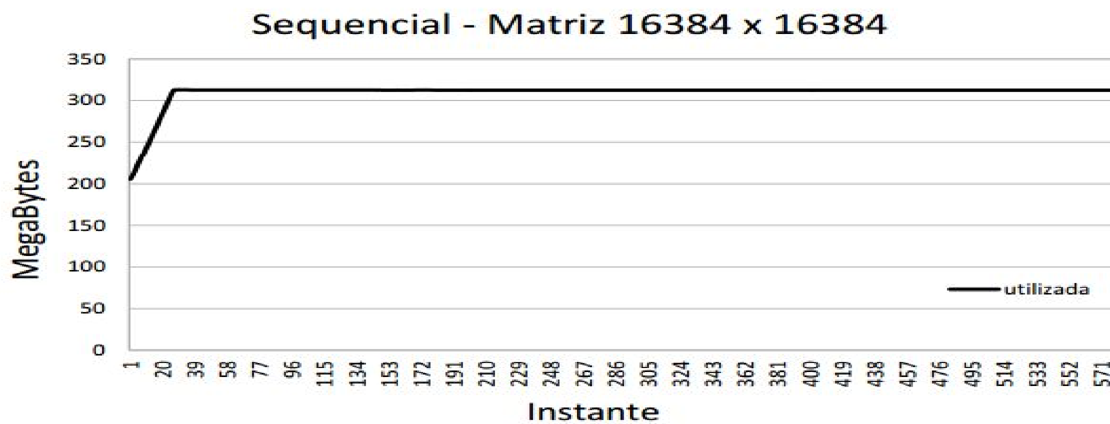
Um comparativo entre as versões sequencial, MPI-1 e MPI-2 com matriz de entrada de dimensões 16384 x 16384 será realizado. Isso se deve ao fato desta configuração apresentar o melhor desempenho dentre os casos de teste utilizados neste trabalho. Os demais gráficos serão adicionados ao Apêndice A pois apresentam comportamento semelhante. As versões paralelas desenvolvidas com MPI-1 e MPI-2 reduziram o tempo médio de execução da aplicação como já foi demonstrado na subseção 5.1.1. A subseção 5.1.2 apresenta o consumo de memória realizado pelas aplicações desenvolvidas por Lorenzon (2013). Serão apresentados os tempos totais de execução de cada um dos casos de teste, e também o percentual referente a cada etapa da execução da aplicação dentro do tempo total.

Analisando os resultados retornados pelo *script* (NEVES, 2015) referentes ao consumo de memória da versão sequencial do Jogo da Vida. É possível notar através do gráfico na Figura 14 o consumo de memória realizado pela execução da versão sequencial do Jogo da Vida. O eixo x apresenta o instante ou segundo em que ocorreu o monitoramento do uso de memória. Já o eixo y apresenta em *MegaBytes* o valor obtido em cada instante ou segundo. Todas as figuras de consumo de memória seguem este mesmo modelo.

O tempo total de execução da versão sequencial foi de **574** segundos. Este gráfico mostra que 4,6% do tempo total de execução é gasto na inicialização da aplicação, sendo este composto pelo carregamento da matriz de entrada nos níveis da memória desde a

leitura no disco, até os registradores do processador. Posterior a esta primeira etapa 94% do tempo de execução é gasto na computação do problema, ou seja, aplicação das leis genéticas por meio dos laços aninhados de repetição *for* responsáveis pela computação. Ao final da execução a aplicação gasta pouco mais 0,6% do tempo total para realizar o retorno e finalização da aplicação. O consumo máximo de memória alcançado foi superior a **310** MB no pico de consumo da aplicação que corresponde ao período mais longo da execução.

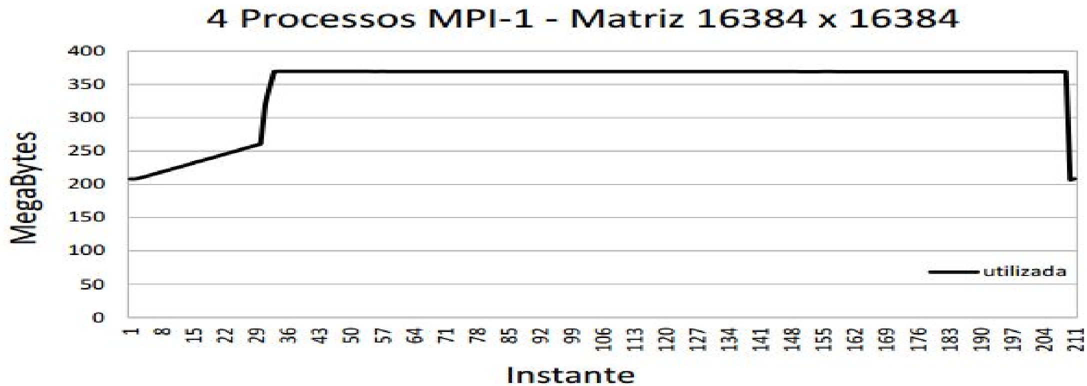
Figura 14 – Jogo da Vida - Consumo de Memória Sequencial



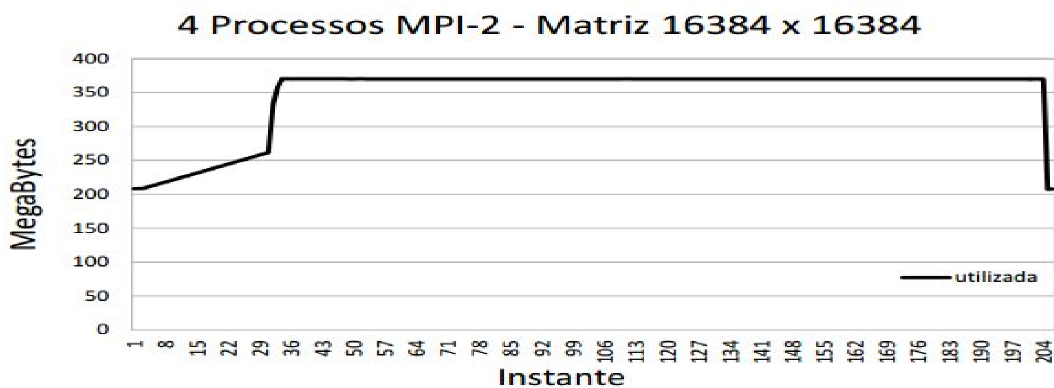
A Figura 15 apresenta o consumo de memória realizado pelas versões MPI-1 na sub-figura *a*, e MPI-2 na sub-figura *b*. A sub-figura *a* mostra que o tempo total de execução foi de **211** segundos para a versão MPI-1. É possível notar que cerca de 15,6% do tempo total de execução foi utilizado na inicialização da aplicação, criação dos processos e carregamento da matriz de entrada nos níveis de memória. Além disso ocorre a divisão e atribuição da carga de trabalho pelo número de processos especificado. Posterior a esta etapa aproximadamente 83,4% do tempo de execução é gasto com a execução dos processos, além das comunicações entre processos que possuem dependências de dados, estas dependências são referentes às atualizações das bordas das sub-matrizes pertencentes a cada processo. Ao final da computação os processos filho retornam os resultados computados para o processo pai, e logo após os processos são liberados da memória e a aplicação finalizada, o que corresponde menos de 1% do tempo total de execução. As versões com 4 processos tiveram aumento percentual no tempo total de execução de mais de 10% nas etapas de inicialização e finalização da aplicação quando comparadas ao sequencial, sendo que este percentual ficou um pouco acima na versão MPI-2 o que se deve ao sobrecusto da criação dinâmica de processos levando cerca de 2% a mais para realizar estas etapas.

Ainda observando a Figura 15 através da sub-figura *b* O tempo total de execução foi de **204** segundos para a versão MPI-2. É possível notar que cerca de 17,4% deste tempo total foi utilizado na inicialização da aplicação, criação dos processos através de uma única chamada `MPI_Comm_Spawn`. Também ocorre o carregamento da matriz de en-

Figura 15 – Jogo da Vida - Consumo de Memória 4 Processos



(a) MPI-1



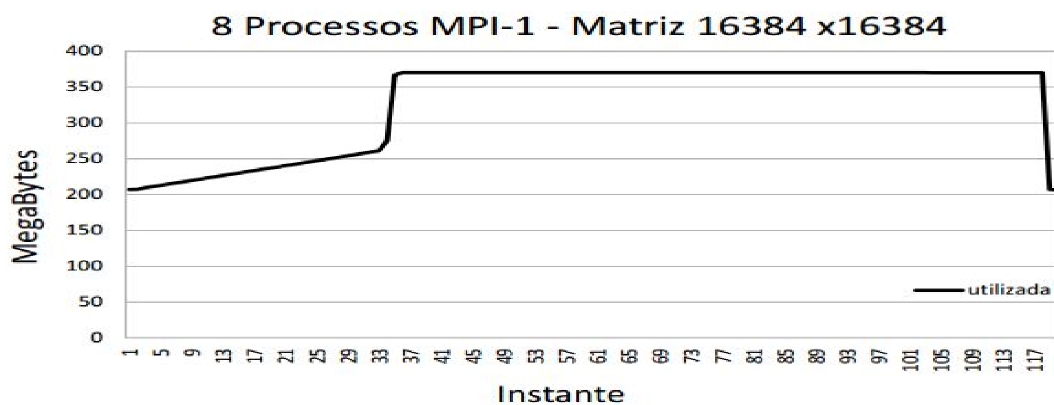
(b) MPI-2

trada nos níveis de memória, além da divisão e atribuição da carga de trabalho pelo respectivo número de processos. Posterior a estas duas etapas aproximadamente 82% do tempo total de execução é gasto com a execução dos processos, além das comunicações entre os processos que possuem dependência entre seus dados. Ao final da computação os processos filhos retornam os resultados computados para o processo pai. Posteriormente na última etapa os processos são liberados da memória e a aplicação finalizada, e isto corresponde a menos de 1% do tempo total de execução. O comportamento das versões MPI-1 e MPI-2 é semelhante devido as implementações que seguem o mesmo molde. A criação dinâmica de processos, a qual ocorre em tempo de execução através da chamada `MPI_Comm_Spawn` gera um sobrecusto para aplicação. Porém o modelo de comunicação adotado na implementação MPI-2 fez com que este sobrecusto fosse atenuado. A comunicação entre processos em ambiente de memória compartilhada se dá através do compartilhamento de um região de memória entre os processos. As versões paralelas com 4 processos apresentaram consumo de memória superior a sequencial, sendo este consumo superior a **360** *MB* tanto para MPI-1 quanto para MPI-2 no pico de consumo da aplicação. As versões paralelas aumentaram o percentual correspondente as etapas de inicialização e finalização da aplicação. Isto se dá ao fato de a aplicação criar um maior número de

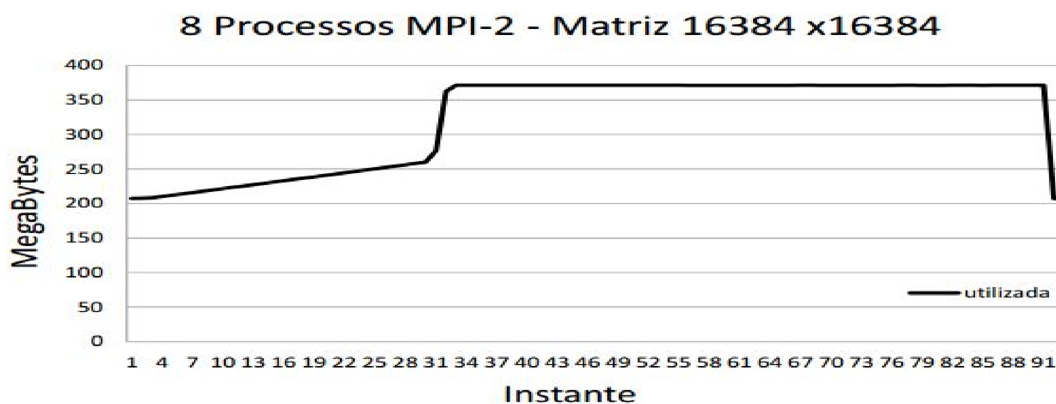
processos ao início, e na etapa de finalização liberar estes processos da memória. Cerca de 16,6% do tempo total de execução é gasto nessas etapas na versão MPI-1 e 18,4% na versão MPI-2, o que demonstra o impacto causado pela criação dinâmica de processos. A versões paralelas aumentaram em mais de 10% as etapas iniciais e finais em comparação a versão sequencial.

A Figura 16 apresenta através da sub-figura *a* o tempo total de **117** segundos com 8 processos MPI-1, sendo cerca de 27,4% deste tempo foi utilizado na inicialização da aplicação. Posterior a estas duas etapas aproximadamente 69,1% do tempo de execução é gasto com a execução dos processos. O final da computação corresponde a cerca de 1,6% do tempo total de execução.

Figura 16 – Jogo da Vida - Consumo de Memória 8 Processos



(a) MPI-1



(b) MPI-2

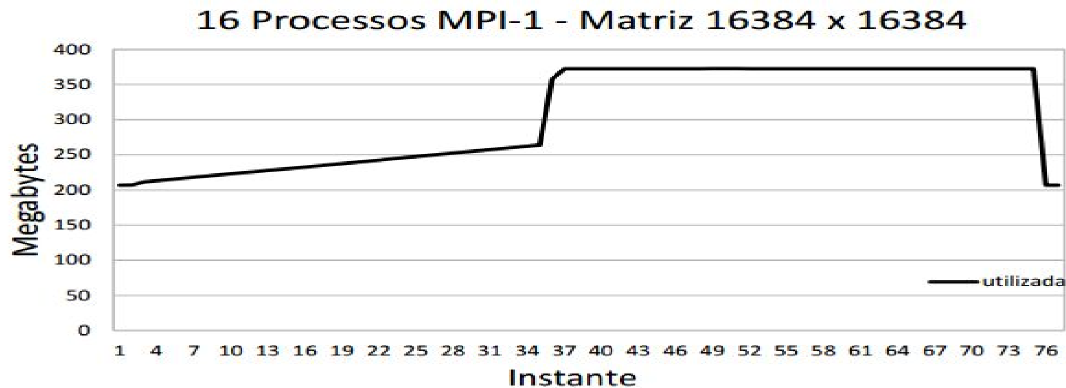
Observando a sub-figura *b* que mostra a execução com 8 processos MPI-2, é possível notar que o tempo total de execução é de **91** segundos. Cerca de 35% do tempo total de execução foi utilizado na inicialização da aplicação e atribuição da carga de trabalho correspondente a cada processo. Posterior a estas duas etapas aproximadamente 62,6% do tempo total de execução é gasto com a execução dos processos. Por fim a aplicação é finalizada o que corresponde a cerca de 2,1% do tempo total de execução. As versões para-

lelas com 8 processos apresentaram consumo de memória superior a sequencial. Consumo este próximo a execução com 4 processos para ambas versões paralelas nos momentos de pico de consumo. Entretanto é possível notar que a versão com 8 processos MPI-2 tem seu pico de consumo por um período de tempo menor que a versão MPI-1. Os percentuais referentes as etapas de inicialização e finalização aumentaram em ambas versões paralelas. As etapas de inicialização e finalização da aplicação tem percentual superior a etapa de execução, levando 30% do tempo total de execução para a versão MPI-1 e 37% para a versão MPI-2, onde a criação dinâmica de processos teve impacto maior que nas execuções com 4 processos anteriores aumentando o percentual destas etapas em 11% para a versão MPI-1, e 17% para a versão MPI-2. Já em comparação a versão sequencial aumentou em 20% para versão com criação estática e em 31% para a versão com criação dinâmica de processos.

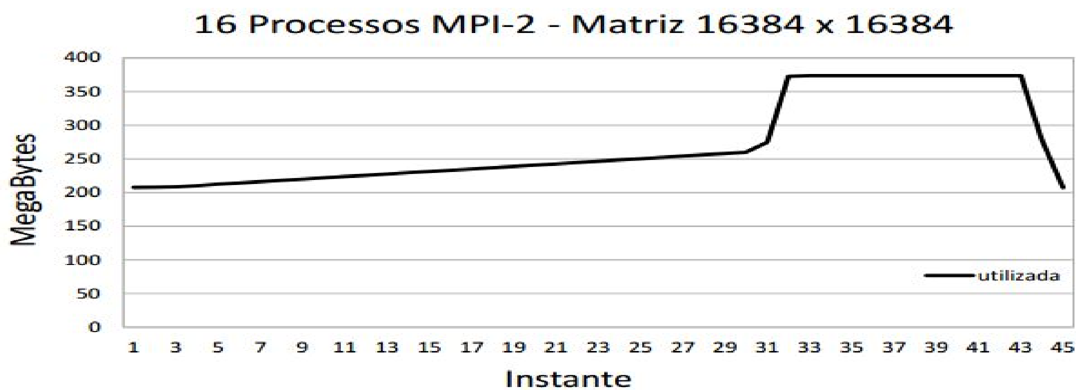
Já na Figura 17 observando a sub-figura *a* que mostra a execução com 16 processos MPI-1, é possível notar que o tempo total de execução foi de **76** segundos. Cerca de 47,9% do tempo total de execução foi utilizado na inicialização da aplicação e atribuição a cada processo sua carga de trabalho. Posterior a estas duas etapas aproximadamente 49,3% do tempo de execução é gasto com a execução dos processos. Ao final a aplicação é finalizada o que corresponde cerca de 2,5% do tempo total de execução.

A sub-figura *b* que mostra a execução com 8 processos MPI-2, mostra que o tempo total de execução foi de **45** segundos, cerca de 73,2% deste tempo foi utilizado na inicialização da aplicação e atribuição da carga de trabalho correspondente a cada processo. Posterior a estas duas etapas aproximadamente 22% do tempo total de execução é gasto com a execução dos processos. Ao final a aplicação finalizada o que corresponde a cerca de 4,4% do tempo total de execução. Neste caso os tempos referentes as execuções dos processos foram inferiores aos tempos de realização das etapas de inicialização e finalização. As versões paralelas com 16 processos apresentaram consumo de memória superior a sequencial. Porém diferente das execuções com 4 e 8 processos, a execução dos processos foi realizada em um período de tempo menor. Ressaltando que a versão paralela MPI-2 consumiu memória por um período de tempo inferior a versão MPI-1. As etapas de inicialização e finalização da aplicação tem percentual superior a etapa de execução, levando 50% do tempo total de execução para a versão MPI-1 e 77,6% para a versão MPI-2, onde a criação dinâmica de processos teve grande impacto nestas etapas. A versão com criação estática com 16 processos aumentou em 20% o percentual destas etapas, já a versão com criação dinâmica aumentou em 40% o percentual referente as mesmas etapas quando comparada a versão com 8 processos. Já em comparação com a versão sequencial, as versões paralelas aumentaram em 45% para MPI-1 e 72% para MPI-2, o que demonstra um grande impacto causado pela criação dinâmica de processos nas etapas de inicialização e finalização da aplicação.

Figura 17 – Jogo da Vida - Consumo de Memória 16 Processos



(a) MPI-1

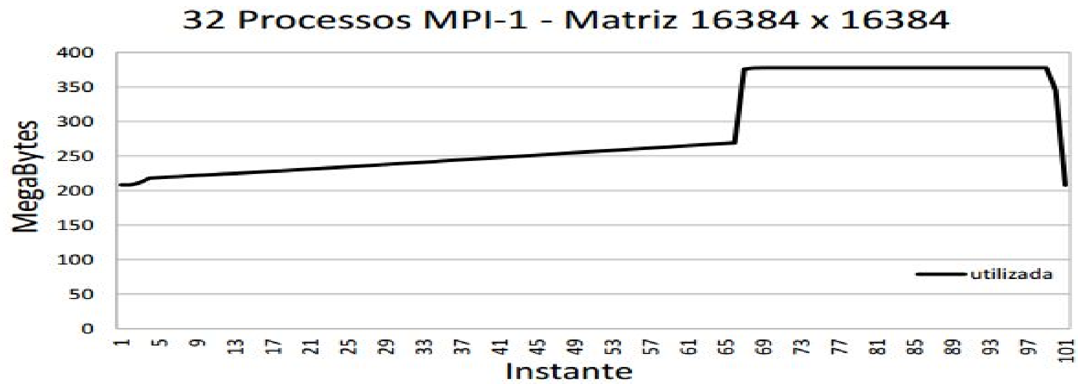


(b) MPI-2

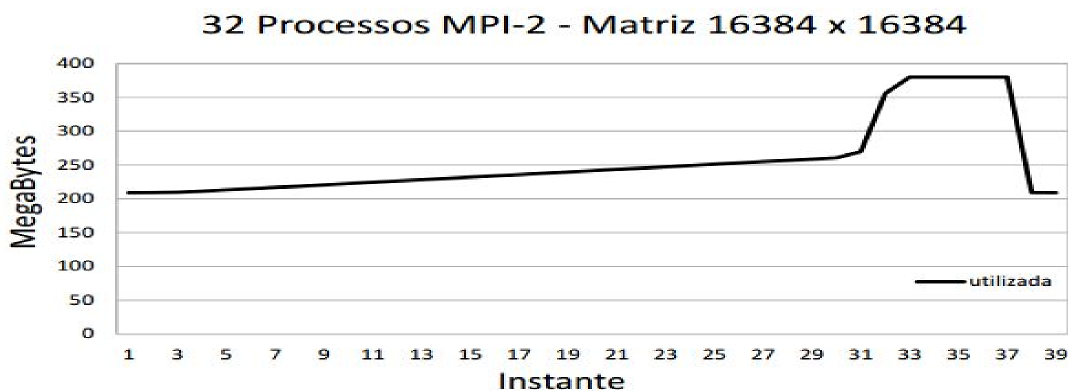
A Figura 18 que mostra a execução com 32 processos MPI-1 através da sub-figura *a*, demonstra que o tempo total de execução foi de **101** segundos. Cerca de 67,8% do tempo total de execução foi utilizado na inicialização da aplicação e atribuição a cada processo de sua carga de trabalho. Posterior a estas duas etapas aproximadamente 30,4% do tempo de execução é gasto com a execução dos processos. Ao final a aplicação é finalizada, o que corresponde cerca de 1,8% do tempo total de execução.

Através da sub-figura *b* que mostra a execução com 32 processos MPI-2, demonstra que o tempo total de execução é de **39** segundos. É também possível notar que cerca de 84,5% do tempo total de execução foi utilizado na inicialização da aplicação e atribuição da carga de trabalho correspondente a cada processo. Posterior a estas duas etapas aproximadamente 10,2% do tempo total de execução é gasto com a execução dos processos. Por fim a aplicação é finalizada, o que corresponde a cerca de 5,1% do tempo total de execução. A versão MPI-2 com 32 processos teve seu pico de consumo de memória por um período inferior a todos os demais casos de teste. As etapas de inicialização e finalização da aplicação tem percentual superior a etapa de execução, levando 69,6% do tempo total de execução para a versão MPI-1 e 89,6% para a versão MPI-2, onde a criação dinâmica

Figura 18 – Jogo da Vida - Consumo de Memória 32 Processos



(a) MPI-1



(b) MPI-2

de processos teve grande impacto nestas etapas. Conforme o número de processos cresce o percentual referente ao tempo em que a inicialização e finalização aumenta em ambas as versões, destacando que o maior aumento foi obtido pela versão com criação dinâmica de processos em todos os casos. As versões paralelas tiveram grande aumento do percentual das etapas de inicialização e finalização da aplicação. Onde a versão com MPI-2 teve aumento de 85% em comparação a versão sequencial.

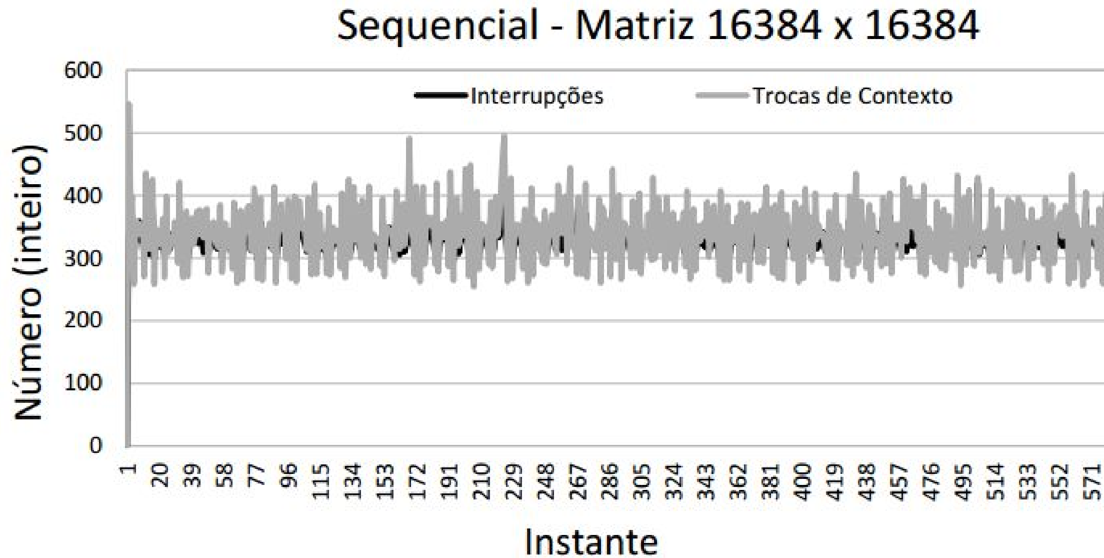
5.1.3 Análise de Trocas de Contexto e Interrupções

As versões paralelas desenvolvidas com MPI-1 e MPI-2 reduziram o tempo médio de execução e aumentaram o consumo de memória nos momentos de pico das aplicação como demonstrado anteriormente na subseção 5.1.2. A Figura 19 mostra as trocas de contexto e as interrupções da versão sequencial do Jogo da Vida. O eixo x apresenta o instante ou segundo em que o valor foi retornado. Já o eixo y apresenta em inteiros o valor obtido em cada instante ou segundo. Todas as figuras de trocas de contexto e interrupções seguem o mesmo modelo.

Ao início da execução da versão sequencial a aplicação chega ao valor máximo de

trocas de contexto próximo a 550 por instante. Posteriormente tem variação entre 250 e 500 trocas de contexto por segundo cada segundo ate a finalização. As interrupções se mantém entre 300 e 400 por instante durante toda a execução.

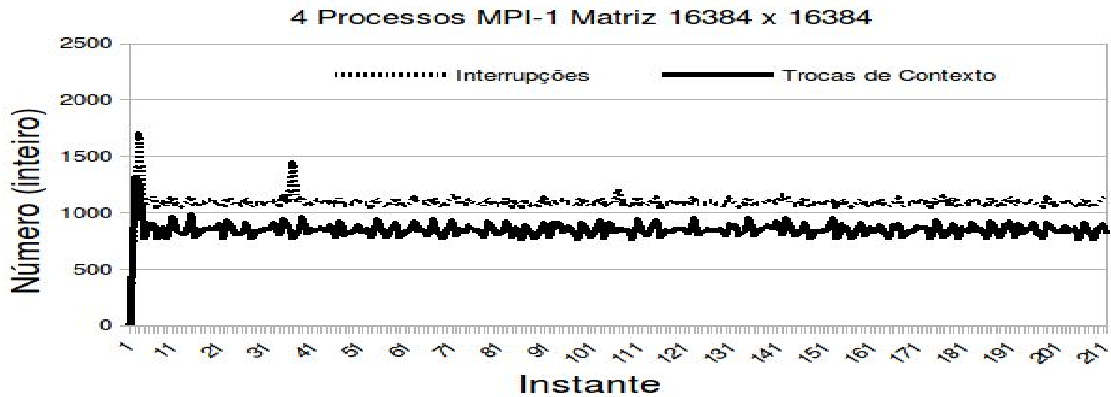
Figura 19 – Jogo da Vida - Trocas de Contexto e Interrupções Sequencial



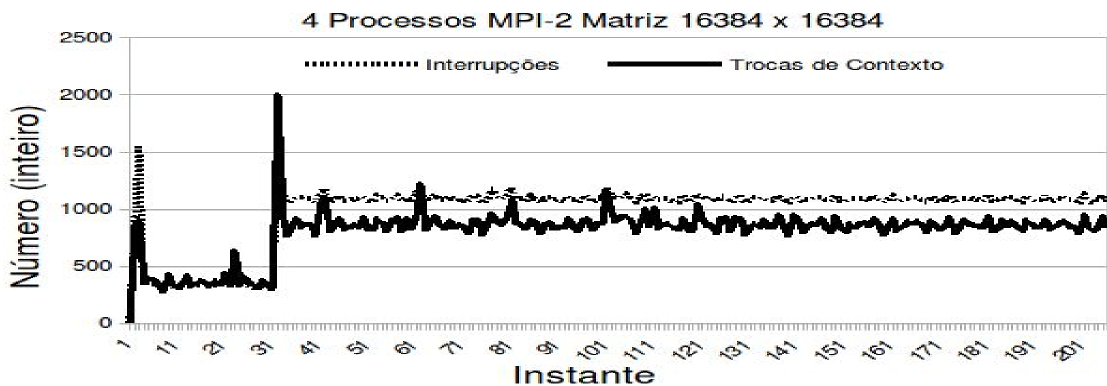
As versões paralelas executadas com 4 processos aumentaram o número de trocas de contexto e interrupções em aproximadamente 3 vezes. A Figura 20 apresenta os valores obtidos durante as execuções. A versão MPI-1 ilustrada pela sub-figura *a* teve seu valor máximo alcançado ao início da execução chegando próximo a 1400 trocas de contexto. Logo após este valor cai para 800 pois os processos já foram criados e ocorre a comunicação entre os processos, mantendo variação entre 800 e 1000 trocas de contexto a cada segundo durante a execução dos processos. As interrupções tem comportamento semelhante alcançando seu valor máximo ao início da execução próximo a 1700 interrupções por segundo. Durante o restante da execução as interrupções variam entre 1000 e 1400 aproximadamente.

A sub-figura *b* mostra os valores obtidos pela versão MPI-2. Ao início da execução as trocas de contexto ficam próximas a 1000 e logo após caem para menos de 500. Mantendo-se com pequena variação entre 800 e 1000, até atingir o valor máximo de 2000 trocas de contexto alguns instantes depois. Posteriormente este valor cai e se mantém em variação entre 700 e 1000 trocas de contexto a cada instante. A versão MPI-1 tem maior número de trocas de contexto e interrupções ao início da execução. Isto se deve ao fato de criar todos os processos da aplicação ao início da execução. Já a versão MPI-2 após criar os processos alcança valores superiores chegando próximo a 2000 pois os processos estão recebendo a sua carga de trabalho. Ambas versões tem comportamento parecido após a criação dos processos se mantendo entre 800 e 1200. A versão que utiliza criação

Figura 20 – Jogo da Vida - Trocas de Contexto e Interrupções 4 processos



(a) MPI-1



(b) MPI-2

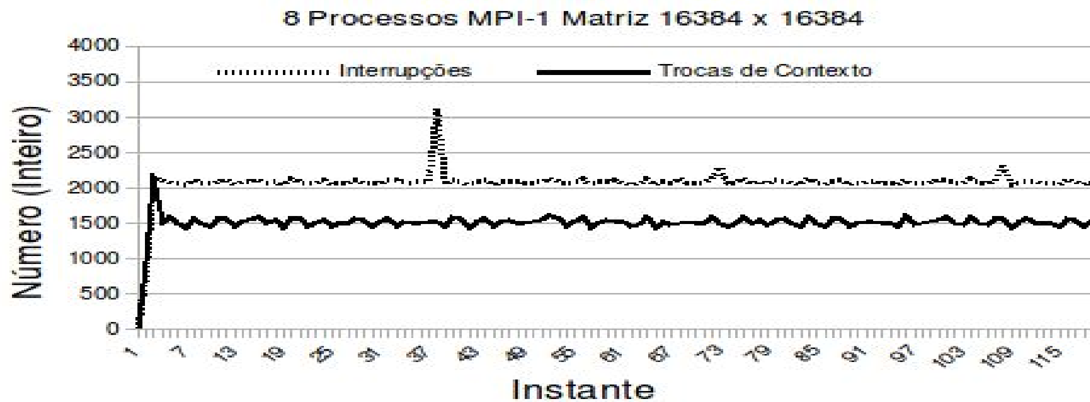
dinâmica de processos impactou em um menor número de trocas de contexto durante o período de criação dos processos quando comparada a versão que utiliza criação estática.

As execuções realizadas utilizando 8 processos são apresentadas na Figura 21. Nela é possível observar através da sub-figura *a*, que a versão MPI-1 tem comportamento semelhante ao apresentando na Figura 20. As trocas de contexto atingem o valor máximo superior a 2000, e posteriormente se mantém variando entre 1500 e 1700 aproximadamente. As interrupções também tem comportamento semelhante variando entre 2000 e 2200, chegando a mais de 3000 quando o valor máximo é atingido.

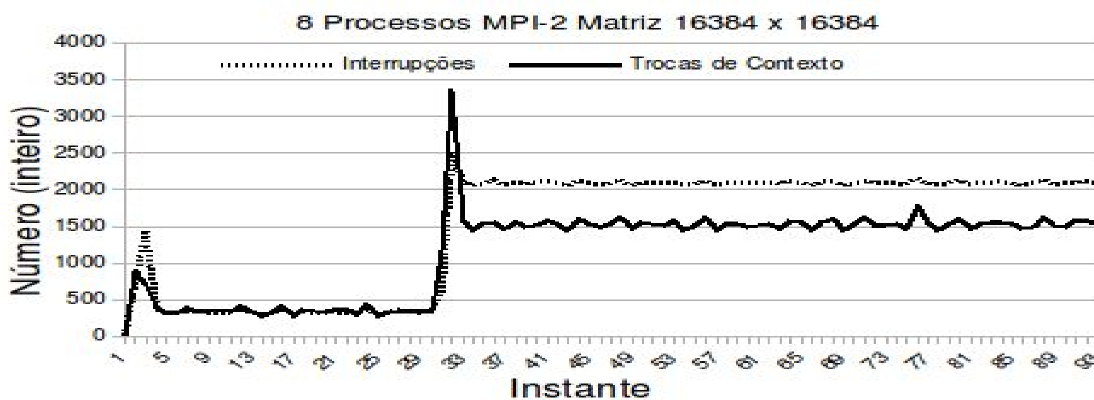
Na sub-figura *b* é apresentada a versão MPI-2. Nela podemos notar que ao início da execução são atingidos valores praticamente iguais aos obtidos na execução com 4 processos, porém quando atinge seu pico chega a quase 3500 trocas de contexto, caindo logo após mantendo variação entre 1500 e 1800. As interrupções tem comportamento semelhante com a execução utilizando 4 processos, porém quando os processos são criados este número é superior chegando próximo a 3000. Logo após este valor cai e se mantém em variação entre 2000 e 2300.

As execuções realizadas utilizando 16 processos são apresentadas na Figura 22.

Figura 21 – Jogo da Vida - Trocas de Contexto e Interrupções 8 processos



(a) MPI-1



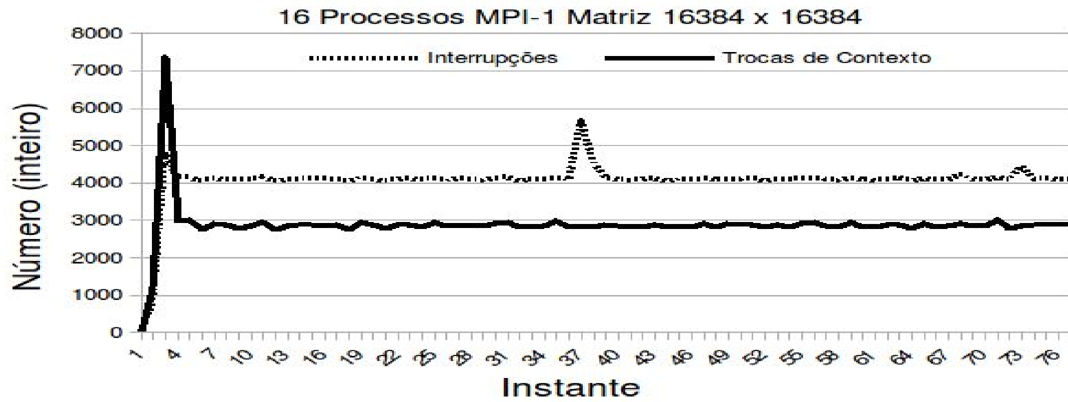
(b) MPI-2

Nela é possível observar através da sub-figura *a*, que a versão MPI-1 tem comportamento semelhante ao apresentando as execuções com menor número de processos. As trocas de contexto atingem o valor máximo superior a 7000, e posteriormente se mantém próximo de 3000. As interrupções também tem comportamento semelhante permanecendo próxima a 4000, chegando a quase 6000 quando atingiu seu valor máximo.

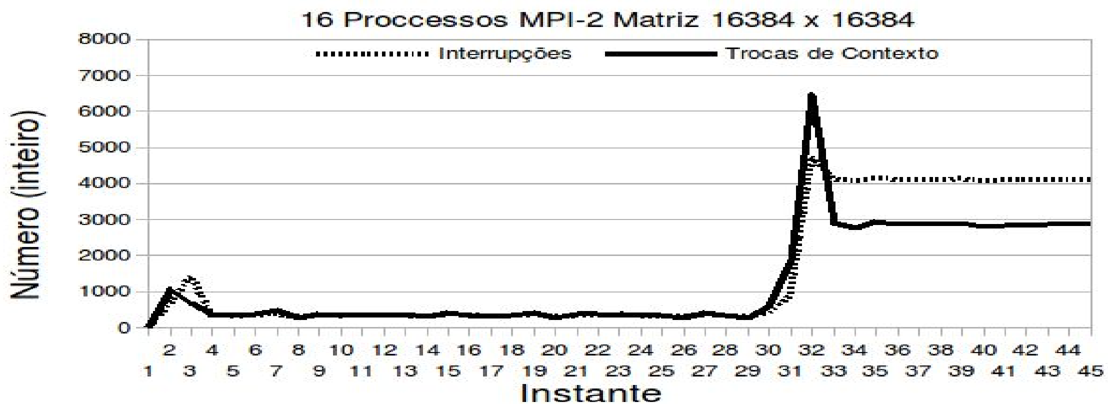
Na sub-figura *b* é apresentada a versão MPI-2. Nela podemos notar que ao início da execução atinge valores praticamente iguais aos obtidos na execução com menor número de processos, porém quando atinge seu pico chega a mais de 6000 trocas de contexto e se mantém próximo a 3000. As interrupções tem comportamento semelhante com a execução com menor número de processos, porém quando os processos são criados este número é superior chegando próximo a 5000. Logo após este valor cai e se mantém próximo a 4000.

As execuções realizadas utilizando 32 processos são apresentadas na Figura 23. Nela é possível observar através da sub-figura *a*, que a versão MPI-1 tem comportamento diferente das execuções com menor número de processos. Este comportamento se dá ao uso da tecnologia *Hyperthreading*. Este recurso faz com que dois processos executem concorrentemente em cada núcleo, gerando maior número de trocas de contexto. As

Figura 22 – Jogo da Vida - Trocas de Contexto e Interrupções 16 processos



(a) MPI-1



(b) MPI-2

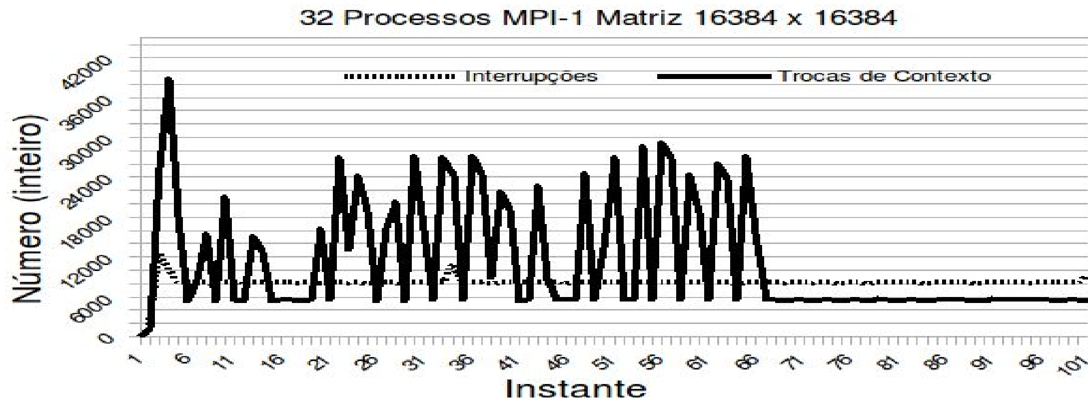
trocas de contexto atingem o valor máximo próximo a 40000, e posteriormente variando entre 5000 e 30000. As interrupções tem comportamento semelhante as execuções com menor número de processos, mas com valor mais elevado permanecendo próxima a 10000, chegando a quase 14000 quando atingiu seu valor máximo.

Na sub-figura *b* é apresentada a versão MPI-2. Nela podemos notar que ao início da execução atinge valores praticamente iguais aos obtidos nas execuções com menor número de processos, porém quando atinge seu pico chega a mais a 12000 trocas de contexto e se após se mantém próximo a 6000. As interrupções tem comportamento semelhante com as execuções com menor número de processos, seu maior valor é próximo a 8000 mil e se mantém até a finalização.

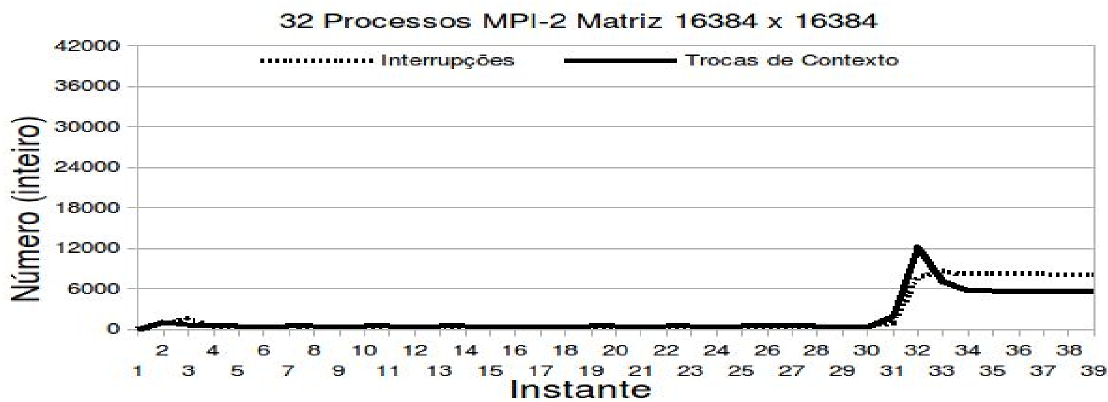
5.1.4 Análise de Utilização de *CPU*

Nas execuções sequenciais e paralelas do problema do Jogo da Vida, os núcleos de processamento que receberam carga de trabalho fizeram uso de 100% durante todo tempo. Na versão sequencial um núcleo de processamento recebeu o problema e durante toda a execução teve 100% de uso. Já nas versões paralelas, tanto para MPI-1 quanto

Figura 23 – Jogo da Vida - Trocas de Contexto e Interrupções 32 processos



(a) MPI-1



(b) MPI-2

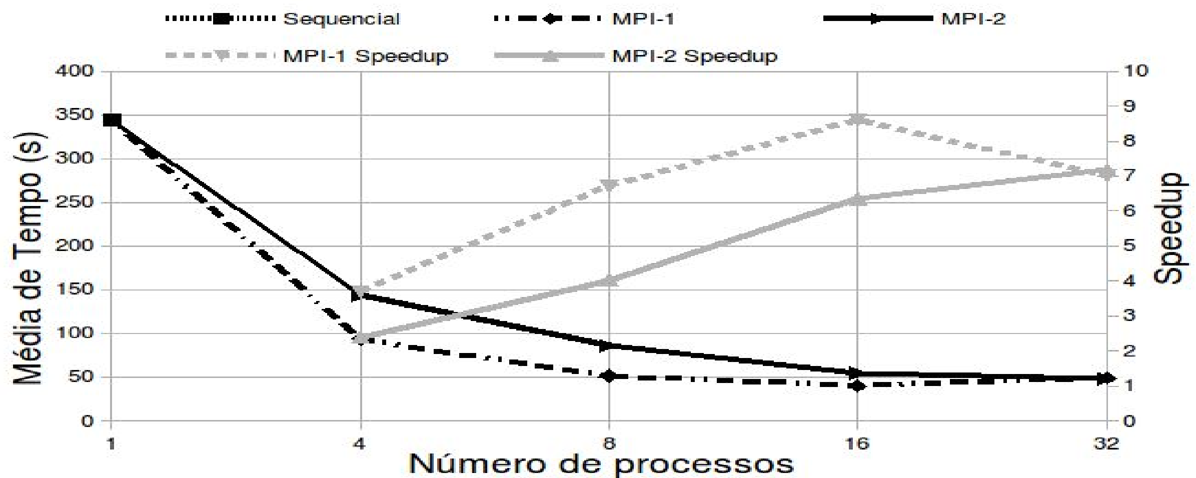
para MPI-2, durante as execuções com 4 processos o ambiente de testes teve 4 núcleos com 100% de uso durante toda execução. Com 8 processos executados teve 8 núcleos utilizados 100% também durante todo o tempo, e assim sucessivamente. Na execução com 32 processos utilizou a tecnologia *Hyperthreading* onde dois processos são executados concorrentemente em cada núcleo, tendo oscilações durante toda a execução.

Dessa forma foram apresentadas as análises de desempenho, consumo de memória, trocas de contexto, interrupções e uso de *CPU* para o Jogo da Vida. A seção 5.2 apresenta as mesmas análises para o problema do *Skyline Matrix Solver*.

5.2 Skyline Matrix Solver

Para o problema do *Skyline Matrix Solver* foram utilizadas como entrada matrizes com tamanho 4096 x 4096, 5120 x 5120. Foram realizadas 10 execuções para cada uma das configurações de teste, onde os números de processos foi de 4, 8, 16 e 32 respectivamente para as implementações paralelas com MPI-1 e MPI-2. Estes números de processos foram adotados com base nas características do ambiente de testes, assim como no problema

Figura 24 – *Skyline Matrix Solver*: Comparativo Sequencial/MPI - Matriz de ordem 4096



do Jogo da Vida. Também foram geradas as médias do tempo total de execução para cada um dos casos de teste. Serão apresentados comparativos entre o tempo médio e o ganho de desempenho (*speedup*) obtido nas execuções sequencial e paralelas. Também são apresentados comparativos entre o consumo de memória, trocas de contexto e interrupções entre as versões sequencial e paralelas.

5.2.1 Análise de Desempenho

Com ambos tamanhos de matriz de entrada as versões paralelas reduziram o tempo médio de execução da aplicação como podemos notar nas figuras 24 e 25. Utilizando matriz de entrada 4096 x 4096 a versão MPI-1 obteve os melhores resultados comparando aos resultados da versão MPI-2. O maior *speedup* foi obtido utilizando 16 processos chegando próximo a 9, mas ficando abaixo do ideal. Nas execuções com 32 processos MPI-1 o *speedup* cai em relação a execução com 16 processos. Isto se deve a utilização da tecnologia *hyperthreading*, e causa impacto no ganho de desempenho de um caso de teste para o outro. Já a versão MPI-2 teve seu maior ganho de desempenho utilizando 32 processos chegando a um *speedup* próximo a 7. Nesta versão, a qual utiliza a criação dinâmica de processos, o uso da tecnologia *hyperthreading* não causa impacto no desempenho da aplicação obtendo o maior ganho entre os casos de teste nesta versão.

A Tabela 2 apresenta uma tabela com todos os *speedups* para melhor compreensão dos dados obtidos. Os maiores valores obtidos em cada versão estão destacados.

5.2.2 Análise de Consumo de memória

As versões paralelas desenvolvidas com MPI-1 e MPI-2 reduziram o tempo médio de execução da aplicação como já foi demonstrado na subseção 5.2.1. A Figura 26

Figura 25 – *Skyline Matrix Solver:* Comparativo Sequencial/MPI - Matriz de ordem 5120

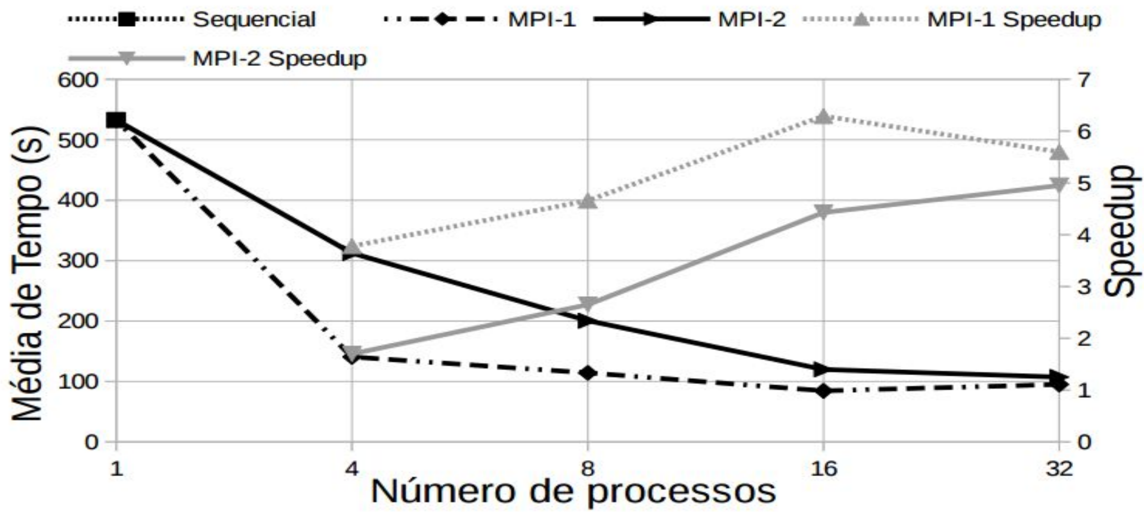
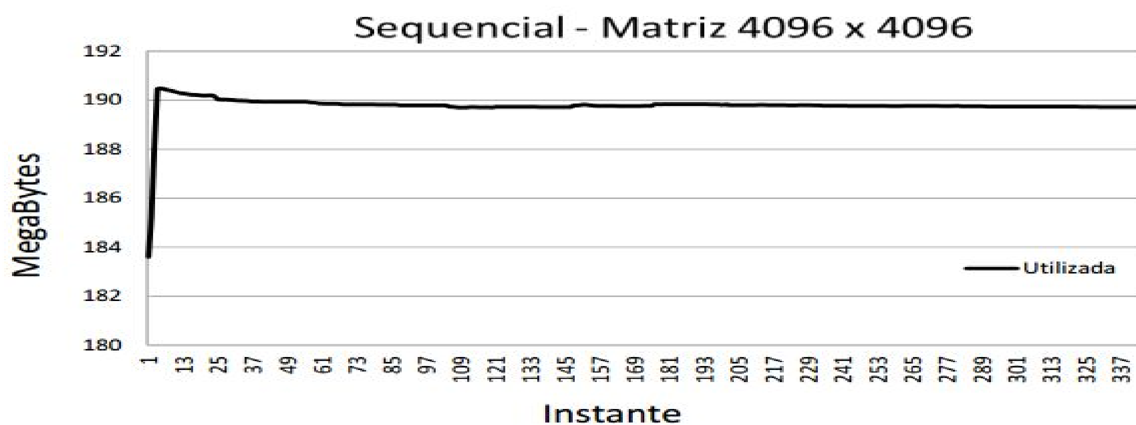


Tabela 2 – *Skyline Matrix Solver*: *Speedups* das versões paralelas

Processos	MPI-1 4096	MPI-2 4096	MPI-1 5120	MPI-2 5120
4	3.69	2.4	3.78	1.7
8	6.72	4.02	4.66	2.65
16	**8.61**	6.35	**6.29**	4.43
32	7.01	**7.17**	5,6	**4.95**

Figura 26 – *Skyline Matrix Solver* - Consumo de Memória Sequencial



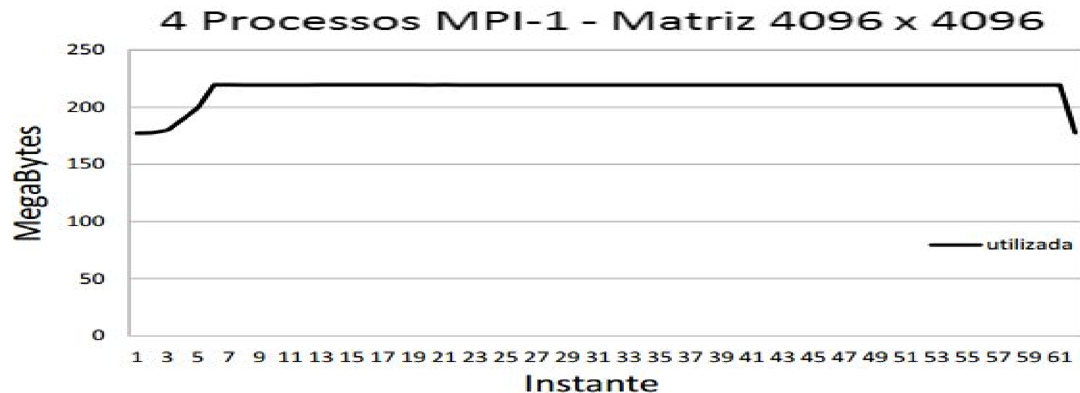
apresenta o consumo de memória realizado pela versão sequencial, nela é possível notar o tempo total de execução da aplicação de **337** segundos. Sendo cerca de 1,15% do tempo total gasto na inicialização do problema, onde a matriz de entrada e carregada na memória. Após os ajustes iniciais 98,8% do tempo é gasto na execução do processo, onde são realizados as operações responsáveis pelos cálculos da decomposição LU. Ao final menos de 1% do tempo foi utilizado para finalização da aplicação.

A versão sequencial teve um crescimento durante poucos segundos ao início da execução chegando a um consumo de memória pouco acima de 190 *MB*. E se mantém até o final da execução em um pouco abaixo de 190 *MB*. A seguir serão apresentados os consumos realizados pelas versões paralelas.

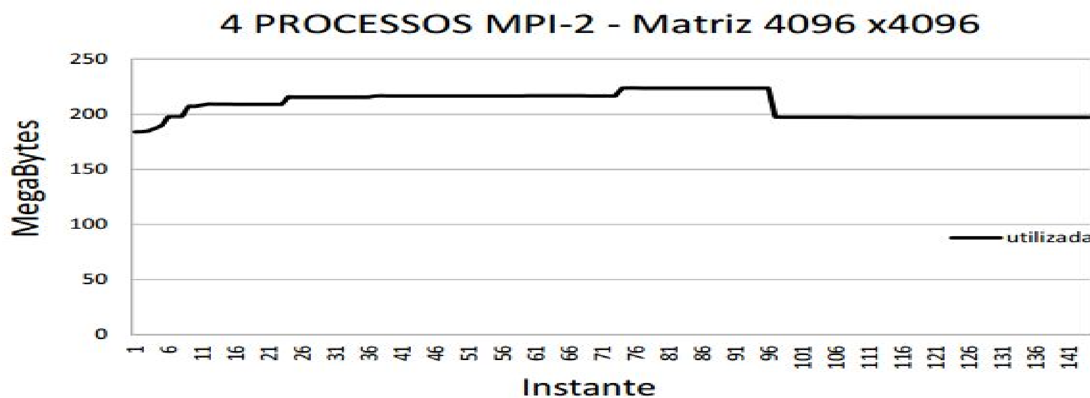
A Figura 27 apresenta o consumo de memória realizado pelas versões paralelas com 4 processos MPI-1 e MPI-2. A sub-figura *a* apresenta o consumo de memória da versão MPI-1. Nela é possível notar que o tempo total de execução foi de **61** segundos. Cerca de 6,4% do tempo é gasto na inicialização do problema. Esta etapa inicial corresponde ao carregamento da matriz de entrada na memória, além da divisão da carga de trabalho pelo respectivo número de processos. Também são criados os processos e atribuída sua respectiva carga de trabalho. Além disso são definidos os vetores auxiliares que armazenam as primeiras posições de linha e coluna válidos, os quais são utilizados para evitar acessos a posições desnecessárias pelo método de *Doolittle*. Logo após aproximadamente 87% é gasto na execução dos processos, onde são realizadas as operações responsáveis pelos cálculos da decomposição *Lower-Upper*. Nela ocorrem os cálculos de linha e coluna e realizadas as comunicações coletivas entre o processo mestre e os processos trabalhadores, através das primitivas `MPI_Bcast()` e `MPI_Reduce()`. Estas comunicações ocorrem para manter a consistência dos dados que possuem dependências entre processos. Ao final 3,3% do tempo total é gasto na finalização da aplicação onde os processos são liberados da memória. O valor de consumo máximo atingido ficou entre **200** e **250** *MB* durante a execução dos processos.

Ainda observando a Figura 27 a sub-figura *b* mostra o consumo de memória da versão MPI-2. Nela é possível notar que o tempo total de execução foi de **141** segundos. Cerca de 5,5% do tempo é gasto na inicialização da aplicação onde ocorre o primeiro aumento de consumo da aplicação chegando próximo a **200** MB. Após esta etapa todos os processos são criados em uma única chamada `MPI_Comm_Spawn`. Conforme os processos recebem sua carga de trabalho começam a ser executados elevando o consumo gradativamente. A execução total dos processos leva aproximadamente 93,7% do tempo total para ser realizada. Ao final 0,8% do tempo é gasto na finalização da aplicação onde os processos ativos são liberados da memória. Esta versão chegou a um consumo máximo entre **200** e **250** *MB* no momento em que todos os processos estão em execução. Desta forma, este o momento de pico de consumo da aplicação. Através Figura 27 notamos que a versão paralela MPI-1 apesar de seu tempo médio de execução menor, tem seu momento de pico de consumo por um período de tempo maior que a versão paralela MPI-2. As versões paralelas tiveram aumento percentual na etapa de inicialização da aplicação. A versão com criação estática de processos aumentou em aproximadamente 6,3% na inicialização e de 2,3% na finalização. Já a versão com criação dinâmica de processos aumentou em aproximadamente 5,3% na inicialização e teve pequena queda de 0,2% na finalização da aplicação.

Figura 27 – *Skyline Matrix Solver* - Consumo de Memória 4 Processos



(a) MPI-1

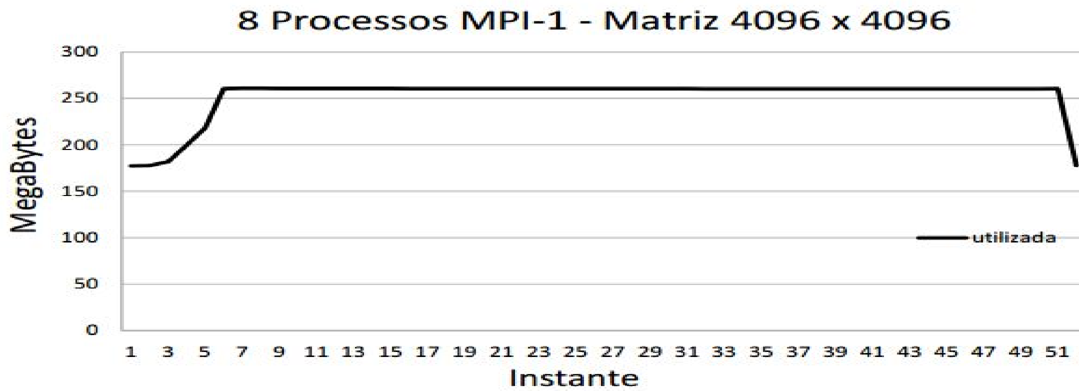


(b) MPI-2

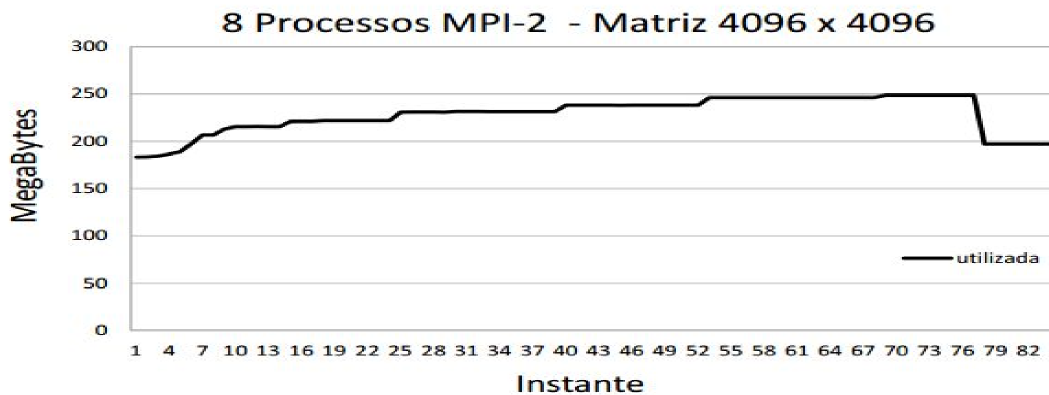
A Figura 28 mostra a execução com 8 processos MPI-1 na sub-figura *a*. Nela é possível notar que o tempo total de execução foi de **51** segundos. Cerca de 9,6% do tempo é gasto na inicialização do problema. Logo após aproximadamente 86,5% é gasto na execução dos processos. Ao final 3,8% do tempo total é gasto na finalização da aplicação. O valor máximo atingido superior **250** *MB* durante a execução dos processos.

A sub-figura *b* mostra a execução com 8 processos MPI-2. Nela é possível notar que o tempo total de execução foi de **82**. Deste tempo total cerca de 8,3% do tempo é gasto na inicialização da aplicação. A execução dos processos leva cerca de 89,2% do tempo total para ser realizada. Ao final 2,3% do tempo é gasto na finalização da aplicação. Esta versão chegou a um consumo máximo próximo a **250** *MB* no momento em que todos os processos estão sendo executados. O comportamento é semelhante a execução com 4 processos para ambas as versões, porém o consumo de memória cresceu na execução com 8 processos. As execuções com 8 processos aumentaram os percentuais das etapas de inicialização e finalização da aplicação quando comparadas a execução com 4 processos. Na versão MPI-1 teve aumento 3,2% na inicialização e de 0,5% na finalização, já a versão MPI-2 de 1,8% na inicialização e de 1,5% na finalização.

Figura 28 – *Skyline Matrix Solver* - Consumo de Memória 8 Processos



(a) MPI-1

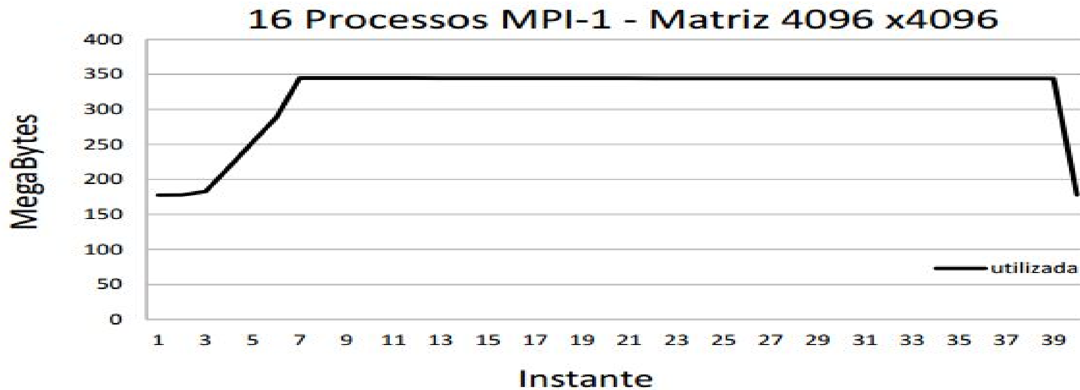


(b) MPI-2

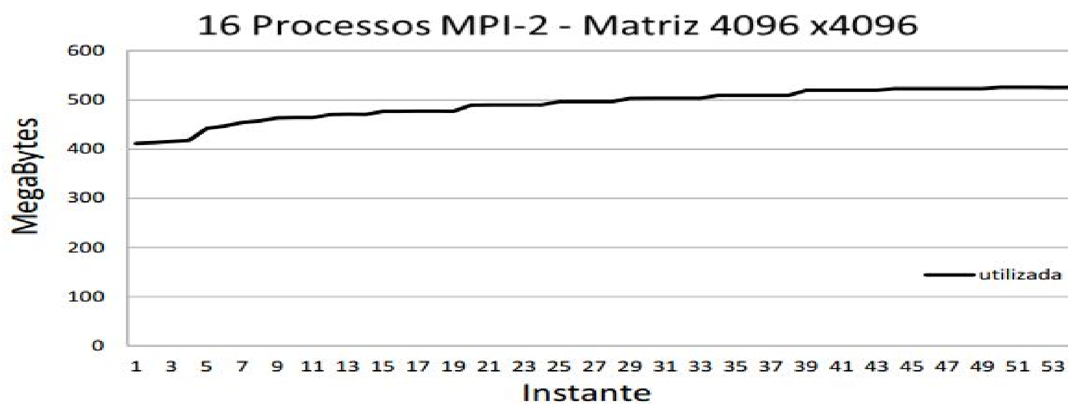
A Figura 29 mostra a execução com 16 processos MPI-1 na sub-figura *a*. Nela é possível notar que o tempo total de execução foi de **39** segundos. Cerca de 15% do tempo é gasto na inicialização do problema. Logo após aproximadamente 80% do tempo total é gasto na execução dos processos. Ao final aproximadamente 5% do tempo total é gasto na finalização da aplicação. O valor máximo foi atingido durante a execução de todos os processos, chegando próximo a **350** *MB*.

A sub-figura *b* mostra a execução com 16 processos MPI-2. Nela é possível notar que o tempo total de execução foi de **53**. Deste tempo total cerca de 12,9% do tempo é gasto na inicialização da aplicação. A execução dos processos leva aproximadamente 83,3% do tempo total para ser executada. Ao final 3,7% do tempo é gasto na finalização da aplicação onde o(s) processos(s) ainda ativos são liberados da memória. Esta versão chegou a um consumo máximo superior a **500** *MB* no momento em que todos os processos estão sendo executados. O consumo de memória de ambas versões utilizando 16 processos cresceu em relação as versões sequencial e paralelas com 4 e 8 processos. Destacando que a versão com 16 processos MPI-1 consumiu mais memória que a versão MPI-2, diferente dos casos de teste anteriores. Comparando com as execuções com 8 processos obteve

Figura 29 – *Skyline Matrix Solver* - Consumo de Memória 16 Processos



(a) MPI-1



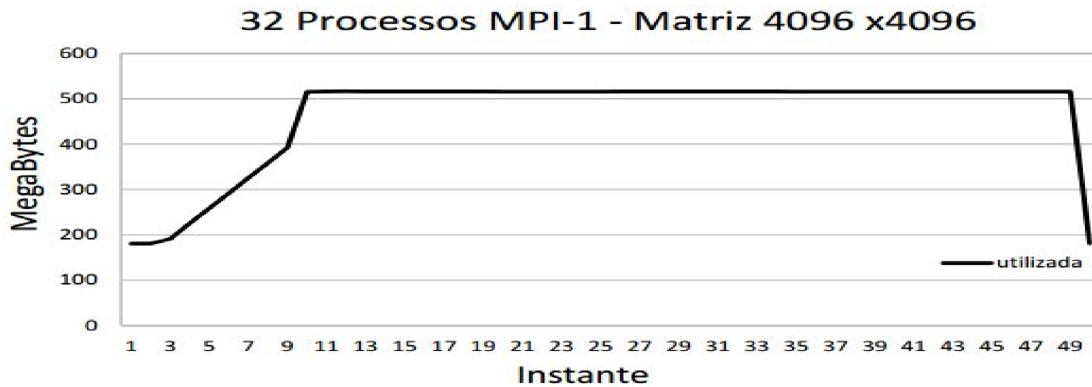
(b) MPI-2

aumento de 5,4% na versão MPI-1 com 16 processos nas etapas de inicialização e de 1,2% na finalização. Já a versão MPI-2 teve aumento de 3,9% na inicialização e de 1,4% com 16 processos.

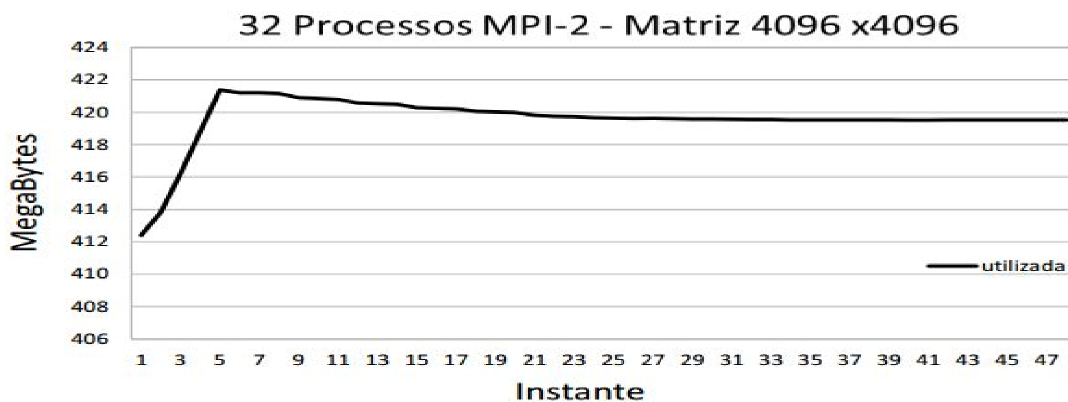
A Figura 30 mostra a execução com 32 processos MPI-1 na sub-figura *a*. Nela é possível notar que o tempo total de execução foi de **49** segundos. Cerca de 12% do tempo é gasto na inicialização do problema. Logo após aproximadamente 84% é gasto na execução dos processos. Ao final aproximadamente 4% do tempo total é gasto na finalização da aplicação. O valor máximo foi atingido durante a execução de todos os processos, chegando próximo a mais de **500** *MB*.

A sub-figura *b* mostra a execução com 32 processos MPI-2. Nela é possível notar que o tempo total de execução foi de **47**. Deste tempo total cerca de 16% do tempo é gasto na inicialização da aplicação. A execução dos processos leva cerca de 81% do tempo total para ser executada. Ao final aproximadamente 3% do tempo é gasto na finalização da aplicação. Esta versão chegou a um consumo máximo superior a **400** *MB* no momento em que todos os processos estão sendo executados. Assim como nas outras execuções com exceção da que utiliza 16 processos, a versão MPI-2 realizou consumo de

Figura 30 – *Skyline Matrix Solver* - Consumo de Memória 32 Processos



(a) MPI-1

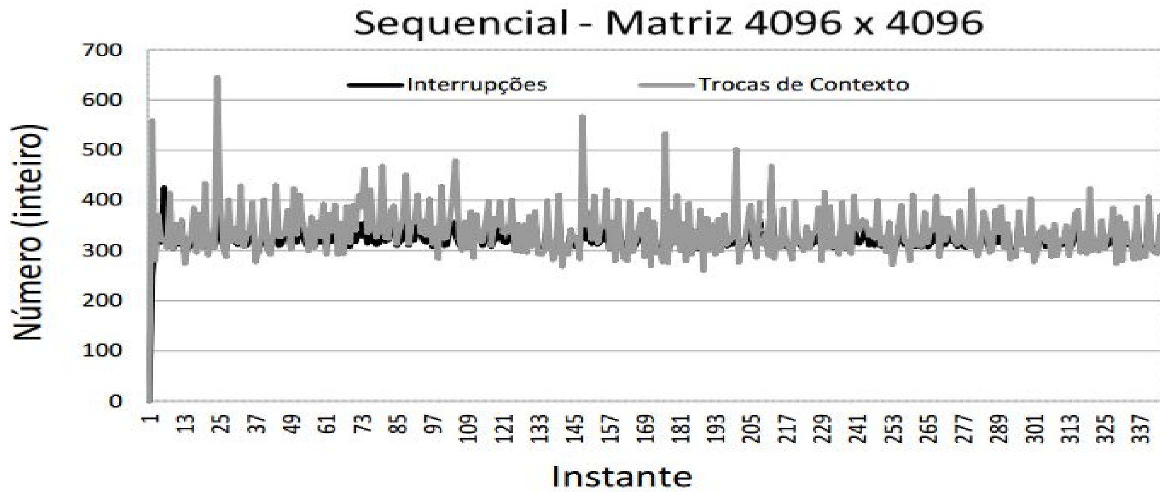


(b) MPI-2

memória inferior a versão MPI-1. Comparando com as execuções com 16 processos, a versão MPI-1 teve redução do percentual referente ao tempo de inicialização e aumento no percentual referente a finalização de menos de 1%. Já a versão MPI-2 aumentou em 3,1% o percentual referente ao tempo de inicialização e reduziu o percentual de finalização em 0,7% o tempo de finalização.

A comunicação irregular do problema *Skyline Matrix Solver* impactou em um aumento no consumo de memória, chegando ao uso de mais que o dobro de memória, conforme aumentou-se o número de processos. Adicionalmente como os *speedups* desta aplicação foram menores que os obtidos com o Jogo da Vida, o percentual de uso de memória para a execução da aplicação também foi superior. Comparando o percentual de tempo de inicialização, com 4 e 8 processos MPI-2 utilizou 1% a menos que MPI-1, sendo este valor ampliado para 3% com 16 processos. Já para 32, este cenário se inverte e MPI-2 precisa de 4% a mais de tempo inicializando

Figura 31 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções Sequencial



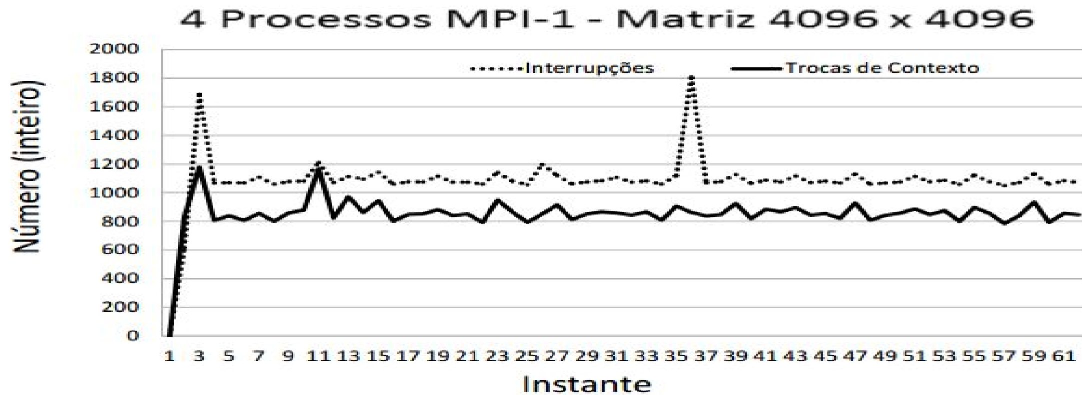
5.2.3 Análise de Trocas de Contexto e Interrupções

As versões paralelas desenvolvidas com MPI-1 e MPI-2 reduziram o tempo médio de execução e aumentaram o consumo de memória máximo como demonstrado anteriormente na subseção 5.2.2. A Figura 31 mostra as trocas de contexto e as interrupções da versão sequencial do *Skyline Matrix Solver*. Ao início da execução da versão sequencial a aplicação chega ao valor máximo de trocas de contexto superior a 550. Posteriormente tem variação entre 250 e 600 trocas de contexto por segundo, atingindo seu valor máximo próximo a 700 no instante 25, logo após se mantém entre 300 e 400 até o final da execução. As interrupções durante toda a execução se mantém entre 300 e 400, atingindo seu máximo nos primeiros instante superior a 400.

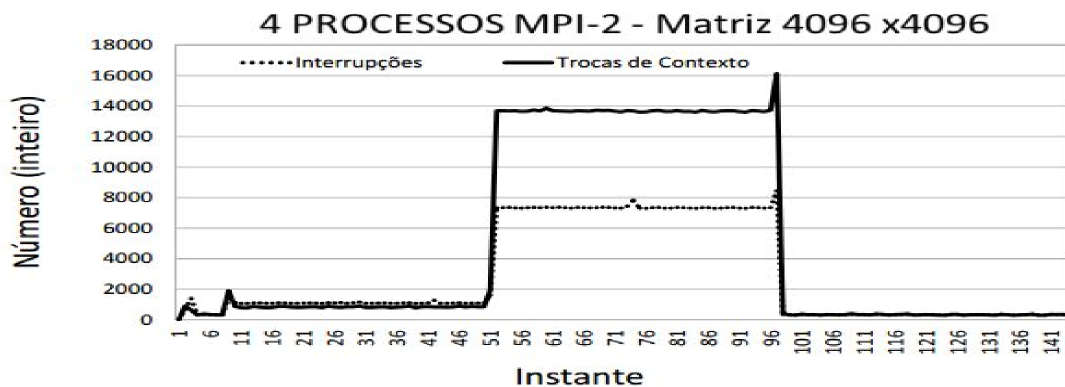
A Figura 32 mostra a execução com 4 processos MPI-1 na sub-figura *a*. O valor máximo alcançado chegou próximo a 1200 trocas de contexto. Logo após quando os processos já foram criados este valor cai para 800, e mantém variação entre 800 e 1000 trocas de contexto. As interrupções tem comportamento semelhante alcançando seu valor máximo superior a 1800 interrupções. Durante o restante da execução as interrupções variam entre 1000 e 1200 interrupções.

A sub-figura *b* mostra a execução com 4 processos MPI-2. Ao início da execução as trocas de contexto ficam entre 1000 e 2000, mantém-se com pequena variação por alguns instantes. Atinge 14.000 trocas de contexto alguns instantes depois quando os processos são criados. Posteriormente este valor se mantém pouco a abaixo de 14000. No momento em que atinge seu máximo chega a mais de 16000 trocas de contexto. Estes valores vão crescendo a medida que os processos vão sendo executados e ocorrem as comunicações entre eles. A versão MPI-2 tem maior número de trocas de contexto e interrupções durante a execução, exceto pela etapa inicial onde tem valores muito próximos. As versões

Figura 32 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 4 Processos



(a) MPI-1



(b) MPI-2

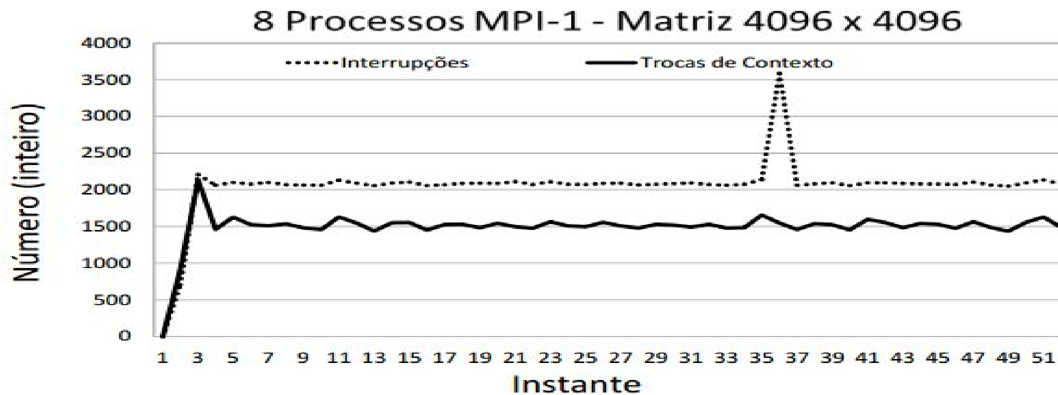
paralelas executadas com 4 processos aumentaram o número de trocas de contexto e interrupções quando comparadas a versão sequencial.

A Figura 33 mostra a execução com 8 processos MPI-1 na sub-figura *a*. O valor máximo alcançado foi superior a 2000 trocas de contexto ao início da execução. Logo após este valor cai para 1500 e se mantém com pequena variação. As interrupções tem comportamento semelhante alcançando seu valor máximo superior a 3500 interrupções. Durante o restante da execução as interrupções variam entre 1000 e 1200.

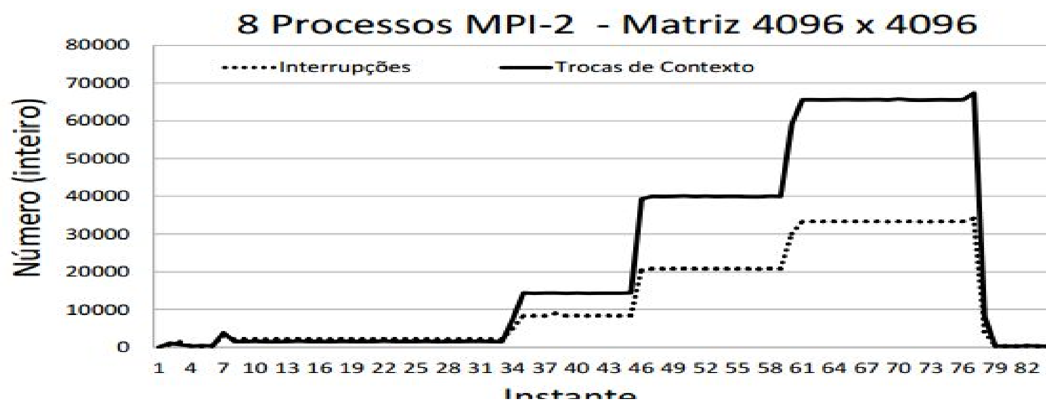
A sub-figura *b* mostra a execução com 8 processos MPI-2. Ao início da execução as trocas de contexto ficam entre 1000 e 2000 se mantendo com pequena variação por alguns instantes. Este número vai crescendo chegando próximo a 70000 trocas de contexto nos instantes finais da execução. A versão MPI-2 tem maior número de trocas de contexto e interrupções durante a execução. As versões paralelas executadas com 8 processos aumentaram aumentaram o número de trocas de contexto e interrupções quando comparadas a versões sequencial e paralelas utilizando 4 processos.

A Figura 34 mostra a execução com 16 processos MPI-1 na sub-figura *a*. O valor máximo alcançado chegou próximo a 8000 trocas de contexto ao início da execução. Logo

Figura 33 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 8 Processos



(a) MPI-1



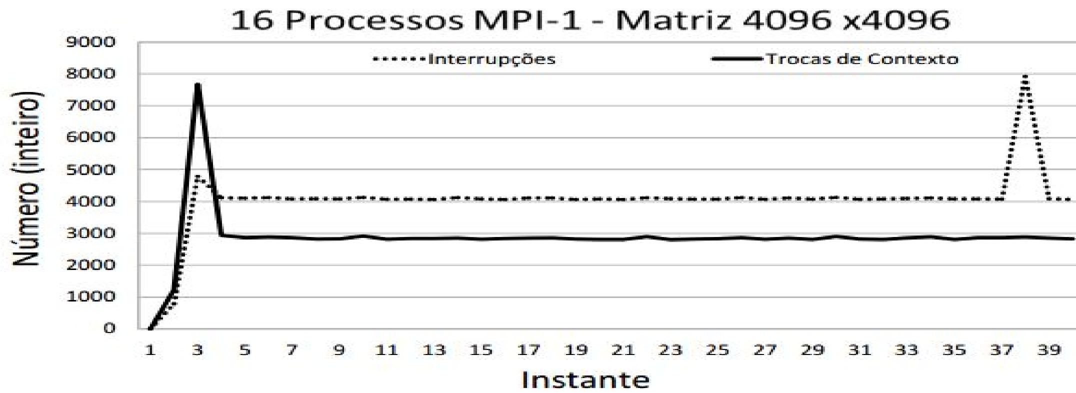
(b) MPI-2

após este valor cai para pouco menos de 3000 e se mantém com pequena variação. As interrupções tem comportamente semelhante alcançando seu valor máximo próximo a 8000 interrupções nos momentos finais da execução. Durante os demais instantes da execução as interrupções ficaram próximas a 4000.

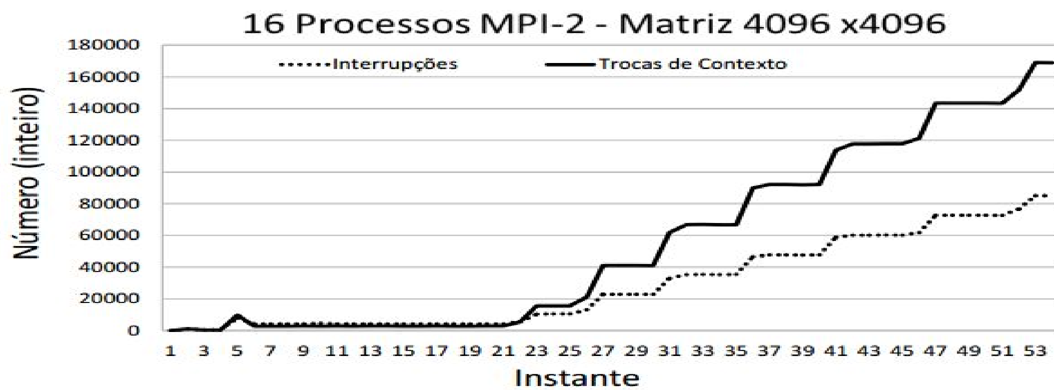
A sub-figura *b* mostra a execução com 16 processos MPI-2. Ao início da execução as trocas de contexto ficam entre 1000 e 2000 se mantendo com pequena variação por alguns instantes. Este número vai crescendo conforme os processos estão sendo executados chegando próximo a 180000 trocas de contexto nos instantes finais da execução. A versão MPI-2 tem maior número de trocas de contexto e interrupções durante a execução. As versões paralelas executadas com 16 processos aumentaram as trocas de contexto e interrupções comparando as versões anteriores.

A Figura 35 mostra a execução com 32 processos MPI-1 na sub-figura *a*. O valor máximo alcançado chegou próximo a 35000 trocas de contexto ao início da execução. Logo após este valor cai para pouco mais de 5000 e se mantém com pequena variação. As interrupções tem comportamento semelhante alcançando seu valor máximo próximo a 15000 interrupções nos momentos iniciais e finais da execução. Durante os demais

Figura 34 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 16 Processos



(a) MPI-1



(b) MPI-2

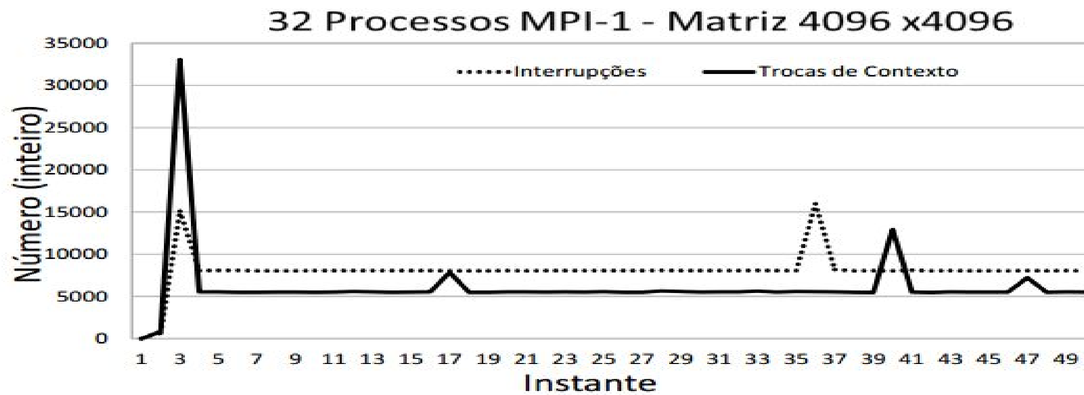
instantes da execução as interrupções variaram entre 6000 e 10000.

A sub-figura *b* mostra a execução com 32 processos MPI-2. Ao início da execução as trocas de contexto chegaram ao valor máximo de 40000. Este valor cai e se mantém próximo a 5000 até o final da execução. A versão MPI-2 tem maior número de trocas de contexto e interrupções ao início da execução. Nos demais instantes ambas versões resultaram valores muito próximos. As versões paralelas executadas com 32 processos aumentaram as trocas de contexto e interrupções quando comparadas as versões anteriores.

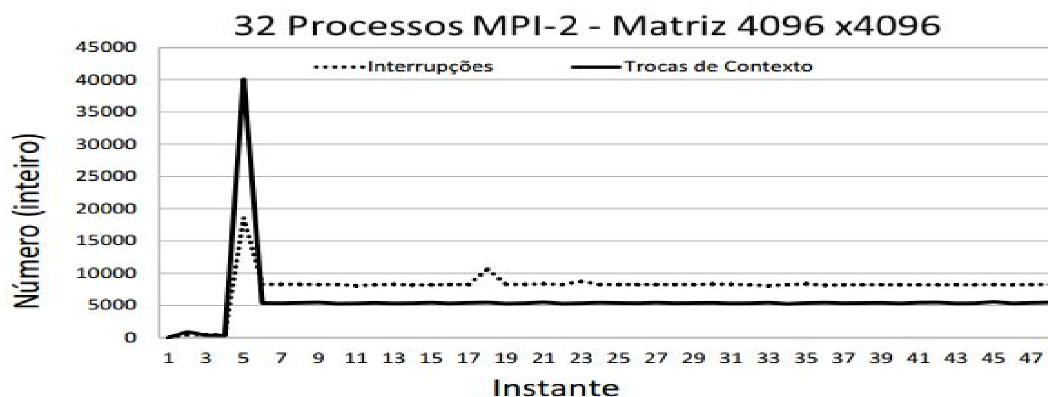
5.2.4 Análise de Utilização de *CPU*

Nas execuções sequenciais e paralelas do problema do *Skyline Matrix Solver* assim como para o problema do Jogo da Vida, os núcleos de processamento que receberam carga de trabalho fizeram uso de 100% durante todo tempo. Na versão sequencial um núcleo de processamento recebeu o problema e durante toda a execução teve 100% de uso. Já nas versões paralelas, tanto para MPI-1 quanto para MPI-2, durante as execuções com 4 processos o ambiente de testes teve 4 núcleos com 100% de uso durante todo o tempo. Com 8 processos executados teve 8 núcleos núcleos com 100% de usos durante toda execução e

Figura 35 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 32 Processos



(a) MPI-1



(b) MPI-2

assim sucessivamente. Com 32 processos utilizou a tecnologia *Hyperthreading* onde dois processos são executados concorrentemente em cada núcleo, tendo oscilações durante as execuções.

5.3 Conclusões do Capítulo

Este capítulo apresentou os resultados obtidos nas execuções dos problemas alvo. Na análise do problema do Jogo da Vida o qual possui carga de trabalho regular, identificou-se que para todos os tamanhos de entrada a versão implementada com MPI-2 é mais eficiente. As versões com 32 processos para todos os tamanhos de matriz de entrada obtiveram os maiores *speedups* chegando próximos a 15. As versões paralelas fizeram uso de memória acima da versão sequencial para todos os casos de teste durante a execução dos processos. Este período onde o consumo de memória é mais elevado apesar de ser menor para a versão MPI-2 em relação a versão MPI-1 é muito próximo. O consumo de memória para ambas as versões sempre se manteve entre 350 e 400 *MB* para todas as execuções, demonstrando que para o problema que a criação dinâmica de processos não

impacta em maior consumo de memória. Porém ao aumentar-se o número de processos as etapas de inicialização e finalização da aplicação sofrem aumento percentual, e com 16 e 32 processos estas etapas foram impactadas pela criação dinâmica de processos. As trocas de contexto e interrupções durante as execuções paralelas também aumentaram em relação a execução sequencial. Além disso conforme o número de processos cresce estes números também sofrem crescimento. Utilizando 4, 8 e 16 processos nas execuções com MPI-2 realizaram mais trocas de contexto e interrupções em determinados momentos, porém nas demais etapas estes valores ficaram muito próximos as execuções com MPI-1. Já com 32 processos as execuções com MPI-1 obtiveram os valores mais elevados. A criação dinâmica de processos impactou em um número menor de trocas de contexto e interrupções para todos os casos, destacando que quando utilizada a tecnologia *hyperthreading* a maior parte da execução estes valores estiveram muito acima da versão MPI-2.

Já o problema do *Skyline Matrix Solver* possui carga de trabalho irregular. Além disso conforme aumentamos o tamanho de entrada e o número de processos, o desempenho aumenta exceto pela execução MPI-2 com 32 processos que tem desempenho inferior a execução com 16 processos. O melhor resultado entre os testes foi obtido utilizando 16 processos com MPI-1, chegando a um *speedup* próximo a 9 com entrada de tamanho 4096 x 4096. Na execução com tamanho de entrada 5120 x 5120 o maior *speedup* chegou próximo a 7. Nas implementações MPI-2 foi possível notar que a criação dinâmica de processos juntamente com a tecnologia *Hyperthreading*, o sobrecusto da criação de processos em tempo de execução é atenuado. Dessa forma obtendo ganho superior ao obtido quando executado com os demais números de processos. Porém a versão MPI-1 foi sempre mais eficiente. As versões paralelas fizeram consumo mais elevado de memória comparando com a versão sequencial. As versões com 4 e 8 processos para ambas implementações tem o consumo máximo próximo. Porém quando executadas com 16 processos a versão MPI-2 essa situação se inverte. Já a versão MPI-1 com 32 processos obteve o maior consumo chegando a mais de 500 *MB*. As trocas de contexto e interrupções durante as execuções paralelas aumentaram em relação a execução sequencial. Assim como no problema do Jogo da Vida, conforme o número de processos aumenta o número de trocas de contexto e interrupções também cresce. Todavia o valor mais elevado entre os obtidos foi utilizando MPI-2 com 16 processos. Já na versão MPI-1 estes valores crescem, mas se mantém sempre abaixo dos obtidos na versão MPI-2.

Em ambos os problemas o uso de *CPU* foi semelhante com exceção do tempo de execução para cada caso. Todos os núcleos de processamento que estavam sendo utilizados na execução se mativeram com 100% de uso, exceto pelas execuções com 32 processos onde o uso teve oscilação durante todas execuções.

O Capítulo 6 apresenta uma discussão sobre os resultados apresentados nessa sessão.

6 Conclusões

Este trabalho realizou um estudo sobre o desempenho de aplicações paralelas implementadas com MPI-1, onde os processos são criados de forma estática, e com MPI-2 na qual os processos são criados dinamicamente. Foi possível avaliar o desempenho das versões paralelas de melhor desempenho implementadas por Lorenzon (2013) dos problemas alvo. A paralelização reduziu o tempo médio de execução das aplicações, assim como realizou consumo mais elevado de memória para a todas as versões quando comparadas a versão sequencial.

Com base nas análises realizadas concluímos que o problema do Jogo da Vida teve seu tempo médio reduzido em todas execuções das versões paralelas. A versão implementada com MPI-2 para todos os casos de teste foi mais eficiente que a versão MPI-1. Destacando que o melhor resultado foi obtido utilizando 32 processos chegando a um *speedup* próximo a 15 com a matriz de entrada de tamanho 16384 x 16384. Além do desempenho as versões paralelas do Jogo da Vida também aumentaram o consumo de memória durante as execuções. Para todos os casos de teste o consumo de memória chegou próximo a **400 MB** nos picos de consumo. Destacando que a versão MPI-2 fez uso do mesmo tamanho de memória, porém por períodos de tempo mais curtos que a versão MPI-1. As trocas de contexto e interrupções também sofrem aumento nas versões paralelas. A versão MPI-1 ao início das execuções realiza um número maior de trocas de contexto e interrupções. Todavia a versão MPI-2 ao criar os processos inverte este cenário. Nos demais momentos da execução ambas versões com o mesmo número de processos tem comportamento semelhante.

Para o problema do *Skyline Matrix Solver* foi observado o ganho de desempenho com MPI-1. Destacando que o melhor resultado foi obtido utilizando 16 processos chegando a um *speedup* próximo a 9 com matriz de entrada de tamanho 4096 x 4096. Na execução com 4 processos MPI-1 onde o *speedup* ficou muito próximo do ideal. Assim como as versões paralelas do Jogo da Vida realizaram consumo mais elevado de memória em alguns momentos das execuções, as versões paralelas do *Skyline Matrix Solver* também tiveram crescimento de consumo. Com o aumento do número de processos o consumo de memória cresce. O valor máximo alcançado chegou acima de **500 MB** utilizando 32 processos na versão MPI-1. As trocas de contexto e interrupções também aumentaram nas execuções paralelas. A versão MPI-2 realizou em todos os casos maiores números de trocas de contexto e de interrupções. Destacando que o valor mais elevado foi obtido utilizando 16 processos chegando a mais de 160.000.

Para os dois problemas a utilização de *CPU* foi de 100% nos núcleos de processa-

mento que foram utilizados em cada caso de teste, exceto pelas execuções com 32 processos que tiveram oscilação durante todas as execuções.

Concluímos que as paralelizações dos problemas utilizando MPI implementadas por Lorenzon (2013) reduziram o tempo médio de execução em todos os casos de teste utilizados neste trabalho. Além disso a paralelização fez uso mais elevado de alguns recursos disponíveis no ambiente de testes como memória e núcleos de processamento. Em decorrência deste uso mais elevado ocorreram maior número de trocas de contexto e interrupções durante as execuções quando comparadas a execução sequencial em um período de tempo inferior. Para o o problema do Jogo da Vida, o qual possui carga de trabalho regular o impacto da criação dinâmica no desempenho foi atenuado pelo modelo de comunicação adotado. Já o consumo de memória, trocas de contexto e interrupções foi semelhante para ambas as versões atingindo praticamente os mesmos valores, porém com momentos de pico com menor duração. Já para o problema do *Skyline Matrix Solver*, o qual possui carga de trabalho irregular a criação de processos em tempo de execução impactou negativamente no desempenho da aplicação quando comparada a criação estática. Para todos os casos de teste com exceção da execução com 16 processos o consumo de memória foi mais elevado na versão MPI-1. A criação dinâmica de processos não causa impacto, porém a comunicação acaba gerando maiores quantidades de trocas de contexto e interrupções. Além disso conforme aumenta o número de processos esses valores sobem ainda mais. O uso de *CPU* da mesma maneira que para o problema do Jogo da Vida é de 100% em todas as execuções, porém com oscilações para 32 processos.

Referências

BEZERRA H. P. NETO; COSTA, J. A. Processamento paralelo com openmp em um simulador dinâmico de linhas de ancoragem e risers, parte ii. *Associação Argentina de Mecânica Computacional – Buenos Aires, Argentina.*, p. 3000–3017, 2010. Citado na página 27.

CERA M. C.; MAILLARD, N. Message-passing interface avançado. *X Escolar Regional de Alto Desempenho*, p. 7–29, 2010. Citado na página 21.

COELHO, S. A. Introdução a computação paralela com o open mpi. *Simpósio Brasileiro de Computação Paralela com o Open MPI*, p. 24–44, 2012. Citado na página 21.

CONWAY, J. The game of life. *Scientific American*, 1970. Citado na página 33.

COSTA, J. a. Implementação de uma técnica de processamento paralelo para o método dos elementos discretos utilizando o padrão de memória compartilhada openmp em ambientes multicore. *CILAMCE'2008, Maceió, AL, Brasil*, 2008. Citado na página 27.

EAGER, D.; ZAHORJAN, J.; LAZOWSKA, E. Speedup versus efficiency in parallel systems. *Computers, IEEE Transactions on*, v. 38, n. 3, p. 408–423, Mar 1989. ISSN 0018-9340. Citado na página 25.

FLYNN, M. Some computer organizations and their effectiveness. *IEEETC:JOURNAL*, p. 948–960, 1972. Citado na página 25.

FORUM, M. P. *MPI: A Message-Passing Interface Standard*. Knoxville, TN, USA, 1994. Citado na página 30.

GEIST G. A.; KOHLA, J. A. P. P. M. Pvm and mpi: A comparsion of features. *Calculateurs Paralleles*, p. 137, 1996. Citado na página 27.

GROPP, W. et al. *MPI - The Complete Reference: Volume 2, The MPI-2 Extensions*. Cambridge, MA, USA: MIT Press, 1998. Citado 2 vezes nas páginas 30 e 31.

GROPP, W.; LUSK, E.; SKJELLUM, A. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, MA: MIT Press, 1994. xx + 307 p. ISBN 0-262-57104-8. Citado na página 30.

KUMAR, V. *Introduction to Parallel Computing*. 2nd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. 144-145 p. ISBN 0201648652. Citado na página 27.

LORENZON, A.; CERA, M.; ROSSI, F. Analysing the impact of mpi-2 dynamic process creation to the game of life problem. In: *Computer Systems (WSCAD-SSC), 2012 13th Symposium on*. [S.l.: s.n.], 2012. p. 133–140. Citado na página 34.

LORENZON, A. F. Um estudo sobre o desenvolvimento de aplicações com criação dinâmica de processos em mpi. *Trabalho de Conclusão de Curso, Universidade Federal do Pampa*, 2013. Citado 15 vezes nas páginas 21, 22, 23, 26, 27, 30, 33, 35, 36, 38, 40, 41, 47, 73 e 74.

MELLO, C. C. Impacto da hierarquia de memória no desempenho e consumo energético de aplicações paralelas em sistemas embarcados e de propósitos gerais. *Trabalho de Conclusão de Curso, Universidade Federal do Rio Grande do Sul*, 2014. Citado na página 28.

MOORE, G. E. Cramming more components onto integrated circuits. *Eletronics 38(8)*, p. 114–117, 1965. Citado na página 21.

NAVARRO, C. A.; HITSCHFELD-KAHLER, N.; MATEU, L. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in Computational Physics*, v. 15, n. 02, 2014. Citado na página 25.

NAVAUX, P. O. Introdução ao processamento paralelo. *RBC-Revista Brasileira de Computação*, sn, v. 5, n. 2, p. 31–43, 1989. Citado na página 25.

NEVES, M. V. *Dstat-Monitor*. 2015. Disponível em: https://github.com/mvneves/dstat-monitor - último acesso em Julho de 2015. Citado 2 vezes nas páginas 43 e 47.

PEZZI, G. P. et al. Escalonamento dinâmico de programas mpi-2 utilizando divisao e conquista. *WSCAD'06-Workshop em Sistemas Computacionais de Alto Desempenho*, v. 7, p. 71–78, 2006. Citado na página 27.

PRESS, W. H. e. a. Numerical recipes in c++. *The Art of Scientific Computing - Second Edition*, 2002. Citado 2 vezes nas páginas 36 e 37.

SCHEPKE, C.; LIMA, J. V. F. Programação paralela em memória compartilhada e distribuída. In: *ERAD-RS 2015 - Minicurso 3*. [S.l.: s.n.], 2015. Citado 2 vezes nas páginas 27 e 30.

SILVA R.; DIAS, P.; SOUZA, E. Modelo olam (ocean-land-atmosphere-model): Descrição, aplicações e perspectivas. *Revista Brasileira de Metereologia*, p. 144–157, 2009. Citado na página 25.

STALLINGS, W. *Arquitetura e organização de computadores: projeto para o desempenho*. Pearson Prentice Hall, 2002. ISBN 9788587918536. Disponível em: <https://books.google.com.br/books?id=wI41AAAACAAJ>. Citado 3 vezes nas páginas 21, 28 e 29.

TANENBAUM, A. *Sistemas operacionais modernos*. Prentice-Hall do Brasil, 2003. ISBN 9788587918574. Disponível em: <https://books.google.com.br/books?id=meCAGQAACAAJ>. Citado na página 29.

WILSON, G. V. Assessing the usability of parallel programming systems: The cowichan problems. *Programming Environments for Massively Parallel Distributed Systems Working Conference of the IFIP WG 103*, p. 183–193, 1994. Citado 3 vezes nas páginas 22, 33 e 40.

APÊNDICE A – Gráficos de Resultados Análises Realizadas

Figura 36 – Jogo da Vida - Consumo de Memória Sequencial - Matriz 8192

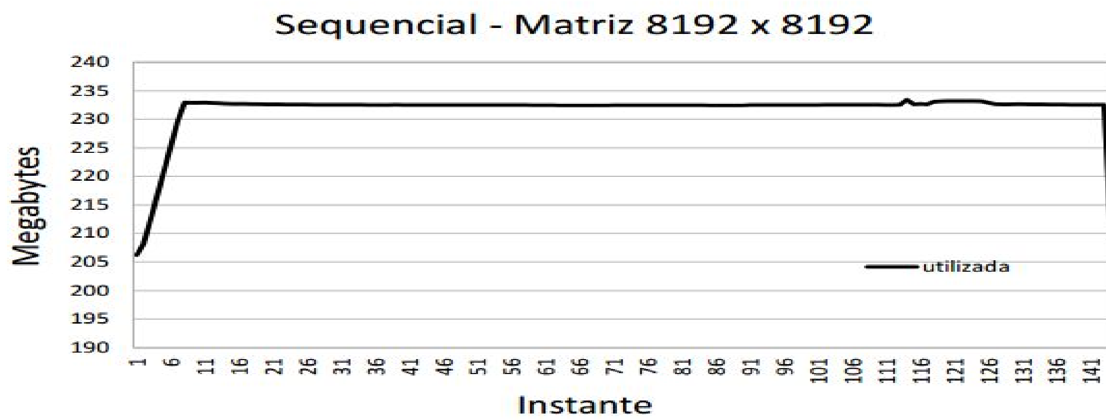
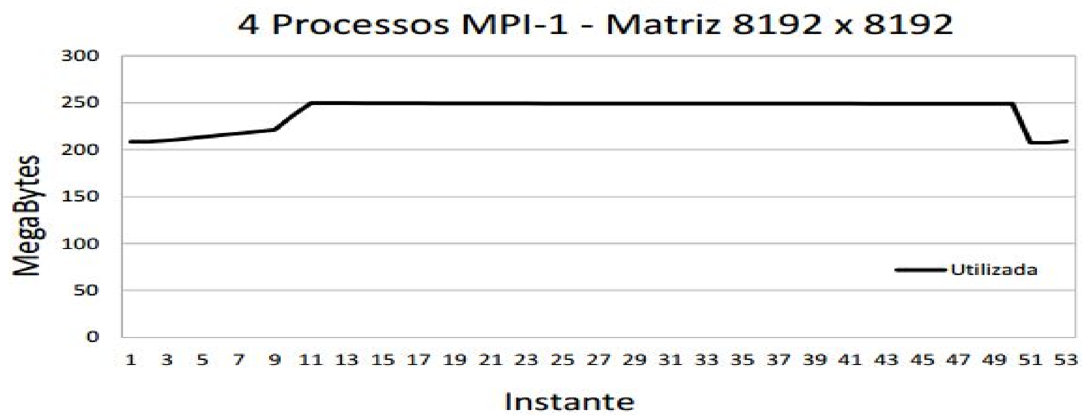
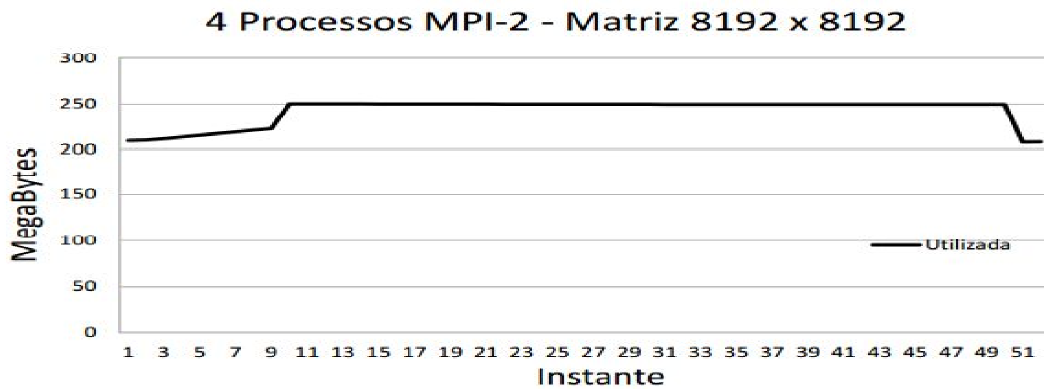


Figura 37 – Jogo da Vida - Consumo de Memória 4 Processos 8192

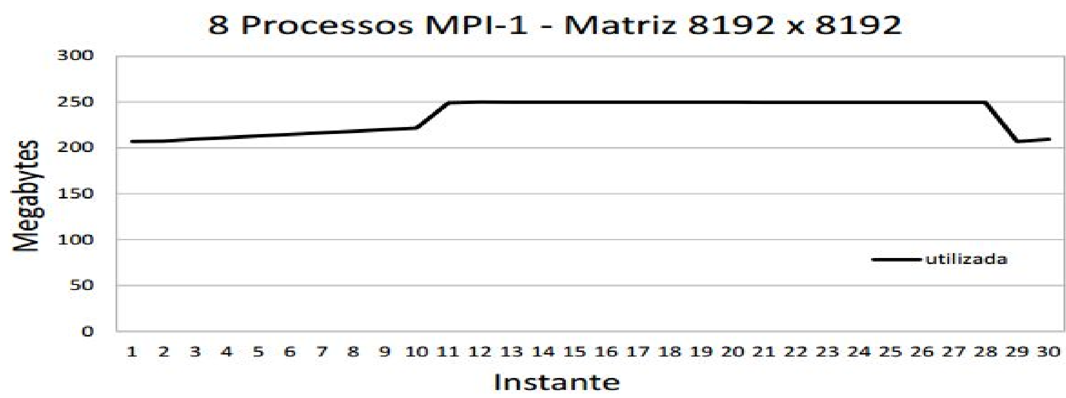


(a) MPI-1

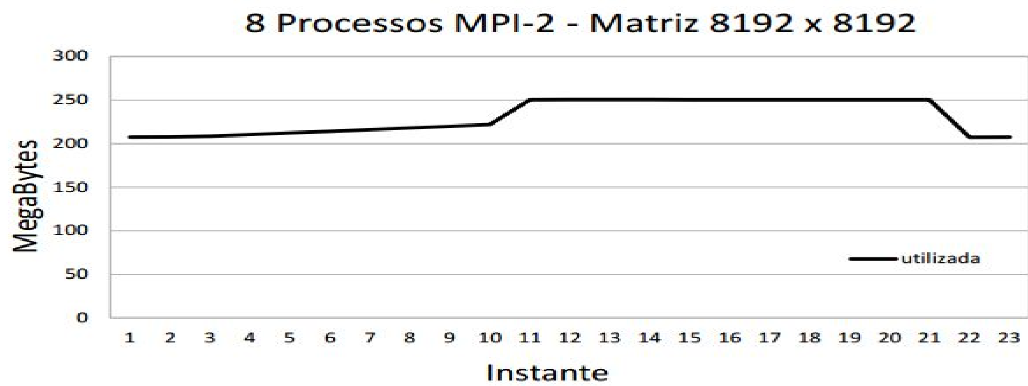


(b) MPI-2

Figura 38 – Jogo da Vida - Consumo de Memória 8 Processos 8192

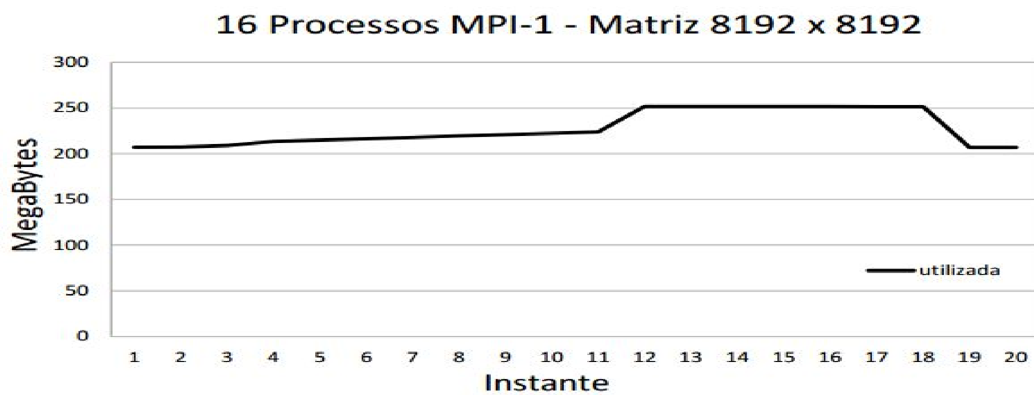


(a) MPI-1

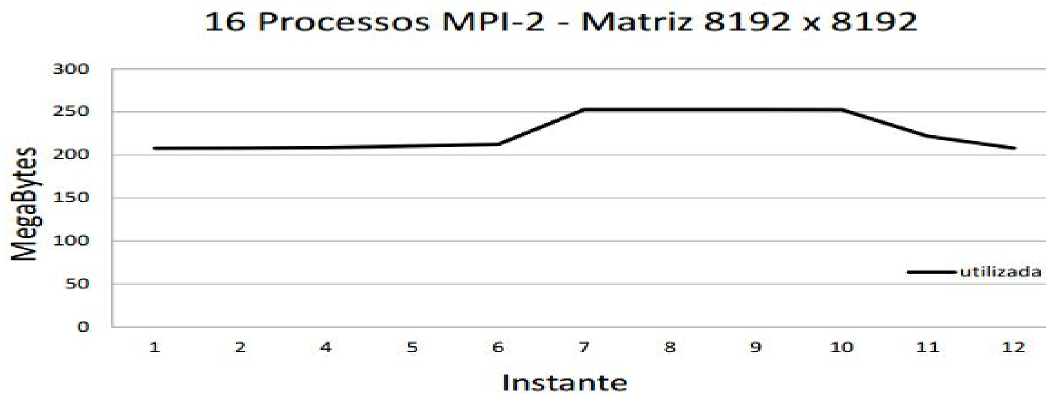


(b) MPI-2

Figura 39 – Jogo da Vida - Consumo de Memória 16 Processos 8192

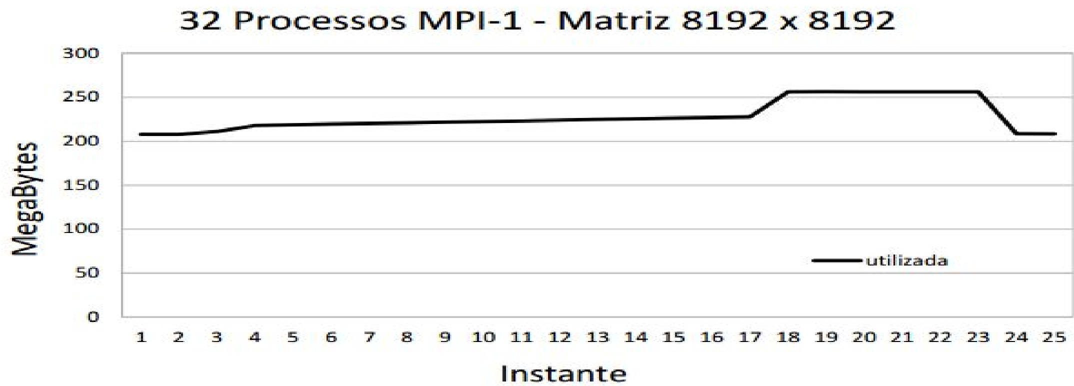


(a) MPI-1

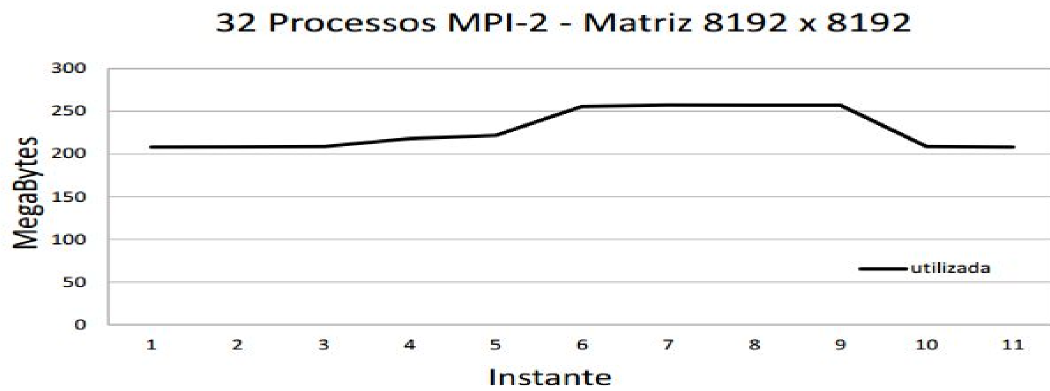


(b) MPI-2

Figura 40 – Jogo da Vida - Consumo de Memória 32 Processos 8192



(a) MPI-1



(b) MPI-2

Figura 41 – Jogo da Vida - Trocas de Contexto e Interrupções Sequencial 8192

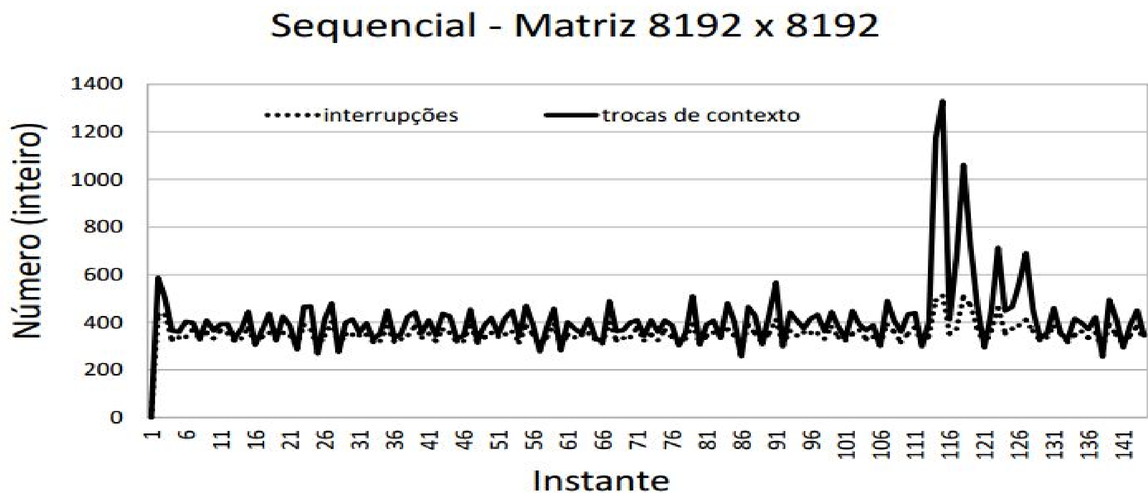
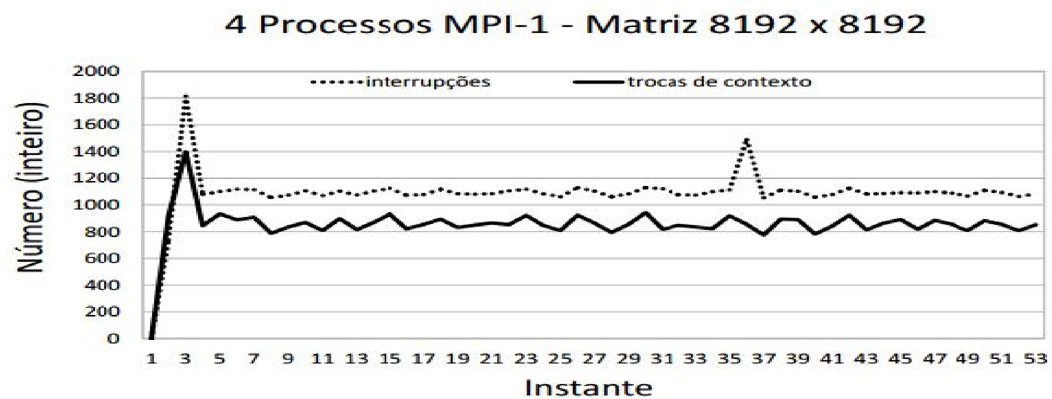
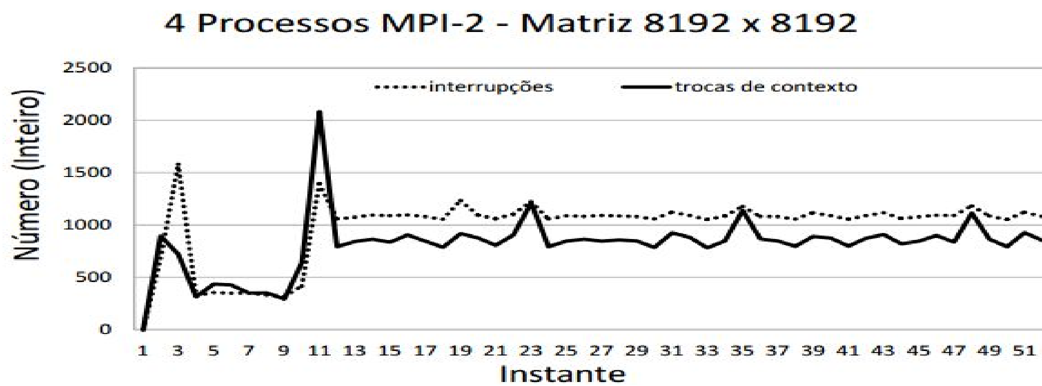


Figura 42 – Jogo da Vida - Trocas de Contexto e Interrupções 4 Processos 8192

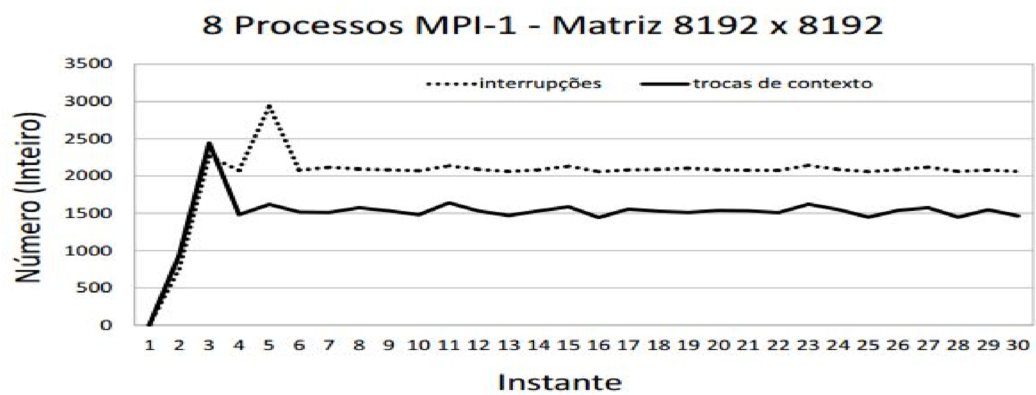


(a) MPI-1

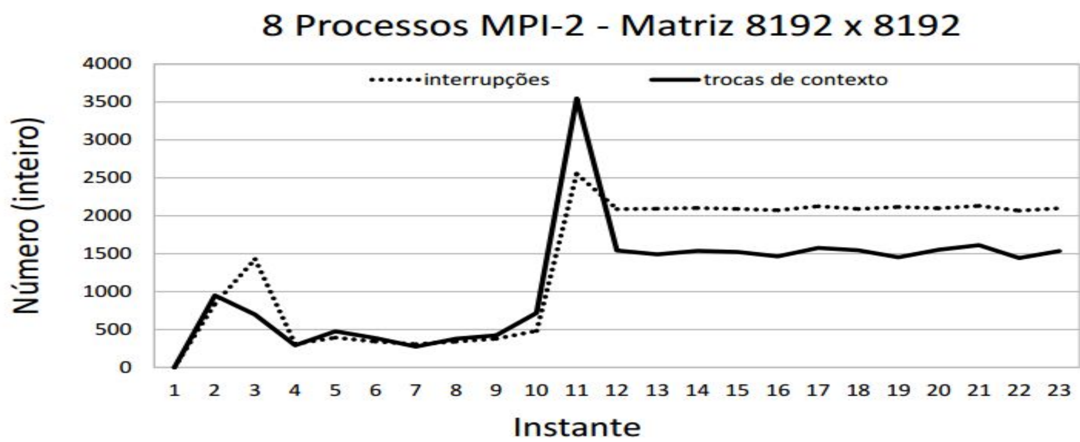


(b) MPI-2

Figura 43 – Jogo da Vida - Trocas de Contexto e Interrupções 8 Processos 8192

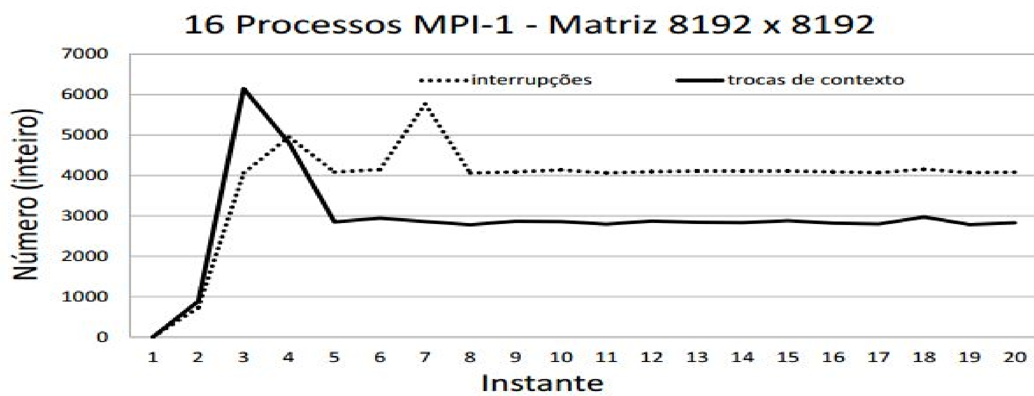


(a) MPI-1

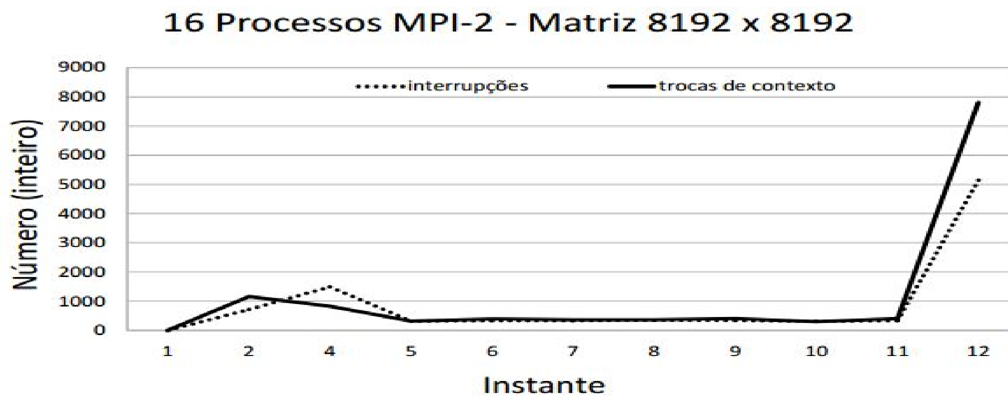


(b) MPI-2

Figura 44 – Jogo da Vida - Trocas de Contexto e Interrupções 16 Processos 8192

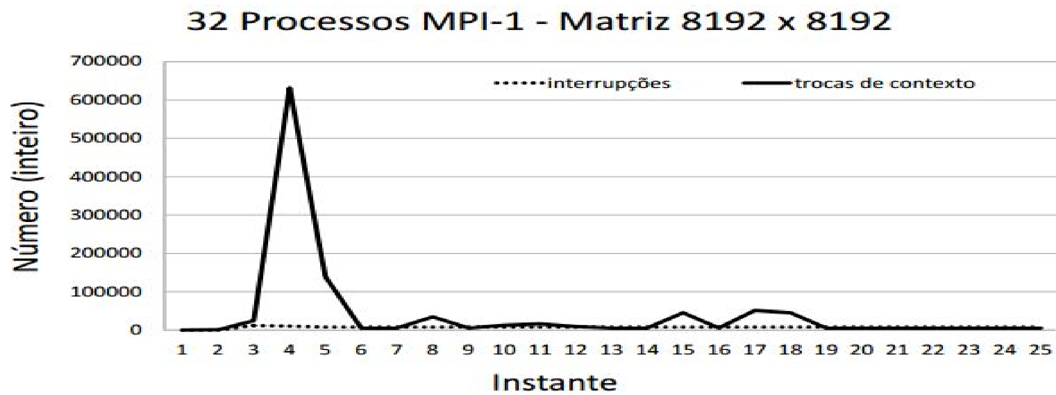


(a) MPI-1

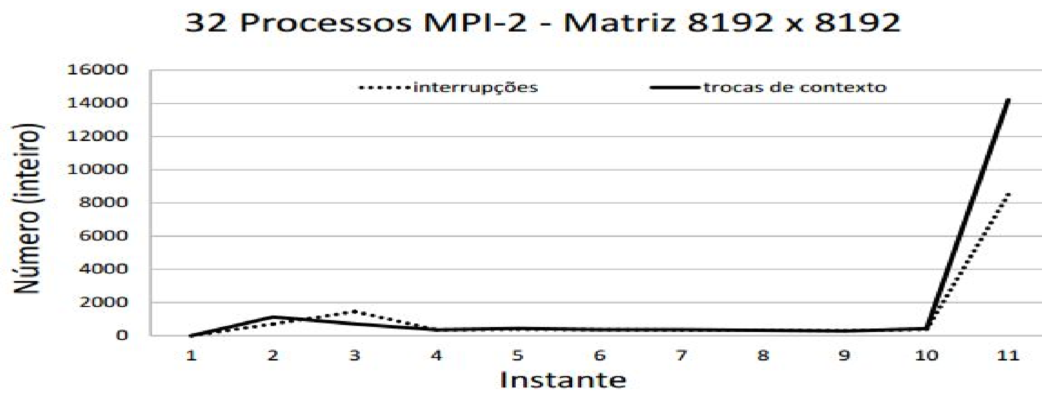


(b) MPI-2

Figura 45 – Jogo da Vida - Trocas de Contexto e Interrupções 32 Processos 8192



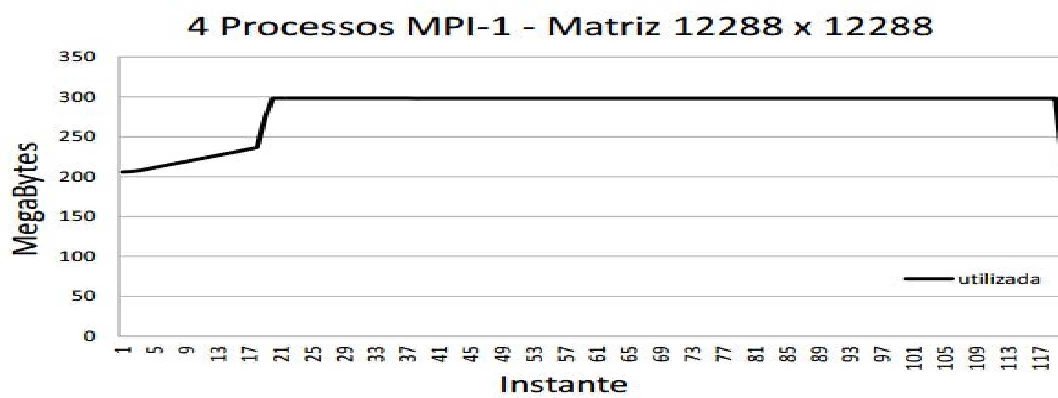
(a) MPI-1



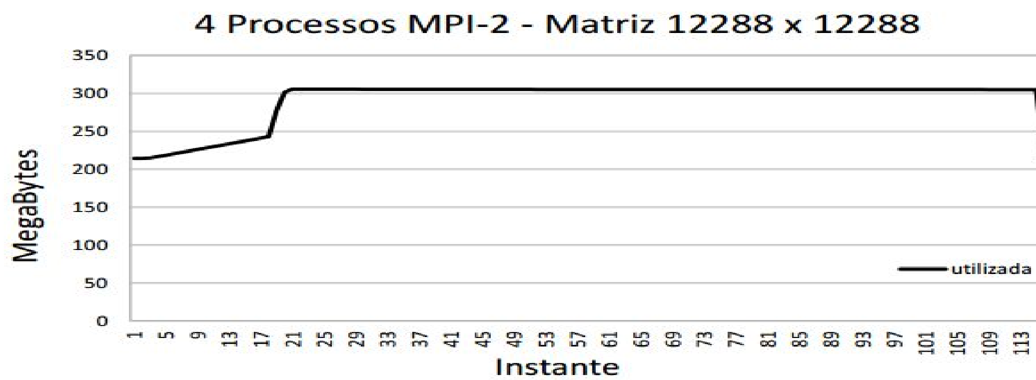
(b) MPI-2

Figura 46 – Jogo da Vida - Consumo de Memória Sequencial - Matriz 12288

Figura 47 – Jogo da Vida - Consumo de Memória 4 Processos 12288



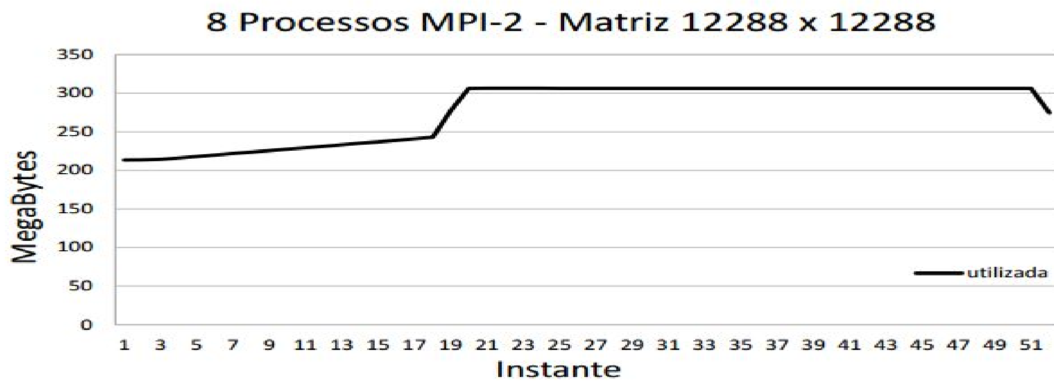
(a) MPI-1



(b) MPI-2

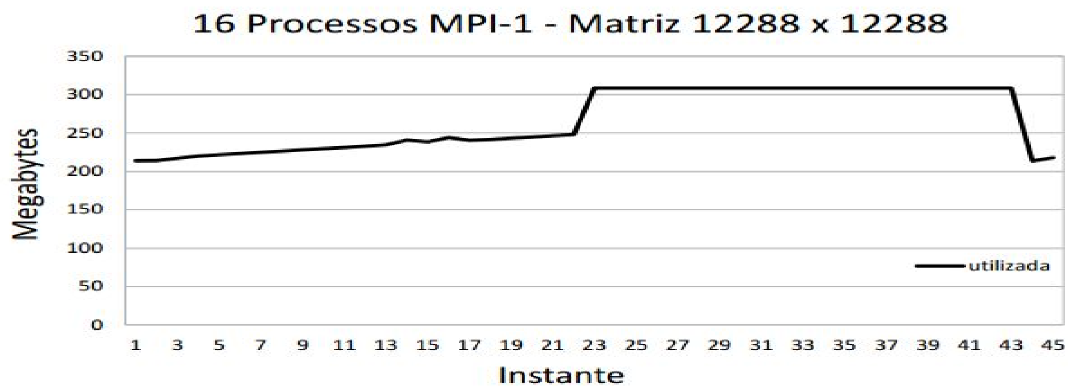
Figura 48 – Jogo da Vida - Consumo de Memória 8 Processos 12288

(a) MPI-1

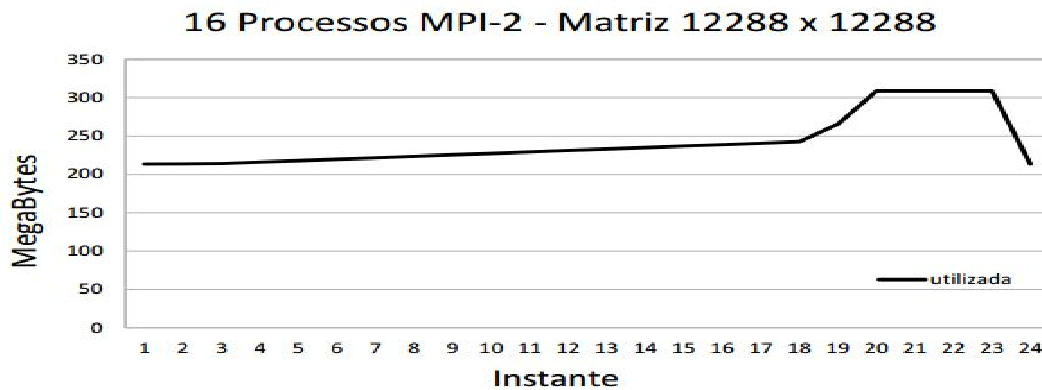


(b) MPI-2

Figura 49 – Jogo da Vida - Consumo de Memória 16 Processos 12288

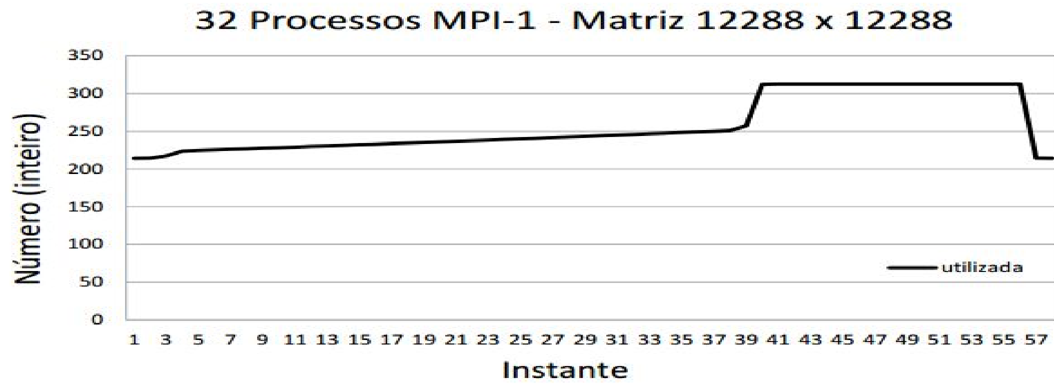


(a) MPI-1

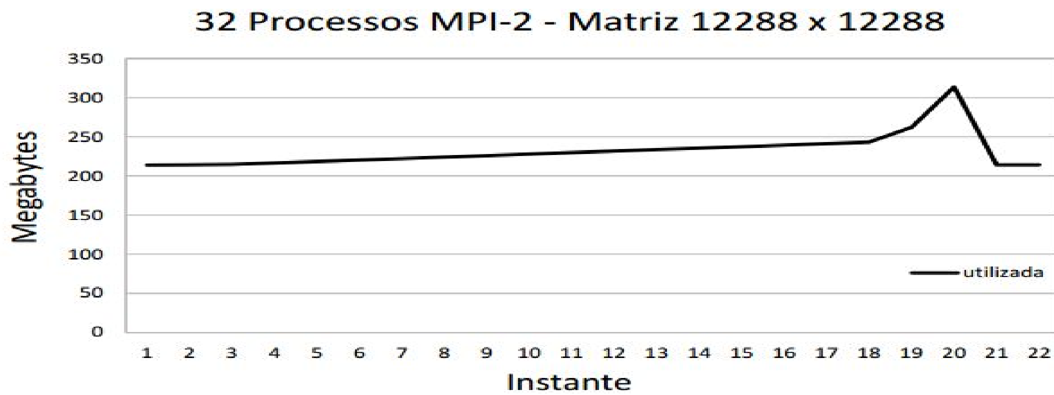


(b) MPI-2

Figura 50 – Jogo da Vida - Consumo de Memória 32 Processos 12288



(a) MPI-1



(b) MPI-2

Figura 51 – Jogo da Vida - Trocas de Contexto e Interrupções Sequencial 12288

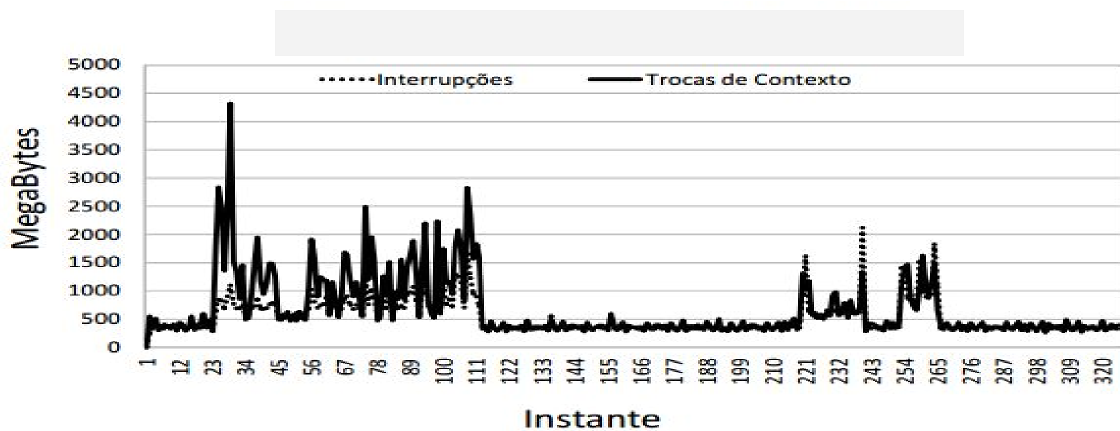
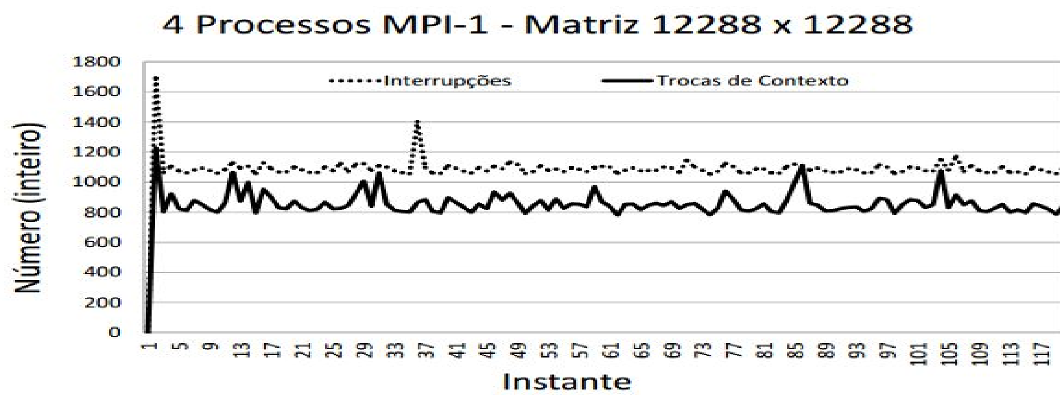
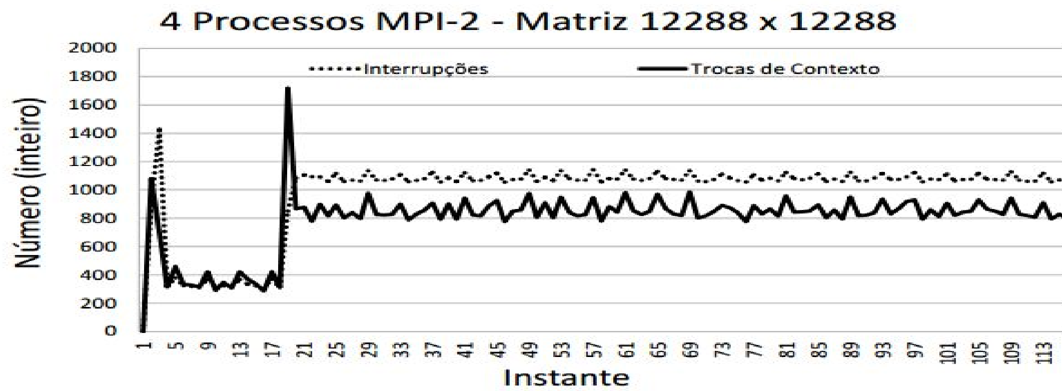


Figura 52 – Jogo da Vida - Trocas de Contexto e Interrupções 4 Processos 12288

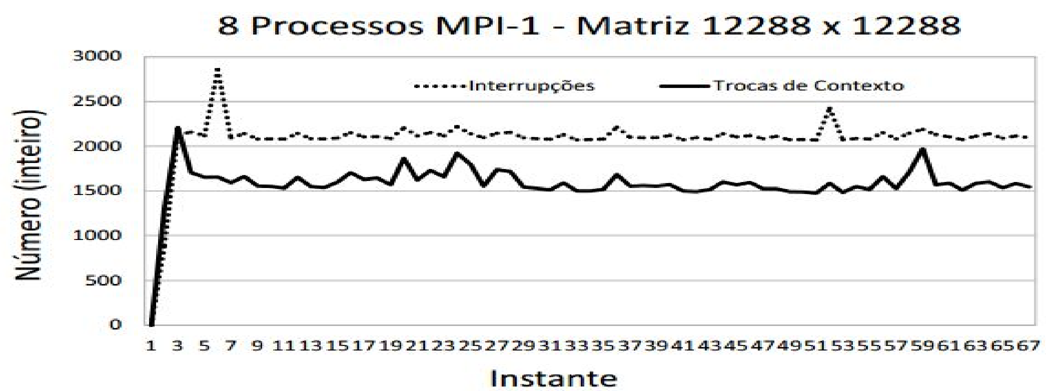


(a) MPI-1

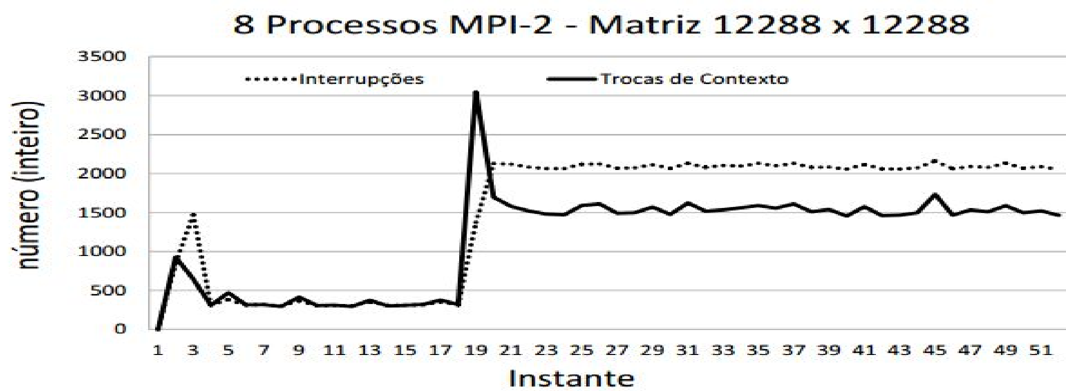


(b) MPI-2

Figura 53 – Jogo da Vida - Trocas de Contexto e Interrupções 8 Processos 12288

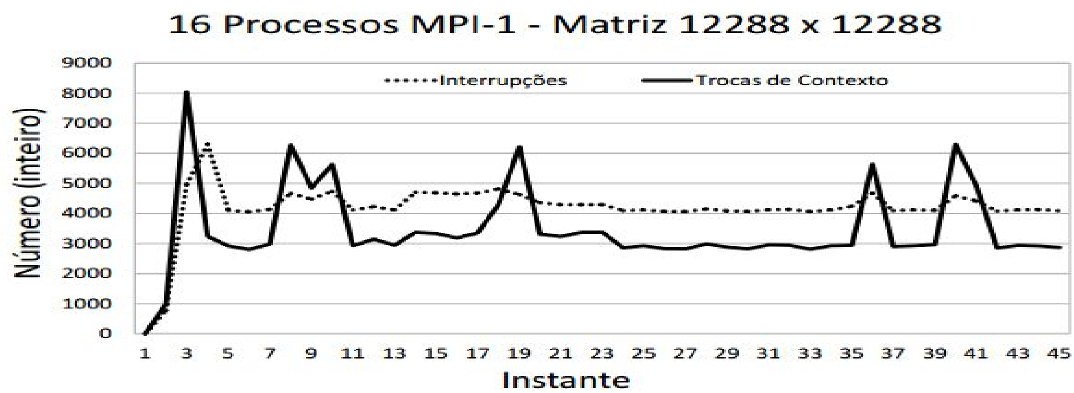


(a) MPI-1

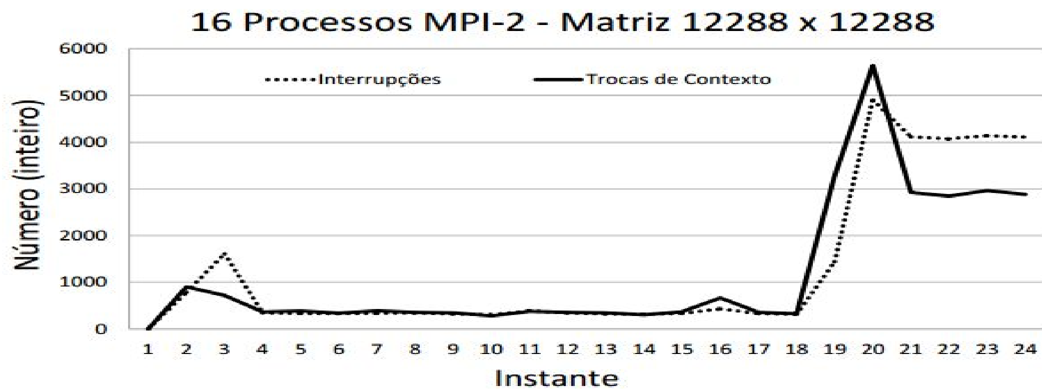


(b) MPI-2

Figura 54 – Jogo da Vida - Trocas de Contexto e Interrupções 16 Processos 12288

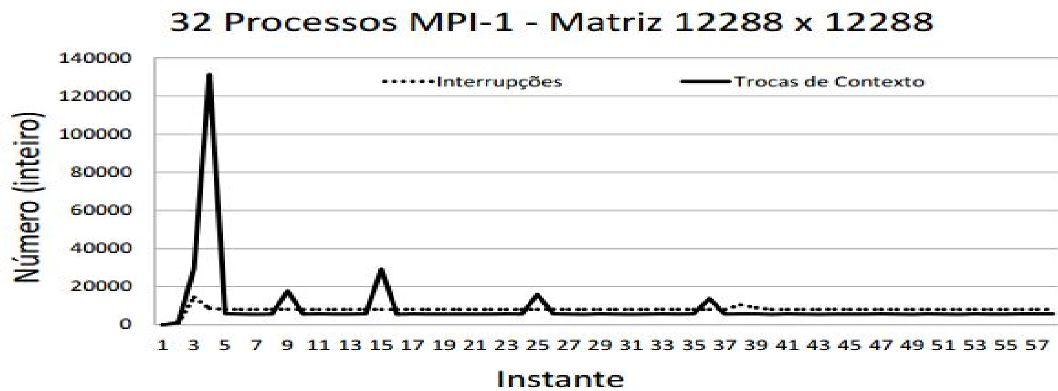


(a) MPI-1

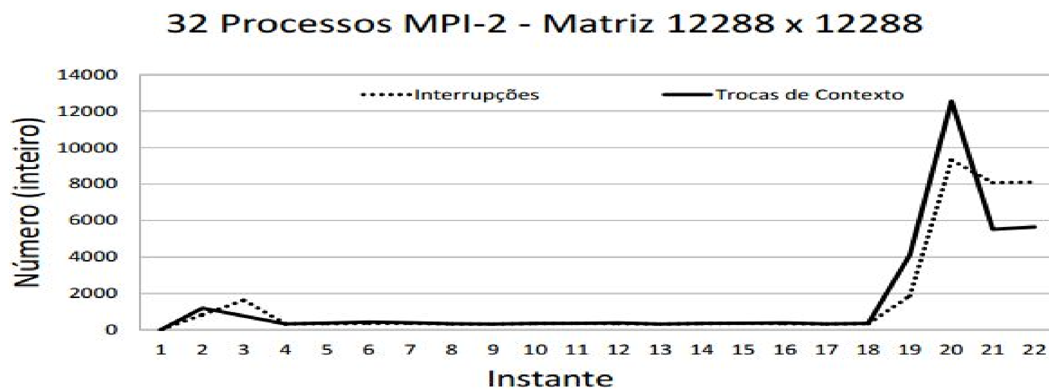


(b) MPI-2

Figura 55 – Jogo da Vida - Trocas de Contexto e Interrupções 32 Processos 12288



(a) MPI-1



(b) MPI-2

Figura 56 – *Skyline Matrix Solver* - Consumo de Memória Sequencial 5120

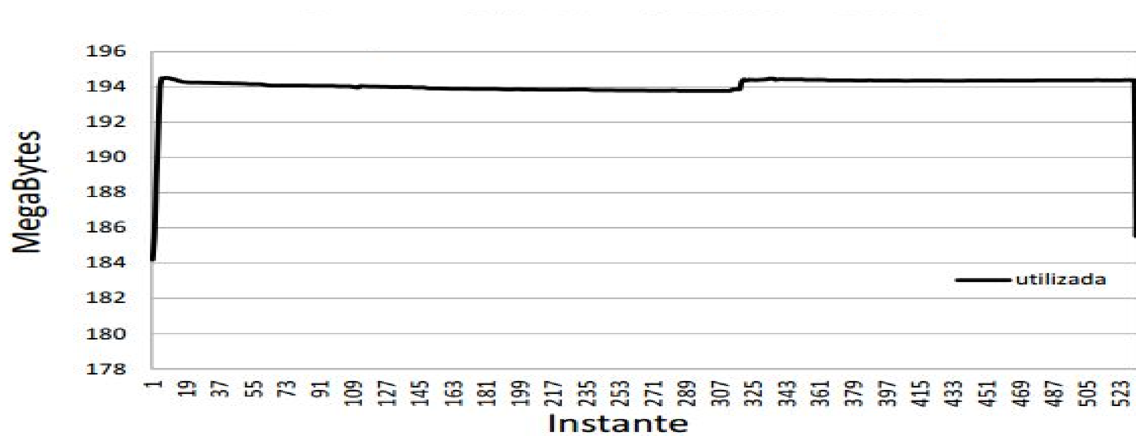
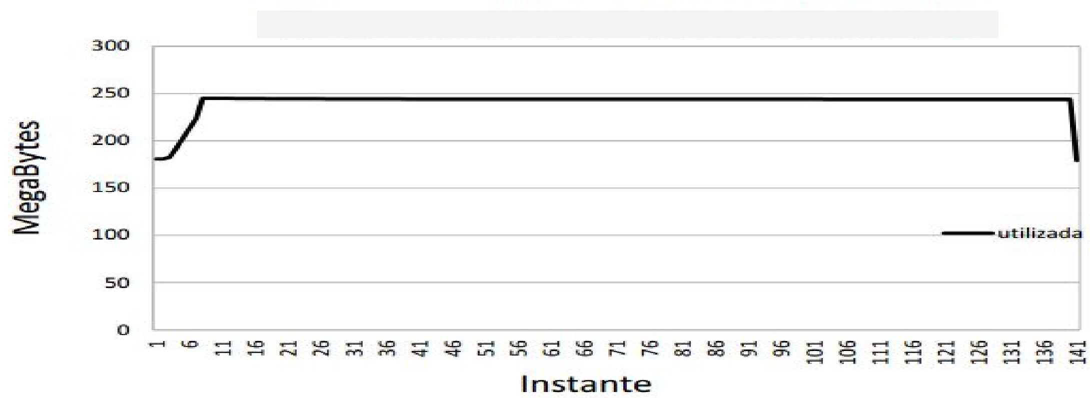
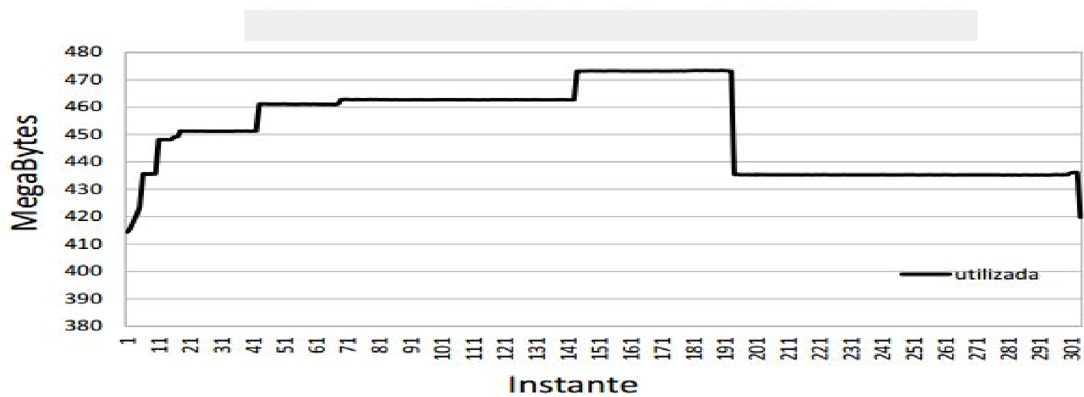


Figura 57 – *Skyline Matrix Solver* - Consumo de Memória 4 Processos 5120

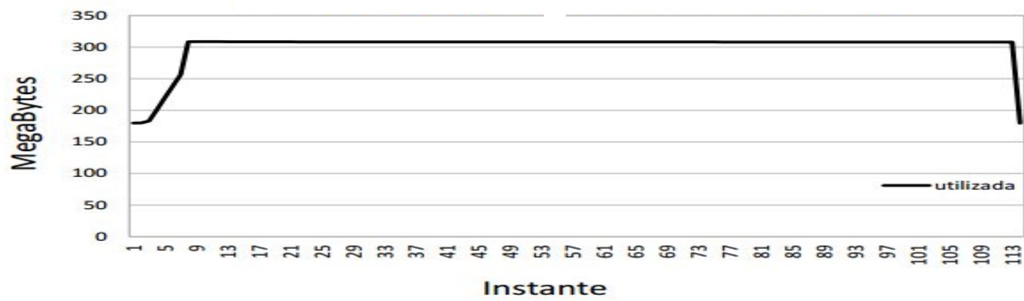


(a) MPI-1

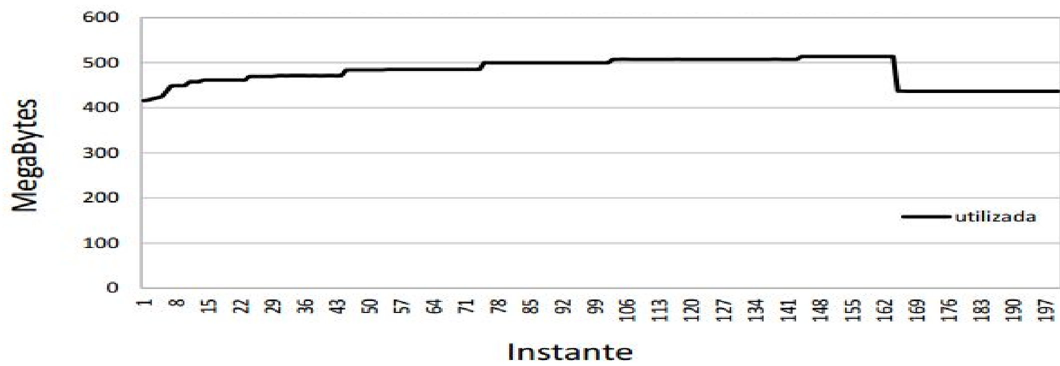


(b) MPI-2

Figura 58 – *Skyline Matrix Solver* - Consumo de Memória 8 Processos 5120

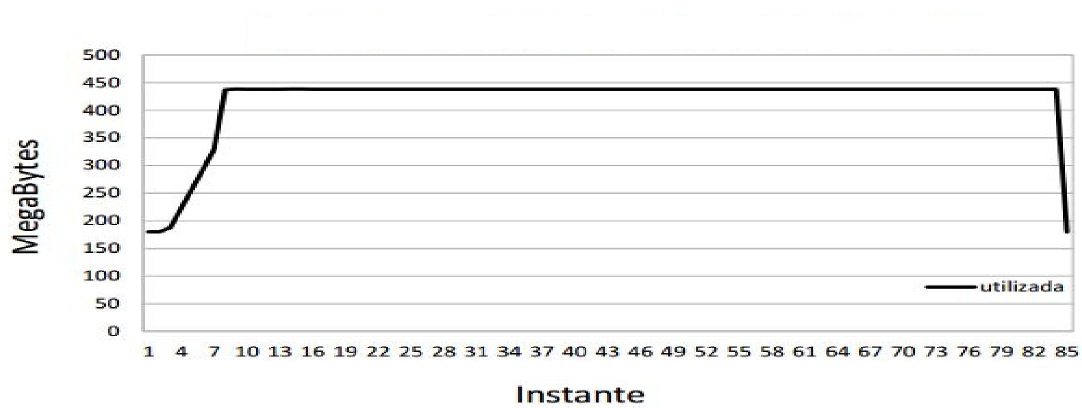


(a) MPI-1

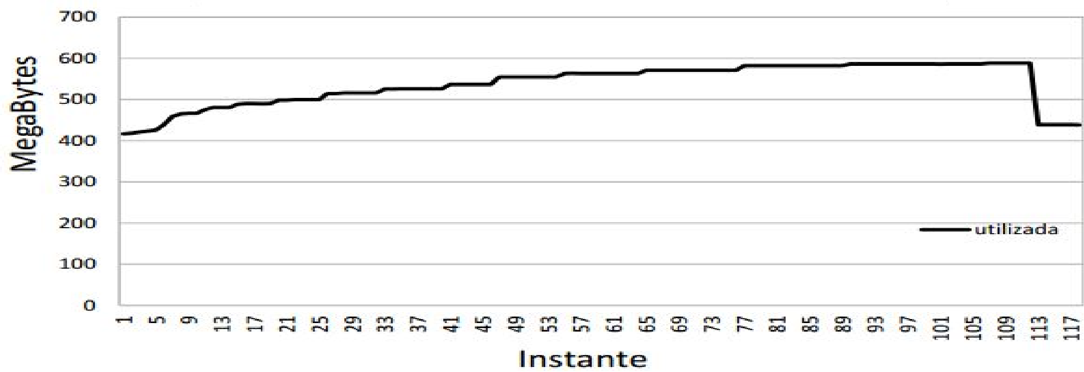


(b) MPI-2

Figura 59 – *Skyline Matrix Solver* - Consumo de Memória 16 Processos 5120

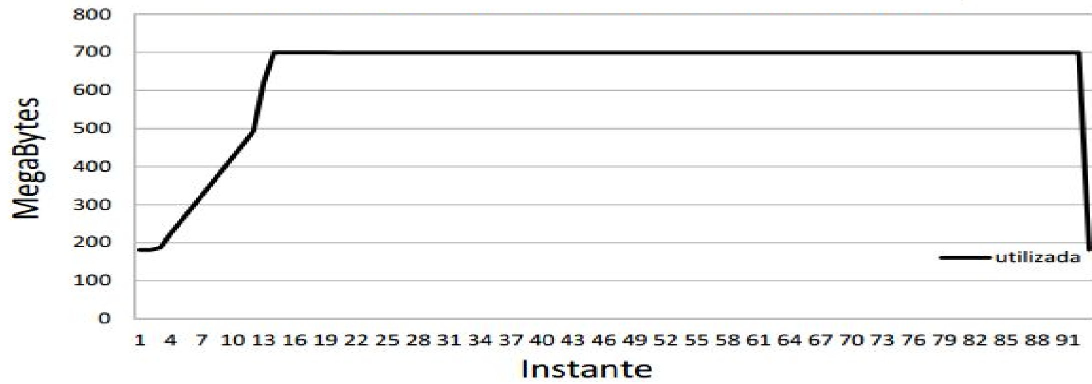


(a) MPI-1



(b) MPI-2

Figura 60 – *Skyline Matrix Solver* - Consumo de Memória 32 Processos 5120

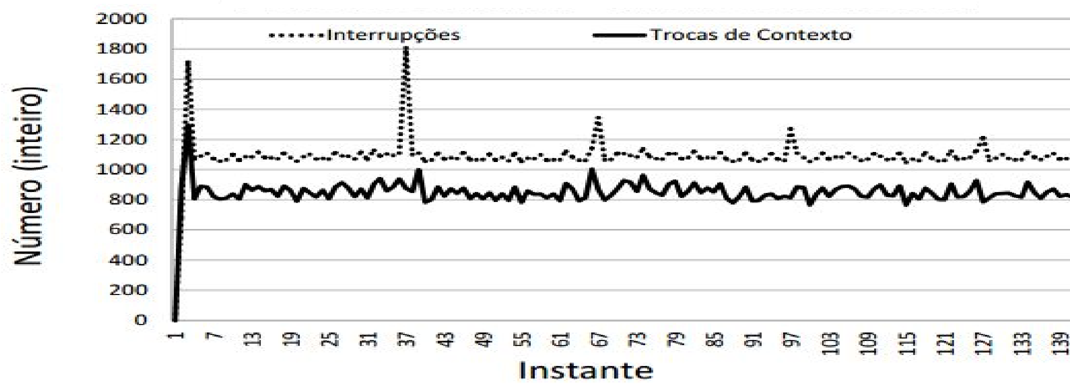


(a) MPI-1

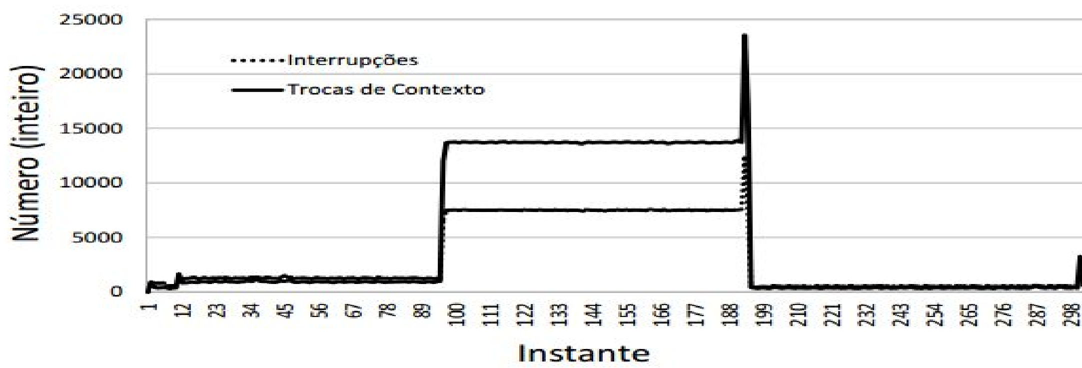
(b) MPI-2

Figura 61 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções Sequencial 5120

Figura 62 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 4 Processos 5120

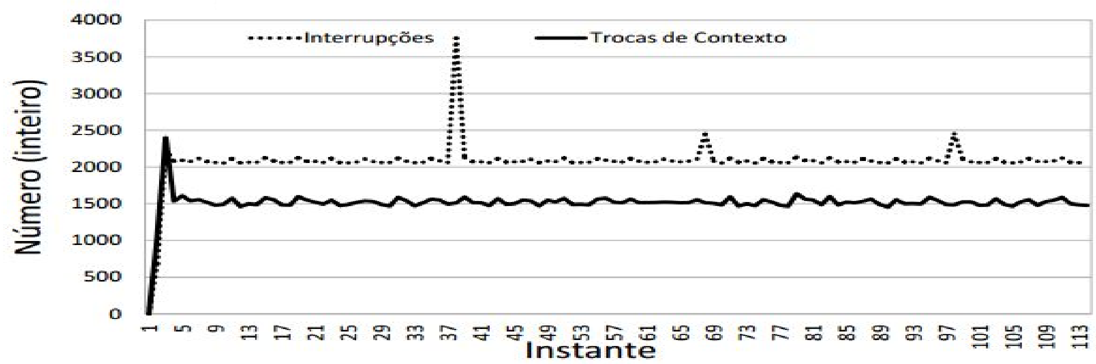


(a) MPI-1

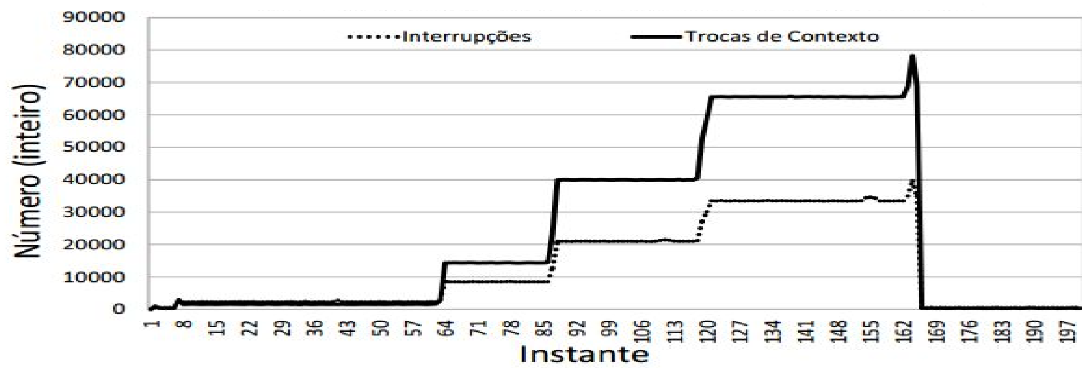


(b) MPI-2

Figura 63 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 8 Processos 5120

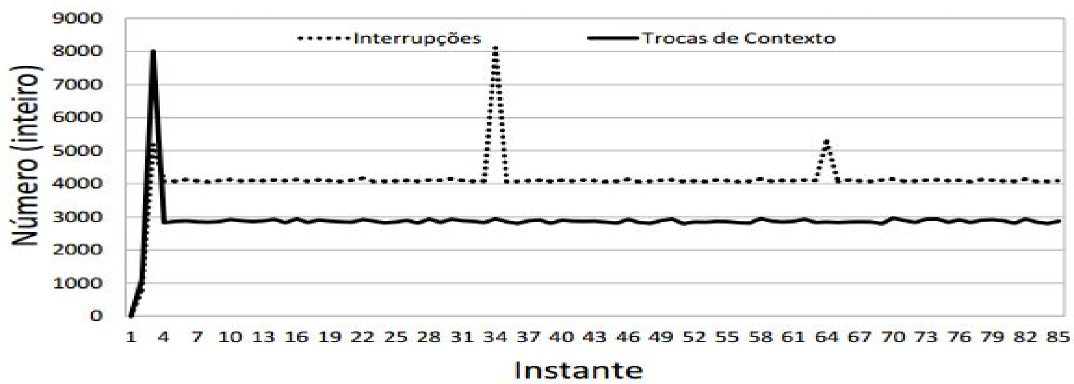


(a) MPI-1

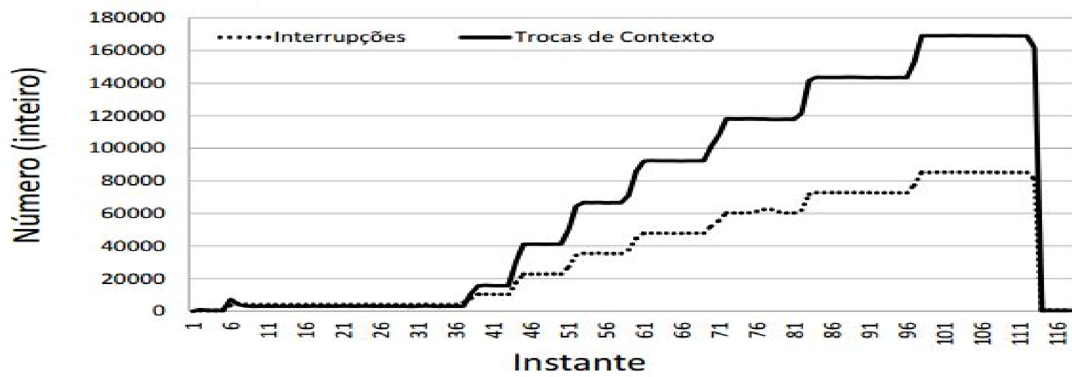


(b) MPI-2

Figura 64 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 16 Processos 5120

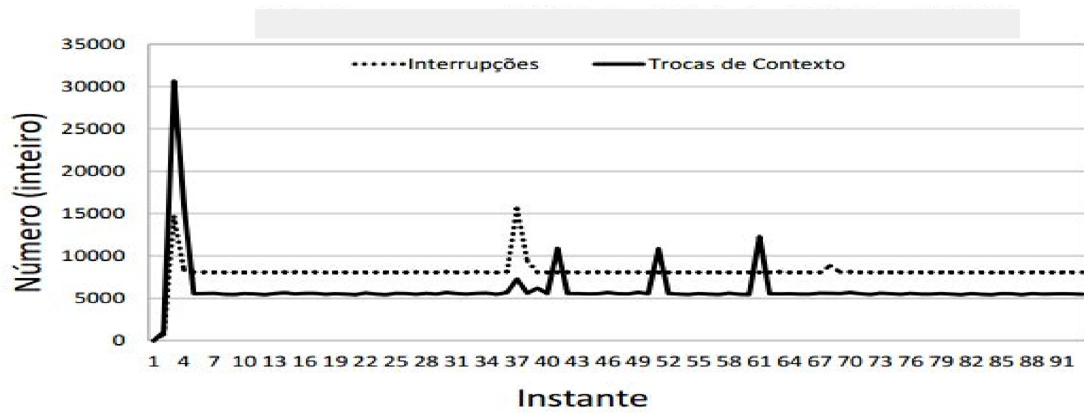


(a) MPI-1

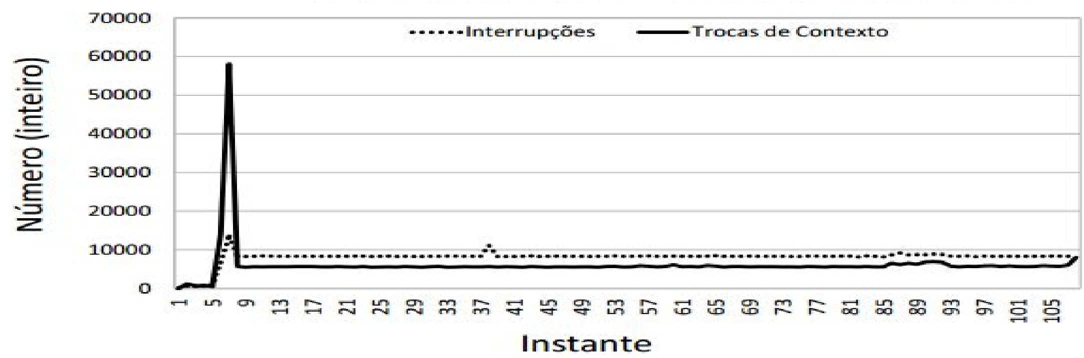


(b) MPI-2

Figura 65 – *Skyline Matrix Solver* - Trocas de Contexto e Interrupções 32 Processos 5120



(a) MPI-1



(b) MPI-2