UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DIEGO FABRICIO SCHLOSSER

AVALIAÇÃO DA INFLUÊNCIA DA QUANTIDADE DE NÚCLEOS DE PROCESSADORES NO DESEMPENHO DA COMPILAÇÃO DO KERNEL LINUX

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2015

DIEGO FABRICIO SCHLOSSER

AVALIAÇÃO DA INFLUÊNCIA DA QUANTIDADE DE NÚCLEOS DE PROCESSADORES NO DESEMPENHO DA COMPILAÇÃO DO KERNEL LINUX

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Paulo Lopes de Menezes

MEDIANEIRA
2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Avaliação da influência da quantidade de núcleos de processadores no desempenho da compilação do Kernel Linux

Por

Diego Fabricio Schlosser

Este Trabalho de Diplomação (TD) foi apresentado às 07:30 h do dia 20 de novembro de 2015, como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Manutenção Industrial, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. Os acadêmicos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. Dr. Paulo Lopes de Menezes
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Me. Hamilton Pereira da Silva
UTFPR – *Campus* Medianeira
(Convidado)

Prof.Me. Glória Patricia Lopez Sepulveda
UTFPR – *Campus* Medianeira
(Convidado)

Prof.Me. Jorge Aikes Junior
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Considerando este trabalho de diplomação como resultado de uma caminhada que começou na UTFPR, agradecer pode não ser tarefa fácil, nem justa. Para não correr o risco da injustiça, agradeço de antemão a todos que de alguma forma passaram pela minha vida e contribuíram para a construção de quem sou hoje.

Agradeço primeiramente a Deus por ter me guiado neste caminho.

E agradeço, particularmente, a algumas pessoas pela contribuição direta na construção deste trabalho:

À minha família pelo apoio, carinho e demais contribuições.

À minha noiva amada Bianca Colombari Peron, que me ajudou nesta etapa a ser concluída, não medindo esforços para me ajudar no que fosse possível.

À meu pai Sr. Vilmar Schlosser porque se não fosse por ele eu não teria entrado na UTFPR.

À minha mãe PhD. Marli Terezinha Szumilo Schlosser por ter insistido que eu prosseguisse nesta jornada acadêmica que esta sendo findada com este trabalho.

À minha avó Sr^{a}. Josefa Szumilo por ter me auxiliado financeiramente em momentos que precisei.

À meu avô Sr. Tadeu Szumilo por sempre estar ao meu lado me passando seus ensinamentos e experiências.

À meu irmão Dean Fael Schlosser por estar sempre me aconselhando.

Ao professor, Me. Nelson Miguel Betzek por ter iniciado como orientador deste trabalho, e ter contribuído para minha formação.

Ao professor Dr. Paulo Lopes de Menezes por ter aceitado dar continuidade como orientador deste trabalho, e pelos ensinamentos voltados a área.

Estendo meus agradecimentos aos demais professores e servidores da UTFPR pelo apoio institucional.

Aos amigos que fiz durante o decorrer do curso, os quais compartilharam comigo bons momentos.

RESUMO

SCHLOSSER, Diego Fabricio. Avaliação da influência da quantidade de núcleos de processadores no desempenho da compilação do Kernel Linux. 2015. 44 f. Trabalho de Diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira, 2015.

Sistema operacional pode ser definido como o conjunto de um ou mais programas que fornece um pacote de serviços, o qual cria uma interface entre aplicações e o *hardware* do computador, que aloca e gerencia recursos compartilhados entre múltiplos processos. Criado em 1991, na Finlândia por Linus Torvalds, o Linux é considerado um clone do Unix, é uma alternativa barata e funcional, destacando-se pelas seguintes vantagens: sistema multitarefa e multiusuário de 32 e 64 bits, sistema gráfico X-Window, suporte a diversas linguagens, como Java, C, C++, Pascal, Lisp, Prolog, entre outras, suporte aos protocolos de rede: TCP/IP, IPX, AppleTalk e NetBios, e memória virtual. Já o Kernel é o núcleo do próprio sistema operacional, código este que controla a interconexão entre os programas do usuário e os dispositivos de hardware, a programação de processos para obter as multitarefas e demais aspectos do sistema. A compilação de Kernel requer itens como: a instalação do compilador C (GCC); as bibliotecas C (libC); um espaço suficiente no disco rígido, aproximadamente 1GB e estar "logado" como root. A utilização do comando de *cores* possibilita acelerar o processo de compilação, sendo liberado a quantidade de *cores* de acordo com a capacidade do *hardware*. Com o objetivo de avaliar a influência da quantidade de núcleos de processadores no desempenho da compilação de Kernel, realizou-se testes em quadruplicata, variando a quantidade de núcleos (0, 2, 4, 6, e 8), para realização dos testes utilizou-se um Macbook-Pro da Apple, com sistema operacional OSX El Capitan versão 10.11, software Parallels Desktop para Mac. Os resultados dos testes foram organizados em tabelas e analisados estatisticamente pelo software Action 2.9. Os parâmetros analisados foram tempo de compilação, utilização de hardware e variação da temperatura ao longo do processo. Pode-se concluir ao final dos testes, que quanto mais *cores* ativos, maior a utilização de hardware, menor o tempo de compilação, porém maior era o aumento de temperatura.

Palavras-chave: Sistemas operacionais, Linux, Compilação de Kernel, núcleos de processador.

ABSTRACT

SCHLOSSER, Diego Fabricio. Evaluation of the influence of the amount of processor cores on the performance of compiling the Linux Kernel. 2015. 44 f. Trabalho de Diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira, 2015.

Operational system can be defined as the set of one or more programs that provides a package of services, which creates an interface between applications and computer hardware, which allocates and manages shared resources among multiple processes. Created in 1991 in Finland by Linus Torvalds, Linux is considered a clone of Unix, is an inexpensive and functional alternative, especially for the following advantages: multitasking system and multiuser 32 and 64 bits, X-Window graphics system, support several languages, such as Java, C, C ++, Pascal, Lisp, Prolog, among others, support for network protocols: TCP / IP, IPX, AppleTalk, and NetBIOS, and virtual memory. The kernel is the core of the operational system, this code controls the interconnection between user programs and hardware devices, the scheduling of processes to achieve multitasking and other aspects of the system. The kernel compilation requires items such as: the installation of the C compiler (GCC); the C libraries (LibC); enough hard disk space, about 1GB and be "logged" as root. The use of cores control allows speed up the build process, and releasing the amount of cores to suit the hardware capacity. In aim to evaluate the influence of the number of processor cores in Kernel compilation of performance test was carried out in quadruplicate, varying the amount of cores (0, 2, 4, 6, and 8). To perform the tests used a Macbook Pro, Apple's OSX operating system with El Capitan version 10:11, Parallels Desktop software for Mac. The test results were organized in tables and analyzed by the Action 2.9 software. The parameters analyzed were compile time, hardware utilization and temperature variation throughout the process. It can be concluded at the end of testing, that the most active cores, greater use the hardware utilization, lower compile time, but greater increase in temperature.

Keywords: Operating systems, Linux, Kernel Compilation, processor cores;

LISTA DE FIGURAS

LISTAS DE GRÁFICOS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Desde o primeiro computador projetado notava-se a necessidade de um sistema operacional. Até chegar ao que se tem hoje, os sistemas operacionais passaram por 4 gerações. Stuart (2011) define sistema operacional como conjunto de um ou mais programas que proporcionam serviços, o qual cria uma ligação entre as aplicações e o hardware do computador, que guarda e administra os recursos compartilhados entre múltiplos processos. Tanembaum (2009) afirma que a função dos sistemas operacionais é primeiramente fornecer aos programas do usuário um modelo de computador melhor, simples e limpo, bem como o gerenciamento de todos os recursos.

O Linux é um dos sistemas operacionais mais populares do mundo, uma versão gratuita distribuída do primeiro UNIX desenvolvido por Linus Torvalds na Finlândia, o desenvolvimento deste sistema operacional teve a contribuição de muitos programadores UNIX e especialistas da Internet. O Kernel Linux disponível para o Linux é desenvolvido pelo projeto GNU na *Free Software Foundation* (Fundação do *Software* Gratuito) em Cambrigde, Massachussetts, entretanto programadores do mundo todo contribuem para o crescimento do *software* Linux (WELSH; KAUFMAN, 1997).

O Kernel Linux é o sistema operacional propriamente dito, sobre o qual são executados os diversos aplicativos que acompanham cada distribuição (FERREIRA, 2008). A manipulação do Kernel Linux pode propiciar o melhor aproveitamento do *hardware*. O Kernel como núcleo do sistema operacional é a parte mais próxima do *hardware*, composta de chamadas ao sistema, acesso aos dispositivos de entrada e saída e de gerência dos recursos da máquina (FERREIRA, 2008).

Desta maneira propõem-se estudar o processo de compilação por meio de testes na distribuição Debian GNU/Linux, ou simplesmente Debian, com variação do número de *cores* do processador utilizado. Serão apresentados os resultados obtidos referentes ao desempenho e tempo de execução em testes aplicados com utilização de 2, 4, 6 e 8 *cores* do processador no processo de compilação.

A utilização dos *cores* para desenvolver o processo de compilação do Kernel será acionada por um comando específico durante o processo. A utilização do comando de *cores* possibilita acelerar o processo de compilação, liberando a quantidade de *cores* de acordo com a capacidade do *hardware*. Um comando é definido pelo sistema operacional como uma ordem, passada pelo usuário para executar uma determinada tarefa (SILVA, 2010).

O objetivo deste estudo é avaliar o desempenho e a redução do tempo de compilação do Kernel Linux, por meio da variação da quantidade de *cores* do processador, possibilitada pela utilização do comando de *cores*.

1.1 OBJETIVO GERAL

Avaliar o desempenho e a redução do tempo de compilação, por meio da utilização de um processo para otimização do Kernel Linux, utilizando controle de núcleos (*core* do processador).

1.2 OBJETIVOS ESPECÍFICOS

- Descrever os métodos utilizados para desenvolver o processo de compilação do Kernel.
- Realizar estudo sobre a variação de desempenho obtido pelo comando.
- Realizar testes com diferentes quantidades de cores do processador.
- Analisar estatisticamente o processo de compilação de Kernel.
- Apresentar propostas de melhorias no processo de compilação, em relação ao desempenho obtido por meio da utilização do controle de *cores*.
- Comparar e avaliar os resultados obtidos por meio de testes com diferentes quantidades de *cores*.

1.3 JUSTIFICATIVA

Sabendo-se que o processo de compilação de Kernel Linux, independente da distribuição, é um processo que exige tempo para ser executado. Notou-se a importância de realizar um estudo sobre o "comando" de controle dos *Core* dos processadores, o qual propõe reduzir o tempo de execução do processo de compilação e aprofundar por meio deste estudo, a utilização do controle de *Core* dos processadores para outras funcionalidades do *Software* livre. Itens como utilização do hardware e temperatura da máquina também foram avaliados ao longo do processo.

1.4 ESTRUTURA DO TRABALHO

Este trabalho é estruturado da seguinte maneira:

a. **Capítulo 1:** Neste capítulo é apresentada a introdução do trabalho, bem como seus objetivos e a justificativa para a sua realização.

b. **Capítulo 2:** Neste capítulo é apresentado no formato de revisão de literatura, os conceitos relativos aos sistemas operacionais, Linux, Debian, definição de processos, compilação de Kernel, e demais abordagens julgadas necessárias para o desenvolvimento deste trabalho.

c. **Capítulo 3:** Contém todos os materiais e métodos que foram utilizados no decorrer da elaboração deste trabalho.

d. **Capítulo 4:** Apresenta todas as técnicas e resultados do trabalho realizado.

e. **Capítulo 5:** Último capítulo deste trabalho tem como objetivo apresentar as considerações finais e sugestões para pesquisas futuras relacionadas a este tema.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são discutidos os principais temas de interesses deste trabalho, tais como: Sistemas operacionais, Linux, Debian, compilação de Kernel, bem como a definição de processos.

2.1 SISTEMA OPERANCIONAL

2.1.1 História dos sistemas operacionais

Os sistemas operacionais têm passado por um processo gradual de evolução.

O primeiro computador digital foi projetado pelo matemático inglês Charles Babbage (1792-1871). O inglês gastou parte da vida e fortuna para construir a ‘máquina analítica’, porém nunca conseguiu vê-la funcionar de maneira apropriada. A máquina era inteiramente mecânica e a tecnologia da época não contribuiu para produção de rodas, engrenagens e correias de alta precisão que eram necessárias. Baddage percebeu que seria preciso um software para sua máquina, então contratou uma jovem chamada Ada Lovelace, primeira programadora do mundo. A linguagem de programação Ada® foi denominada em sua homenagem (TANEMBAUM, 2009).

Até os dias de hoje a evolução dos sistemas operacionais passou por 4 gerações: a primeira durou de 1945 a 1955, com a utilização de válvulas, a segunda durou de 1955 a 1965, com a presença de transistores e sistemas em lote (batch), a terceira foi de 1965 a 1980, com a utilização de CIs (circuitos integrados) e multiprogramação. A quarta e atual surge a partir de 1980, com o desenvolvimento de microcomputadores pessoais (TANEMBAUM, 2009).

2.1.2 Definições de sistema operacional

O sistema computacional moderno é um sistema complexo, que consiste em um ou mais processadores, memória principal, discos, impressoras, teclado, mouse, monitor, interfaces de rede e outros dispositivos de entrada e saída. Por tanto, existe um dispositivo de software denominado sistema operacional (TANEMBAUM, 2009).

De acordo com Stuart (2011), um sistema operacional é o conjunto de um ou mais programas que fornece um pacote de serviços, o qual cria uma interface entre aplicações e o hardware do computador e que aloca e gerencia recursos compartilhados entre múltiplos processos.

O sistema operacional é a base do software onde as aplicações são executadas. Ele é responsável pelo gerenciamento e pela alocação do hardware do sistema, incluindo a CPU (*Central Processing Unit*), memória física e dispositivos de E/S (STUART, 2011).

Existem diversas maneiras de montar um sistema operacional completo. Alguns sistemas são estruturados como programas monolíticos convencionais, outros seguem o modelo em camadas, alguns se baseiam em microkernels e ainda existem os que são modelos de máquinas virtuais. O sistema operacional deve-se carregar na memória por um tipo de boot, após ele ser carregado e estar em execução está pronto para aceitar solicitações de serviços, processos por meio do mecanismo e chamadas ao sistema (STUART, 2011).

Segundo Tanembaum (2009), a grande maioria dos computadores possui dois níveis de operação: modo núcleo e modo usuário. O sistema operacional opera em modo núcleo, ou seja, ele tem acesso total a todo o *hardware* e pode executar qualquer instrução que a máquina suportar. O resto do *software* opera em modo usuário, no qual apenas um subconjunto de instruções da máquina está disponível.

2.2 LINUX

2.2.1 Origens do Linux

O UNIX criado no final da década de 1960 e início da década de 1970, é um sistema operacional multiusuário, permite que vários usuários utilizem o mesmo computador simultaneamente, por meio de terminais remotos, e também por um sistema operacional multitarefa, pois possibilita que vários programas sejam executados ao mesmo tempo. Além disso, apresenta inúmeras possibilidades relacionadas a rede, como sistema de cota de fisco, FTP, e-mail, WWW, DNS, diferentes níveis de acesso, execução de programas em background etc. (FERREIRA, 2008).

A partir do UNIX, como resposta a interrupção de publicação do mesmo, as universidades decidiram criar seus próprios sistemas operacionais, foi quando Andrew Tanenbaum criou o MINIX, lançado em 1987, foi desenvolvido para ser executável na maioria dos hardwares da IBM na época. Com a detenção dos direitos de distribuição pela Prentice Hall, o MINIX alcançou sucesso e tornou-se um sistema educacional, e teve algumas extensões criadas para ele, como não era gratuito não teve seu desenvolvimento amplamente divulgado (STUART, 2011).

Em 1991, na Finlândia Linus Torvalds, comprou o livro e o S.O do MINIX e desenvolveu um projeto, com dois objetivos, aprofundar-se sobre como utilizar recursos do processador 386 da Intel que o MINIX utilizava e desenvolver um emulador de terminal que lhe permitisse utilizar seu computador para conectar-se aos maiores computadores da universidade para utilizá-los (STUART, 2011).

No decorrer do desenvolvimento do projeto Linus detectou que o emulador de terminal executado sem nenhum sistema operacional de suporte estava começando assumir muito da funcionalidade de um sistema operacional, foi onde o objetivo do projeto se tornou o desenvolvimento de um sistema operacional (STUART, 2011).

A disponibilização do seu primeiro sistema operacional gratuitamente ocorreu em 1991 na internet. Conforme o desenvolvimento do sistema, aumentava-se as ambições e objetivos. Em 1994 foi lançado a versão 1.0 do Linux, sistema que passou de um projeto pessoal para um projeto coletivo, com grande número de colaboradores envolvendo pessoas do mundo todo. O sistema tornou-se base para diversas empresas comerciais, deu origem a

milhões de linhas de código, foram lançadas inúmeras revisões e disponibilizadas diversas distribuições (STUART, 2011).

O Linux é considerado um clone do UNIX, criado como alternativa barata e funcional para quem não está disposto a pagar o alto preço de um sistema UNIX comercial ou não possui um computador suficientemente rápido (FERREIRA, 2008).

Os programas do Linux foram juntados em “pacotes”, chamados de distribuições, por empresas e organizações voluntárias, aos quais os mesmos fornecem suporte. Entre as mais famosas e utilizadas distribuições destacam-se: *RedHat Enterprise / Fedora Core, Madriva, Debian, Slackware e Suse* (FERREIRA, 2008).

Dentre as vantagens que o Linux oferece, Ferreira (2008) destaca: sistema multitarefa e multiusuário de 32 e 64 bits, sistema gráfico X-Window, suporte a diversas linguagens, como Java, C, C++, Pascal, Lisp, Prolog, entre outras, suporte aos protocolos de rede: TCP/IP, IPX, AppleTalk e NetBios, e memória virtual.

2.2.2 Arquitetura básica

Em resumo, o Linux é um kernel monolítico convencional com suporte para módulos carregáveis.

A Figura 1 apresenta a arquitetura conceitual do Linux:

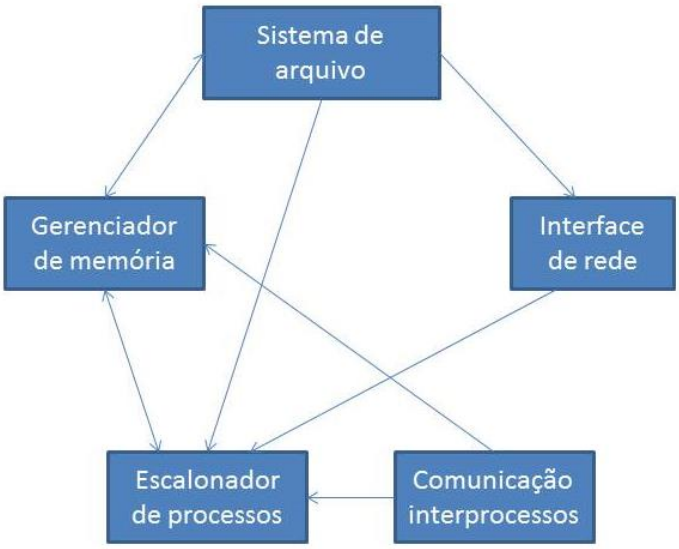


Figura 1 - Arquitetura conceitual do Linux
Fonte: Stuart (2011).

De acordo com Stuart (2011), as caixas na Figura 1 representam os principais subsistemas do Kernel do Linux. Uma seta saindo de uma caixa para outra indica a dependência uma da outra. Cada subsistema representa uma parte das responsabilidades gerais do sistema operacional.

- Escalonador (Scheduler) de Processo: Realiza a troca de contexto e suporta o escalonamento para o gerenciamento do procedimento. Subsistema bem centralizado, mesmo sendo relativamente pequeno.
- Gerenciador de memória: É responsável por alocar memória entre vários processos do sistema. Ele também lida com os casos em que a quantidade de memória exigida é maior que a fisicamente disponível.
- Sistema de arquivo: O Linux suporta uma ampla variedade de layouts de sistema de arquivo, esse subsistema lida com as funções esperadas para gravar e acessar arquivos em um dispositivo de armazenamento.
- Interface de rede: Fornece vários protocolos que são suportados pelo Linux.
- Comunicação inter-processos: Permite que processos troquem dados entre si. Há três mecanismos que suportam essa troca: operações com troca direta de mensagens, áreas de memória compartilhada e primitivas que suportam sincronização entre processos.

2.3 DEBIAN

O Debian é um sistema operacional livre, logo respeita a liberdade dos usuários, comparado com uma pirâmide, onde a base é o Linux e no topo estão todas as ferramentas básicas disponibilizadas pelo GNU, no restante estão todas as aplicações de *software*. Os desenvolvedores do Debian atuam como arquitetos e coordenadores cuidadosamente organizando o sistema e o mantendo integrado e atualizado (GOERZEN; OTHMAN, 2001).

O Projeto Debian é formado por grupo mundial de voluntários, que se dedicam para produzir um sistema operacional livre, inteiramente composto por software livre. O produto principal do projeto é a distribuição Debian GNU/Linux, que inclui o kernel do sistema operacional Linux e centenas de aplicações pré-empacotadas. Vários tipos de processadores são suportados, incluindo o Intel i386 e superior, Alpha, ARM, Motorola 68k, MIPS, PowerPC, Sparc, e UltraSparc, HP PA-RISC, IBM S/390 e Hitachi SuperH (GARBEE, 2002).

O Projeto Debian foi fundado por Ian Murdock em 16 de Agosto de 1993. Na época, o conceito de uma "distribuição" de Linux era novo, Ian pretendia que o Debian fosse uma distribuição criada abertamente, no mesmo espírito do Linux e do GNU. A criação do Debian teve o apoio do projeto GNU da FSF durante um ano (GARBEE, 2002).

A distribuição Debian pretendia ser cuidadosamente e conscientemente criada em conjunto, ser mantida e suportada com similar cuidado. Iniciou-se tudo a partir de um pequeno e forte grupo de hackers do Software Livre e cresceu gradualmente, se tornando uma grande e bem organizada comunidade de desenvolvedores e usuários. A Debian é a única distribuição que é aberta para que todo desenvolvedor e usuário possa contribuir com seu trabalho, distribuidor significativo de Linux que não é uma entidade comercial, grande projeto com uma constituição, um contrato social e documentos com políticas para organizar o projeto. A Debian também é a exclusiva distribuição que é micro-empacotada, usando informações detalhadas de dependência de pacotes para garantir a consistência do sistema em atualizações (GARBEE, 2002).

O Debian GNU/Linux possui suporte a língua portuguesa, sendo privilegiado por conter 14 arquiteturas diferentes, entre elas pode-se citar: i386, IA64, AMD64, Alpha, Sparc, PowerPc, Macintosh, Arm, ainda possuindo 15 sub-arquiteturas (SILVA, 2010).

2.4 PRINCÍPIOS DE GERENCIAMENTO DE PROCESSOS

Processos são uma das mais antigas e importantes abstrações que o sistema operacional oferece. São eles quem mantem a capacidade de operações (pseudo) concorrentes, mesmo quando há apenas uma CPU disponível. Os processos transformam uma única CPU em múltiplas CPUs virtuais. Sem a existência de processos não existiria a ciência da computação moderna (TANEMBAUM, 2009).

Um sistema operacional gerencia os recursos da CPU, por meio do gerenciamento dos processos que a utilizam. O processo é algo intangível, pode-se também afirmar que processo é a sequência de configurações que uma máquina percorre durante determinada computação (STUART, 2011).

De acordo com Tanembaum (2009), um processo é basicamente um programa em execução. Associado a cada processo está seu espaço de endereçamento, uma lista de posições de memória, que vai de 0 até o máximo. Portanto, esse processo pode ler e escrever, também associado ao processo está um conjunto de recursos, normalmente incluindo registradores, uma lista dos arquivos abertos, alarmes pendentes, listas de processos relacionados e todas as demais informações necessárias para executar um programa. Um processo também pode ser definido como um contêiner que armazena todas as informações necessárias para executar um programa.

2.4.1 Gerenciamento de processos no Linux

Os processos no Linux são criados de maneira bem simples. A chamada de sistema *fork* cria uma cópia exata do processo original. O processo criador é chamado de processo pai. O novo processo é chamado de processo filho. Cada um tem a sua própria imagem na memória privada, se após a criação, o pai alterar suas variáveis, essas alterações não serão visíveis pelo processo filho e vice-versa (TANEMBAUM, 2009).

2.5 KERNEL

O Kernel é o centro do próprio sistema operacional, é o código que controla a interface entre os programas do usuário e os dispositivos de hardware, a programação de processos para obter as multitarefas e muitos outros aspectos do sistema. O Kernel não é um processo separado executado no sistema, ele pode ser considerado como um conjunto de rotinas, constantemente na memória, que cada processo tem que acessar (WELSH; KAUFMAN, 1997).

O Kernel do Linux é conhecido como um Kernel monolítico, ou seja, todos os drivers do dispositivo fazem parte do próprio Kernel. Ele é capaz de emular sozinho as instruções FPU 387, para que os sistemas sem um coprocessador matemático possam executar programas que requerem as instruções matemáticas como ponto flutuante. Suporta os executáveis carregados com solicitação de páginas, ou seja, apenas os segmentos de um programa de fato usados, serão lidos na memória a partir do disco. O Kernel também programa um pool de memória único para os programas do usuário e o cache de disco, assim toda a memória livre será usada para o cache e ele será reduzido ao executar programas grandes (WELSH; KAUFMAN, 1997).

2.5.1 Compilação de Kernel

O Kernel do Linux é o sistema operacional propriamente dito, nele são executados diversos aplicativos. O Kernel possui módulos de *hardware* e *software* que podem ser carregados ou descarregados da memória de acordo com a necessidade, ou o *hardware* presente no computador. A compilação de Kernel requer os seguintes itens: a instalação do compilador C (GCC); as bibliotecas C (libC); um espaço suficiente no disco rígido, aproximadamente 1GB e estar "logado" como root (FERREIRA, 2008).

Existem duas maneiras de utilização do Kernel, a primeira é incluindo *drivers* de *hardware* e subsistemas de *software*, neste caso o crescimento do Kernel é diretamente proporcional à quantidade de dados inclusos, chamado de monolítico. A segunda maneira é conhecida como modular, é possível gerar drivers de dispositivos de *hardware* e subsistemas de *software* na forma de módulos, programa que o Kernel pode carregar na memória após

entrar em execução. Os módulos são comumente utilizados para acrescentar suporte a um dispositivo sem recompilar o Kernel (FERREIRA, 2008).

2.5.2 Ferramentas de compilação e desenvolvimento

No caso de utilizar um ambiente similar ao UNIX para realizar a compilação, é bastante provável que as ferramentas corretas estejam à disposição. Na maioria dos casos, os mkfiles esperam que o comando cc seja o compilador correto, nos sistemas de código aberto, esse comando chamará o compilador gcc. O mkfile do HP-UX espera que o compilador se chame c89. Para o NT (e outros sistemas do Microsoft Windows baseados em NT), é necessário compilador Microsoft Visual C (STUART, 2011).

2.6 *CENTRAL PROCESSING UNIT* (CPU)

A CPU, unidade central de processamento, é a parte do computador que controla o processamento de dados, desde sua entrada até a sua saída. Quando um processo começa, é a CPU que busca a instrução no disco rígido, faz a decodificação e executa cada instrução do programa. Os principais fabricantes de processadores são a Intel e AMD (GLÓSSARIO TÉCNICO, 2015).

A forma e a implementação de CPUs tiveram alterações ao longo de sua história, mas sua operação fundamental permanece quase inalterada. Os componentes principais de uma CPU incluem a Unidade Lógica Aritmética (ALU), que realiza operações aritméticas e lógicas. Os registradores do processador fornecem operandos a ALU e armazenam os resultados das operações, e a unidade de controle é quem busca as instruções da memória e as executa, dirigindo as operações coordenadas da ALU, registos e outros componentes.

A maioria dos processadores modernos são na verdade microprocessadores , o que significa que estão contidos em um único circuito integrado (IC) de chip. Um IC que contém uma CPU pode também conter memória, as interfaces periféricas, e outros componentes de um computador; tais dispositivos integrados são chamados de microcontroladores ou sistemas

em um chip (SoC). Alguns computadores utilizam um processador multi-core, ou seja, um único chip contendo dois ou mais CPUs chamados de "núcleos".

2.6.1 Diferença entre processadores

O que diferencia um processador do outro é basicamente a capacidade de processamento, ou seja, quantos cálculos por segundo ele pode fazer. Essa capacidade é medida em Hertz (Hz), dessa forma, um processador de 100 Hz, pode fazer 100 cálculos por segundo, atualmente, a maioria dos computadores novos vem com processadores na casa dos GHz (ANDRADE, 2015)**.**

O clock interno (ou apenas clock) serve justamente a este fim, ou seja, basicamente, atua como um sinal para sincronismo. Quando os dispositivos do computador recebem o sinal de executar suas atividades, dá-se a esse acontecimento o nome de "pulso de clock". Em cada pulso, os dispositivos executam suas tarefas, param e vão para o próximo ciclo de clock (ALECRIM, 2015).

É importante destacar, que se dois processadores diferentes - um da Intel e outro da AMD, por exemplo - tiverem clock interno de mesmo valor - 3,2 GHz, por exemplo -, não significa que ambos trabalham com a mesma velocidade. Cada processador possui um projeto distinto e conta com características que determinam o quão rápido podem ser. Assim, um determinado processador pode levar, por exemplo, 2 ciclos de clock para executar uma instrução, enquanto outro processador, esta mesma instrução pode requerer 3 ciclos (ALECRIM, 2015).

Os bits de um processador também interferem no seu desempenho, ou seja, quanto mais bits internos o processador possuir, mais rapidamente ele poderá fazer cálculos e processar dados em geral, dependendo da execução a ser feita. Isso acontece porque os bits dos processadores representam a quantidade de dados que os circuitos desses dispositivos conseguem trabalhar por vez (ALECRIM, 2015).

A memória cache consiste em uma pequena quantidade de memória SRAM embutida no processador. O trabalho da memória cache, também compromete o desempenho do processador, de nada serve possuir um processador rápido se a memória for lenta. Quando este precisa ler dados na memória RAM, um circuito especial chamado "controlador de cache" transfere blocos de dados muito utilizados da RAM para a memória cache. Assim, no

próximo acesso do processador, este consultará a memória cache, que é bem mais rápida, permitindo o processamento de dados de maneira mais eficiente (ALECRIM, 2015).

A potência dos processadores também é um fator que os diferencia, pois a produção de energia está associada com a quantidade que o processador possui. Quanto mais eletricidade for utilizada, maior será o calor resultante, por isso normalmente os dissipadores são acompanhados de ventiladores (cooler), esse dispositivo tem a função de amenizar o intenso calor gerado pela potência (ALECRIM, 2015).

Usuários normalmente possuem interesses distintos, por isso existem diversos modelos de processadores. Um processador Intel i3, por exemplo, são para os menos exigentes, traz dois núcleos de processamento, parecem fracos, mas vieram para substituir a antiga linha Core2Duo. Atualmente os processadores de múltiplos núcleos são predominantes, assim a Intel decidiu criar modelos que pudessem simular uma quantia ainda maior de núcleos. Se considerar que os CPUs da linha Core i3 possuem apenas dois núcleos, pode-se imaginar que eles não durem muito. Contudo, com a utilização da Intel Hyper-Threading, os processadores i3 “ganham” dois núcleos a mais. Hyper-Threading é ideal para momentos em que se precisa efetuar várias atividades simultaneamente. Essa tecnologia serve para que um núcleo consiga realizar duas atividades ao mesmo tempo, daí o motivo pelo qual a tecnologia, supostamente, faz os núcleos dobrarem em quantidade (JORDÃO, 2010).

O Intel Core i5, disponível em modelos de dois ou quatro núcleos, é encarregado de suprir as necessidades do mercado de porte intermediário, ou seja, aqueles mais exigentes que realizam tarefas mais pesadas. Possui tecnologia Intel *Hyper-Threading*, tecnologia Turbo *Boost* e muito mais. A tecnologia Turbo *Boost* da Intel promete aumentar a velocidade do processador automaticamente, esta tecnologia é inteligente e trabalha 100% do tempo verificando frequência, voltagem e temperatura do processador. Ao notar uma baixa em um dos valores-padrão utilizados pelo CPU, este novo recurso aumenta a frequência e consegue um desempenho muito maior em qualquer aplicação (JORDÃO, 2010).

O i7 é o que se tem por último em tecnologia de processamento, esta linha de processadores, é voltada ao público entusiasta e profissional, traz diversos benefícios e especificações excelentes. Todos os CPUs da série Core i7 possuem quatro núcleos (o i7-980X possui seis núcleos), memória cache L3 de 8 MB, controlador de memória integrado, tecnologia Intel Turbo *Boost*, tecnologia Intel *Hyper-Threading*, tecnologia Intel HD *Boost* e ainda o recurso Intel QPI (JORDÃO, 2010).

3 METODOLOGIA E TECNOLOGIAS UTILIZADAS

Nesta seção, é apresentada a metodologia deste trabalho, ou seja, a forma de como foram realizados os testes de compilação, bem como as tecnologias utilizadas.

3.1 METODOLOGIA

A preferência pelo sistema operacional Linux foi de livre escolha, este por sua vez tem à disposição várias distros Linux de acordo com suas características e funcionalidades, o processo de compilação de Kernel otimizado pode ser aplicado a qualquer distribuição Linux.

A compilação de Kernel Linux é um processo que requer demanda de tempo, foi criada uma forma simplificada para acelerar o processo, onde se utilizou o *hardware* disponível para fazê-lo.

O processo para acelerar a compilação de Kernel consistiu na utilização do hardware disponível, entretanto a forma que se desenvolveu trouxe como optativa a quantidade de núcleos do processador, no qual a quantidade de núcleos escolhidos era responsável por desenvolver especificamente o processo de compilação do Kernel.

Primeiramente realizou-se a coleta de dados de temperatura, tempo e processamento de hardware com a máquina ligada, somente com o sistema operacional, posteriormente foi realizada a coleta dos dados com a máquina ligada e com a máquina virtual carregada.

A instalação da Distro Linux Debian 2.30.2 ocorreu utilizando-se do pacote de Kernel Headers de acordo com a versão da Distro escolhida. O pacote de Kernel Headers era composto por um conjunto de arquivos e ponteiros, necessários para que o compilador fosse capaz de gerar módulos adequados ao Kernel em uso.

Iniciou-se o processo de compilação do Kernel, navegando na raiz de arquivos do sistema descompactou-se no diretório “/usr/src” o pacote de Kernel Headers. Acessou-se o mesmo “cd /usr/src/linux-source-3.2-rc4” e executou-se o comando “makemenuconfig” com objetivo de otimizar o Kernel para versão personalizada de acordo com o hardware disponível, como apresentado na Figura 2.

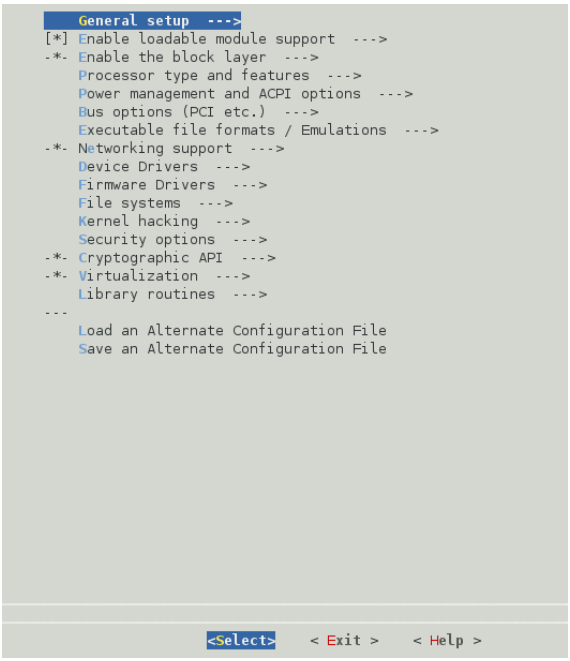


Figura 2 - Menu de configuração do Kernel
Fonte: Autoria própria.

Após configuração do novo Kernel iniciou-se o processo de compilação de fato, a linha de comando “time make-kpkgkernel_imagekernel_sourcekernel_headers”, foi executada com objetivo de cronometrar o processo através da inclusão do “time” no contexto do comando.

Na primeira fase de teste de compilação, o sistema Linux Debian 2.30.2 controlava o processamento da compilação do Kernel, adicionou-se o pacote de Kernel Headers Linux-source-3.2-rc4, utilizou-se a configuração default do processo de compilação, como apresentado na Figura 3.

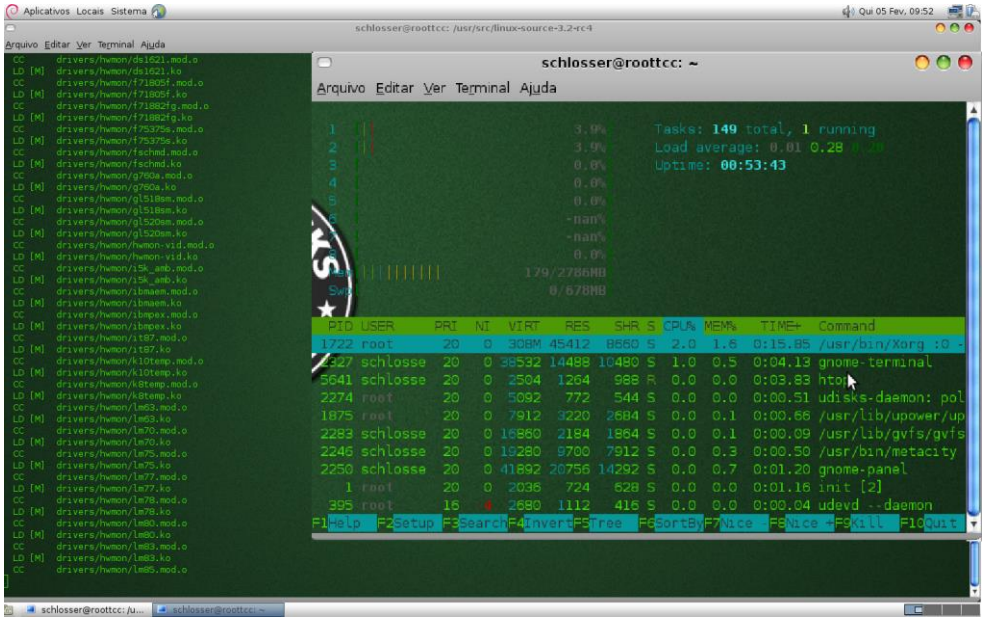


Figura 3 - Processo de compilação de Kernel default.
Fonte: Autoria própria.

Conforme segue na figura 3, o processamento do novo Kernel Linux-source-3.2-rc4 observa-se a utilização de padrão do hardware para desenvolver o processo, contudo o sistema gerenciou por default a utilização do hardware disponível.

Entre cada fase de teste foi realizada uma limpeza do cache da memória e buffers do Linux a fim de se obter resultados mais precisos, para este procedimento utilizou-se o comando “sudo sync && sudo sysctl -w vm.drop_caches=3 && sudo sysctl -w vm.drop_caches=0” , o comando sync assegurou que os dados pendentes no sistema de cache fossem escoados, o comando sysctl -w vm.drop_caches=3 limpava o cache na memória e posteriormente o comando sysctl -w vm.drop_caches=0 reiniciava o drop cache. Para finalizar foi utilizado o comando Free –m o qual apresentava os dados relacionados à utilização do cache (PRACIANO, 2015).

Para fins estatísticos estabeleceu-se 4 repetições a cada fase de teste, na coleta dos dados de: temperatura, tempo e uso de hardware. A coleta de dados da primeira fase do teste ocorreu no tempo de 1 minuto, em seguida nos tempos 5, 10 e 30 minutos.

A segunda fase do teste de compilação do Kernel Linux-source-3.2-rc4 procedeu de forma peculiar, com apenas uma alteração no padrão da configuração, onde se adicionou uma linha de código na prévia do comando “makemenuconfig”.

Na linha de código se especificou a quantidade de núcleos do processador que era responsável pelo processo de compilação, a estrutura do comando seguiu como: “export CONCURRENCY_LEVEL= “quantidade de cores”.

A segunda fase de teste, implementada utilizou-se do comando de otimização com 2 *cores* do processador “time make-kpkgkernel_imagekernel_sourcekernel_headers export CONCURRENCY_LEVEL=2”, para que fosse possível desempenhar o processo, segue na Figura 4.

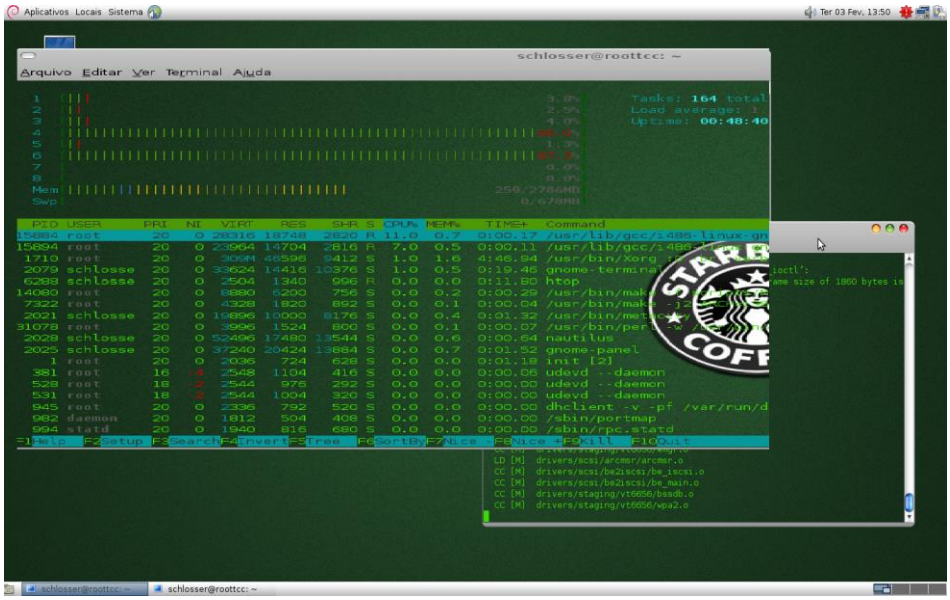


Figura 4 - Processo de compilação utilizando 2 núcleos.
Fonte: Autoria própria.

Através da utilização do comando de otimização, observou-se mudança significativa na utilização do hardware por parte do sistema operacional, para desempenhar o processo de compilação do Kernel. Na segunda fase de teste o comando disponibiliza 2 *cores* do processador, para executar o processo conforme exemplificado na Figura 4.

Os dados de temperatura, tempo e consumo de hardware da segunda fase do teste foram coletados no tempo de 1 minuto, em seguida aos 5, 15, 25 minutos. Com repetição de 4 vezes do mesmo teste. Posteriormente aos testes, novamente executaram-se os comandos de limpeza do cache da memória e buffers do sistema.

Na terceira fase efetuou-se o aumento da utilização do hardware, manteve-se o mesmo processo de limpeza de memória e configuração de Kernel da segunda fase, porém

alterou-se a quantidade de núcleos a serem utilizados. Neste processo disponibilizou-se 4 cores do processador para o desenvolvimento do processo de compilação de Kernel, como apresentado na Figura 5.

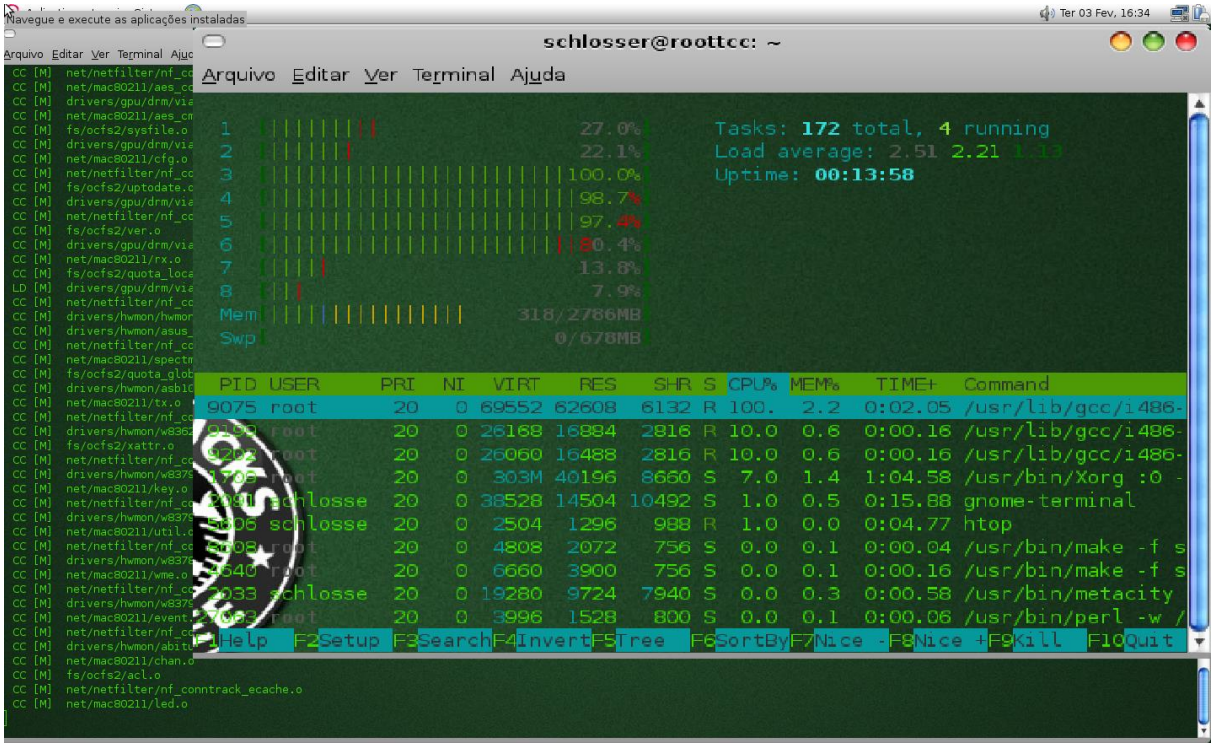


Figura 5 - Processo de compilação de Kernel com 4 núcleos.
Fonte: Autoria própria.

No decorrer do processo de compilação de Kernel com a utilização de 4 núcleos, o processo ocorreu de forma contínua, visível na Figura 5. A coleta de dados de temperatura, tempo e consumo de hardware da terceira fase dos testes ocorreu no tempo de 1 minuto, em seguida aos 5, 10, 15 minutos. Após testes desta etapa, executou-se os comandos de limpeza do cache da memória e buffers do sistema com a finalidade de garantir a integridade da coleta de dados.

Na quarta fase do teste de compilação de Kernel foram optados 6 núcleos do processador, mantendo a configuração da compilação de Kernel e a limpeza de memória *swap* e *buffer* utilizada no intercalar dos testes, apresentado na Figura 6 .

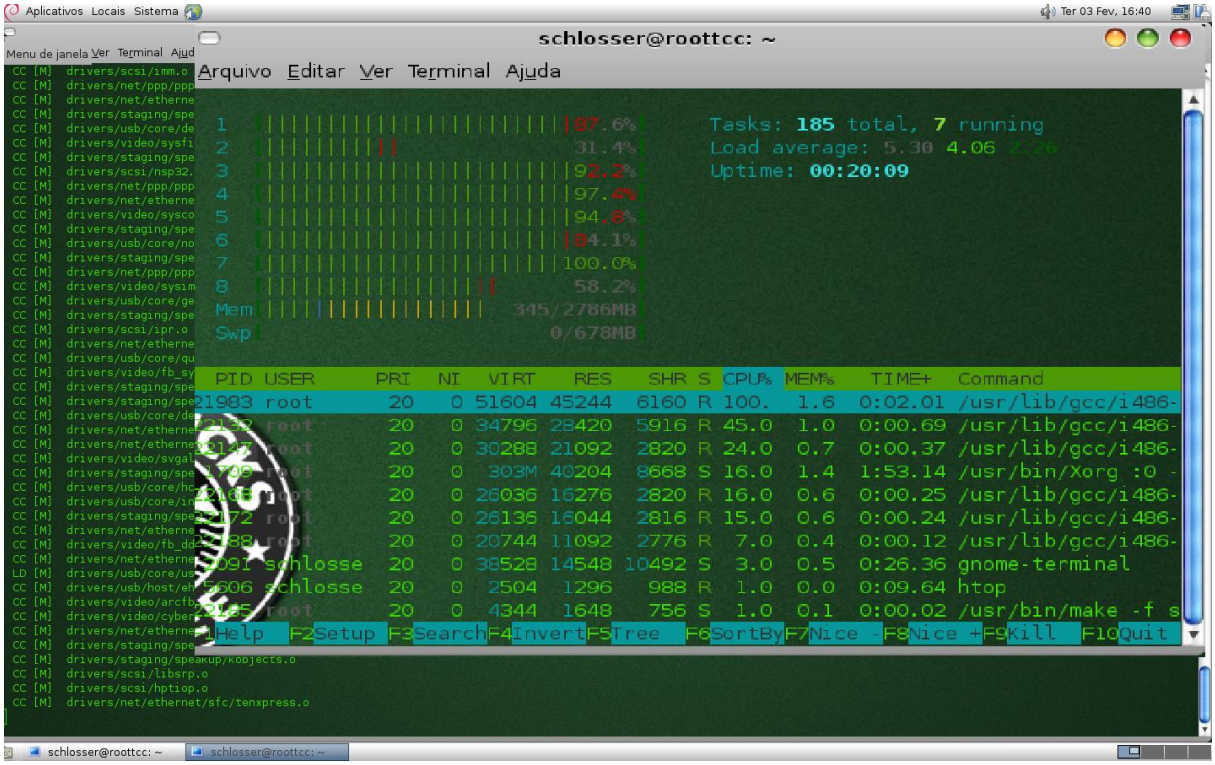


Figura 6 - Processo de compilação de Kernel utilizando 6 núcleos.
Fonte: Autoria própria.

O aumento da quantidade de núcleos expôs o hardware a um stress diferenciado, o uso de 6 cores do processador para efetuar o processo de compilação exigiu da máquina um esforço físico considerável. No decorrer dos 4 testes da quarta fase, efetuou-se a coleta de dados para fins estatísticos, no tempo de 1 minuto, e na sequência 5 ,10,e 15 minutos.

Findado os testes de cada fase executaram-se os comandos de limpeza do cache da memória e buffers do sistema operacional.

A última fase de teste, teve como utilização máxima de cores do processador em uso, utilizou 8 cores para desempenhar o processo de compilação de Kernel Linux, a capacidade de cada núcleo atingiu quase 100%. Segue apresentado na Figura 7.

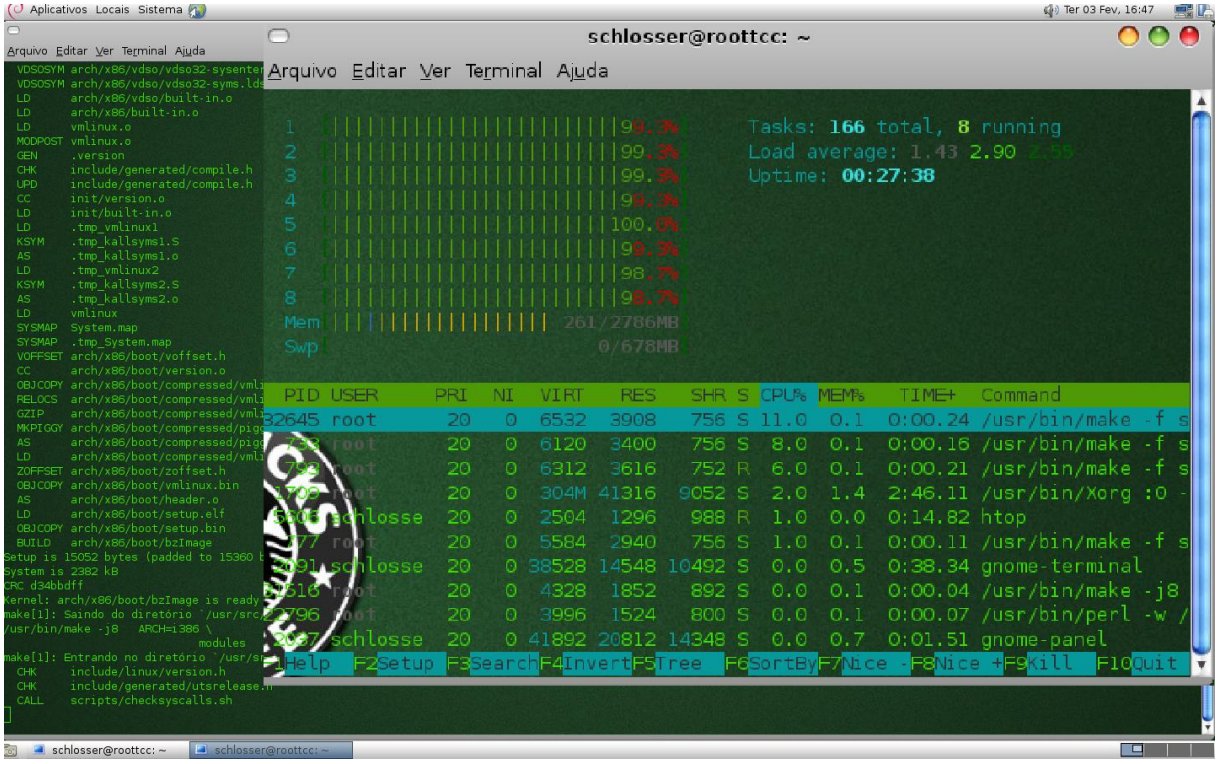


Figura 7 - Processo de compilação Kernel utilizando 8 núcleos.
Fonte: Autoria própria.

Nesta condição o *hardware* sendo utilizado ao máximo para desempenhar o processo de compilação, tem-se obtido resultado significativo em termos de desempenho, bem como na diminuição considerável no tempo de execução do processo de compilação. Efetuou-se o acompanhamento durante as 4 fases de testes para obter dados de temperatura, tempo de consumo de hardware, a coleta ocorreu ao tempo de 1 minuto, em seguida em 3, 4, 5 minutos.

3.1.1 Análise estatística

Para a análise dos dados obtidos utilizou-se o software Action.

3.2 TECNOLOGIAS UTILIZADAS

3.2.1 Parallels Desktop

Lançado em junho de 2006, é o primeiro produto de *software* a trazer a virtualização *mainstream* para computadores Macintosh que utilizam a arquitetura da Apple-Intel (produtos de software anteriores correu software PC em um ambiente emulado). Em janeiro de 2007, o Parallels Desktop 3.0 para Mac recebeu o prêmio "*Best in Show*" na MacWorld 2007.

Parallels Desktop para Mac é um *software* de virtualização de *hardware* de emulação, utilizando *hypervisor*, tecnologia que funciona mapeando os recursos de *hardware* do computador *host* diretamente para os recursos da máquina virtual.

Cada máquina virtual opera de forma idêntica a um computador autônomo, com praticamente todos os recursos de um computador físico. Como todas as máquinas virtuais convidadas a usar os mesmos *drivers* de *hardware*, independentemente do *hardware* real no computador *host*, a instância da máquina virtual é altamente compatível entre computadores.

O *software* Parallels Desktop está disponível na versão 11 para Mac, última versão disponibilizada pela empresa. A versão utilizada para os testes deste trabalho foi a versão 9 (Build 9.0.24172), oficialmente anunciado em 29 de agosto de 2013 e lançado em 05 de setembro de 2013 como apresentado na Figura 8.

Figura 8 - Parallels Desktop 9.
Fonte: Autoria própria.

3.2.2 Configuração de *hardware* e *software*

Para o desenvolvimento dos testes deste trabalho utilizou-se um Macbook-Pro da Apple, com as configurações de *hardware* do produto descritos na Figura 9.

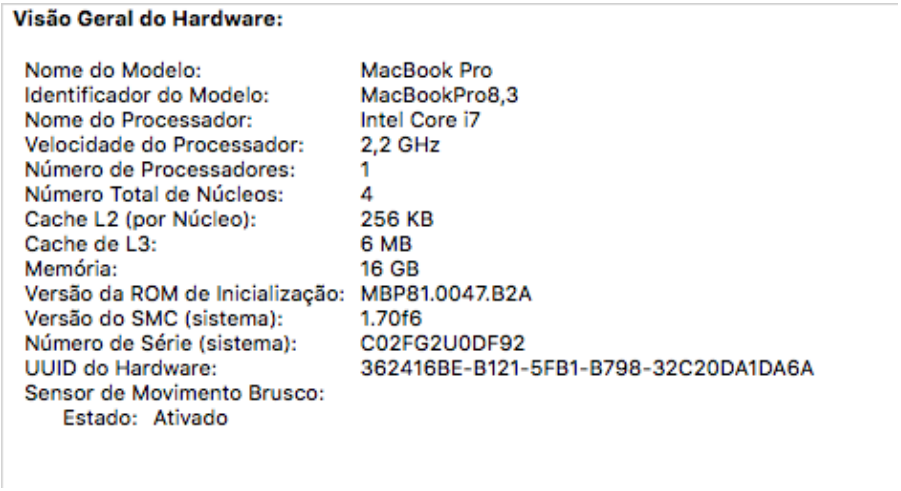


Figura 9 - Detalhamento do *hardware* do Macbook-Pro utilizado.
Fonte: Autoria própria.

O Sistema Operacional utilizado foi o OSX El Capitan versão 10.11. A Figura 10 apresenta a descrição do processador Intel Core i7 2,2 GHz, Memória Ram 16G 1333 DDR3, para gráficos Intel HD Graphics 3000 512Mb memória dedicada.

Figura 10 - Descrição do sistema operacional OSX El Capitan.
Fonte: Autoria própria.

O disco utilizado na máquina foi um HD SSD Corsair Force GS de 240 G, segue na Figura 11 a descrição do disco.

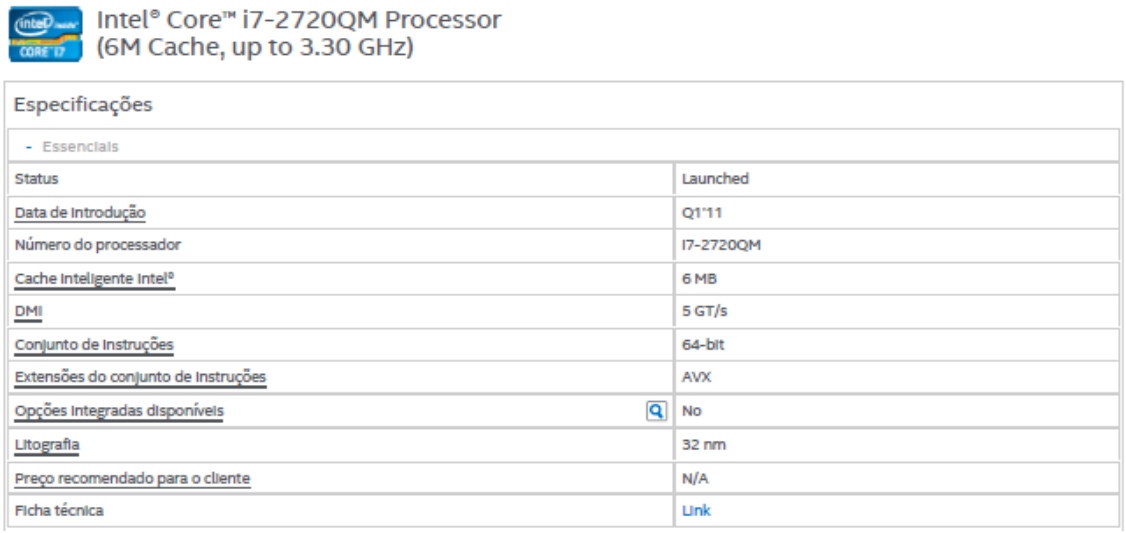
Nome da Mídia:	Corsair Force GS Media
Tamanho:	239,2 GB (239.197.536.256 bytes)
Tipo de Mídia:	SSD
Protocolo:	SATA
Interno:	Sim
Tipo de Mapa de Partição:	GPT (Tabela de Partição GUID)
Estado:	Online
Estado S.M.A.R.T.:	Verificado
PV UUID:	EFE58651-44FC-4546-821A-850AFA62E34F

Figura 11 - Configurações do *Hard Drive*.
Fonte: Autoria própria.

3.2.3 Unidade central de processamento

O MacBook Pro de 15 polegadas modelo 2011, equipado com um processador de 32 nm, Intel "Core i7" 2.2 GHz modelo (2720QM), com quatro núcleos independentes de processadores sobre um único chip de silício.

Baseado na arquitetura "Sandy Bridge", oferecia *Hyperthreading* para controlar 8 *threads* simultaneamente (para um melhor uso do *pipeline*). As melhorias notáveis nesta geração são: as novas instruções 265Bit AVX, o Turbo 2.0 e a integração da placa de vídeo no núcleo do CPU, desta forma compartilhava *cache* nível 3 com os núcleos do CPU (usando a nova tecnologia *ring bus*) e funcionava com uma frequência de 650-1300MHz (Turbo *Boost*). Segue especificações do processador disponibilizadas pela Intel nas Figuras 12 e 13.



Intel® Core™ i7-2720QM Processor
(6M Cache, up to 3.30 GHz)

Especificações

- Essenciais	
Status	Launched
Data de Introdução	Q1'11
Número do processador	i7-2720QM
Cache Inteligente Intel®	6 MB
DMI	5 GT/s
Conjunto de Instruções	64-bit
Extensões do conjunto de Instruções	AVX
Opções Integradas disponíveis	No
Litografia	32 nm
Preço recomendado para o cliente	N/A
Ficha técnica	Link

Figura 12 - Especificações essenciais do processador de acordo com o fabricante
Fonte: Intel (2015).

- Desempenho	
Número de núcleos	4
Nº de threads	8
Frequência baseada em processador	2.2 GHz
Frequência turbo max	3.3 GHz
TDP	45 W
- Especificações de memória	
Tamanho máximo de memória (de acordo com o tipo de memória)	32 GB
Tipos de memória	DDR3 1066/1333/1600
Nº máximo de canais de memória	2
Largura de banda máxima da memória	25,6 GB/s
Compatibilidade com memória ECC ‡	No

Figura 13 - Especificações de desempenho e memória do processador de acordo com o fabricante
Fonte: Intel (2015).

A tecnologia Intel de processadores Intel Sandy Bridge é chamada de segunda geração da família Intel *Core.* A Intel foi capaz de alcançar com a nova arquitetura em base de 32 nanômetros, melhorias significativas na eficiência em economia de energia. A nova função *Vector Extensions* Avançado (AVX) está em aplicativos de ponto flutuante (FPU), que preveem um processamento mais rápido de instruções correspondente a um aumento significativo na performance.

Os processadores Sandy Bridge tem um núcleo gráfico integrado Intel HD *Graphics* 3000, utilizando uma frequência gráfica de 650 MHz. A Intel incorporou a unidade de processamento gráfico para o silício, juntamente aos núcleos da unidade central de processamento. Apresentando na Figura 14.

- Especificações gráficas	
Gráficos do processador ‡	Intel® HD Graphics 3000
Frequência da base gráfica	650 MHz
Máxima frequência dinâmica da placa gráfica	1.3 GHz
Saída gráfica	eDP/DP/HDMI/SDVO/CRT
Intel® Quick Sync Video	Yes
Tecnologia Intel® InTru™ 3D	Yes
Intel® Insider™	Yes
Intel® Wireless Display	Yes
Interface de Vídeo Flexível Intel® (Intel® FDI)	Yes
Tecnologia de alta definição Intel® Clear Video	Yes
Necessário Licença Macrovision®	No

Figura 14 - Especificações gráficas do processador de acordo com o fabricante
Fonte: Intel (2015).

3.2.4 Software de informação do *sistema*

O Software CPUID, oferece um conjunto de rotinas relacionadas a *hardware* desenvolvido para qualquer linguagem de programação moderna. A ferramenta CPUID é um poderoso mecanismo de detecção de *hardware*, baseado em um *driver* de *software* de modo Kernel (CPUID, 2015).

Através da utilização do software CPUID efetuou-se a verificação do processador em questão, o qual foi utilizado para desenvolver os testes deste trabalho. Seguem nas Figuras de 15 a 20 as especificações do processador Intel i7 2.2 (CPUID, 2015).

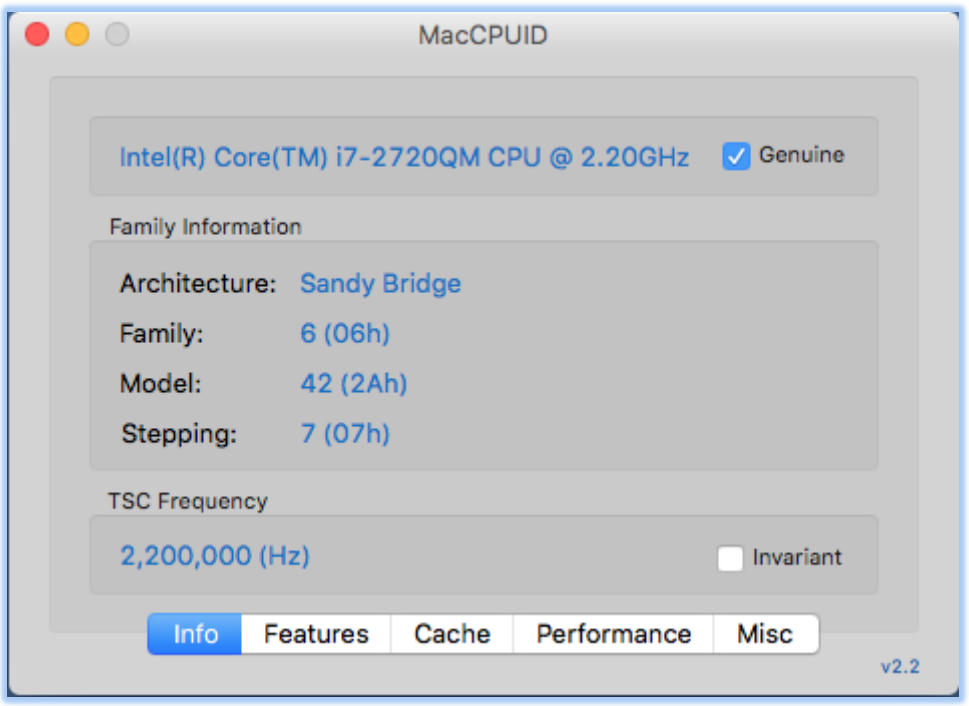


Figura 15 - Processador i7 2.2 informações da arquitetura Sandy Bridge frequência 2.20GHz
Fonte: Autoria própria.

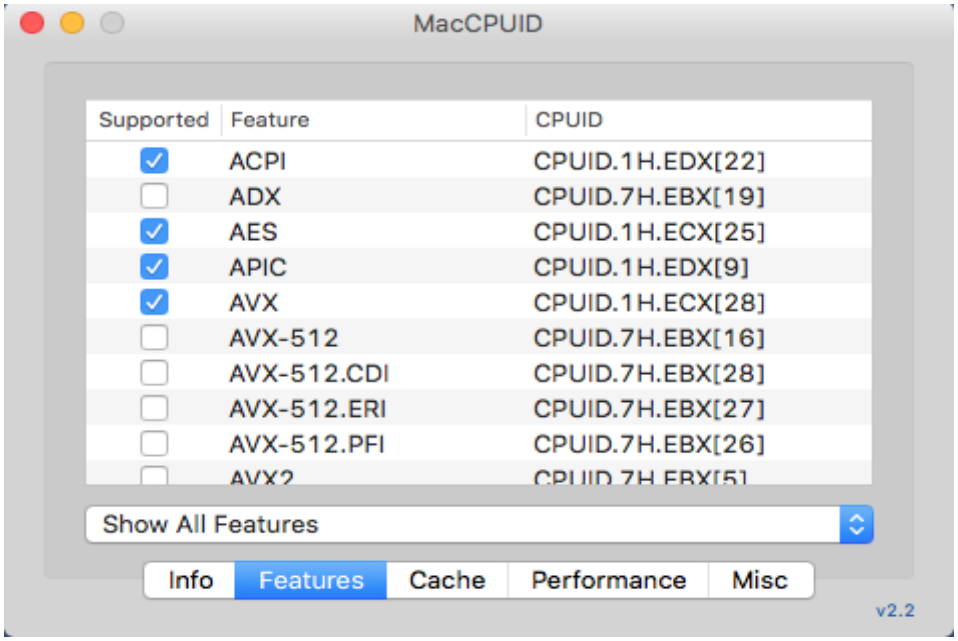


Figura 16 - Características suportadas
Fonte: Autoria própria.

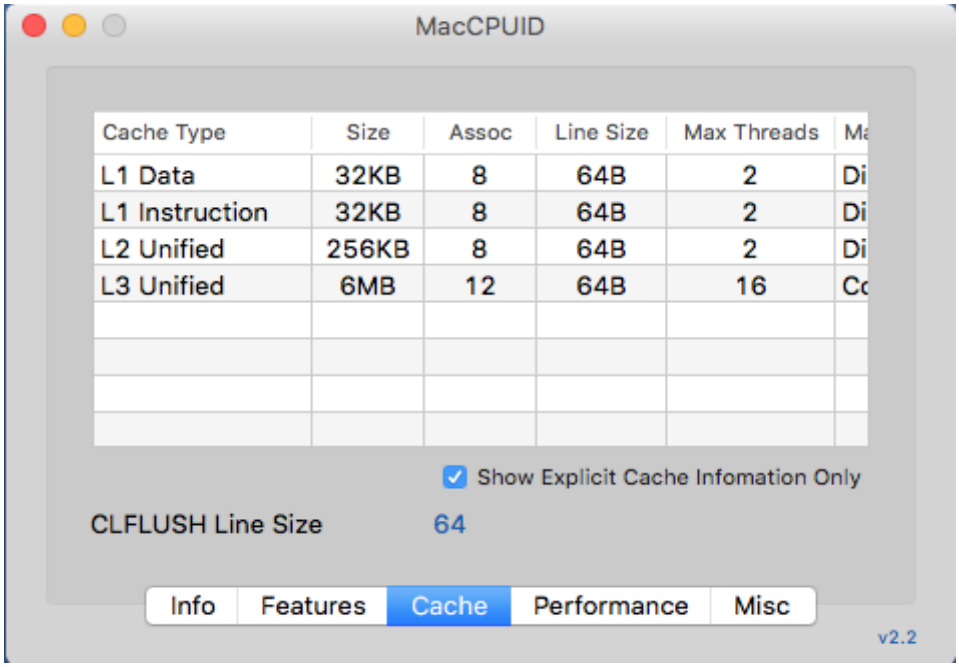


Figura 17 - Informações do cache.
Fonte: Autoria própria.

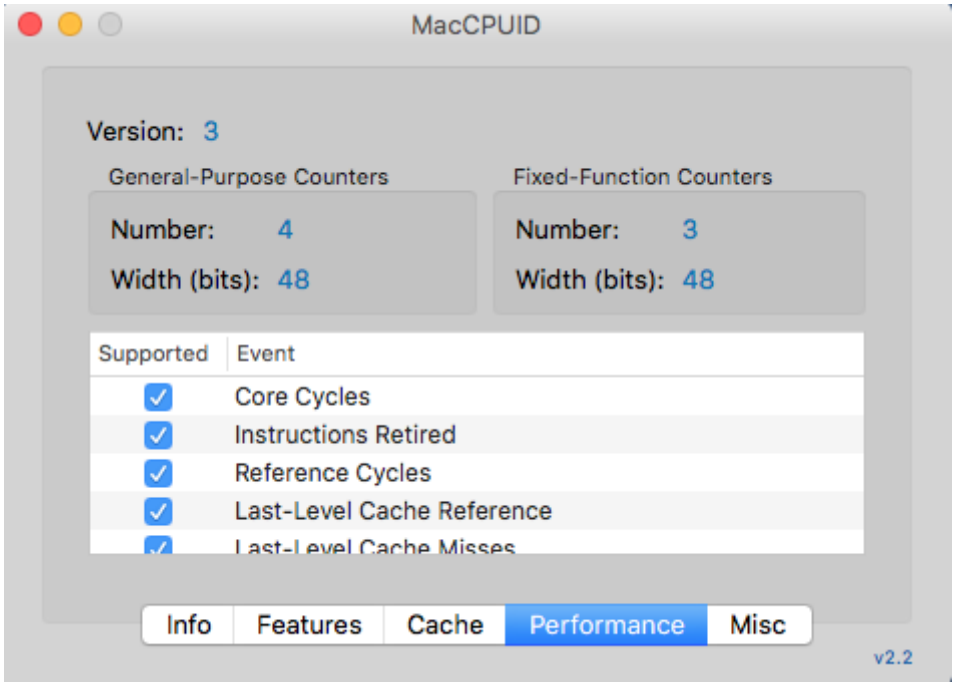


Figura 18 - Itens de performance.
Fonte: Autoria própria.

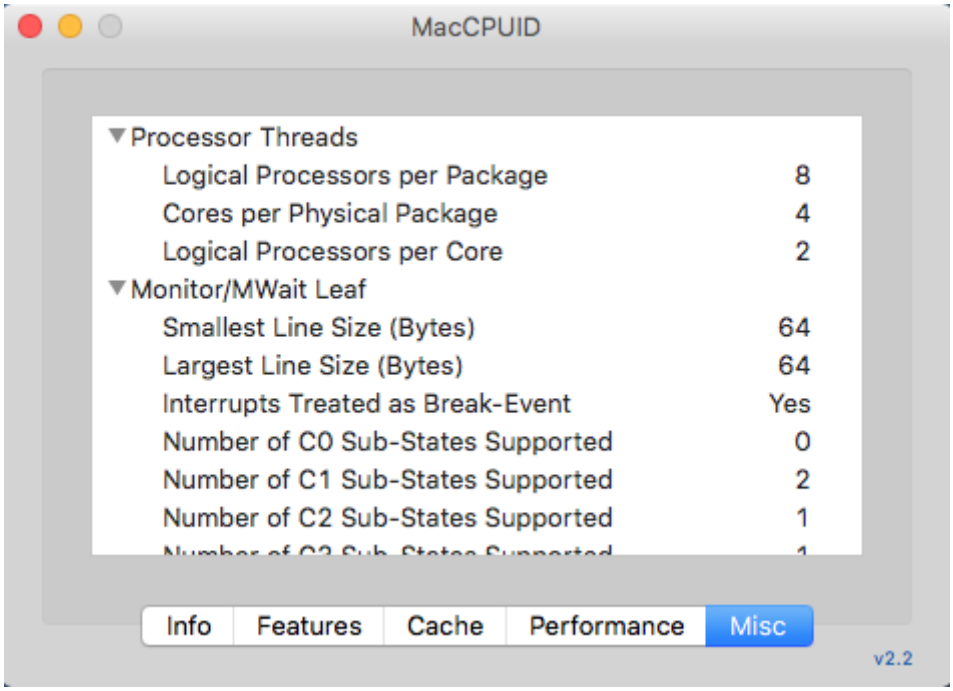


Figura 19 - Informações das *Threads* do processador oferece *Hyperthreading* para controlar 8 *threads* simultaneamente.
Fonte: Autoria própria.

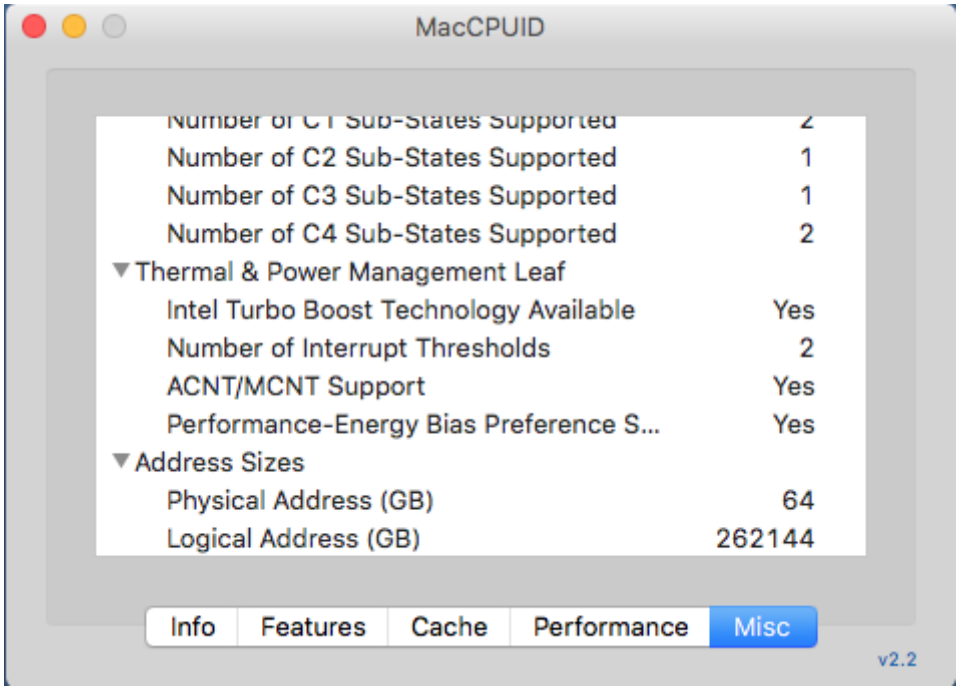


Figura 20 - Informações de tecnologia.
Fonte: Autoria própria.

4 RESULTADOS E DISCUSSÃO

Diretamente proporcional à quantidade de cores do processador ativo durante o processo de compilação, obteve-se alterações nos seguintes atributos: tempo de compilação, na temperatura do processador e aumento do ”stress” do hardware utilizado.

Após a realização dos testes com 2, 4, 6 e 8 cores ativos, foi registrado o tempo de compilação para cada teste respectivamente.

A Tabela 1 apresenta os resultados do tempo de compilação em quadruplicada para cada quantidade de *cores* ativos.

Tabela 1 - Resultado dos tempos de compilação de acordo com o número de cores ativos.

Número de *Cores* Ativos	Tempo (minutos)				Média	Desvio padrão
	Teste 1	Teste 2	Teste 3	Teste 4		
0	52,37	51,12	53,20	51,46	52,04	0,812
2	30,31	30,25	28,50	27,48	24,19	1,201
4	20,05	19,20	18,46	19,28	19,25	0,563
6	16,39	16,21	16,33	16,25	16,30	0,070
8	5,27	6,15	5,51	6,12	5,76	0,382

Fonte: Autoria própria.

Diante dos tempos obtidos, observou-se que quanto maior a quantidade de *cores* ativos no processador, mais rápido ocorreu a compilação. Os tempos de compilação variaram de aproximadamente 53 minutos até 6,15 minutos.

A partir da Tabela 1, geraram-se os Gráficos do 1 ao 4.

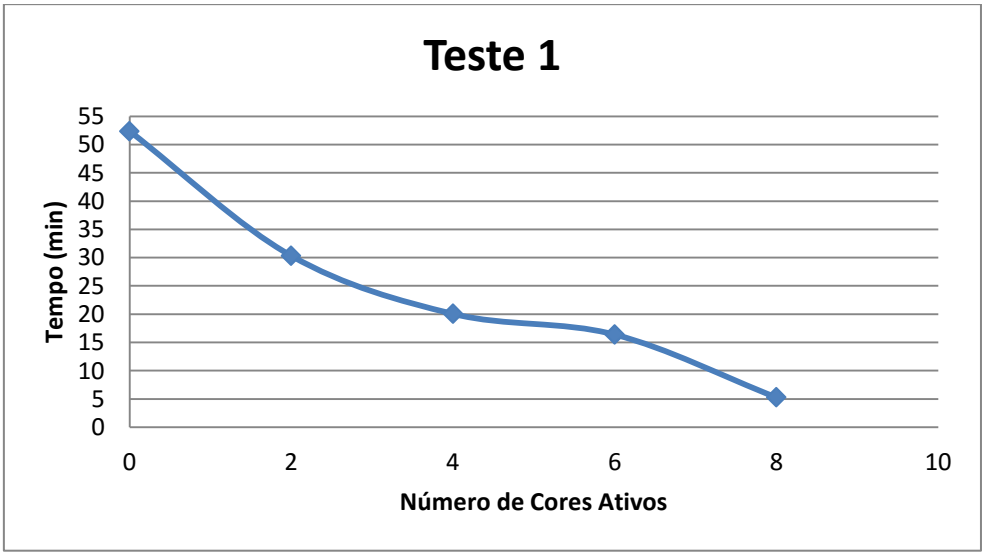


Gráfico 1 - Primeiro teste de compilação para cronometragem de tempo
Fonte: Autoria própria.

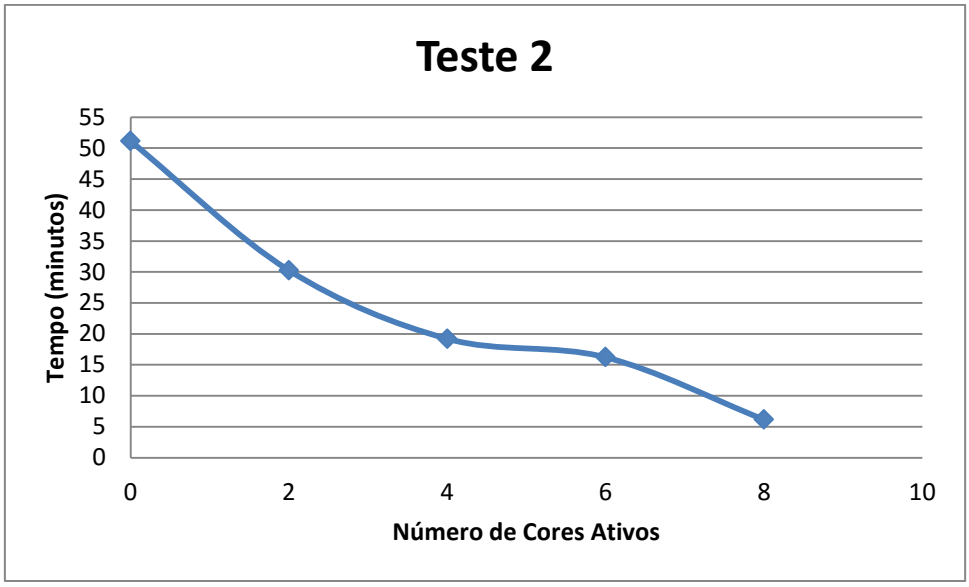


Gráfico 2 - Segundo teste de compilação para cronometragem de tempo
Fonte: Autoria própria.

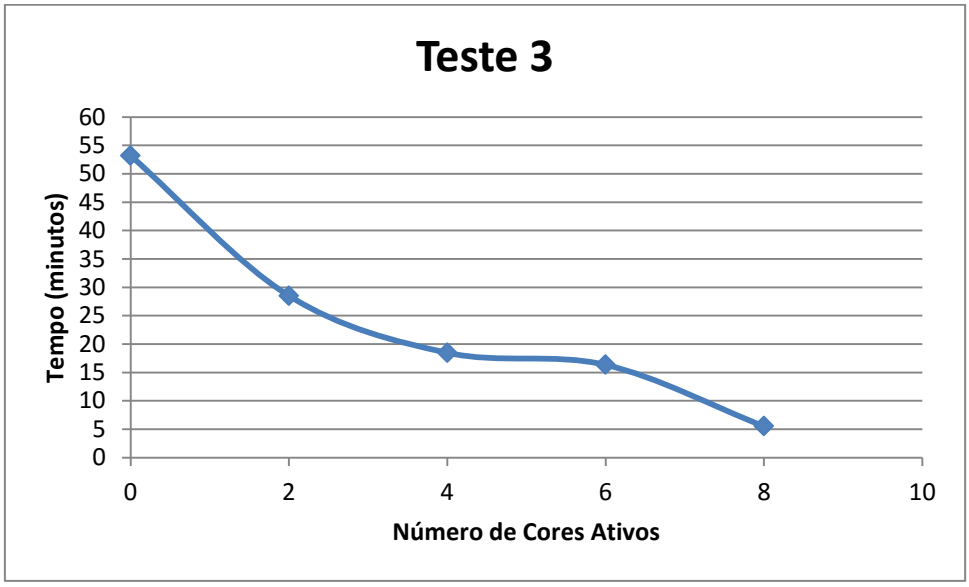


Gráfico 3 - Terceiro teste de compilação para cronometragem de tempo
Fonte: Autoria própria.

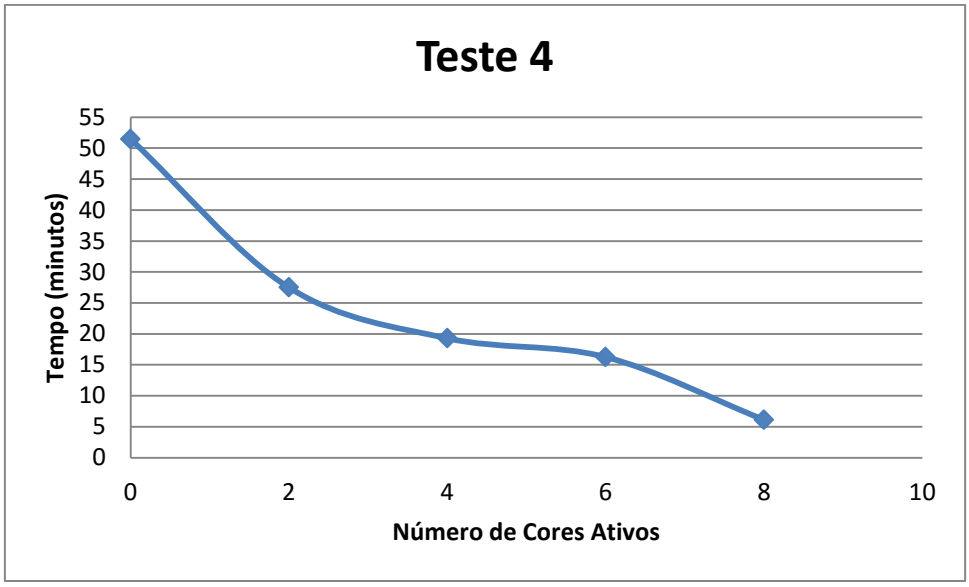


Gráfico 4 - Quarto teste de compilação para cronometragem de tempo.
Fonte: Autoria própria.

Os dados coletados no decorrer dos testes, plotados nos Gráficos 1 ao 4 demonstram o objetivo deste trabalho atingido, a diminuição do tempo de execução da compilação de Kernel de acordo com a quantidade fornecida de cores de processadores.

Aplicou-se o teste de normalidade, considerando que Ho: a amostra provém de uma distribuição normal e H1: a amostra não provém de uma população normal, ao nível de significância de 5% (α=0,05), sendo que para n (número de amostras) = 4, Wα = 0,748. Utilizando o software Action e aplicando a metodologia de Shapiro-Wilk., onde deve-se rejeitar Ho ao nível de significância α se $W_{calculado} <$ Wα. De acordo com a Tabela 2, constatou-se que os dados seguiam uma distribuição normal, o que permite a utilização da média dos testes para discussão dos resultados.

Tabela 2 - Resultado do teste de normalidade do tempo de compilação.

Número de *cores* ativos	W calculado
0	0,947632858
2	0,867120689
4	0,965196449
6	0,853707679
8	0,962716354

Fonte: Autoria própria.

O Gráfico 5 representa a média dos 4 testes.

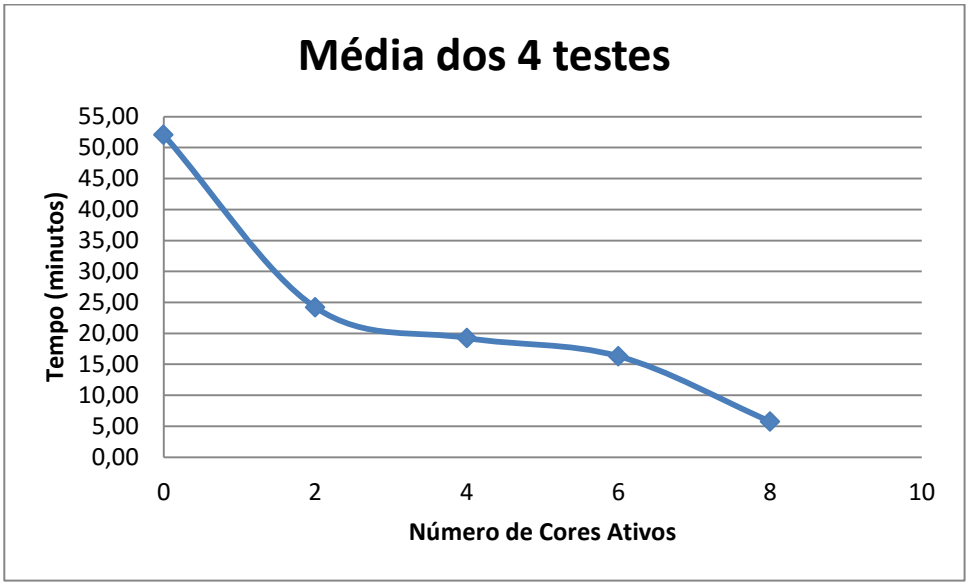


Gráfico 5 - Média dos tempos dos quatro testes de compilação para cada quantidade de *cores* ativos.
Fonte: Autoria própria.

Após os tempos de compilação serem tabelados e plotados em gráficos, mediu-se a porcentagem de utilização de *hardware*, os mesmos foram organizados na tabela 3.

Tabela 3 - Resultado da % de utilização do hardware de acordo com o número de cores ativos.

Número de *Cores* Ativos	Uso de Hardware (%)				Média	Desvio padrão
	Teste 1	Teste 2	Teste 3	Teste 4		
0	15	14	16	15	15	0,0071
2	30	31	28	33	31	0,0180
4	53	54	52	55	54	0,0112
6	81	77	75	78	78	0,0217
8	88	91	94	95	92	0,0274

Fonte: Autoria própria.

É visível a variação na porcentagem de utilização de hardware de acordo com o aumento da quantidade de *cores* ativos. Com nenhum *core*, a utilização é baixa tendo uma

média de 15%, já com 8 *cores* a utilização do hardware é quase completa, obtendo-se valores próximos a 100%.

A partir da Tabela 3, obtiveram-se os Gráficos de 6 a 10.

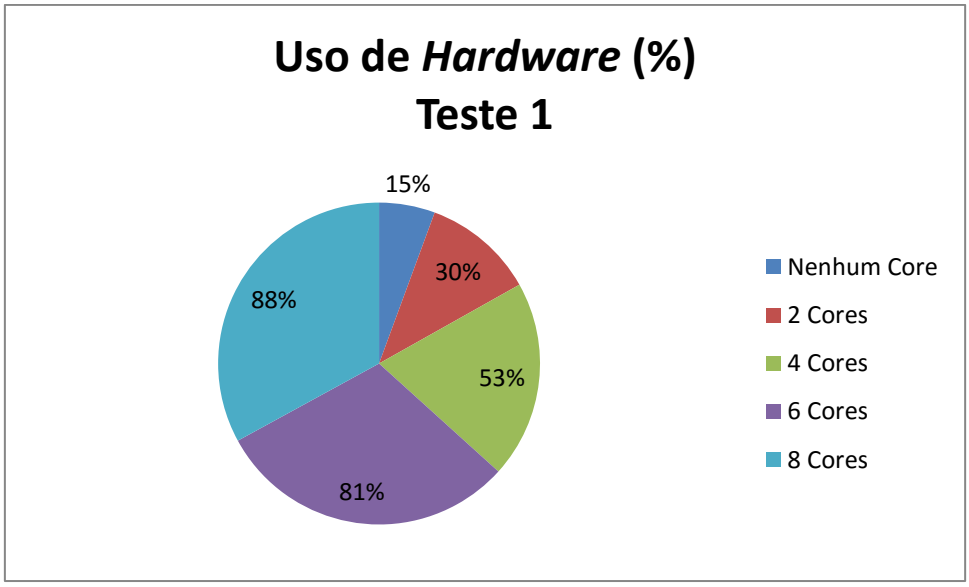


Gráfico 6 - Primeiro teste de utilização de *hardware* de acordo com a quantidade de *cores* ativos.
Fonte: Autoria própria.

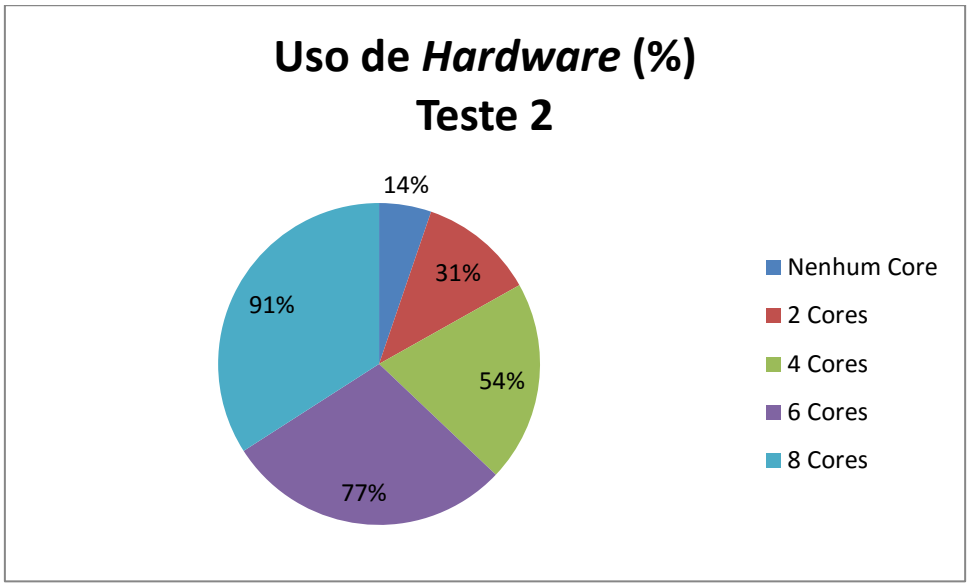


Gráfico 7 - Segundo teste de utilização de *hardware* de acordo com a quantidade de *cores* ativos.
Fonte: Autoria própria.

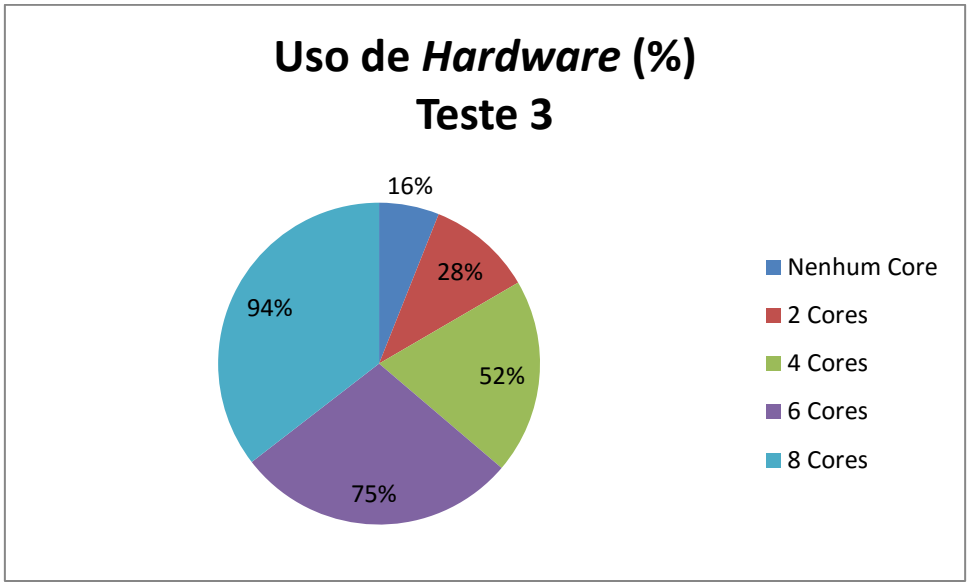


Gráfico 8 - Terceiro teste de utilização de *hardware* de acordo com a quantidade de *cores* ativos.
Fonte: Autoria própria.

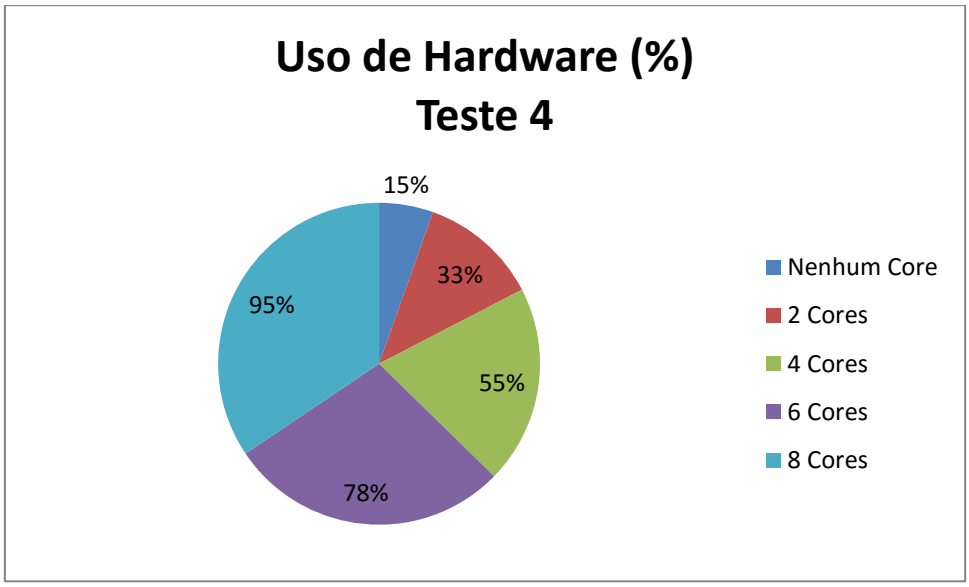


Gráfico 9. Quarto teste de utilização de *hardware* de acordo com a quantidade de *cores* ativos.
Fonte: Autoria própria.

Ao aplicar o teste de normalidade como descrito para o tempo de compilação, observou-se que os dados provêm de uma distribuição normal. Apresentado na tabela 4 os valores do W calculado.

Tabela 4 - Resultado do teste de normalidade da porcentagem de utilização do hardware.

Número de *cores* ativos	W calculado
0	0,944664
2	0,998396
4	0,992912
6	0,981516
8	0,939820

Fonte: Autoria própria.

Sendo assim, é válida a utilização do Gráfico 10, o qual apresenta médias dos testes.

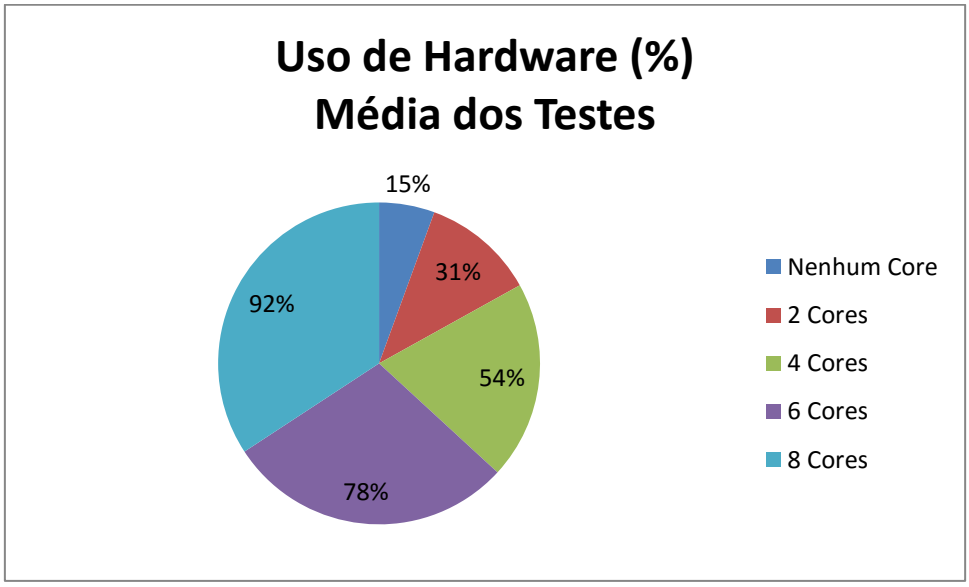


Gráfico 10 - Gráfico da média dos testes de utilização de hardware.
Fonte: Autoria própria.

Através da interpretação dos gráficos da utilização do *hardware*, conclui-se que a variação de utilização conforme a quantidade de *cores* ativos é alta. Desta forma, pode-se afirmar que o objetivo de diminuir o tempo de execução da compilação de Kernel utilizando o *hardware* disponível é válida.

O modo *default* da compilação de Kernel utiliza entre 14% e 16% do *hardware* disponível para desenvolver o processo, mantendo uma demanda maior de tempo para completar uma compilação. O comando de *cores*, possibilita disponibilizar mais *hardware* para o processo de compilação de Kernel, desta forma fica visível a variação da utilização do *hardware*.

Com 2 *cores* ativos utilizou-se em média 28 % a 33% do *hardware* para completar o processo de compilação, consequentemente com o aumento de cores para 4 a porcentagem de utilização aumentou variando entre 52% a 55% do processador para o mesmo processo. A quantidade de 6 *cores* utilizou de 78 % a 81 % do consumo do processador, findando a última fase de testes com 8 *cores*, variou entre 88% a 95% do uso do processador para desenvolver o processo de compilação de Kernel do Debian Linux.

A temperatura também foi um fator importante avaliado durante o processo de compilação. A Tabela 5 apresenta a variação de temperatura em função do tempo, de acordo com a quantidade de *cores* ativos.

Tabela 5 - Variação da temperatura ao longo do tempo de acordo com a quantidade de *cores* ativo.

Tempo (minutos)	Temperatura (°C)						
	Onn[a]	Onn VM[b]	D[c]	2 Cores	4 Cores	6 Cores	8 Cores
1	44	60	83	93	94	94	92
3	-	-	-	-	-	-	94
4	-	-	-	-	-	-	93
5	42	61	81	95	93	90	95
10	-	-	-	-	93	93	-
15	44	63	79	89	95	93	-
25	-	-	-	91	-	-	-
30	43	64	80	-	-	-	-

[a]Onn (ligada); [b]Onn VM (ligada com sistema operacional Linux em VM); [c]D (compilação de kernel padrão)
Fonte: Autoria própria.

A variação dos testes foi obtida através do acompanhamento do tempo e as respectivas temperaturas no decorrer do mesmo. A partir da Tabela 5 obtiveram-se os Gráficos de 11 a 17.

O modo Onn representou a máquina ligada não havendo nenhuma utilização específica de *software*, apenas o sistema operacional iniciado. Os dados de temperatura coletados a cada 5 minutos em um período total de 30 minutos, a temperatura variou entre 42º a 44º, variação esta, apresentada no Gráfico 11.

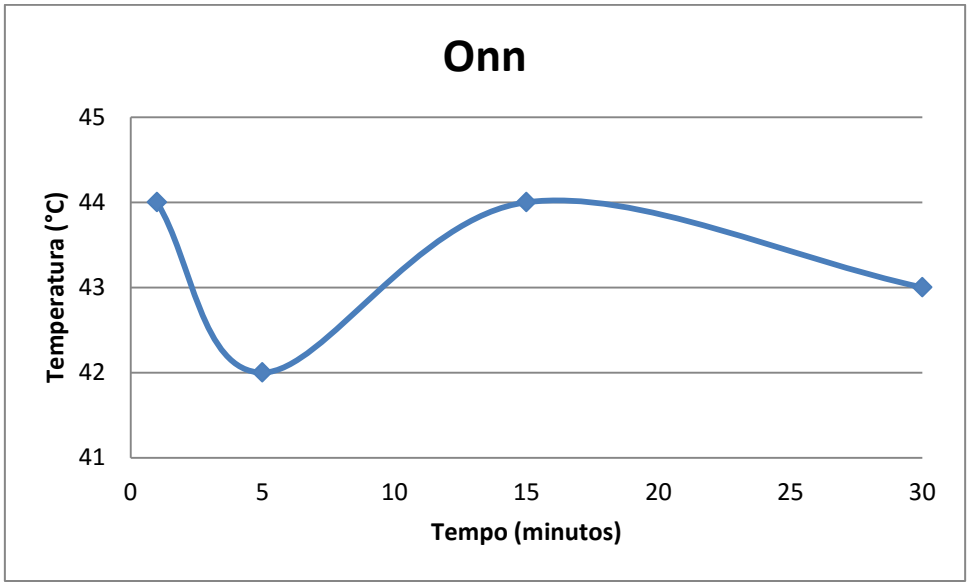


Gráfico 11 - Variação da temperatura ao longo do processo no modo Onn.
Fonte: Autoria própria.

O Modo Onn VM consistiu na máquina iniciada, juntamente com a mesma o software de máquina virtual (Parallels *desktop*), contendo o sistema operacional Linux Debian Lenny, onde se obteve uma variância de temperatura e tempo apresentados no Gráfico 12.

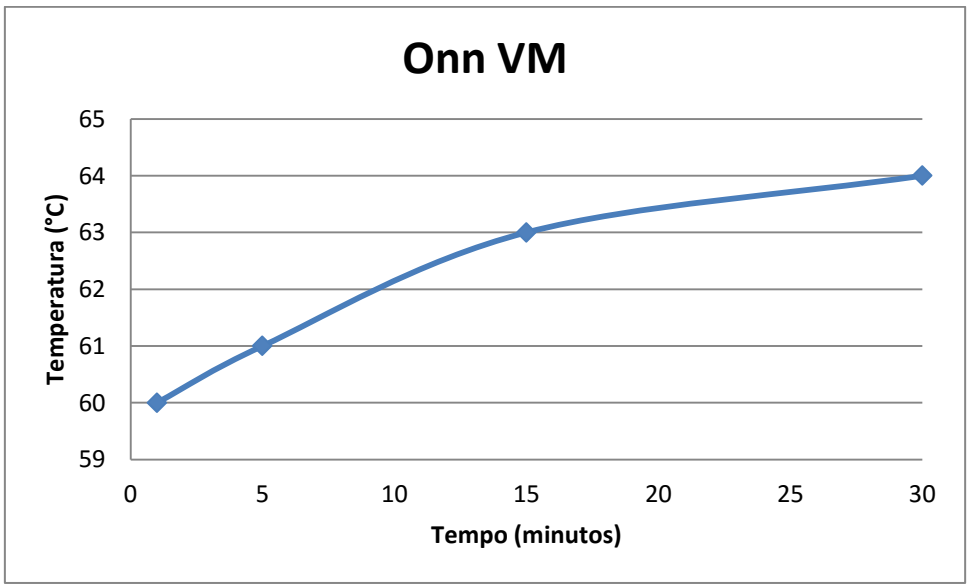


Gráfico 12 - Variação da temperatura ao longo do processo no modo Onn VM.
Fonte: Autoria própria.

Neste modo apresentado no Gráfico 12, obteve-se um aumento na temperatura ocasionado pela utilização da máquina virtual.

O modo D (*default*) representou o processo de compilação de Kernel Linux na sua forma original, onde foi mantida a máquina virtual ligada juntamente com o sistema operacional Linux executou-se o processo de compilação de Kernel. O Gráfico 13 apresenta a variação de temperatura e tempo.

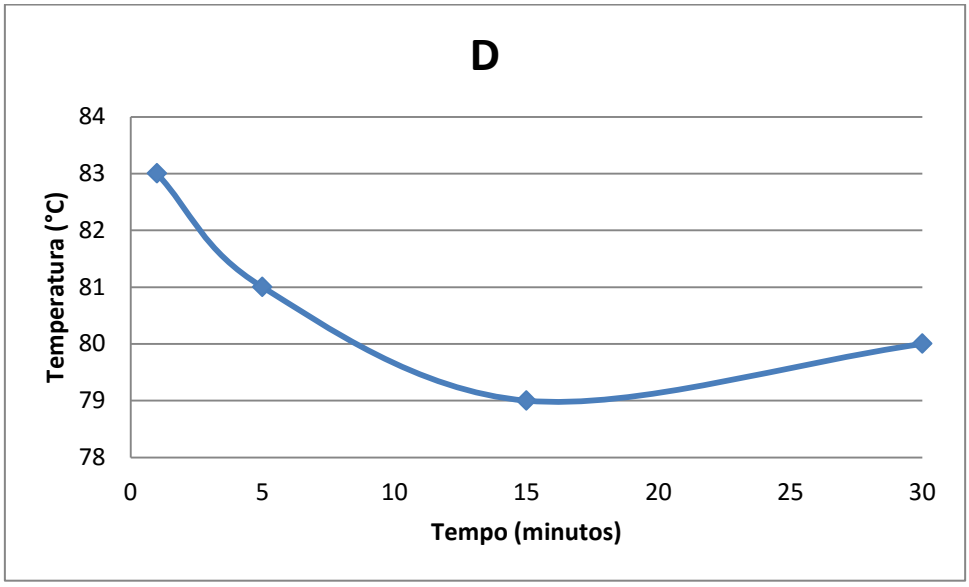


Gráfico 13 - Variação da temperatura ao longo do processo no modo *default*.
Fonte: Autoria própria.

Durante o processo de compilação de Kernel no modo D(*default*) ocorreu novamente um aumento na temperatura com variação entre 79ºC a 83ºC.

Durante os Modos Onn (ligado), Onn VM (máquina virtual) e D (*default*), foram coletados dados para apresentar a variação de temperatura e tempo em que a máquina foi exposta as condições dos testes.

O Gráfico 14 apresenta a variação de temperatura em função do tempo para o comando de utilização de 2 *cores* no processo de compilação de Kernel.

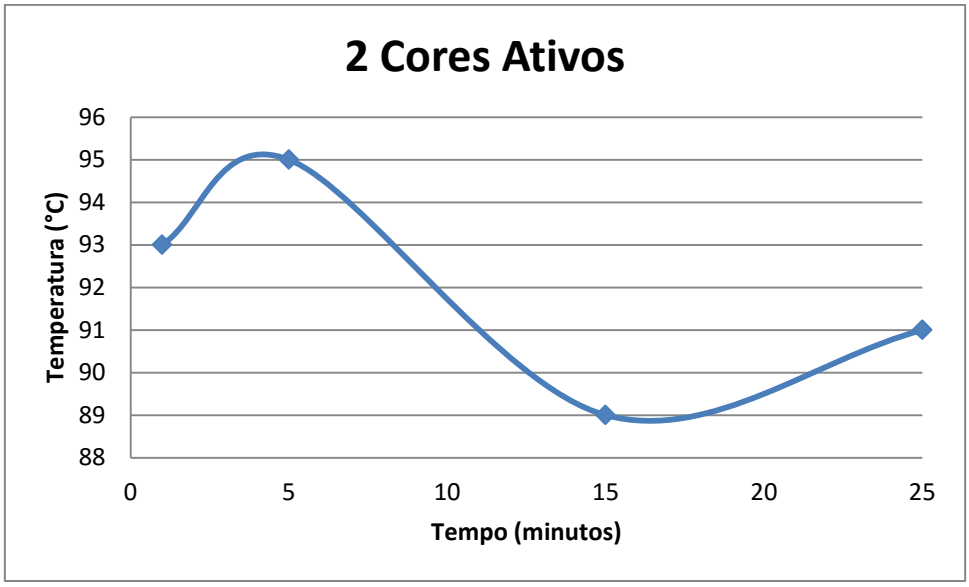


Gráfico 14 - Variação da temperatura ao longo do processo com a utilização de 2 *cores* ativos.
Fonte: Autoria própria.

Ao utilizar o comando de ativação de *cores* apresentado no Gráfico 14, sendo 2 *cores* para este processo, verifica-se o aumento da temperatura variando de 89º a 95º, obtendo um resultado minimizando o tempo do processo.

A utilização no processo com 4 *cores* ativos apresentando a variância de temperatura e tempo no Gráfico 15.

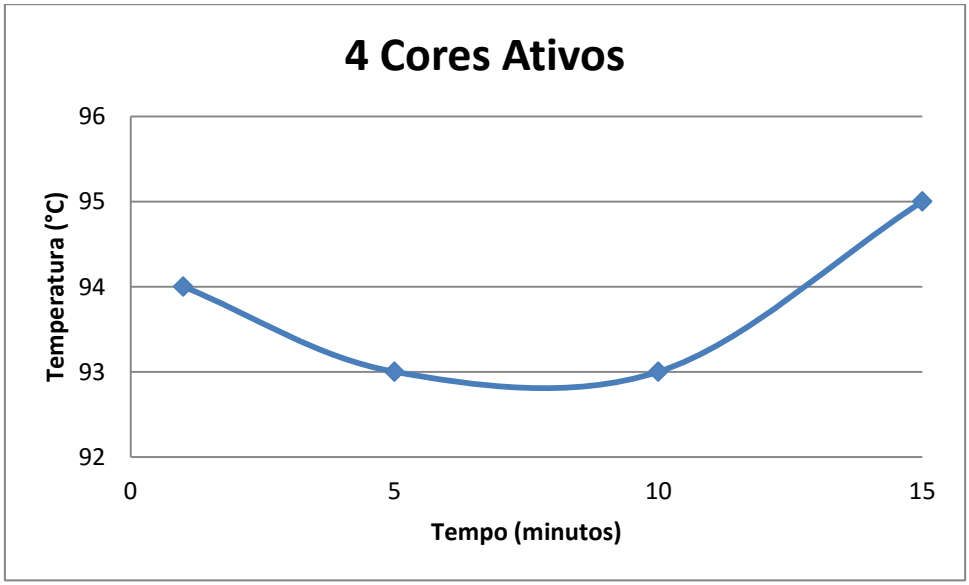


Gráfico 15 - Variação da temperatura ao longo do processo com a utilização de 4 *cores* ativos.
Fonte: Autoria própria.

Com o aumento de 4 *cores*, para desenvolver o processo de compilação de Kernel, observou-se uma diminuição do tempo a cada teste realizado e a temperatura manteve-se entre 93ºC e 95ºC, mesmo com o aumento da utilização do *hardware*.

No processo apresentando a utilização de 6 *cores* ativos, observa-se a variação de temperatura e tempo durante o período de compilação de Kernel, representado no Gráfico 16.

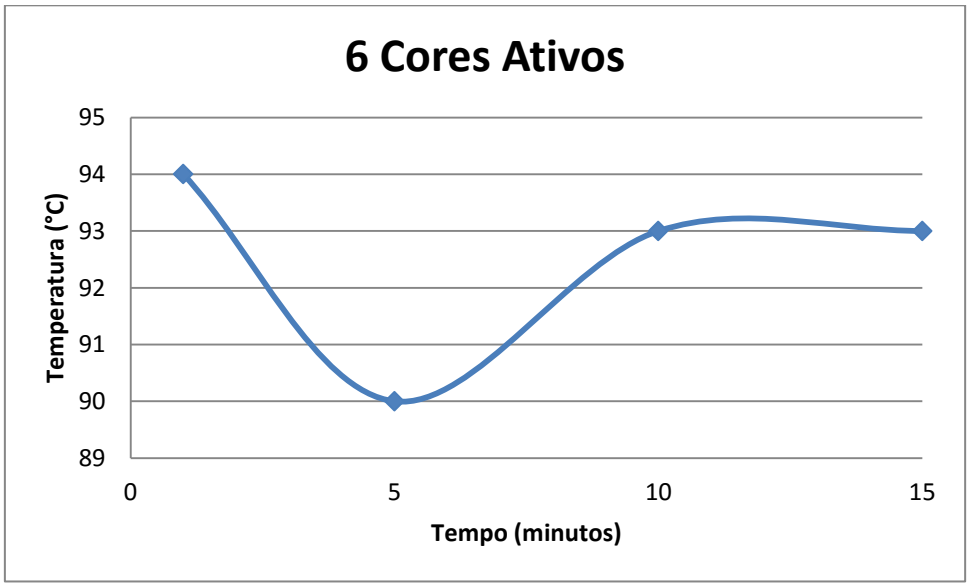


Gráfico 16 - Variação da temperatura ao longo do processo com a utilização de 6 *cores* ativos.
Fonte: Autoria própria.

A variação da temperatura do teste com 6 *cores* ativos manteve-se entre 90º e 94º , o tempo de processo manteve-se em média de 15 minutos de execução.

No teste com 8 *cores* ativos, utilizou-se da capacidade máxima de *hardware* disponível na máquina, apresentado a variação de temperatura e tempo no Gráfico 17.

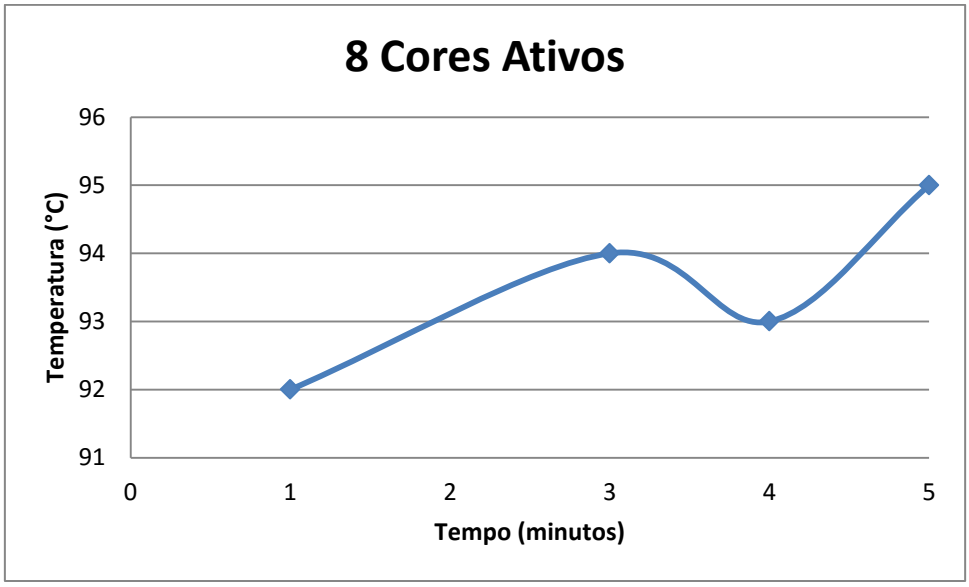


Gráfico 17 - Variação da temperatura ao longo do processo com a utilização de 8 *cores* ativos.
Fonte: Autoria própria.

No último teste realizado detectou-se a variação entre temperatura e tempo, observou-se a diminuição do tempo para 5 minutos, na realização do processo a temperatura manteve-se entre 92ºC a 95ºC.

5 CONSIDERAÇÕES FINAIS

Esta seção apresenta a conclusão obtida após a elaboração e análise dos testes, bem como ideias para melhorar ou continuar trabalhos na área de compilação de Kernel.

5.1 CONCLUSÃO

Levando em consideração todos os resultados obtidos nos testes de compilação variando a utilização de cores no processador, conclui-se que quanto maior a quantidade de *cores* ativos no processador, menor será o tempo do processo de compilação de Kernel. Por outro lado, como ponto negativo tem-se o aquecimento do *hardware*, o qual é exposto a temperaturas próximas a 100° C, consideradas destrutivas para equipamentos de uso comum.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Como sugestões para trabalhos futuros podendo utilizar este trabalho como base, tem-se:

a) Implementação do comando de gerenciamento de *cores* para outros processos além da compilação de Kernel Linux.

b) Estudo dos benefícios com a implementação deste gerenciamento em projetos e *software* livre.

c) A utilização do comando em *softwares* que necessitam uma demanda maior de *hardware* visando melhorar o desempenho do mesmo.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ALECRIM, Emerson. **Processadores: clock, bits, memória cache e múltiplos núcleos.** Disponível em:<http://www.infowester.com/processadores.php>. . Acesso em: 17 de out. 2015.

ANDRADE, Gabriel. O que você precisa saber para comprar seu computador. Disponível em: <http://www.infoescola.com/informatica/o-que-voce-precisa-saber-para-comprar-seu-computador/>. Acesso em: 17 de out. 2015.

CPUID. 2015. Disponível em: < http://www.cpuid-pro.com/products-services.php>. Acessado em: 16 out. 2015.

FERREIRA, Rubem E. **Linux: guia do administrador do sistema**. 2 ed. São Paulo: Novatec Editora, 2008.

GARBEE, Bdale. Uma breve história do Debian. Traduzido por: Michelle Ribeiro. 2002. Disponível em: https://www.debian.org/doc/manuals/project-history/index.pt.html#contents. Acessado em: 09 de set. 2015.

GOERZEN, John; OTHAMAN, Ossama. **Debian GNU/Linux:** Guide to Installation and Usage. 2001. Disponível em: <http://www.gutenberg.org/dirs/etext04/dguid10pdf.pdf>. Acesso em: 09 de set. 2015.

JORDÃO, Fabio. **Quais as diferenças entre os processadores intel core i3, i5 e i7?.** Disponível em: < http://www.tecmundo.com.br/processadores/3904-quais-as-diferencas-entre-os-processadores-intel-core-i3-i5-e-i7-.htm>. Acessado em: 17 de out. de 2015.

PRACIANO, Elias. **Use sync e sysctl para esvaziar a memória cache no Linux**. Disponível em: < http://elias.praciano.com/2013/07/como-limpar-a-memoria-cache-no-linux/>. Acessado em: 15 de set. 2015.

SILVA, Gleydson M. da. **Guia Foca:** GNU/Linux. 2010. Disponível em: <static.efetividade.net/archive/img/focalinux-iniciante.pdf>. Acesso em: 09 de set. 2015.

STUART, Brian L. Sistemas operacionais – Projetos e Aplicações. 1 ed. São Paulo: Cengage Learning, 2011.

TANEMBAUM, Andrew S. **Sistemas operacionais modernos**. 3 ed. São Paulo: Pearson Prentice Hall, 2009.

WELSH, Matt; KAUFMAN, Lar. Dominando o Linux. Rio de Janeiro: Editora Ciência Moderna Ltda, 1997.