sid.inpe.br/mtc-m21c/2020/02.14.00.18-TDI

EXPLORAÇÃO DE OPORTUNIDADES DE MELHORIA DE DESEMPENHO EM UM MODELO CLIMÁTICO

Rhuan Edson Caldini Costa

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Celso Luiz Mendes, aprovada em 17 de fevereiro de 2020.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3UU6PU2>

INPE
São José dos Campos
2020

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

sid.inpe.br/mtc-m21c/2020/02.14.00.18-TDI

EXPLORAÇÃO DE OPORTUNIDADES DE MELHORIA DE DESEMPENHO EM UM MODELO CLIMÁTICO

Rhuan Edson Caldini Costa

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Celso Luiz Mendes, aprovada em 17 de fevereiro de 2020.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3UU6PU2>

INPE
São José dos Campos
2020

Dados Internacionais de Catalogação na Publicação (CIP)

Costa, Rhuan Edson Caldini.

C823e Exploração de oportunidades de melhoria de desempenho em um modelo climático / Rhuan Edson Caldini Costa. – São José dos Campos : INPE, 2020.

xviii+54 p. ; (sid.inpe.br/mtc-m21c/2020/02.14.00.18-TDI)

Tese (Doutorado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2020.

Orientador : Dr. Celso Luiz Mendes.

1. Modelos climáticos. 2. Otimização. 3. Processamento concorrente. 4. Desempenho de sistemas computacionais. I.Título.

CDU 004.4'416:551.509.313.4

Aluno (a): *Rhuan Edson Caldini Costa*

Título: "EXPLORAÇÃO DE OPORTUNIDADES DE MELHORIA DE DESEMPENHO EM UM MODELO CLIMÁTICO"

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Doutor(a)* em

Computação Aplicada

Dr. Stephan Stephany

Presidente / INPE / SJCampos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dr. Celso Luiz Mendes

Orientador(a) / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dr. Pedro Ribeiro de Andrade Neto

Membro da Banca / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dr. Álvaro Luiz Fazenda

Convidado(a) / UNIFESP / São José dos Campos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 17 de fevereiro de 2020

AGRADECIMENTOS

Agradeço ao Dr. Celso Luiz Mendes pelo tempo dispendido na orientação deste trabalho e por todo o conhecimento transmitido.

Ao CNPq pela bolsa de mestrado concedida.

Ao LNCC pelo acesso ao sistema Santos Dumont, utilizado na execução dos experimentos aqui relatados.

Ao CPTEC que, através do Me. Manoel Baptista, nos forneceu o código-fonte do BESM e as instruções iniciais de como executá-lo.

Ao Dr. Nandamudi L. Vijaykumar pela recepção no INPE, orientação no primeiro ano do mestrado e pela amizade formada desde então.

Aos professores da CAP pelos ensinamentos nas disciplinas ministradas.

E aos membros da banca examinadora pela disponibilidade em avaliar este trabalho.

RESUMO

Modelos climáticos são programas complexos, que envolvem a implementação computacional de inúmeros cálculos matemáticos de forma a representar fenômenos físicos que interagem entre si. Tais fenômenos geralmente são representados por módulos específicos (atmosférico, oceânico, solo, etc.) que são executados de forma acoplada, isto é, realizam a simulação dos respectivos fenômenos para um certo intervalo de tempo, trocando informações entre si quando necessário. Desta complexidade, surge a necessidade de paralelizar e otimizar sua execução de modo a se obter resultados em tempo viável. Além da paralelização do modelo, técnicas de otimização voltadas à arquitetura de hardware utilizada também podem ser aplicadas, tais como vetorização e blocagem de *loops*. Este trabalho tem como objetivo explorar oportunidades de otimização de desempenho em modelos climáticos globais, utilizando como estudo de caso o *Brazilian Earth System Model* (BESM), um modelo desenvolvido pelo CPTEC/INPE com o principal objetivo de simular o clima global de modo a entender as causas das mudanças climáticas. Inicialmente, foi avaliado o desempenho original do BESM, verificando como o modelo estava sendo compilado, quais eram as técnicas de paralelização utilizadas, como seus módulos eram executados, e quais eram os pontos com maior consumo de CPU. Em seguida, baseado nas informações encontradas na análise inicial, foram exploradas possíveis otimizações de desempenho do modelo. Foram aplicadas melhorias no processo de compilação e execução do modelo, e também foram realizadas as alterações necessárias para permitir a execução dos módulos atmosférico e oceânico de forma concorrente, com diferentes quantidades de CPUs alocados para cada módulo. Após todas as otimizações aplicadas, foi obtida redução no tempo de execução de até quatro vezes quando utilizados 24 processadores, e foi reduzido pela metade o tempo mínimo de execução das simulações empregando centenas de processadores. Apesar dos resultados consideráveis alcançados, ainda há espaço para melhorias no BESM em trabalhos futuros, voltadas ao balanceamento de carga entre os processos MPI, especialmente no módulo atmosférico.

Palavras-chave: Modelos climáticos. Otimização. Processamento concorrente. Desempenho de sistemas computacionais.

EXPLORING OPPORTUNITIES FOR PERFORMANCE IMPROVEMENT IN A GLOBAL CLIMATE MODEL

ABSTRACT

Climate models are complex programs that involve the computational implementation of numerous mathematical calculations in order to represent physical phenomena that interact with each other. Such phenomena are usually represented by specific modules (atmospheric, oceanic, land, etc.) that operate in a coupled way, that is, they perform the simulation of the respective phenomena for a certain interval of time, exchanging information among them when necessary. Due to this complexity, the need arises to parallelize its execution in order to obtain results in a viable time. In addition to the model parallelization, optimization techniques aiming the employed hardware architecture can also be applied, such as vectorization and loop blocking. This work aims to explore opportunities of performance improvements in global climate models, using as case study the Brazilian Earth System Model (BESM), a model developed by CPTEC/INPE with the main goal of simulating the global climate in order to understand the causes of climate changes. The work started with an evaluation of the original BESM's performance, verifying how the model was being compiled, which parallelization techniques were applied, how its modules were executed, and which were the points with the highest CPU consumption. Then, based on the information found in the initial analysis, the model's performance was optimized. Improvements were made in the process of compiling and executing the model, and the necessary changes were also made to allow the execution of the atmospheric and oceanic modules concurrently, with different amounts of CPUs allocated for each module. After all the applied optimizations, a reduction in the execution time of up to four times was obtained when 24 processors were used, and the minimum time for executions with hundreds of processors was reduced to half of the original time. Despite the substantial results achieved, there is still room for improvement in BESM in future works, aimed at load balancing between the MPI processes, especially in the atmospheric module.

Keywords: Climate models. Optimization. Concurrent processing. Computer systems performance.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

AGCM	–	Atmospheric Global Circulation Model
BESM	–	Brazilian Earth System Model
CMIP	–	Coupled Model Intercomparison Project
CPTEC	–	Centro de Previsão de Tempo e Estudos Climáticos
ESM	–	Earth System Model
FMS	–	Flexible Modular System
GCM	–	Global Climate Model
GFDL	–	Geophysical Fluid Dynamics Laboratory
GHG	–	Greenhouse Gas
INPE	–	Instituto Nacional de Pesquisas Espaciais
IPCC	–	Intergovernmental Panel on Climate Change
LNCC	–	Laboratório Nacional de Computação Científica
MOM	–	Modular Ocean Model
MPI	–	Message Passing Interface
NCAR	–	National Center for Atmospheric Research
OGCM	–	Oceanic General Circulation Model
OMP	–	OpenMP
SIMD	–	Single Instruction, Multiple Data
SIS	–	Sea Ice Simulator
SST	–	Sea Surface Temperature

SUMÁRIO

Pág.

1 INTRODUÇÃO

A principal ferramenta para estudar futuras mudanças climáticas é o Modelo Climático Global (*Global Climate Model* - GCM), também conhecido como Modelo do Sistema Terrestre (*Earth System Model* - ESM). Para representar a variabilidade do clima, modelos globais incorporam a complexidade de vários componentes da Terra, geralmente oceano, atmosfera e superfície terrestre em um sistema acoplado (CHOU et al., 2014) . Exemplos de ESMs produzidos pelos maiores centros de pesquisa do mundo são: *Community Climate System Model version 4* (CCSM4) pelo *National Center for Atmospheric Research* (NCAR); *Geophysical Fluid Dynamics Laboratory Climate Model version 2* (GFDL CM2); e *Hadley Centre Global Environmental Model Earth System* (HadGEM-ES) (NOBRE et al., 2013).

Outros centros climáticos ao redor do mundo também estão desenvolvendo seus próprios modelos climáticos, seja adaptando um ESM global, ou combinando modelos de oceano, atmosfera e solo, de diferentes fontes, criando seus próprios modelos globais. Esta foi a opção escolhida pelo Brasil, que optou por desenvolver seu ESM (NOBRE et al., 2013).

A implementação de modelos climáticos globais exige a representação de vários componentes do planeta, como atmosfera, oceano e solo, citados anteriormente. Cada componente envolve múltiplos elementos que interagem entre si, tanto dentro do próprio componente quanto com elementos de outros componentes. Desta forma, a implementação destes modelos é bastante complexa, fazendo com que equipes de pesquisadores se dediquem à modelagem e implementação de um determinado componente. Assim, são disponibilizados modelos isolados (somente atmosfera, somente oceano, somente solo, etc.), conhecidos como módulos, que visam representar mais precisamente tal componente.

Os modelos climáticos globais acoplam estes módulos buscando ter uma representação de todo o sistema terrestre. Entretanto, como cada módulo, muito provavelmente, foi desenvolvido por um conjunto diferente de pesquisadores, há grandes chances das implementações terem visado direções diferentes de otimização de código. Além disso, para que estes módulos possam ser executados operacionalmente em tempo hábil, é necessário, quase sempre, implementá-los utilizando técnicas de paralelização. Desta forma, fazer com que os módulos trabalhem em harmonia de forma eficiente, com o mínimo de desperdício de recursos computacionais, torna-se uma tarefa complexa, que envolve a análise de como cada módulo se comporta durante a execução, e a identificação de técnicas de paralelização e otimização que

sejam viáveis para todos os módulos.

Um exemplo de modelo climático global desenvolvido utilizando este esquema de módulos acoplados é o *Community Earth System Model* (CESM), que engloba o CCSM4 citado anteriormente. O CESM foi desenvolvido pelo NCAR em conjunto com a comunidade científica, e é composto de cinco módulos: atmosfera, oceano, solo, gelo do solo e gelo oceânico; mais um componente acoplador central (NATIONAL CENTER FOR ATMOSPHERIC RESEARCH - NCAR, 2018a).

Neste modelo, a execução dos módulos é realizada de forma paralela, ou seja, há um conjunto de CPUs dedicadas a computar cada módulo, simultaneamente; e a cada determinado intervalo de tempo as informações são sincronizadas entre os módulos através do acoplador. Cada módulo pode utilizar uma quantidade diferente de CPUs de forma a balancear a intensidade computacional exigida, uniformizando o tempo de processamento (NATIONAL CENTER FOR ATMOSPHERIC RESEARCH - NCAR, 2018b).

Esta dissertação tem como objetivo investigar técnicas de otimização do desempenho computacional de modelos climáticos, compostos tipicamente pela combinação de diversos módulos relativos a diferentes processos físicos da natureza. Essa investigação foi realizada através do estudo do modelo climático brasileiro (BESM), utilizando técnicas de otimização de desempenho que sejam efetivas neste modelo. Foram analisadas a forma em que o modelo estava sendo compilado e executado, e quais eram as técnicas de paralelização utilizadas. Foram realizados testes de desempenho no modelo original com utilização de diversas quantidades de CPUs, a fim de verificar suas limitações e identificar *hotspots*.

Com estes dados em mãos, foram aplicadas técnicas de otimização no processo de compilação, tais como a troca do compilador e teste de diversas *flags* de otimização automática pelo compilador. Também foi alterada a biblioteca MPI utilizada. Em seguida, o modelo foi modificado de forma a executar seus módulos concorrentemente, para o qual foi necessário criar novos subcomunicadores MPI e alterar os scripts de execução do modelo. Foi avaliada a alocação de diversas proporções de CPUs para cada módulo do BESM, com o intuito de encontrar uma proporção ou quantidade de CPUs que trouxesse melhor desempenho. Também foram exploradas as técnicas de vetorização e blocagem nos *loops* de maior relevância. Em sua grande maioria, os testes descritos nesta dissertação foram conduzidos no Santos Dumont, o sistema de computação petaflópica do LNCC (LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA - LNCC, 2019).

Embora este trabalho tenha focado no BESM, as técnicas de análise e otimização aqui apresentadas podem ser aproveitadas em modelos computacionais similares, que requerem grande quantidade de processamento e são estruturados em módulos correspondentes a diferentes processos físicos.

1.1 Organização do trabalho

O restante deste documento está organizado da forma indicada a seguir. O Capítulo 2 descreve o modelo climático brasileiro (BESM), destacando sua relevância para o desenvolvimento científico no país, e aponta a necessidade de tais modelos serem executados em supercomputadores, explicando brevemente as técnicas de paralelização mais utilizadas no meio meteorológico para otimizar a execução. O capítulo também apresenta outros trabalhos relacionados.

O Capítulo 3 explora o desempenho original do BESM, analisando como este é executado, sua escalabilidade e identificando *hotspots*. No Capítulo 4 são apresentadas as otimizações exploradas, as investigações acerca delas e os correspondentes resultados obtidos. Por fim, o Capítulo 5 relata as conclusões deste trabalho e aponta para temas que podem ser explorados em trabalhos futuros.

2 CONCEITOS PRELIMINARES

Neste capítulo, primeiramente é apresentada uma descrição do BESM, incluindo sua origem, constituição e participação em trabalhos e pesquisas. Na seção seguinte, são apresentadas algumas técnicas utilizadas na execução e otimização de modelos de tempo e clima. Em seguida, a necessidade de otimizar o BESM é destacada, e são citados trabalhos que também exploraram a otimização de desempenho de modelos similares.

2.1 Brazilian Earth System Model

Brazilian Earth System Model (BESM) é um esforço de várias instituições e pesquisadores, liderado pelo Instituto Nacional de Pesquisas Espaciais (INPE) através do Centro de Previsão de Tempo e Estudos Climáticos (CPTEC), para construir um *framework* de pesquisa multidisciplinar com o intuito de entender as causas das mudanças climáticas globais, seus efeitos e impactos na sociedade (NOBRE et al., 2013).

A primeira versão do modelo, denominada *Brazilian Earth System Model Ocean-Atmosphere version 2.3* (BESM-OA 2.3), foi desenvolvida seguindo os critérios para participar na quinta fase do *Coupled Model Intercomparison Project* (CMIP5), explicado adiante. O objetivo era simular o comportamento do sistema acoplado oceano-atmosfera, em escalas de tempo decadais, sob variação da concentração de gases do efeito estufa na atmosfera (NOBRE et al., 2013).

O BESM-OA 2.3 é uma evolução de versões anteriores do modelo acoplado oceano-atmosfera do CPTEC, que vinha sendo utilizado para estudar os impactos do desflorestamento da Amazônia no clima (NOBRE et al., 2009); a dinâmica da zona de convergência do Atlântico Sul sobre águas frias (CHAVES; NOBRE, 2004; NOBRE et al., 2012); a importância do acoplamento oceano-atmosfera para representar a profundidade da termoclina do Atlântico (SIQUEIRA; NOBRE, 2006); e previsões sazonais regionais com o modelo regional Eta sobre a América do Sul (PILOTTO et al., 2012). Todos estes trabalhos destacaram a importância de um modelo acoplado oceano-atmosfera para melhor prever o fenômeno a ser estudado (NOBRE et al., 2013).

O BESM-OA 2.3 é constituído pelo Modelo de Circulação Geral Atmosférica (*Atmospheric Global Circulation Model* - AGCM) do CPTEC/INPE, acoplado ao *Modular Ocean Model version 4p1* (MOM4p1) do *Geophysical Fluid Dynamics Laboratory* (GFDL). O acoplamento é feito pelo *Flexible Modular System* (FMS), também do

GFDL. Apesar da simplicidade do BESM-OA 2.3 quando comparado com ESMs completos, seus resultados são comparáveis com outros modelos de clima global (NOBRE et al., 2013).

Um pesquisador do INPE, colaborador no desenvolvimento do BESM, entrevistado por Miguel e Monteiro (2015) diz:

> "Quando acoplamos outros modelos, não estamos somente copiando e colando, estamos acrescentando novos desenvolvimentos, novos parâmetros que não estão representados neles, como a interação da floresta Amazônica e do oceano Atlântico Sul."

2.1.1 Módulo atmosférico

O AGCM do CPTEC/INPE tem sido constantemente modificado, incluindo novos núcleos dinâmicos (Euleriano e semi-Lagrangiano), algoritmos para uso de grades reduzidas, módulos de biosfera, turbulência, ondas gravitacionais, convecção, radiação e nuvens (NOBRE et al., 2013).

As trocas de calor, umidade e momento entre a superfície e atmosfera são computadas de forma diferente sobre o oceano e sobre os continentes. Vários processos físicos definem os fluxos de superfície (NOBRE et al., 2013).

Sobre os oceanos, os fluxos de momento, calor e umidade são parametrizados considerando o esquema aerodinâmico global, onde os fluxos são considerados proporcionais à velocidade do vento e potencial superficial (diferença de temperatura ou umidade entre superfície do oceano e o ar próximo à superfície) (NOBRE et al., 2013).

Sobre os continentes, a parametrização dos fluxos de momento, calor e umidade é mais sofisticada. É considerada uma variedade de tipos de vegetação, solo e informações geográficas (NOBRE et al., 2013).

2.1.2 Módulo oceânico

No BESM-OA 2.3, o modelo oceânico utilizado é o *Modular Ocean Model version 4p1* (MOM4p1) do GFDL, que inclui o modelo de gelo *Sea Ice Simulator* (SIS). O módulo oceânico recebe como parâmetros: vento, temperatura, umidade específica a 10 m acima da superfície, pressão a nível do mar, fluxos radiativos, cobertura e espessura de gelo do mar, e descarga dos rios (NOBRE et al., 2013).

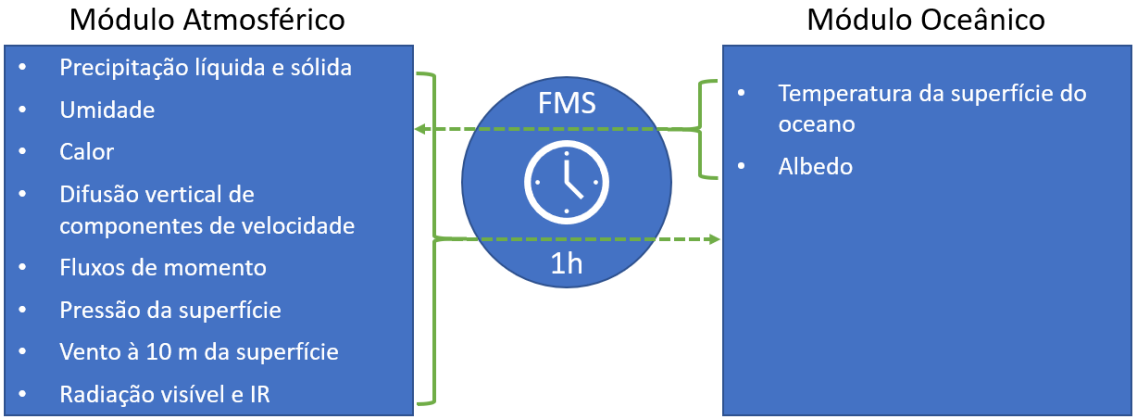
O SIS possui três camadas verticais (uma de neve e duas de gelo) e cinco categorias de espessura de gelo. O modelo calcula a concentração, espessura, temperatura, teor de salmoura, e cobertura de neve em um número arbitrário de categorias de espessura de gelo, bem como o movimento do conjunto como um todo. Além disso, o modelo também é responsável por calcular os fluxos gelo/oceano, e fluxos entre oceano e atmosfera (CASAGRANDE et al., 2016).

A inicialização do OGCM é feita com velocidades de corrente nulas, então são simulados 13 anos sendo forçados com campos atmosféricos climatológicos, seguido por mais 58 anos de simulação forçada por campos interanuais. A descarga dos rios e as variáveis do gelo marítimo são mantidas climatológicas em ambos os casos. Durante a execução do modelo forçado, a estrutura dinâmica e termodinâmica do oceano é salva em períodos regulares, de modo a ser usada como estado inicial do oceano para experimentos com o modelo acoplado (NOBRE et al., 2013).

2.1.3 Acoplamento

O BESM utiliza o *Flexible Modeling System* (FMS), do GFDL, para acoplar o AGCM do CPTEC ao MOM4p1 e SIS do GFDL (Figura 2.1). O AGCM recebe do MOM e SIS a temperatura da superfície marítima (SST) e o albedo oceânico, a cada hora simulada (*timestep* de acoplamento) (NOBRE et al., 2013).

Figura 2.1 - Acoplamento dos módulos do BESM-OA 2.3



Fonte: Produção do autor.

As variáveis de acoplamento fornecidas pelo AGCM são: água doce (precipitação liquida e sólida), umidade específica, calor, difusão vertical de componentes de velocidade, fluxos de momento e pressão da superfície. Campos de tensão do vento são computados, no MOM, a partir dos campos de vento 10 m acima da superfície do oceano, recebidos do AGCM. Ajustes são feitos nos parâmetros de penetração de ondas curtas no oceano de acordo com a radiação visível e infravermelha de ondas curtas fornecido pelo AGCM (NOBRE et al., 2013).

2.2 Participação no CMIP

Climate Model Intercomparison Project (CMIP) é um *framework* padronizado para avaliar e comparar simulações de clima geradas por modelos acoplados oceano-atmosfera. Este *framework* é importante no avanço do entendimento da variabilidade do clima e mudança do clima global, uma vez que ele disponibiliza a qualquer pesquisador ou instituição um conjunto atualizado de dados multimodelos livremente acessível. Os dados do CMIP estão disponíveis no site do *Earth System Grid Federation* (ESGF), que mantém uma rede global de data-centers de modo a disponibilizar mundialmente o acesso a um grande acervo de dados climáticos (CASAGRANDE et al., 2016).

O CMIP5 fez parte do *Intergovernmental Panel on Climate Change* (IPCC), contribuindo com dados para compor o *Fifth Assessment Report* (AR5). O conjunto de dados foi analisado pela comunidade científica mundial, e continua disponível para estudos do clima, sua variabilidade e mudança, além das implicações sociais e ambientais das mudanças climáticas em termos de impactos, adaptação e vulnerabilidade (EMORI et al., 2016).

A participação do BESM no IPCC5 ocorreu com o modelo atmosférico do CPTEC acoplado apenas ao modelo oceânico MOM. Isso causou sérias dificuldades para a aceitação das simulações do modelo no IPCC5. Para o próximo IPCC, o BESM não poderá participar do projeto se não tiver acoplado o modelo de aerossóis aos já utilizados (MIGUEL; MONTEIRO, 2015).

Está sendo buscado acoplar ao BESM: o modelo de química atmosférica e aerossóis (ECHAN - HAMMOZ, *Max-Planck Institut*, Alemanha (HAMMOZ, 2019)); e o modelo de vegetação da superfície (IBIS, *University of Wisconsin*, EUA (UNIVERSITY OF WISCONSIN-MADISON, 2019)). Somente quando todos esses componentes estiverem executando conjuntamente o BESM será um modelo do sistema terrestre "padrão IPCC" (MIGUEL; MONTEIRO, 2015).

Segundo um pesquisador do INPE entrevistado por Miguel e Monteiro (2015),

> "Alcançar o padrão IPCC significa participar de um grupo seleto de países com a capacidade de gerar futuros climáticos que compõem a base do conhecimento em mudanças climáticas avaliado pelo painel internacional. Este é o mais alto nível científico das pesquisas em modelagem das mudanças climáticas."

2.3 Relevância do BESM

Como citado anteriormente, o BESM foi desenvolvido visando a participação no CMIP5. Além da participação nesta comunidade mundial de estudos do clima global, o BESM tem dado suporte para a publicação de diversos artigos, teses de doutorado, dissertações de mestrado, e até mesmo iniciações científicas.

Em sua tese de doutorado, Casagrande (2017) avaliou a capacidade do BESM de representar o gelo marinho Ártico e a sensibilidade ao forçamento radiativo de CO2. Os resultados foram validados comparando-os com observações de satélite e também com os dados do CMIP5. Os resultados do BESM se mostraram consistentes com os modelos do CMIP5 e com as observações de satélite.

Fialho (2017) estudou em sua dissertação de mestrado a importância do acoplamento oceano-atmosfera para a previsão da Zona de Convergência do Atlântico Sul. Foram realizados experimentos com o modelo acoplado e somente com sua componente atmosférica. O modelo acoplado teve resultados de precipitação mais próximos do observado, apesar de serem valores subestimados em todos os casos.

Callegare e Giarolla (2016) estudaram em sua iniciação científica os impactos da utilização de uma grade de alta resolução horizontal na componente oceânica do BESM. Embora as simulações com alta resolução ainda estejam divergindo após seis anos simulados, as correções topográficas realizadas no trabalho aumentaram a estabilidade do modelo.

Giarolla et al. (2015) comparam uma versão mais nova do BESM (2.3.1), que inclui mudanças no esquema de cobertura de nuvens e propriedades ópticas da atmosfera, com a versão anterior (2.3) e mais sete modelos do CMIP5. A nova versão apresenta melhorias na variação de temperatura da superfície na região equatorial.

Chou et al. (2014) testaram um modelo climático regional (RCM) acoplado a três modelos globais: HadGEM2-ES, BESM e MIROC5. O RCM cobre América do Sul,

América Central e Caribe. Os autores destacam que RCMs embutidos em modelos globais fazem o papel de prover detalhes necessários para estudos de impacto local, como áreas urbanas, plantações, etc.

Além dos trabalhos já publicados, os pesquisadores do CPTEC/INPE continuam aprimorando o BESM. De acordo com Nobre et al. (2016), estão em processo de implementação: modelo de superfície com vegetação dinâmica e hidrologia, efeitos dos aerossóis, e química atmosférica; bem como a inclusão de novas parametrizações de convecção atmosférica que melhor representem a precipitação sobre a América do Sul.

2.4 Execução de GCMs e técnicas relacionadas

Devido à inclusão da modelagem matemática de diversos fenômenos físicos, os modelos globais se tornam programas computacionalmente complexos, o que exige elevada quantidade de processamento para sua execução. Por este motivo, tais modelos são geralmente executados em supercomputadores, que atualmente são em maioria clusters.

Clusters são aglomerados de computadores (nós) conectados por uma rede de comunicação de alta velocidade, que permite que o processamento dos dados seja distribuído entre os nós. Além de ser distribuído entre os nós, o processamento ainda pode ser dividido entre os vários núcleos de processamento existentes em cada nó. A técnica de dividir um programa em "pedaços" que são executados ao mesmo tempo (em paralelo) é chamada de paralelização.

Duas técnicas fortemente consolidadas de paralelização são a ulização das bibliotecas OpenMP (OPENMP, 2019) para paralelização intra-nó, e *Message Passing Interface* (MPI) (MPI FORUM, 2019) para paralelização tanto intra quanto inter-nó.

2.4.1 OpenMP

O OpenMP executa um programa como um único processo no sistema operacional. Este processo pode inicializar várias *threads* que trabalham em paralelo em determinadas regiões do código, especificadas pelo programador através da inserção de diretivas no código-fonte. Este esquema é também conhecido como *fork-join*, onde a execução ocorre sequencialmente em uma *thread* principal e, ao atingir uma região paralelizável (geralmente *loops*), diversas *threads* são inicializadas e o trabalho é dividido entre elas (*fork*). Ao fim da execução desta região, os resultados são unificados (*join*) e a execução volta a ser realizada sequencialmente até atingir outra

região paralelizável (BARNEY, 2020).

Neste tipo de execução, as *threads* têm acesso à mesma região de memória, ou seja, a memória é única para todo o processo. Este comportamento possui pontos positivos e negativos, sendo os principais: o acesso direto à memória agiliza a troca de informações entre as *threads*; entretanto, deve-se tomar cuidado com a concorrência de acesso a memória, ocasião onde múltiplas *threads* tentam acessar e/ou modificar um mesmo endereço de memória simultaneamente, causando inconsistência nos dados.

2.4.2 MPI - Message Passing Interface

Ao contrário do OpenMP, o MPI executa um programa inicializando vários processos. Cada processo possui sua região privada de memória, e a comunicação entre eles é feita através de troca de mensagens. Os processos são cópias do programa principal, e cabe ao programador definir quais tarefas serão executadas em cada processo de acordo com seu identificador (*rank*). Também é comum que um processo escolhido como principal gerencie a entrada e saída de dados, dividindo o processamento dos mesmos entre os demais processos.

Assim como no OpenMP, existem pontos positivos e negativos nesta abordagem de paralelização: como cada processo possui sua região de memória, não há concorrência de acesso entre os mesmos; e a existência de vários processos permite que sejam executados em múltiplos nós computacionais. Por outro lado, a comunicação por troca de mensagens se torna mais lenta que o acesso direto à memória devido ao *overhead* causado pelo empacotamento e transmissão das mensagens.

2.4.3 Paralelização híbrida

Existem ainda implementações que fazem proveito das duas técnicas, utilizando comunicação entre nós via MPI, e paralelização intra-nó pelo OpenMP. Assim, tira-se proveito da agilidade do acesso direto à memória ofertado pelo OpenMP, somado à escalabilidade do MPI trazida pela possibilidade de distribuir a execução do programa em múltiplos nós computacionais. Entretanto, esta abordagem aumenta a complexidade de programação e divisão de tarefas podendo até mesmo reduzir o desempenho caso a distribuição do processamento não seja feita de forma efetiva e balanceada. Esta mistura de técnicas é conhecida como paralelização híbrida.

2.4.4 Otimizações relacionadas à arquitetura de hardawre

Além das técnicas de paralelização, ainda existem opções de otimização voltadas à arquitetura de hardware utilizada. As mais comuns são vetorização e blocagem. A primeira tem como objetivo processar múltiplos dados com uma única instrução do processador, conhecida como instrução vetorial. A segunda busca melhorar a utilização da memória cache do processador, processando um conjunto reduzido de dados por vez.

2.4.4.1 Vetorização

Processadores modernos fornecem conjuntos de instruções otimizados para determinados tipos de operações, como por exemplo SSE (*Streaming SIMD Extensions*) e AVX (*Advanced Vector Extensions*). Essas instruções tem como objetivo executar uma mesma operação em múltilos dados simultaneamente, em vez de operar em um único dado por vez; e são conhecidas como instruções SIMD (*Single Instruction, Multiple Data*), ou instruções vetoriais. Por este motivo, a utilização de instruções SIMD (vetorização) aumenta consideravelmente o desempenho de processamento (INTEL, 2018).

Loops são grandes candidatos à vetorização, pois geralmente realizam uma mesma operação em uma sequência de dados. Exemplos de operações que podem ser realizadas via instruções SIMD são: aritmética (soma, multiplicação, subtração, divisão, mínimo, máximo, raiz quadrada, etc); comparação; embaralhamento de dados; conversão de tipo de dado; operações bit a bit; movimentação de dados entre memória e registradores; e gerenciamento de memória cache.

É papel do compilador converter o código-fonte (C, Fortran, Java, etc) em instruções para o processador, bem como decidir qual conjunto e sequência de instruções trará maior desempenho ao programa resultante. Desta forma, existem *flags* de otimização que habilitam a vetorização no momento da compilação.

Por vezes, ocorre de o compilador não conseguir identificar se um determinado trecho de código é vetorizável, compilando-o com instruções comuns. Em algumas linguagens de programação, como Fortran e C/C++, quando o programador sabe de antemão que um determinado trecho é vetorizável, é possível inserir diretivas de compilação no código que informam ao compilador que aquele trecho pode ser vetorizado, aumentando o desempenho do programa.

2.4.4.2 Blocagem

Os processadores atuais possuem uma memória interna, pequena porém extremamente rápida, conhecida como memória cache. Esta memória serve como intermediário entre a memória RAM e os registradores da CPU, aumentando a velocidade de acesso aos dados que serão processados. Quando um endereço de memória é acessado, primeiramente é verificado se o dado correspondente se encontra na cache. Em um primeiro momento este não estará, resultado em um *cache miss* (dado não encontrado na cache). Então, o dado é buscado na memória RAM e copiado para a cache. No caso de um segundo acesso à este dado, o mesmo já se encontrará na cache, resultando em um *cache hit* (dado encontrado na cache). Logo, o dado é acessado de forma mais rápida, sem a necessidade de buscá-lo na memória RAM.

Quando um dado é copiado para a cache, este vem acompanhado de outros dados presentes nos endereços de memória adjacentes. Mais especificamente, um bloco de memória é copiado para uma linha de cache que armazena vários elementos. Desta forma, na ocorrência de um acesso sequencial aos endereços de memória, o próximo elemento já estará na cache.

Como a memória cache é pequena, o acesso a novos endereços de memória causa a sobrescrita dos dados mais antigos da cache (*cache thrashing*). Uma vez que a cópia dos dados é realizada em blocos que ocupam toda uma linha de cache, o *cache miss* de um único endereço de memória pode fazer com que diversos outros endereços que seriam utilizados em um próximo instante sejam sobrescritos, fazendo com que ocorra um novo *cache miss*.

Um exemplo de situação onde isto ocorre é em uma operação com matrizes onde uma delas é acessada no sentido das linhas, outra no das colunas. As matrizes são armazenadas de forma contínua na memória, com as colunas (ou linhas - dependendo da linguagem de programação utilizada) dispostas uma seguida da outra. Desta forma, quando o acesso é realizado percorrendo uma linha, o passo será correspondente à quantidade de elementos na coluna; enquanto o acesso no sentido da coluna terá passo unitário. O código abaixo exemplifica (em Fortran) uma operação onde a matriz `A` é acessada com passo unitário, enquanto `B` é acessada com passo `N`.

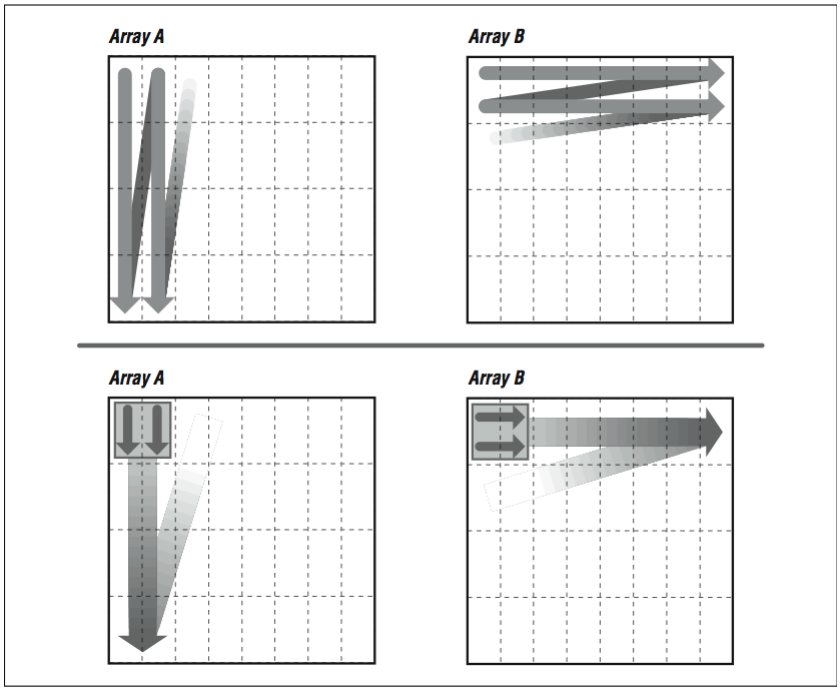
```
DO I=1,N
    DO J=1,N
      A(J,I) = A(J,I) + B(I,J)
    ENDDO
ENDDO
```

Em casos deste tipo, onde apenas inverter a ordem do aninhamento dos *loops* não resolve o problema, a técnica de blocagem pode ser utilizada. A blocagem consiste em processar pequenos blocos da matriz por vez, de forma que os endereços de memória manipulados estarão próximos e caberão na memória cache sem a sobrescrita dos dados, aumentando assim a quantidade de *cache hits* (SEVERANCE; DOWD, 2010).

A primeira linha da Figura 2.2 ilustra as matrizes e como ocorrem os acessos de acordo com o código acima. Os retângulos pontilhados indicam a quantidade de dados correspondente a cada linha de cache. No código original, sem aplicação de blocagem, a matriz A tem seus elementos acessados sequencialmente (no sentido vertical para a linguagem Fortran); logo, a maior parte dos acessos terá *cache hit*.

Considerando que as matrizes tenham uma grande quantidade de linhas, cada coluna conterá muitos elementos, de forma que a quantidade de linhas de cache não será suficiente para armazenar múltiplas colunas de B. Logo, como os elementos de B são acessados no sentido horizontal e os dados são armazenados na memória no sentido vertical (considerando Fortran), a matriz B tem seus elementos acessados com saltos correspondentes à quantidade de linhas da matriz, então cada acesso sofrerá um *cache miss* impactando negativamente no desempenho do programa.

Figura 2.2 - Ilustração da técnica de blocagem



Fonte: Severance e Dowd (2010).

A segunda linha da Figura 2.2 mostra como se torna o acesso após a aplicação da técnica de blocagem. Considerando que cada linha de cache (retângulo pontilhado) armazene dois elementos da matriz. Limitando a operação à blocos 2x2, serão utilizadas apenas quatro linhas de cache (duas para A e duas para B), o que provavelmente caberá na memória cache sem necessidade de sobrescrita. Desta forma, no acesso ao elemento (1,1) da matriz B será carregada uma linha de cache que também contém o elemento (2,1); em seguida o elemento (1,2) será utilizado, também carregando o elemento (2,2). Nas iterações posteriores os elementos (2,1) e (2,2) já estarão na cache, evitando que sejam buscados na RAM. Enquanto isso, os acessos aos elementos de A continuam sendo realizados de forma sequencial em cada linha de cache.

2.5 Necessidade de otimização do BESM

As versões iniciais do BESM foram implementadas com o intuito de executar os experimentos utilizando o supercomputador do CPTEC, o Tupã (INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS - INPE, 2012). Atualmente, o modelo vem sendo portado para também ser executado no supercomputador do Laboratório Nacional de Computação Científica (LNCC), o Santos Dumont, que possui arquitetura de hardware mais recente que o primeiro (LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA - LNCC, 2019). Com mais supercomputadores aptos a executar o modelo, amplia-se o acesso dos pesquisadores ao mesmo.

Como tais supercomputadores são de uso compartilhado por vários pesquisadores da comunidade científica brasileira, por vezes se torna concorrida a execução de experimentos, fazendo-se necessário aguardar em uma fila de execução que pode se prolongar por dias. Quanto maior o tempo de execução e quantidade hardware requisitados, maior é o tempo de espera na fila de execução até que os recursos computacionais sejam alocados.

Este fator por si só já se mostra um bom incentivo para que os modelos implementados façam o maior proveito possível do hardware disponibilizado, viabilizando que experimentos sejam realizados mais rapidamente e sem a necessidade de alocação de grande quantidade de recursos computacionais. Isto agiliza tanto o início da execução do experimento quanto o tempo necessário para a simulação em si, facilitando estudos científicos que dependam dos resultados do modelo.

Porém, o principal motivo para se otimizar um modelo climático é que, com uma implementação mais eficiente, é possível realizar previsões para futuros mais distantes e/ou mais precisas com o uso de modelos mais refinados, que a partir de um

código não otimizado se tornariam computacionalmente impraticáveis.

Confirmando o acima dito, em uma apresentação realizada pela equipe de desenvolvimento do BESM (KUBOTA, 2015), é destacada a necessidade de se otimizar o modelo para que simulações com resolução mais refinada (∼10 km) sejam viáveis. Naquela apresentação foram mostrados testes onde a utilização de 30.000 CPUs (quase o supercomputador Tupã todo) ofereceu ganho de apenas 34% em relação à execução com 8.400 CPUs. Isto representa uma escalabilidade com eficiência de somente 37% entre esses dois casos, indicando um desperdício expressivo de poder computacional.

Desta forma, um modelo computacionalmente mais eficiente agiliza a execução de previsões e experimentos, bem como possibilita que mais detalhes sejam inseridos na modelagem, aumentando a fidelidade da representação matemática dos fenômenos.

2.6 Trabalhos relacionados

Além dos trabalhos diretamente ligados ao BESM citados anteriormente, vale destacar outros estudos sobre o desempenho de modelos de tempo e clima.

Rodrigues et al. (2010) realizam uma análise comparativa de algoritmos de balanceamento de carga aplicados ao BRAMS, um modelo de previsão de tempo brasileiro. O trabalho destaca que o desbalanceamento de carga causado por variações no estado da atmosfera é comum nos modelos, e o mais difícil de tratar.

A técnica de balanceamento aplicada pelos autores não necessita de modificações relevantes no código do modelo, pois emprega o conceito de virtualização de processadores. Em vez de inserir o balanceador de carga no código da aplicação, é utilizada uma implementação virtualizada do MPI para desacoplar a estratégia de balanceamento do modelo em si. A motivação para estudo desta técnica é a dificuldade de implementar o balanceamento de carga em códigos legados, comuns em modelos de tempo e clima. A virtualização de processadores foi utilizada por meio do AMPI, uma implementação do MPI que permite instanciar mais processadores virtuais do que a quantidade de CPUs reais da máquina, e migrar o trabalho entre as CPUs dinamicamente, buscando balancear a carga entre eles.

Na comparação dos algoritmos de balanceamento de carga foram utilizados 1024 processadores virtuais em uma máquina com 64 CPUs reais. Apenas a virtualização das CPUs, sem balanceamento de carga, trouxe uma redução no tempo de execução de 25%. Aplicando algoritmos de balanceamento, a redução chegou a 32%. Ajustando

o intervalo entre cada invocação do balanceamento (em número de *timesteps*) e o *threshold* (em porcentagem de desbalanceamento), foi alcançada a redução máxima de 37% no tempo de execução do modelo.

Em sua dissertação de mestrado, Souza (2019) otimiza o desempenho computacional de um modelo de previsão de irradiação solar, o BRASIL-SR. O modelo originalmente era executado de forma sequencial e os arquivos de entrada e saída eram em formato binário ou texto.

O autor avaliou técnicas de paralelização de *loops* com diretivas OpenMP, aplicação das técnicas de blocagem e vetorização, e também a utilização da biblioteca NetCDF para gerenciar as operações de entrada/saída do modelo.

A paralelização dos *loops* com diretivas OpenMP aliada à manipulação dos arquivos por meio da NetCDF permitiram que, com o uso de 24 *threads*, o tempo de execução do modelo fosse reduzido de 27 horas para 90 minutos. Já as técnicas de vetorização e blocagem não apresentaram ganhos no tempo de execução do modelo. O autor alega que a presenta de desvios nos *loops* (`go to`) impedia a vetorização dos mesmos e deixa esta alteração como sugestão de trabalho futuro.

As melhorias conseguidas permitiram que as simulações que antes eram executadas com resolução temporal mensal passassem a utilizar resolução diária. Também foram criados scripts que automatizaram todo o processo de execução do modelo, originalmente realizado manualmente, que inclui: coleta de dados de entrada de servidores FTP, pré-processamento destes dados, configuração do ambiente, geração de arquivos de configuração da execução do modelo, execução do modelo e tratamento dos dados de saída.

3 DESEMPENHO ORIGINAL DO BESM

Para estudo neste trabalho, foi fornecido pelo CPTEC o código-fonte do BESM versão 2.6, que possui aproximadamente 1,6 milhão de linhas de código distribuídas em mais de 1200 arquivos, sendo quase toda a implementação em linguagem Fortran 90, e alguns arquivos em C/C++. A principal diferença entre esta versão e a 2.3 explicada na seção anterior é a utilização do módulo oceânico MOM5 em substituição ao MOM4p1. Em sua versão mais recente, o MOM disponibiliza a opção de uso do submodelo Lagrangiano com interação dinâmica, ainda em fase de testes e não recomendado para ambiente de produção (GRIFFIES; ZADEH, 2019).

O BESM está preparado para usar MPI como seu principal método de paralelização. Esta abordagem lança vários processos que são gerenciados a nível de sistema operacional. Cada processo é responsável por processar uma fração do problema, tipicamente correspondente a uma sub-região do domínio representado no modelo.

O ponto principal de execução do BESM é o acoplador FMS. Ele é responsável por inicializar os módulos individuais, gerenciar suas sequências de execução e a troca de informação entre eles. O FMS implementa uma interface de paralelização chamada MPP. Seu propósito é abstrair tarefas relacionadas à paralelização, como sincronização dos processos e seleção de comunicador (conjunto de processos que estarão envolvidos em uma determinada troca de mensagens). As chamadas MPI são implementadas sob o MPP.

Embora a implementação do modelo acoplado vise paralelizar sua execução via processos MPI, os modelos isolados (atmosfera e oceano) possuem diretivas OpenMP (OMP) em seus códigos-fonte. Entretanto, tais implementações são voltadas para a execução isolada, sem acoplamento entre eles. O fato de possuírem diretivas OMP também não garante que estas estejam sendo usadas de forma eficiente.

Conforme relatado pelo pesquisador do CPTEC que forneceu o código-fonte do modelo, até aquele momento, o modelo acoplado não havia sido testado com o OMP dos módulos ativado, apenas com MPI. Desta forma, a execução dos testes de desempenho sobre o modelo foi dividida em duas etapas: a primeira utilizando somente MPI para paralelização de processos; e a segunda utilizando conjuntamente MPI e OMP, onde cada processo MPI dispara um determinado número de *threads* em suas regiões paralelizáveis, o que caracteriza uma paralelização híbrida.

3.1 Plataforma de execução do BESM

Uma das plataformas utilizadas para executar o BESM é o sistema Santos Dumont, do LNCC (LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA - LNCC, 2019). O sistema possui uma capacidade total de processamento de 1,1 Petaflops, é constituído de 18.144 núcleos de CPU, distribuídos em 756 nós computacionais (24 núcleos por nó - 2x Intel Xeon E5-2695v2), com 64 GB de memória RAM DDR3 cada. Os nós são interconectados por meio de uma rede InfiniBand FDR com *throughput* de 58 Gbps por porta. Neste trabalho, os testes e melhorias aplicados ao BESM foram executados no Santos Dumont. Todas as execuções tiveram um núcleo de CPU dedicado para cada processo MPI.

Os testes foram executados baseados em simulações de 1 mês de clima, com resolução T062L28 ($1,8758^0$ de latitude e longitude – 200 km no Equador, e 28 camadas verticais) para o módulo atmosférico (AGCM); e 1^0 de longitude, resolução latitudinal de $0,25^0$ na região tropical a 2^0 nos hemisférios, e 50 camadas verticais para o módulo oceânico (MOM).

3.2 Análise MPI

A análise do BESM sendo executado somente com MPI foi dividida em duas fases: escalabilidade do modelo e identificação de *hotspots*.

3.2.1 Escalabilidade do modelo

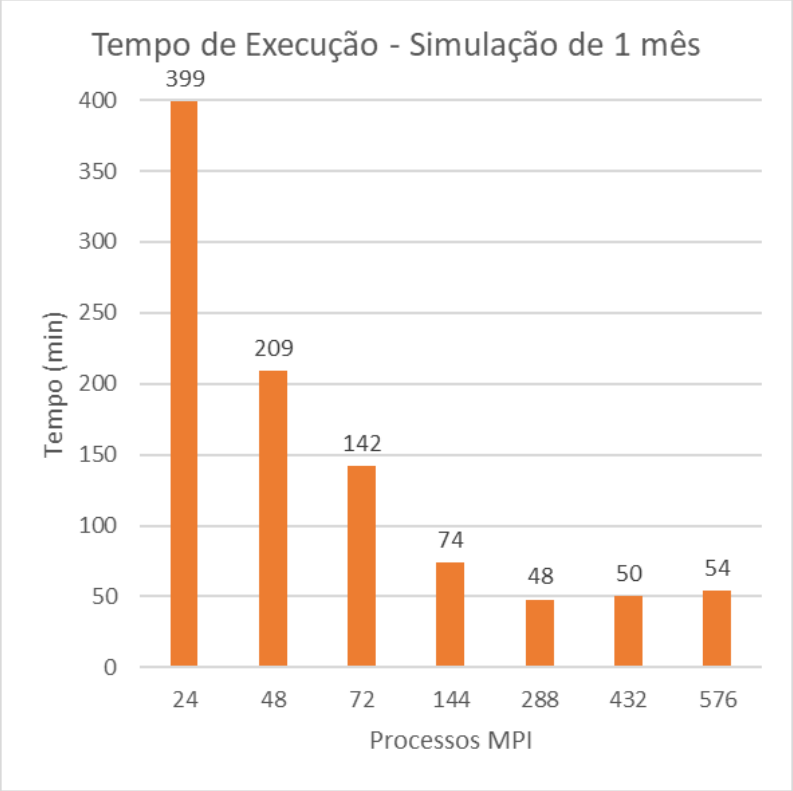
A primeira análise realizada foi sobre como o BESM estava sendo compilado, incluindo o compilador utilizado, sua versão e *flags* de otimização. O modelo estava sendo originalmente compilado com GCC 5.3.0 e OpenMPI 2.0.1. A única *flag* de otimização utilizada era `-O2`, que inclui apenas otimizações que não envolvem a troca de espaço (memória) por velocidade. Então, o modelo foi compilado utilizando todos os valores originais que vieram nos scripts, e foram extraídas estatísticas da execução de simulações de um mês para diferentes quantidades de CPUs alocados para executar o modelo. Os dados extraídos foram: tempo de execução, *speedup* e eficiência.

3.2.1.1 Tempo de execução

Os testes foram executados com 24, 48, 72, 144, 288, 432 e 576 processos MPI, sendo cada processo executado em uma CPU, com o intuito de verificar até que ponto haveria ganho de desempenho ao se adicionar mais processos (consequentemente

mais CPUs) na execução do modelo. A Figura 3.1 mostra o tempo (em minutos) necessário para executar a simulação com as quantidades de CPUs citadas. Iniciando com 399 minutos para simular um mês de clima usando 24 processos MPI, o tempo de execução mínimo foi atingido com 288 processos (48 minutos). Após este ponto, quando a quantidade de processos foi aumentada, o tempo de execução passou a aumentar, atingindo 50 minutos para 432 processos e 54 minutos para 576 processos.

Figura 3.1 - Tempos de execução do BESM original para simulações de um mês

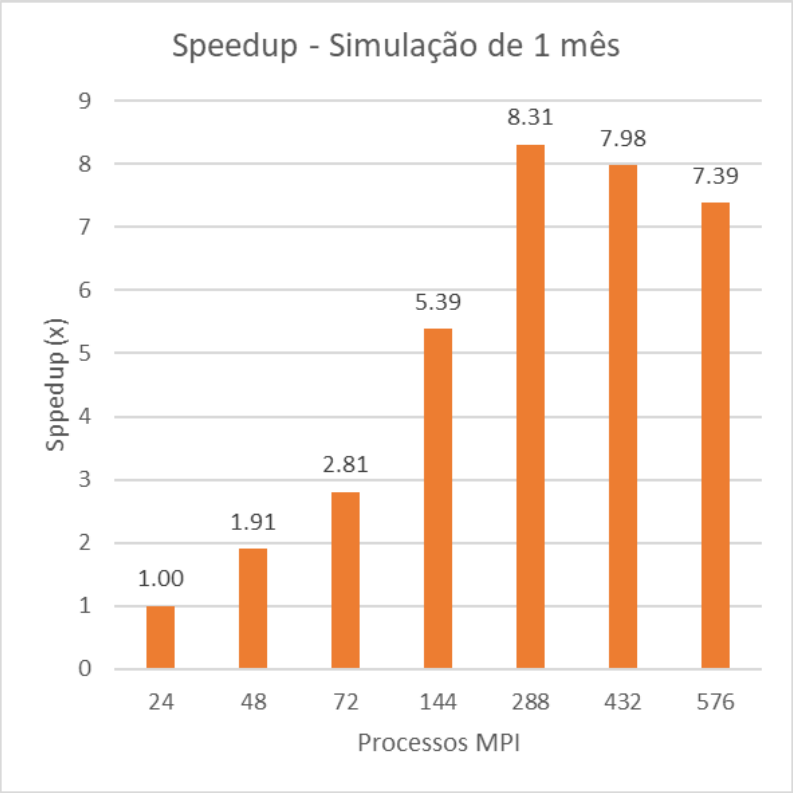


Fonte: Produção do autor.

3.2.1.2 *Speedup*

A partir dos tempos obtidos nos testes anteriores, foi calculado o *speedup*, que é o ganho relacional de tempo entre duas execuções. No caso, a execução com 24 processos foi tomada como referência para as demais, sendo considerada com *speedup* unitário. A Figura 3.2 mostra o *speedup* relacionado a cada quantidade de processos MPI. Neste gráfico, é fácil notar o pico de *speedup* aos 288 processos, com pouco mais de 8 vezes o desempenho da execução com 24 processos.

Figura 3.2 - *Speedups* obtidos para simulações de um mês com o BESM



Fonte: Produção do autor.

Este comportamento indica que o BESM, neste caso, não tira proveito de mais de 300 CPUs, e qualquer hardware adicional alocado para sua execução seria um desperdício de recursos computacionais. Isto também marca o limite de escalabilidade do modelo, considerando a maneira em que ele estava originalmente implementado.

3.2.1.3 Eficiência

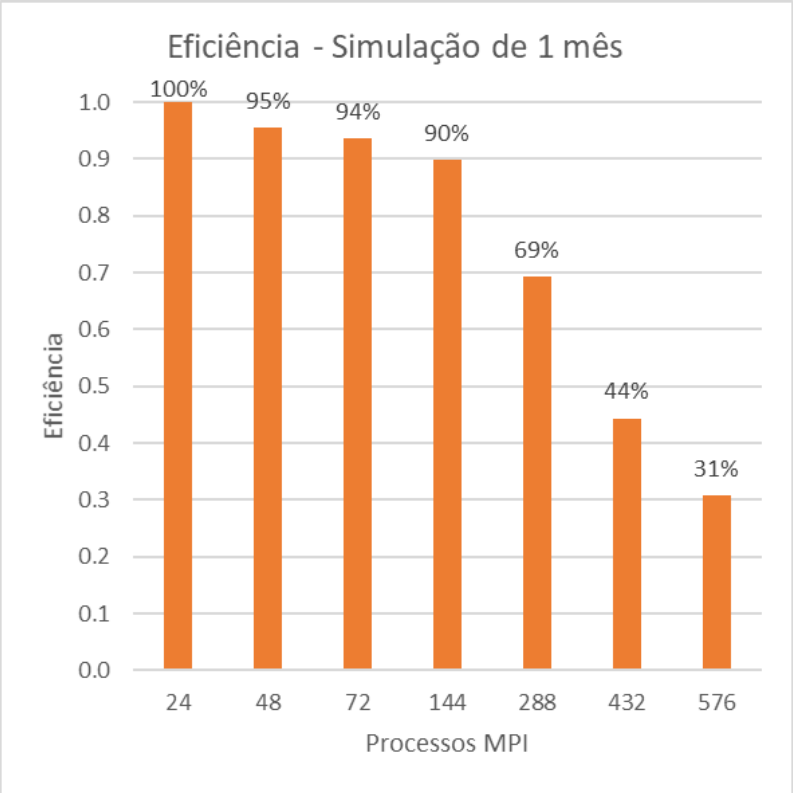
De forma a se ter uma avaliação quantitativa sobre o aproveitamento adequado do hardware adicionado para execução de um programa, calcula-se a eficiência de paralelização. Esta é definida como a relação entre o *speedup* obtido e a quantidade de processos, *threads*, ou CPUs necessários para alcançar aquele desempenho. A eficiência ideal é unitária, indicando total aproveitamento do hardware adicional. Por exemplo, ao dobrar o número de processadores, idealmente o tempo de execução cairia pela metade, resultado em *speedup* de duas vezes e eficiência um.

Caso um programa tenha eficiência aproximadamente unitária e constante diante da adição de mais processos, entende-se que este está aproveitando adequadamente o hardware adicional, e o desperdício de recursos computacionais em relação à pa-

ralelização é desprezível. Porém, este é um caso utópico e raramente encontrado em aplicações reais.

A Figura 3.3 ilustra a eficiência de paralelização encontrada no BESM, respectiva a cada número de processos MPI. A execução com 24 processos foi tomada como referência, representando eficiência de 100%. O cálculo de eficiência expõe que com até 144 processos, havia ao menos 90% de aproveitamento do hardware. Embora a execução com 288 processos tenha resultado no melhor tempo e *speedup*, esta teve apenas 69% de eficiência. Com mais de 500 processos, a eficiência diminuiu para aproximadamente 30%.

Figura 3.3 - Eficiências para simulações de um mês com o BESM



Fonte: Produção do autor.

Após este ponto, as análises subsequentes foram realizadas usando o compilador Intel. Além do melhor desempenho da aplicação, o compilador Intel oferece melhor integração com as ferramentas de análise de desempenho da suíte Intel PSXE. Entretanto, para usar o compilador Intel foi necessário realizar mudanças no processo de compilação. Tais mudanças serão explicadas no Capítulo 4.

3.2.2 Identificação de *hotspots*

Sabendo que a escalabilidade do BESM está consideravelmente limitada, como mostram os resultados anteriores, a próxima etapa da análise foi buscar os motivos desta limitação. Uma primeira abordagem foi procurar por regiões do código onde o tempo de processamento é elevado em relação aos demais, pois é bastante provável que nestes pontos exista maior potencial de otimização. Estas regiões são conhecidas como *hotspots*. Um *hotspot* pode também indicar uma zona de alerta, onde codificação pobre ou lógica ineficiente pode estar presente.

Para tal tarefa, foi utilizada a ferramenta Intel VTune Amplifier, contida na suíte Intel PSXE. Testes foram executados com 72, 144 e 288 processos MPI, a fim de verificar o comportamento dos *hotspots* em relação ao número de processos.

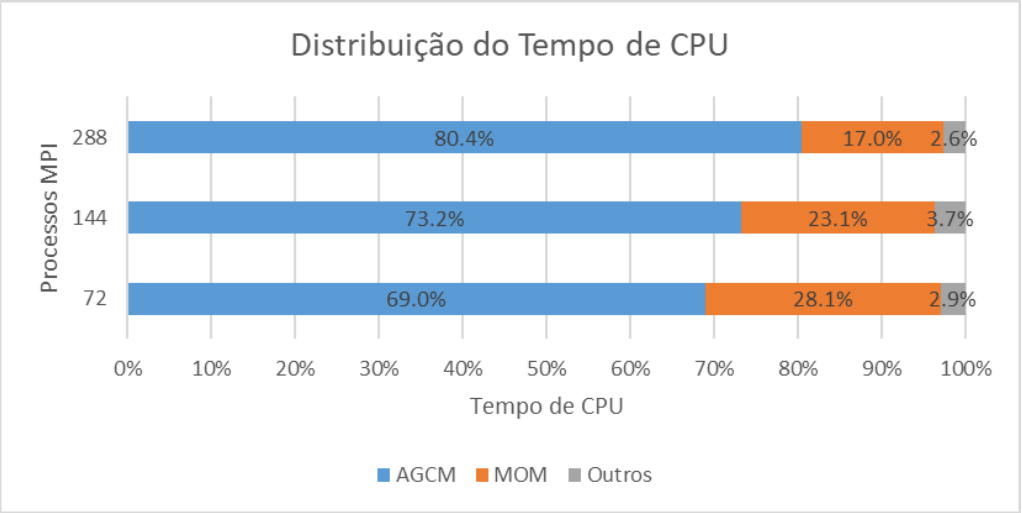
Uma das informações fornecidas pela ferramenta é a porcentagem do tempo de CPU gasto por cada função do programa. Esta proporção inclui também o tempo das subfunções contidas em cada função. Desta forma, o relatório gerado pela ferramenta começa a partir da função "`main`" com 100% de tempo de CPU, e vai distribuindo esses tempos nas subfunções.

3.2.2.1 Distribuição do tempo de CPU entre AGCM e MOM

No BESM, logo abaixo da função principal, existe a função `coupler_main`, responsável pelo acoplamento entre os módulos oceânico e atmosférico. Dentro da `coupler_main`, as chamadas aos modelos atmosférico e oceânico ocupam quase a totalidade do tempo de CPU.

A Figura 3.4 ilustra a proporção de tempo de CPU utilizada pelos modelos atmosférico (AGCM), oceânico (MOM), e pelas demais funções (incluindo inicialização e acoplamento). Nota-se que a fração referente ao AGCM aumenta proporcionalmente à quantidade de processos MPI, enquanto a fração respectiva ao MOM diminui. Já a inicialização do BESM e a comunicação entre os módulos não teve alteração significativa. Isto indica que, enquanto o MOM consegue escalar bem com a adição de mais processos, o AGCM possui deficiências neste aspecto.

Figura 3.4 - Proporção de tempo de CPU utilizada pelo AGCM e MOM



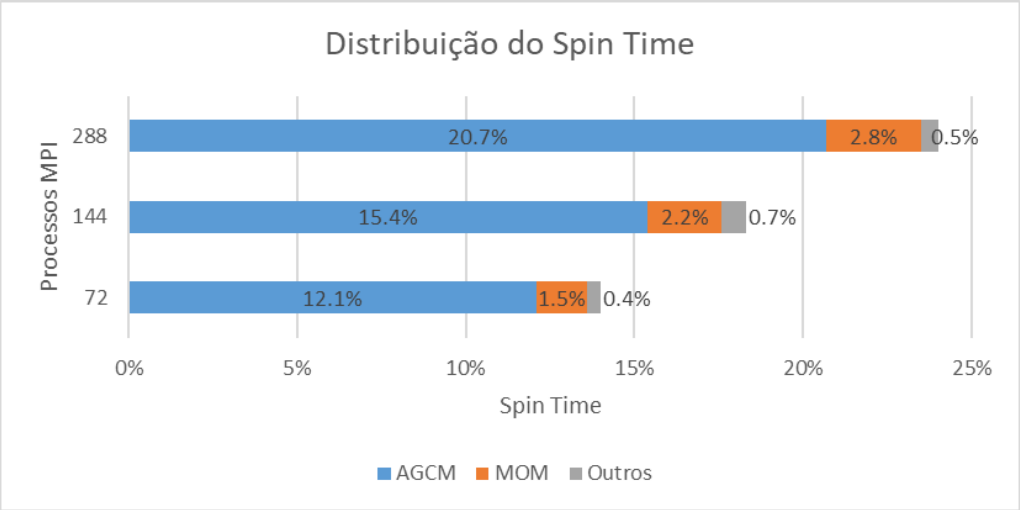
Fonte: Produção do autor.

3.2.2.2 *Spin time*

Antes de aprofundar em uma avaliação de *hotspots* a nível das subfunções, foi encontrada uma situação alarmante. O Intel VTune Amplifier reportava uma alta porcentagem de *spin time* na execução do BESM. A Intel define o *spin time* como tempo de espera no qual a CPU permanece ocupada, geralmente causado por APIs de sincronização (INTEL, 2019). Nestes casos, a CPU constantemente checa pelo sincronismo com outros processos ou *threads*, enquanto a execução do programa principal permanece aguardando.

Nos testes realizados, o *spin time* total do programa cresceu proporcionalmente à quantidade de processos utilizados (Figura 3.5). Parte expressiva desse *spin time* é proveniente do modelo atmosférico, chegando a quase 8 vezes a parcela correspondente ao MOM. No caso com 288 processos, o AGCM desperdiçou 20,7% do tempo total de execução somente com espera ocupada, enquanto o MOM foi responsável por 2,8%. Esta é uma situação preocupante, uma vez que quase um quarto do tempo total de CPU do modelo está sendo desperdiçado quando executado com 288 processos. A tendência é que esta proporção cresça conforme o aumento no número de processos utilizados.

Figura 3.5 - *Spin time* conforme a quantidade de processos MPI utilizados na execução do BESM



Fonte: Produção do autor.

Sabendo que mitigar o *spin time* poderia melhorar a escalabilidade do BESM, foram exploradas as causas deste tempo desperdiçado. Foi identificado que o mesmo era majoritariamente proveniente de uma única chamada da função coletiva `MPI_Allreduce`, dentro do AGCM. Esta função realiza uma operação (soma, multiplicação, mínimo, média, etc) sobre um vetor distribuído em vários processos, e envia o resultado da operação para todos os processos. Então, para que esta função seja concluída, é necessário que todos os processos forneçam seus dados para gerar o resultado final, exigindo que os mesmos estejam sincronizados. Além do *spin time* da `MPI_Allreduce`, também havia até 13% de tempo de CPU (variando de acordo com o número de processos) atribuídas à esta função. Outras operações coletivas do MPI distribuídas pelo modelo também contribuíram para o *spin time*, porém, de forma menos significativa.

3.2.2.3 Teste com barreira explícita

Com o intuito de identificar se o tempo de CPU e *spin time* da `MPI_Allreduce` eram características intrínsecas do cálculo distribuído, ou se havia falta de sincronia entre os processos, foi adicionada uma barreira (`MPI_Barrier`) antes da chamada da `MPI_Allreduce`. A barreira faz com que os processos envolvidos aguardem naquele ponto até que todos o atinjam, somente então é liberada a execução das próximas instruções. Assim, ao iniciar a `MPI_Allreduce`, os processos já estarão sincronizados.

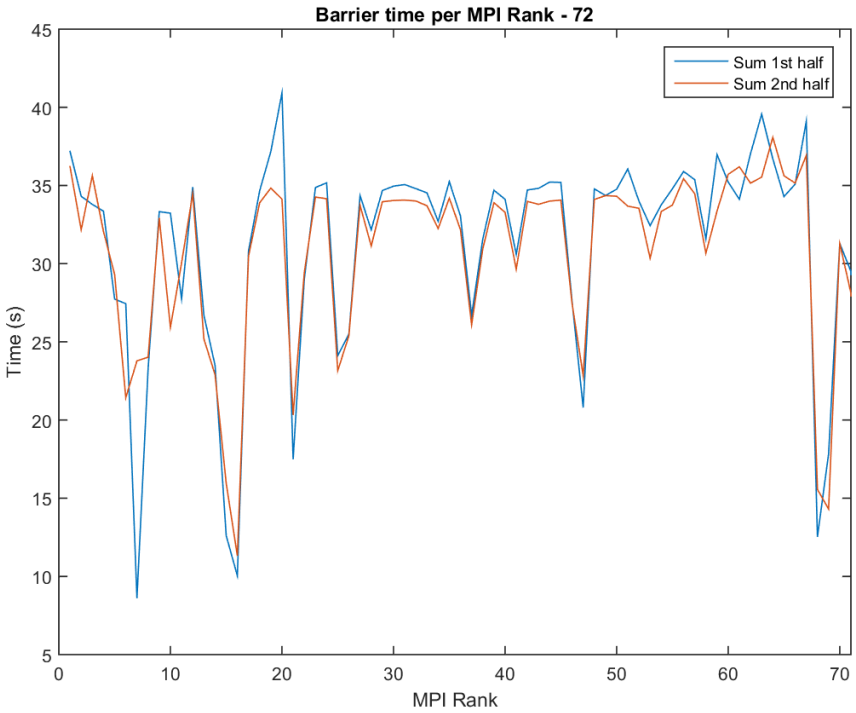
Feita esta modificação, todo o tempo de CPU e *spin time* foi transferido para a barreira, deixando a `MPI_Allreduce` com 0.0% de CPU e *spin time*. Isto indica que o *spin time* desta função estava totalmente relacionado à dessincronização entre os processos MPI.

Sabendo que a barreira não realiza processamento útil, foi investigado em que estava sendo gasto o tempo de CPU considerado como efetivo pelo Intel VTune Amplifier. Este tempo de processamento era proveniente da subfunção `opal_progress`, responsável por consultar *timers* e *triggers* do sistema operacional. Ambos estão relacionados à verificação da sincronia entre os processos MPI. Então, além do *spin time* já identificado automaticamente pelo VTune Amplifier como "CPU desperdiçado", ainda há mais uma quantidade considerável de tempo de CPU sendo utilizada para verificar a sincronia entre os processos enquanto estes aguardam na barreira, ou originalmente na `MPI_Allreduce`.

Com o objetivo de identificar a distribuição destes tempos de espera entre os processos, o código-fonte foi modificado de modo a contabilizar a quantidade de tempo que cada processo aguarda na barreira. Assim, de acordo com a variação entre os tempos de espera, pode-se ter uma noção se há uma tendência de um determinado grupo de processos sempre demorar mais para executar que os demais.

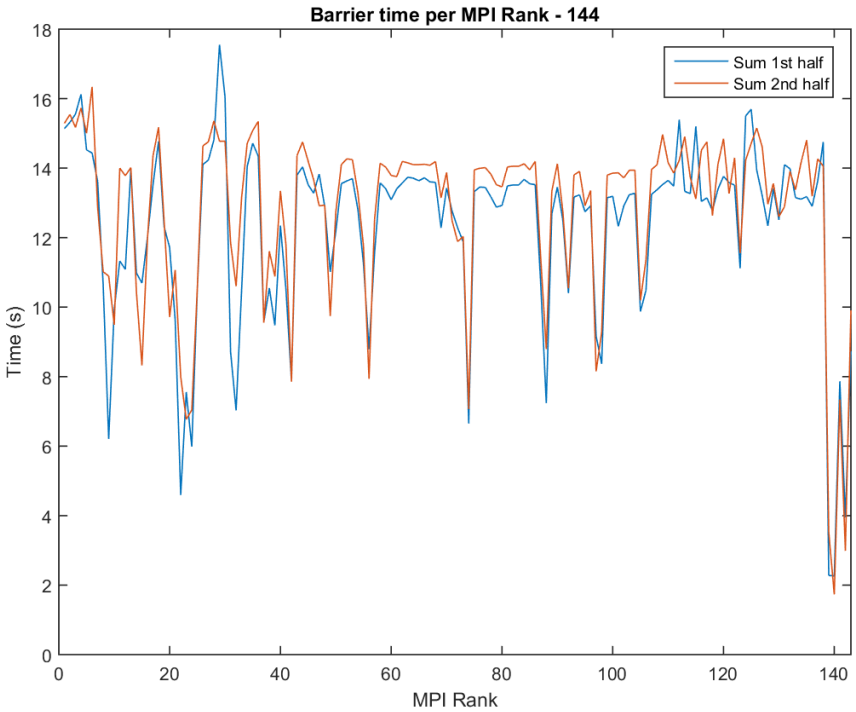
A análise dos tempos de espera na barreira foi dividida entre a primeira e a segunda metade do tempo de execução do modelo. Assim, buscava-se identificar se existia uma tendência de o tempo de espera aumentar ou diminuir durante a execução. Os testes foram executados utilizando 72, 144 e 288 processos MPI, com uma simulação de 5 dias. Os resultados são mostrados nas Figuras 3.6, 3.7 e 3.8, respectivamente.

Figura 3.6 - Tempo de espera na barreira por processo - 72 processos



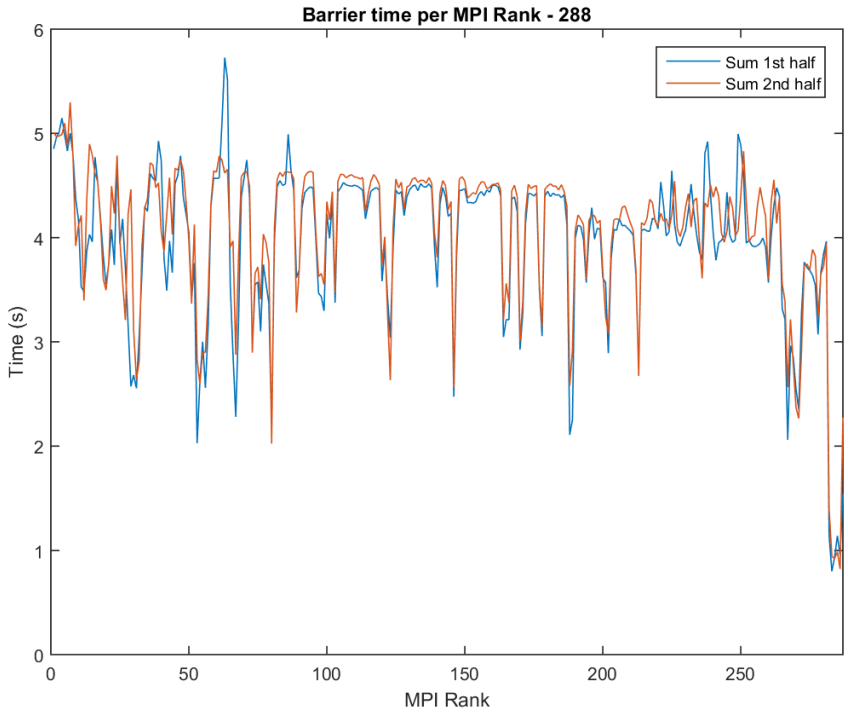
Fonte: Produção do autor.

Figura 3.7 - Tempo de espera na barreira por processo - 144 processos



Fonte: Produção do autor.

Figura 3.8 - Tempo de espera na barreira por processo - 288 processos



Fonte: Produção do autor.

Nas figuras, é visível a semelhança entre as curvas para a primeira e segunda metade da execução, o que indica que não há tendência do tempo de espera de cada processo aumentar ou diminuir da primeira para a segunda metade da execução.

Por outro lado, há uma considerável variação do tempo de espera entre os processos de cada teste. Foi identificada uma tendência de os últimos processos (processos com maior *rank* MPI) esperarem menos que os demais, ou seja, estes são os processos que no geral terminam suas execuções por último, fazendo com que os demais aguardem por mais tempo.

Para verificar se existe tendência desta variação no tempo de espera estar relacionada com o número de processos, foi calculada a razão entre o tempo de espera máximo e mínimo para ambas as metades da execução de cada teste. Conforme mostra a Tabela 3.1, esta relação tende a diminuir conforme o aumento do número de processos, o que indica uma maior uniformidade nos tempos de espera.

A ordem de grandeza dos valores de tempo de espera (eixo vertical nos gráficos) diminuiu conforme o aumento do número de processos, tal como o tempo total de execução do modelo. Assim, com mais processos o modelo executa mais rapidamente, impactando na soma dos tempos de espera de cada processo.

Tabela 3.1 - Relação entre os tempos máximo e mínimo de espera na barreira

Relação de tempo max/min		
Processos	**Metade**	**Relação**
72	1	28.29
	2	19.12
144	1	7.91
	2	9.43
288	1	7.17
	2	6.45

Fonte: Produção do autor.

Tais resultados evidenciam um possível desbalanceamento de carga entre os processos MPI; e este tem se mostrado, até então, o principal fator na limitação de escalabilidade do BESM, quando considerada a utilização somente de MPI.

3.3 Análise MPI + OpenMP

A implementação dos módulos atmosférico e oceânico inclui diretivas OMP. Entretanto, estas implementações são originalmente destinadas à execução isolada destes módulos. Como a execução do modelo acoplado não havia sido testada pelo CPTEC com o OMP ativado, foi verificado se estas diretivas funcionariam adequadamente nesta situação; e caso funcionassem, se trariam ganhos de desempenho.

Ao compilar o modelo com OMP ativado, foi gerado um erro na compilação de um dos arquivos do MOM. Na região de código delimitada pelas diretivas OMP existe uma chamada de função a qual é implementada em um arquivo que não é incluído na compilação do modelo acoplado, apenas do modelo isolado. Esta região só é compilada se o OMP estiver ativado no momento da compilação. Por este motivo não existia este erro ao compilar somente com MPI.

Portanto, foi decidido dividir os testes com OMP em duas fases: ativar somente no AGCM, que compilava corretamente; e depois tentar corrigir o problema do MOM e ativar nos dois módulos.

3.3.1 OMP somente no AGCM

Os testes com OMP ativado somente no AGCM foram realizados utilizando 72 processos MPI, com a quantidade de *threads* por processo sendo 1, 2 ou 4. A quantidade de nós computacionais foi aumentada entre os testes para acomodar a crescente quantidade de *threads*, consequentemente de CPUs necessárias para executar o modelo. Cada nó computacional possui 24 CPUs; então, para acomodar a execução dos 72 processos com uma *thread* cada, foram necessários três nós. Dobrando o número

de *threads*, foi necessário dobrar o número de nós, e assim por diante. As distribuições usadas nos testes, e seus resultados, são sumarizados na Tabela 3.2.

Tabela 3.2 - Resultados dos testes com OMP ativado somente no AGCM

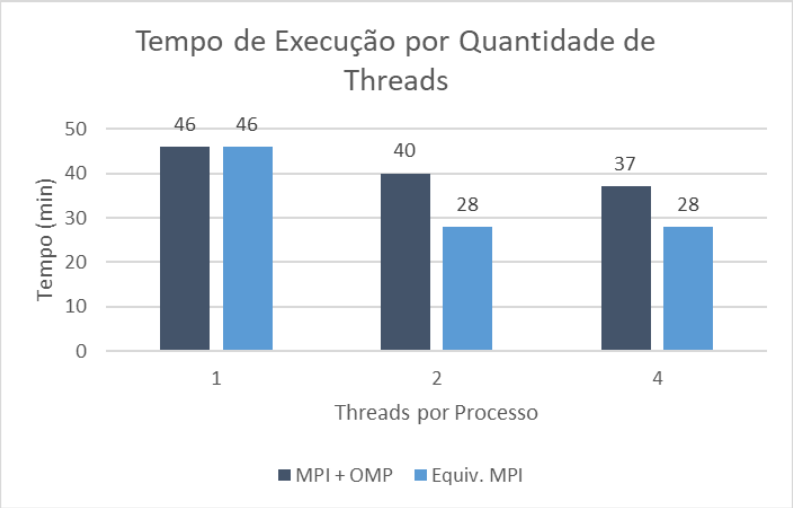
OMP somente no AGCM - Simulação de 1 mês						
Intel 2019 + Intel MPI + OMP						
Processos	**Nós**	**Processos/nó**	**Threads/processo**	**Tempo (min)**	**Equiv. MPI (min)**	**Speedup**
72	3	24	1	46	46	1.00
	6	12	2	40	28	1.15
	12	6	4	37	28	1.24

Fonte: Produção do autor.

Nesta tabela, também foi incluído o tempo levado pela execução do BESM somente com MPI (Equiv. MPI), caso todas as CPUs fossem utilizadas para processos no lugar de *threads*. Desta forma, o equivalente a 72 processos com 2 *threads* cada seria 144 processos; e para 72 processos com 4 *threads* seria 288 processos. Esta comparação foi feita para identificar qual técnica, na situação em que está implementada, é mais vantajosa para uma mesma quantidade de hardware.

Nota-se que apesar de existir um ganho com o aumento da quantidade de *threads*, o *speedup* é baixo. Por exemplo, com duas *threads* houve aumento de desempenho de apenas 15% em relação ao processo com uma *thread*, onde o ideal seria dobrar o desempenho. *Consequentemente, a eficiência da utilização do OMP (da forma em que está implementado) somente no AGCM não é vantajosa, dado que os tempos conseguidos com a mesma quantidade de hardware, porém apenas MPI, são menores.* A Figura 3.9 ilustra os resultados dos testes, bem como esta comparação com a execução puramente MPI.

Figura 3.9 - Resultados dos testes com OMP ativado somente no AGCM



Fonte: Produção do autor.

3.3.2 OMP no AGCM e MOM

Como descrito anteriormente, no código-fonte do MOM havia uma região delimitada por diretivas OMP que chamava uma função a qual o arquivo que a implementava não era incluído na compilação do modelo acoplado. Isto causava um erro de compilação quando o OMP era ativado neste módulo, impedindo a conclusão da mesma.

Então, foi investigado qual era esta função, e qual era sua finalidade. O propósito da mesma era estipular em qual CPU uma *thread* deveria executar (*CPU affinity*), evitando que fosse transferida para outra CPU durante a execução do programa. Este esquema é utilizado a fim de aumentar o desempenho, principalmente pelo aproveitamento dos dados que já estariam na memória cache da CPU. Como esta seria uma função supostamente voltada apenas para melhoria de desempenho, sua chamada foi removida, permitindo assim que a compilação do modelo fosse concluída.

Diversos testes foram executados, com 12 e 24 processos MPI, cada um com número de *threads* variando entre 1 e 12. A maior parte dos testes resultou em *segmentation fault* durante a execução. Os testes que inicialmente executaram até o fim foram repetidos, e nestes casos também houve a ocorrência de *segmentation fault*.

Diante do comportamento aleatório destes erros, há suspeitas de que esteja ocorrendo alguma situação de "race-condition", ou seja, que as diretivas OMP tenham sido inseridas em trechos de código que não são totalmente seguros para uso de tais diretivas no modelo acoplado. Desta forma, é considerado que o BESM, em sua implementação original, não está preparado para utilização de OpenMP no MOM.

4 TÉCNICAS DE MELHORIAS DE DESEMPENHO

O capítulo anterior destaca o *spin time* como o principal responsável pela fraca escalabilidade do BESM. Este comportamento caracteriza um possível desbalanceamento de carga entre os processos do AGCM. O desbalanceamento de carga é típico em modelos atmosféricos, e é frequentemente o principal fator limitante da escalabilidade destes modelos (RODRIGUES et al., 2010). Uma vez que a remoção deste desbalanceamento pode exigir uma reestruturação profunda do modelo, esta abordagem se encontra fora do escopo deste trabalho. Então, são exploradas certas técnicas que podem ao menos mitigar os efeitos do desbalanceamento de carga, e são apresentados os resultados da aplicação destas no BESM.

As técnicas exploradas são: otimizações relacionadas ao processo de compilação do modelo; execução concorrente dos módulos que compõem o BESM; e, por fim, vetorização e blocagem de *loops* relevantes.

4.1 Otimizações relacionadas à compilação

A primeira tarefa conduzida para otimizar o desempenho do BESM foi identificar otimizações em tempo de compilação. Para alcançar melhores otimizações e integração com ferramentas de análise de desempenho, foram criados novos scripts de compilação para usar a suíte Intel PSXE 2019, disponível no Santos Dumont. Esta suíte inclui compiladores Fortran e C/C++, uma biblioteca MPI que implementa a especificação MPI-3.1, e ferramentas para análise de desempenho.

Além da substituição da biblioteca MPI, foram adicionadas *flags* de otimização nos comandos de compilação buscando obter um programa com melhor eficiência computacional. Entre diversas *flags* testadas, as que trouxeram resultados significativos foram:

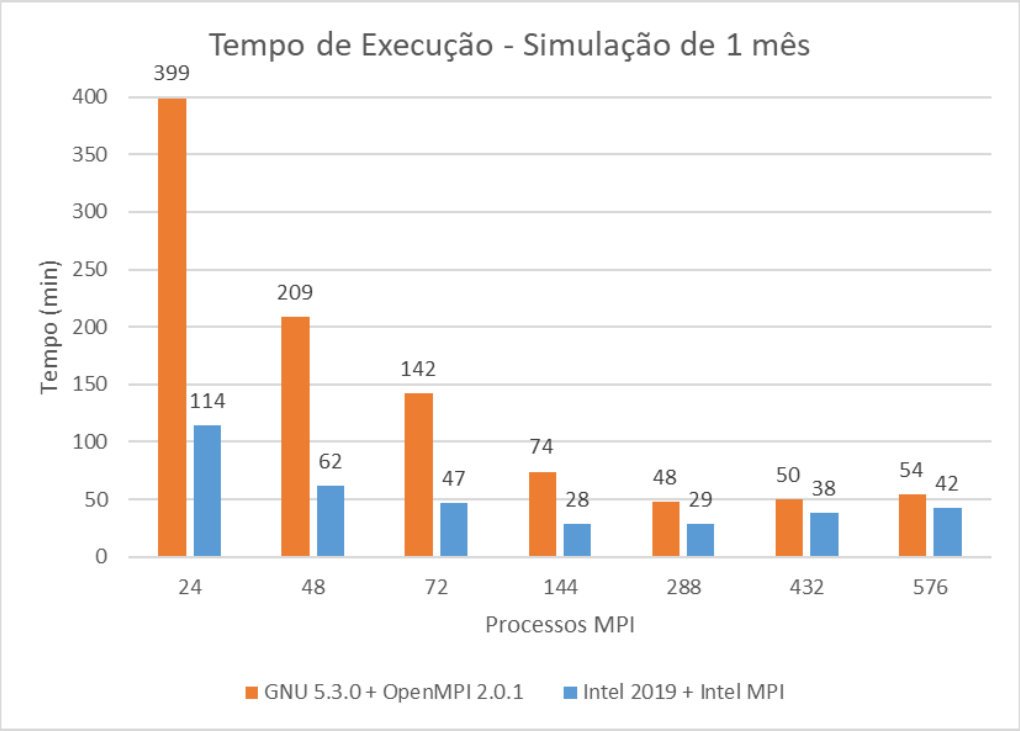
- `-O3` → habilita otimizações mais agressivas, como *prefetching*, transformações de *loops* e de acesso à memória, replicação de código, e modificações em estruturas de dados para utilização mais eficiente da memória cache;
- `-no-prec-div` → permite ao compilador mudar a divisão em ponto-flutuante para a multiplicação pelo inverso do denominador (A/B = A*(1/B)), de modo a melhorar a velocidade do cálculo;
- `-fp-model fast=2` → habilita otimizações mais agressivas em cálculos com ponto-flutuante, podendo impactar negativamente na precisão dos resultados;

- `-xHost` → habilita a utilização de instruções de vetorização otimizadas para o processador no qual o código está sendo compilado.

É interessante apontar que usando estas *flags*, o tempo de compilação do modelo saltou de aproximadamente 5 minutos para mais de 2 horas. De forma geral, a troca de compilador aliada ao uso das *flags* de otimização na compilação resultou em tempos de execução até 3,5x mais rápidos, de acordo com o número de CPUs utilizados para executar o modelo.

Nos testes realizados, houve ganho de desempenho na execução com até 144 processos MPI. Quando o valor foi dobrado para 288 processos, o tempo total de execução aumentou levemente, indicando que ali estaria o limite de paralelização do modelo da forma em que estava implementado. A Figura 4.1 representa graficamente os tempos de execução obtidos nos testes, comparados aos tempos originais (compilador GCC e apenas *flag* `-O2`). O melhor tempo de execução mudou de 48 minutos com GCC para 28 minutos após nossas modificações. Entretanto, o ponto de máximo *speedup* foi movido de 288 processos MPI para 144 processos. No geral, após a troca do compilador e adição de *flags* de otimização, foram obtidos melhores tempos de execução em todos os testes.

Figura 4.1 - Tempos de execução após otimizações no processo de compilação

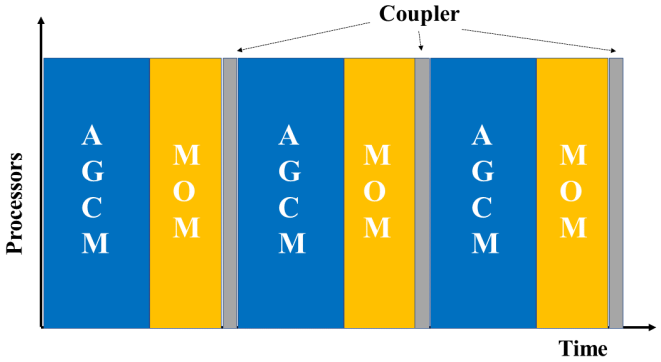


Fonte: Produção do autor.

4.2 Execução concorrente dos módulos

Um dos principais objetivos era explorar as implicações de desempenho ao executar os módulos AGCM e MOM de forma concorrente, em vez de um após o outro (modo alternado) como atualmente feito no CPTEC. A Figura 4.2 mostra um esquemático de funcionamento do modo de execução alternado, onde o AGCM executa seus *timesteps* internos em todos os processos MPI. Então, o MOM executa seus *timesteps* internos também usando todos os processos MPI. Após ambos os módulos executarem um conjunto de *timesteps*, o *timestep* de acoplamento é alcançado, quando os módulos trocam informações.

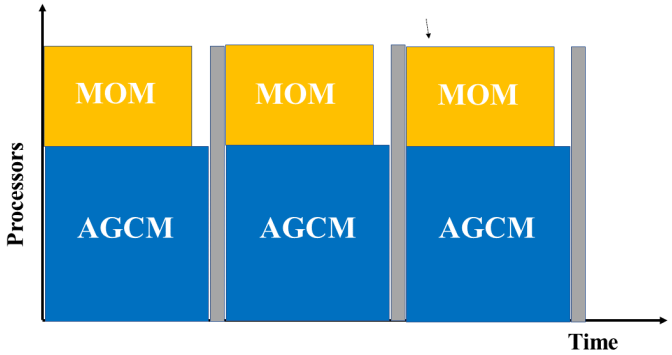
Figura 4.2 - Esquemático do modo de execução alternado



Fonte: Produção do autor.

No modo concorrente (Figura 4.3), uma parte dos processos MPI é dedicada a executar cada módulo. Desta forma, o AGCM é executado em um conjunto de processos, e ao mesmo tempo, o MOM é executado em outro conjunto. Após ambos os módulos finalizarem sua execução, o *timestep* de acoplamento é alcançado. Uma desvantagem deste modo de execução é que o módulo mais rápido terá que aguardar pelo *timestep* de acoplamento até que o módulo mais lento finalize sua execução. Isto pode resultar em desperdício de recursos computacionais, mas pode ser mitigado ao distribuir adequadamente o número de processos MPI dedicados a cada módulo.

Figura 4.3 - Esquemático do modo de execução concorrente



Fonte: Produção do autor.

O acoplador FMS é projetado para dar suporte à execução concorrente dos módulos. O MOM, que veio já integrado ao FMS, também é preparado para executar de forma concorrente. Por outro lado, as investigações que serão apresentadas a seguir apontaram que o AGCM não foi originalmente implementado visando ser parte de um modelo acoplado, e menos ainda a executar de forma concorrente com outros módulos. Portanto, foi necessário modificar o código do AGCM para fazê-lo funcionar com o método de concorrência disponível no FMS.

4.2.1 Modificações para execução concorrente

Enquanto o MOM usa uma interface de paralelização chamada MPP (disponibilizada pelo FMS), o AGCM faz chamadas diretas ao MPI. Portanto, além de algumas mudanças pequenas, a principal modificação realizada foi a criação de um subcomunicador MPI dedicado ao AGCM, para substituir o comunicador global originalmente utilizado. Esta abordagem exigiu menos esforço do que modificar todo o AGCM para usar o MPP.

O código do AGCM utilizava o comunicador `MPI_COMM_WORLD` em suas chamadas MPI. Isso significa que suas funções coletivas esperavam a participação de todos os processos MPI do programa. Quando executado em "modo alternado", isto não era um problema, visto que na vez do AGCM executar ele tinha a participação de todos os processos. Mas quando executando concorrentemente, os processos do MOM não alcançam as funções do AGCM; então, as chamadas MPI coletivas do AGCM ficavam bloqueadas aguardando por todos os processos alcançarem aquela chamada MPI, o que nunca acontecia.

Este problema foi resolvido criando um subcomunicador do `MPI_COMM_WORLD`, chamado `COMM_ATMOS`. Este subcomunicador contem apenas os processos MPI pertencentes ao AGCM. Também foram feitas as mudanças de código necessárias para o novo comunicador funcionar corretamente. Uma vez que o subcomunicador é criado na região de código pertencente ao FMS (onde todos os processos estão disponíveis), foi necessário modificar uma série de funções para passar sua referência até o ponto onde ele seria usado no AGCM. O código original do AGCM também cria um subcomunicador para cálculo de Fourier, chamado `COMM_FOUR`. Portanto, este também foi modificado para ser um subcomunicador do novo `COMM_ATMOS` em vez do comunicador global `MPI_COMM_WORLD`.

Após resolver este problema do comunicador MPI e configurar os arquivos de entrada do modelo com o número apropriado de processos a ser utilizado por cada módulo na execução concorrente, foi possível executar o modelo em modo concorrente. Então, foi checado se os arquivos de saída continham resultados equivalentes aos produzidos pelo modelo original.

A validação foi realizada utilizando a aplicação OpenGrADS para comparar as diferentes variáveis contidas nos arquivos de saída. Os resultados mostraram diferenças de até 5%, similar à variação encontrada quando o número total de processos é alterado no modelo original. Com estas mudanças validadas, foi prosseguido para a etapa de análises de desempenho do modelo sendo executado em modo concorrente; começando por uma divisão uniforme dos CPUs entre os módulos, e depois variando a proporção de CPUs para cada módulo.

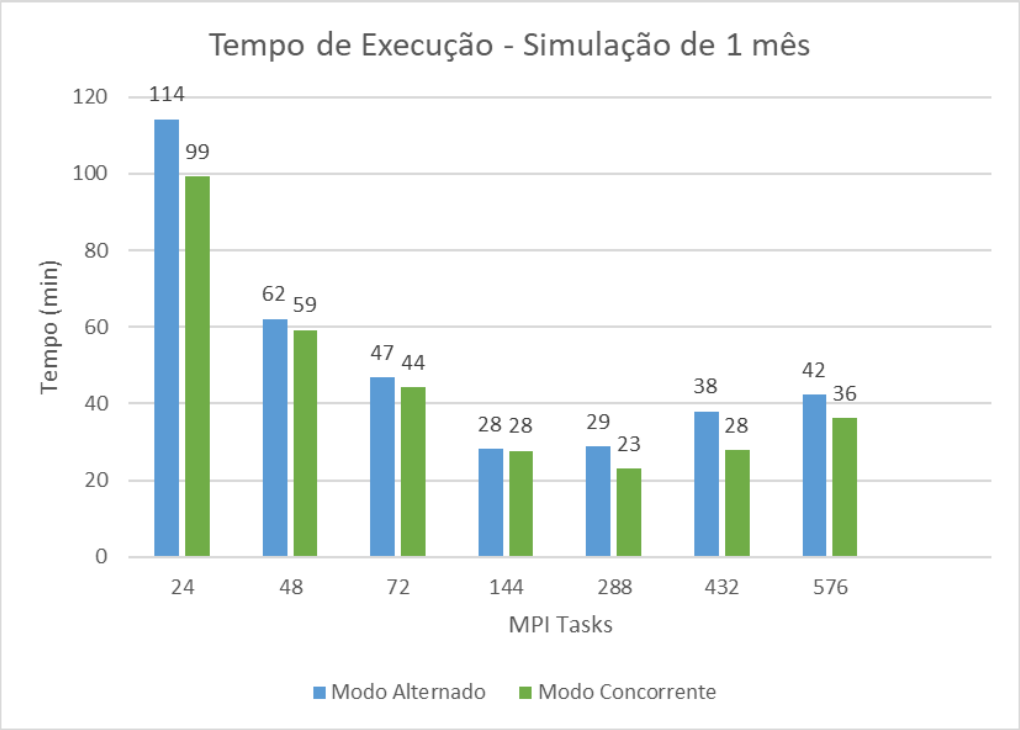
4.2.2 Divisão uniforme dos CPUs

Os primeiros testes executados foram simulações de 1 mês de clima, usando metade dos CPUs para cada módulo (AGCM e MOM), com um processo MPI sendo executado em cada CPU. Em uma situação ideal, espera-se mover o limite de *speedup* para o dobro de CPUs. Isto é esperado porque, hipoteticamente, com cada módulo tendo sua saturação em um número P de CPUs, utilizando $2P$ CPUs (P para cada módulo), ambos os módulos ainda estariam em seus pontos de saturação.

Uma segunda expectativa ideal seria que uma execução concorrente com $2P$ processos (P para cada módulo) levaria metade do tempo de uma execução alternada com P processos. Este é apenas um cenário ideal, uma vez que é sabido (da seção sobre *hotspots*) que os módulos não utilizam a mesma quantidade de recursos computacionais.

Como previsto, o modo concorrente com metade dos CPUs para cada módulo não trouxe os resultados ideais. A Figura 4.4 mostra os tempos de execução concorrente e alternado para diferentes números de processos MPI. Embora o limite de *speedup* tenha sido movido de 144 para 288 processos MPI, e melhores tempos de execução tenham sido obtidos em modo concorrentemente, o desempenho não melhorou de forma significativa. Houve uma redução máxima no tempo de execução de 26% para a rodada com 432 processos; por outro lado, o pior resultado ocorreu com 144 processos, onde o tempo de execução se manteve igual.

Figura 4.4 - Comparação do tempo de execução: modo alternado vs. concorrente



Fonte: Produção do autor.

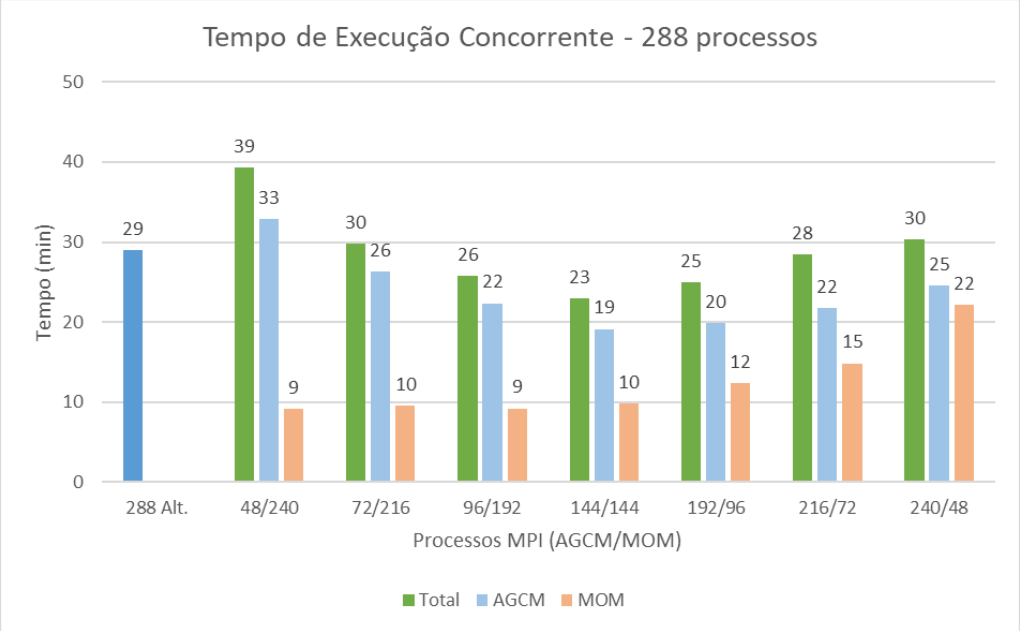
4.2.3 Balanceamento da quantidade de CPUs

Dado que os módulos não consomem a mesma quantidade de recursos computacionais, nossa próxima abordagem foi testar diferentes proporções de distribuição de CPUs para estudar seu efeito no desempenho do modelo. Isto permitiria identificar uma tendência de uma dada proporção de CPUs para o AGCM/MOM desempenhar melhor no BESM. É sabido que o AGCM utiliza mais tempo de CPU do que o MOM nas execuções em "modo alternado". Portanto, a expectativa inicial era que atribuindo mais CPUs ao AGCM o faria executar mais rapidamente; enquanto o

MOM, executando concorrentemente com menos CPUs, poderia finalizar seus *timesteps* aproximadamente ao mesmo tempo que o AGCM. Desta forma, ambos os módulos poderiam alcançar o *timestep* de acoplamento quase simultaneamente.

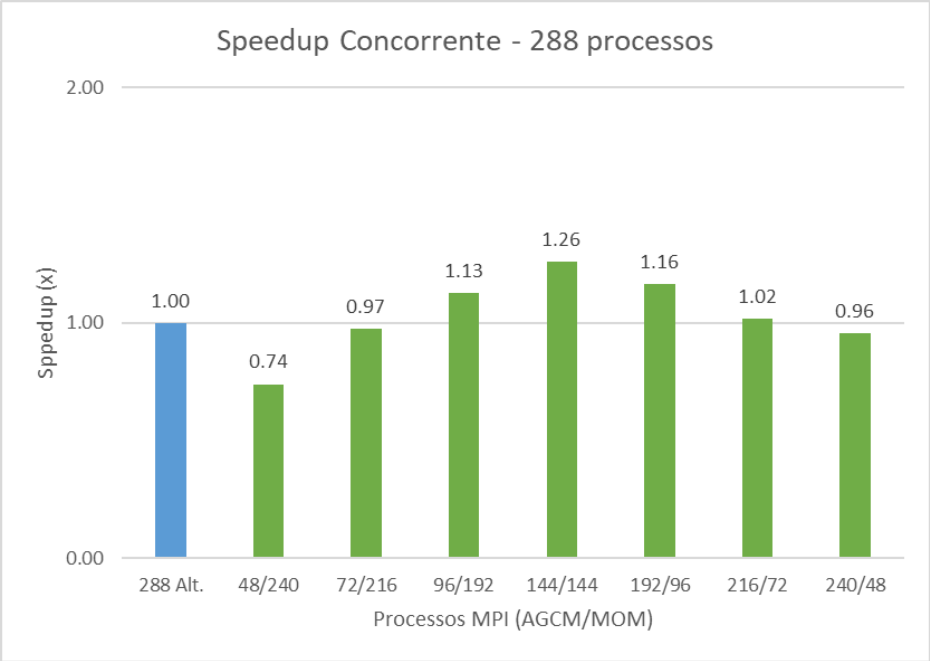
Os testes foram iniciados pelo modo de execução mais rápido obtido até então: a execução com 288 processos MPI. Foram verificados os tempos de execução de uma série de proporções de processos para o AGCM/MOM, obedecendo o mesmo total de 288 processos. A Figura 4.5 mostra os tempos de execução para as distribuições testadas. A melhor proporção de distribuição de processos obtida (em termos de tempo de execução) foi a 144/144. Aumentando ou diminuindo a parte do AGCM dos processos, o tempo de execução aumentou. A primeira barra na Figura 4.5 mostra, para comparação, o tempo de execução do modo alternado equivalente, com o mesmo total de 288 processos MPI. Na Figura 4.6, que mostra os *speedups* correspondentes, é mais fácil visualizar o pico na distribuição 144/144. Para as distribuições mais desiguais, (ex.: 48/240 e 240/48), o desempenho foi pior quando comparado ao modo alternado.

Figura 4.5 - Tempo de execução de diferentes distribuições de 288 processos MPI



Fonte: Produção do autor.

Figura 4.6 - *Speedup* de diferentes distribuições de 288 processos MPI

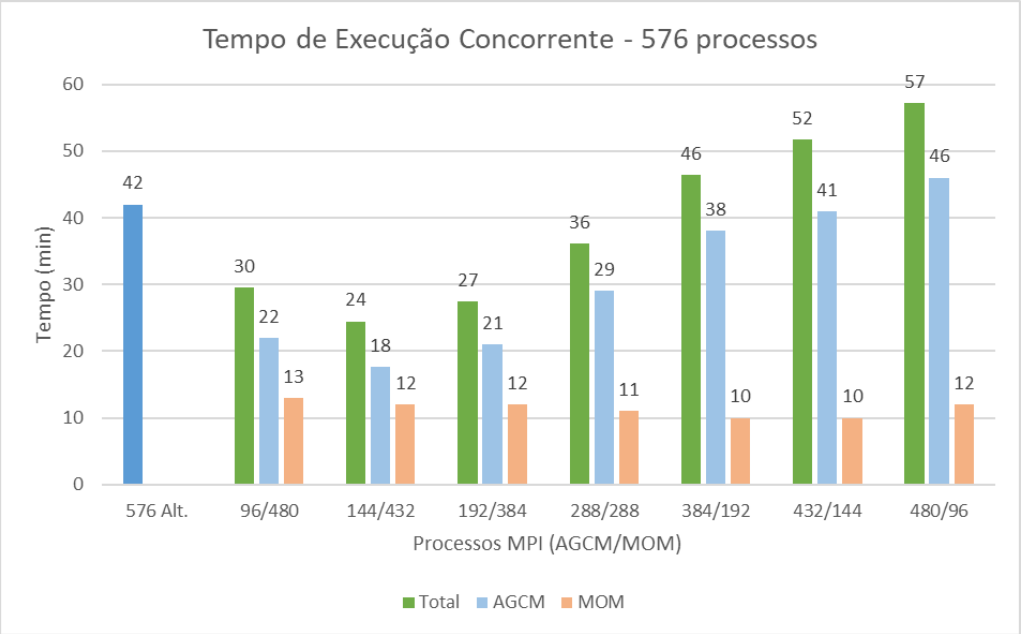


Fonte: Produção do autor.

Os próximos testes buscaram verificar se o modelo poderia efetivamente escalar para mais de 288 CPUs, escolhendo o número máximo de CPUs utilizados até então (576) e variando as proporções de distribuição. A Figura 4.7 mostra o tempo de execução de diferentes distribuições dos 576 processos MPI; a primeira barra mostra o tempo gasto pela execução em modo alternado, para comparação.

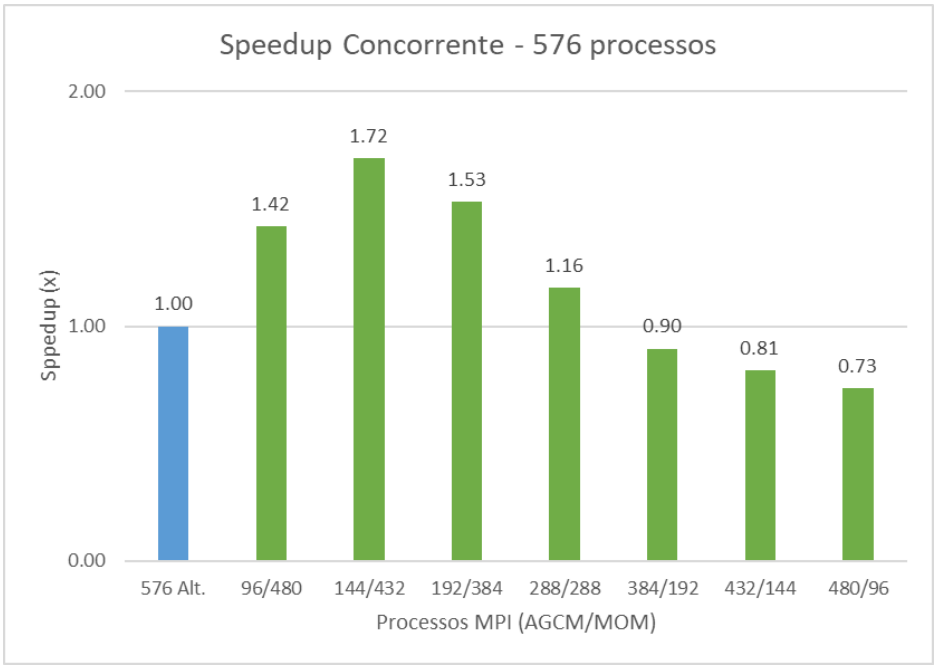
A Figura 4.8 ilustra o *speedup* sobre a execução em modo alternado. Oposto ao que a intuição levaria a esperar, designar mais CPUs para o MOM resultou em melhor desempenho. O que não havia sido levado em consideração era que o *spin time* do AGCM aumenta proporcionalmente ao número de processos MPI em uso. Por outro lado, o MOM mostra melhor escalabilidade com a adição de mais CPUs.

Figura 4.7 - Tempo de execução de diferentes distribuições de 576 processos MPI



Fonte: Produção do autor.

Figura 4.8 - *Speedup* de diferentes distribuições de 576 processos MPI



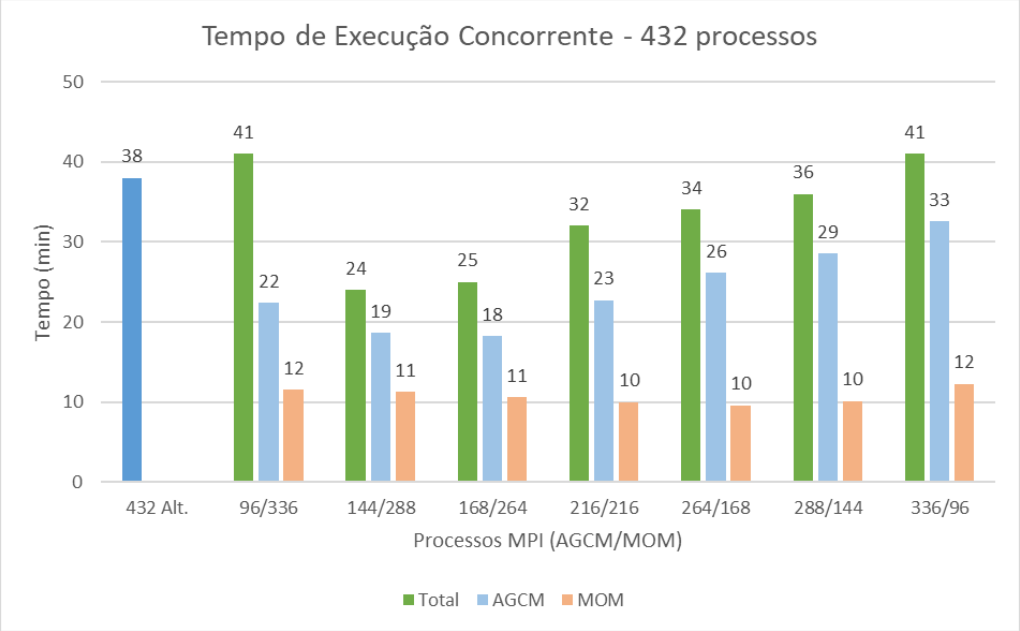
Fonte: Produção do autor.

Houve perda de desempenho em todas as execuções onde a maior parte dos CPUs foi designada ao AGCM (as três barras mais à direita nas Figuras 4.7 e 4.8); enquanto a execução com 75% dos CPUs alocados para o MOM (144/432 CPUs para o AGCM/MOM, respectivamente) resultou em um *speedup* de 1,72. Mesmo com uma distribuição balanceada dos 576 processos, não foi possível superar o tempo de execução dos 288 processos com distribuição 144/144. Os resultados foram: 24 minutos para a distribuição 144/432, e 23 minutos para 144/144.

Dado que nos testes com 288 e 576 CPUs a distribuição com melhor desempenho foi a que empregou 144 CPUs para o AGCM, foi investigado qual seria o comportamento do modelo em uma situação intermediária, com 432 CPUs. Novamente, dedicar 144 CPUs ao AGCM resultou no melhor desempenho, sendo a distribuição 144/288 a com menor tempo de execução, 24 minutos (Figura 4.9).

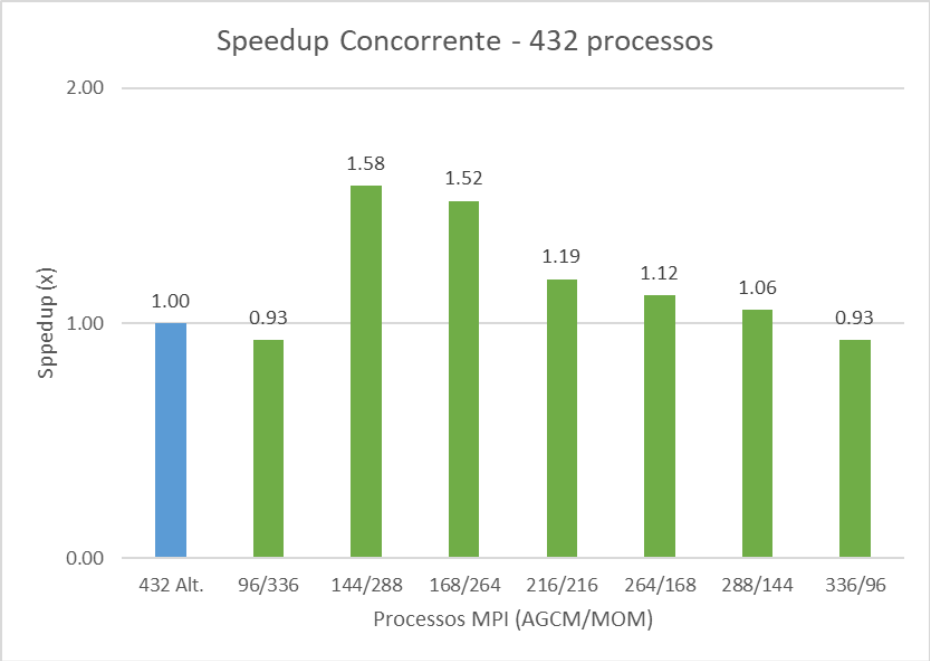
Assim como nos testes com 288 CPUs, as alocações mais desiguais (96/336 e 336/96) trouxeram tempos de execução maiores que a execução alternada, resultando em um *speedup* inferior a um nestes casos (Figura 4.10). A partir de 144 CPUs alocados ao AGCM, o desempenho do modelo é inversamente proporcional à quantidade de CPUs alocados ao modelo atmosférico.

Figura 4.9 - Tempo de execução de diferentes distribuições de 432 processos MPI



Fonte: Produção do autor.

Figura 4.10 - *Speedup* de diferentes distribuições de 432 processos MPI



Fonte: Produção do autor.

As distribuições com melhores resultados para cada quantidade de CPUs testada (288, 432 e 576) é resumida na Tabela 4.1. Em todos os testes, atribuir 144 CPUs ao AGCM trouxe o menor tempo de execução. Ao adicionar mais CPUs para o modelo atmosférico o desempenho diminui. Isso indica que o AGCM está, em sua implementação atual, saturando seu desempenho aos 144 CPUs.

Tabela 4.1 - Melhores distribuições para cada quantidade de CPUs

# CPUs			**Tempo**	**Tempo**	**Speedup**
Total	**AGCM**	**MOM**	**(balanceado)**	**(alternado)**	**(vs. alternado)**
288	144	144	23 min	29 min	1.26
432	144	288	24 min	38 min	1.58
576	144	432	24 min	42 min	1.72

Fonte: Produção do autor.

De forma geral, o BESM demonstrou melhor desempenho ao ser executado com um total de 288 CPUs, sendo 144 para cada módulo (AGCM e MOM). Cada CPU era dedicado a executar um único processo MPI do modelo.

Mesmo que o MOM tenha melhor escalabilidade e se beneficie da execução com mais de 144 CPUs, isto não traria ganhos no tempo de execução do BESM, uma vez que, ao terminar seus *timesteps*, o MOM precisaria aguardar o AGCM para executar o *timestep* de acoplamento. Ou seja, o MOM utilizaria mais recursos computacionais, executaria mais rapidamente, e então seus processos ficariam ociosos aguardando a conclusão da execução do AGCM, que como identificado, satura seu desempenho com aproximadamente 144 CPUs.

Além de o MOM ter que esperar pela execução do AGCM, a adição de mais CPUs aumenta o *overhead* de comunicação entre os processos MPI, uma vez que a quantidade de mensagens trocadas e o número de processos a serem sincronizados são maiores. Desta forma, executar o BESM com mais de 144 CPUs para o MOM acaba aumentando o tempo de execução, como pode ser visto na Tabela 4.1.

A tabela também mostra o *speedup* de cada distribuição em relação ao modo alternado com mesmo total de CPUs. Nota-se que quanto maior o número total de CPUs, maior o *speedup* alcançado. Isso se explica pelo fato de, na execução alternada, o AGCM ser executado com o número total de CPUs, que é maior do que o ponto de saturação identificado (144), fazendo com que seu tempo de execução aumente, impactando negativamente no tempo total do modelo. Desta forma, o *speedup* é maior porque o tempo de referência é pior, e não porque maiores melhorias foram alcançadas.

Na execução alternada, o ganho de desempenho do MOM com 576 CPUs, por exemplo, não é suficiente para compensar o atraso do AGCM com as mesmas 576 CPUs. Desta forma, a execução concorrente dos módulos com 144 CPUs para o AGCM e 432 para o MOM resulta em um *speedup* de 1,72 em relação à execução alternada.

4.3 Vetorização e blocagem

Após verificar o balanceamento da quantidade de CPUs para cada módulo na execução concorrente, foi investigado se o modelo se beneficiaria da adição manual de diretivas de vetorização, e também da modificação de *loops* para aproveitamento da técnica de blocagem. Para tal, o modelo foi compilado com as *flags* `-qopt-report=2 -qopt-report-phase=vec`, que habilitam a emissão de *logs* relacionados à vetorização automática dos *loops*. O objetivo era analisar tais *logs* e identificar *loops* relevantes (que consomem uma quantidade significativa de tempo de CPU) que não puderam ser vetorizados automaticamente pelo compilador.

Os *logs* de compilação foram analisados, considerando todos os *loops* do modelo contidos em funções que possuíssem mais de 1% do tempo total de CPU. *Loops* com tempos menores de CPU foram desconsiderados pois a possível otimização dos mesmos não traria ganhos relevantes no modelo como um todo.

De acordo com os resultados encontrados, o compilador Intel realizou um excelente trabalho de vetorização automática. Das 14 funções analisadas, apenas três não tiveram os *loops* vetorizados, uma função teve parte dos *loops* vetorizados, e as 10 funções restantes tiveram todos os seus *loops* vetorizados automaticamente pelo compilador.

Então, foi analisada a estrutura dos *loops* que não foram vetorizados e foi verificado que, de fato, a vetorização não era possível devido à dependência de iterações anteriores, dependência de saída, *loop* com quantidade indefinida de iterações e *loop* com múltiplos pontos de saída. A Tabela 4.2 mostra as funções analisadas com seus respectivos consumos de CPU, e o resultado da vetorização dos *loops* contidos em cada função.

Tabela 4.2 - *Loops* analisados quanto à vetorização

CPU	Função	Status dos Loops
3.6%	PhysicalFunctions.f90:scalelength2	VECTORIZED
3.6%	PhysicalFunctions.f90:qsih2	VECTORIZED
1.8%	PhysicalFunctions.f90:qsim2	VECTORIZED
1.8%	PhysicalFunctions.f90:t2mt	NOT VECTORIZED > OUTPUT dependence
1.8%	PhysicalFunctions.f90:q2mt	NOT VECTORIZED > OUTPUT dependence
1.8%	Micro_Ferrier.f90:gsmcolumn	NOT VECTORIZED > loop with multiple exits, do while
1.8%	Micro_Ferrier.f90:gsmdrive	VECTORIZED
1.7%	ocean_bihgen_friction.F90:bihgen_friction	VECTORIZED
1.7%	ocean_bih_friction.F90:bih_friction	VECTORIZED
1.5%	Surface.f90:calc2mseaice	VECTORIZED
1.5%	ocean_velocity.F90:ocean_explicit_accel_a	VECTORIZED
1.1%	diag_manager.F90:send_data_3d	VECTORIZED, FLOW dependence
1.1%	ocean_nphysicsB.F90:fz_terms	VECTORIZED
1.1%	ocean_util.F90:diagnose_3d_mask	VECTORIZED

Fonte: Produção do autor.

Seguindo o mesmo critério de seleção, *loops* em funções com mais de 1% de CPU, foi verificado se estes se beneficiariam da técnica de blocagem. Os *loops* que haviam sido vetorizados pelo compilador foram considerados já otimizados o suficiente. Portanto, os *loops* restantes foram analisados em termos da viabilidade de aplicação de blocagem.

Os *loops* das funções `t2mt` e `q2mt` possuíam apenas um nível de iteração (uma dimensão). Logo, a técnica de blocagem não é aplicável neste caso.

Os *loops* da função `gsmcolumn`, apesar de serem aninhados (o que permitiria a blocagem), não apresentaram características que sugerissem ganho de desempenho com a aplicação da técnica. Se tratavam na maior parte de acessos sequenciais aos dados, e diversas operações em variáveis escalares locais (independentes dos índices dos *loops*).

Na função `send_data_3d`, boa parte dos *loops* foram vetorizados. Alguns já possuíam a blocagem aplicada. Nos demais, não foi identificada a viabilidade de aplicação de blocagem. Como esta função pertence ao código do MOM, que possui código aberto em um repositório público (GitHub), é compreensível que o código tenha sido melhor otimizado por possibilitar mais pessoas contribuírem em seu desenvolvimento.

Portanto, de forma geral não foram identificadas possibilidades vantajosas de aplicação da técnica de blocagem no BESM. Certamente, em um modelo complexo como este, existem oportunidades da aplicação da técnica, mas as mesmas não foram encontradas entre os *loops* de maior consumo computacional.

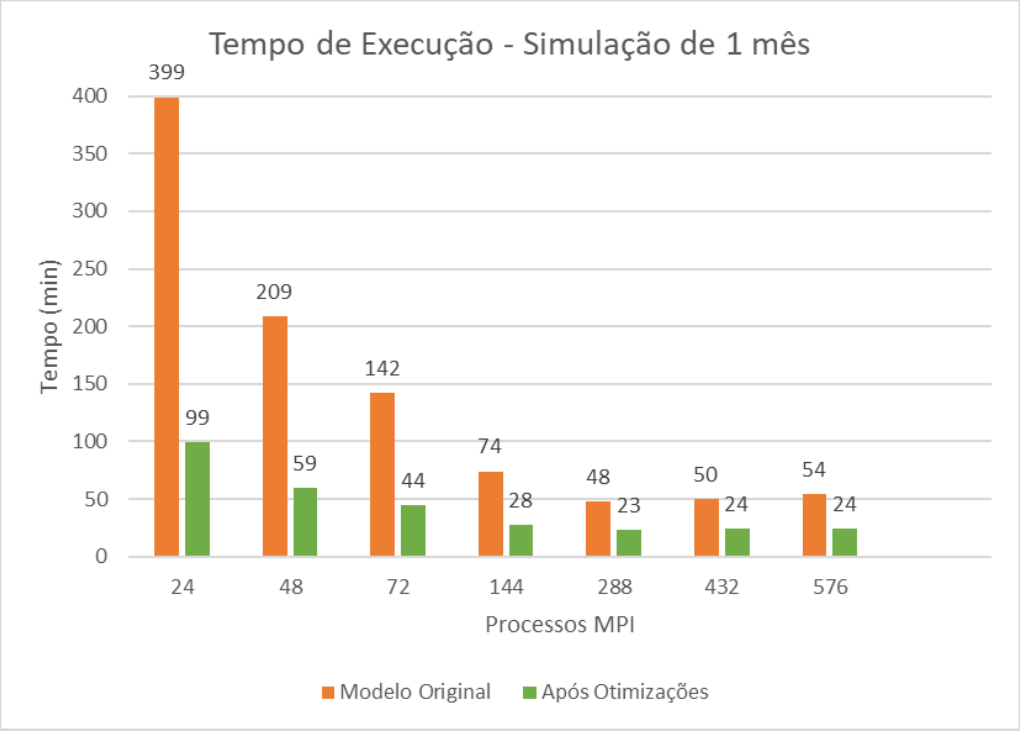
4.4 Resultados gerais

As diversas execuções do BESM realizadas durante todo este trabalho tiveram tempos de execução bastante consistentes, sem variação considerável para uma mesma versão compilada. Um ponto bastante importante no entendimento deste fato é que os programas submetidos na fila de execução do Santos Dumont são executados em nós alocados exclusivamente para tal, sem interferência de outros programas ou processos. Portanto, devido a esta consistência observada no tempo de execução, os resultados finais apresentados não tiveram tratamento estatístico, sendo os tempos aferidos em apenas uma execução para cada configuração de execução.

Apesar de o desbalanceamento de carga do modelo não ter sido explorado, o desempenho do BESM foi melhorado a uma margem promissora. Comparado com a maneira em que o modelo estava sendo executado em produção, foi alcançada uma melhora no tempo de execução de 403% quando utilizando 24 CPUs (de 399 para 99 minutos para uma simulação de um mês); e o tempo mínimo de simulação foi reduzido de 48 para 23 minutos, uma melhoria de 209%, ao compilar e executar o BESM de maneira mais apropriada. A Figura 4.11 mostra os tempo de execução do modelo original comparados com os que foram obtidos após as modificações re-

alizadas neste trabalho. Para as execuções com 288, 432 e 576 processos MPI, foi considerado o melhor resultado após o balanceamento de CPUs, explicado na Seção 4.2.3.

Figura 4.11 - Comparação do tempo de execução após otimizações realizadas



Fonte: Produção do autor.

5 CONCLUSÕES

Este trabalho apresentou os esforços para otimizar a eficiência computacional do BESM, um modelo climático global desenvolvido pelo CPTEC/INPE. O BESM tem mostrado sua relevância fornecendo material para pesquisas acadêmicas e relatórios do IPCC, que trata das preocupações atuais relacionadas ao aquecimento global. Como mostrado, a implementação original do modelo tem espaço para otimizações computacionais de modo a fornecer resultados em um tempo de execução significativamente menor, o que favoreceria simulações cobrindo décadas e séculos.

O trabalho teve início com uma análise da situação inicial do BESM, incluindo: como o modelo estava sendo compilado, quais paradigmas de paralelização estavam sendo usados, como seus módulos eram executados, e quais eram os pontos com maior consumo de CPU (*hotspots*). Em seguida, os esforços foram aplicados para melhorar o desempenho do modelo, baseado no que foi encontrado na análise inicial.

Foram obtidos bons resultados trocando o compilador, ajustando opções de compilação, e realizando as alterações no código necessárias para permitir a execução dos módulos atmosférico e oceânico de forma concorrente com diferentes quantidades de CPUs alocados para cada módulo. Comparando com a maneira em que o BESM estava sendo usado em produção, o tempo de execução do modelo foi melhorado em 403% quando executado com poucas CPUs (24 processos MPI); e o tempo mínimo de execução foi reduzido de 48 para 23 minutos para uma simulação de um mês, uma melhoria de 209%, quando executando com 288 CPUs.

Após cada modificação relevante no código do modelo ou no modo de execução, os resultados de uma simulação de um mês foram comparados com a saída produzida pelo modelo original, de forma a validar as alterações realizadas.

Nos testes do modo concorrente, inicialmente era esperado que designar uma maior proporção das CPUs para o modelo atmosférico (AGCM) traria melhores tempos de execução. Isto porque, em testes preliminares, o AGCM consumia mais tempo de CPU que o modelo oceânico (MOM) quando executado em modo alternado. Portanto, em uma execução concorrente, seria lógico balancear o tempo de execução dos módulos designando mais recursos computacionais ao AGCM. Entretanto, nos testes foram obtidos melhores tempos de execução quando o MOM recebeu a maior parte das CPUs. Isto ocorreu devido à pobre escalabilidade do AGCM, que sofre de um crescente *spin time* quando mais processos MPI são adicionados. Desta forma, maior tempo de CPU é desperdiçado quando o AGCM possui mais CPUs que o

MOM. De acordo com os testes, o AGCM não consegue ganhos de desempenho com mais de 144 CPUs em execuções com a resolução espacial utilizada neste trabalho.

Também foi verificada a possibilidade de otimizações voltadas à arquitetura de hardware utilizada: a aplicação das técnicas de vetorização e blocagem. Neste quesito o BESM se mostrou um modelo bem acurado, sem muito espaço para otimizações que surtissem efeitos notáveis. Foram estudados os *loops* com maior consumo de CPU, tanto o código-fonte quanto os *logs* de vetorização automática do compilador. A maior parte dos *loops* investigados foi vetorizada automaticamente pelo compilador. Os demais possuíam características que realmente impossibilitavam a vetorização dos mesmos. Quanto à técnica de blocagem, a maior parte dos *loops* analisados não se beneficiaria da utilização desta devido às suas características; enquanto outros *loops* já estavam originalmente implementados utilizando blocagem.

Embora o desempenho do BESM tenha sido melhorado em uma margem considerável, ainda há espaço para melhorias adicionais. A investigação da causa do *spin time* observado revelou que o desbalanceamento de carga é o principal fator levando à pobre escalabilidade do AGCM. Devido a este gargalo, não foram obtidos benefícios de desempenho usando mais de 144 CPUs no AGCM. Mitigar este efeito é um desafio que ainda permanece aberto. Entretanto, abordar o problema de desbalanceamento de carga provavelmente requererá a reestruturação do código do modelo atmosférico, o que está além do escopo deste trabalho.

As análises de desempenho e aplicação de técnicas de otimização realizados neste trabalho foram voltadas ao modelo climático brasileiro, o BESM. Entretanto, o estudo aqui apresentado pode ser aplicado em modelos de tempo e clima similares, que são baseados no acoplamento de módulos e destinados à execução paralela em *clusters*.

REFERÊNCIAS BIBLIOGRÁFICAS

BARNEY, B. **OpenMP**. Lawrence Livermore National Laboratory, 2020. Disponível em: <https://computing.llnl.gov/tutorials/openMP/>. 11

CALLEGARE, A. O.; GIAROLLA, E. Impactos da utilização de uma grade de alta resolução horizontal na componente oceanica do modelo brasileiro do sistema terrestre (BESM). In: SEMINÁRIO DE INICIAÇÃO CIENTÍFICA DO INPE (SICINPE), 2016, São José dos Campos, SP. **Anais...** São José dos Campos: INPE, 2016. Disponível em: <http://urlib.net/rep/8JMKD3MGP3W34P/3N5DQQ5>. Acesso em: 11 jan. 2020. 9

CASAGRANDE, F. **Sea ice study and Arctic polar amplification using BESM model**. 89 p. Tese (Doutorado em Ciência do Sistema Terrestre) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2017. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m21b/2016/05.12.04.17>. Acesso em: 29 abr. 2018. 9

CASAGRANDE, F.; NOBRE, P.; SOUZA, R. B.; MARQUEZ, A. L.; TOURIGNY, E.; CAPISTRANO, V.; MELLO, R. L. Arctic sea ice: decadal simulations and future scenarios using BESM-OA. **Atmospheric and Climate Sciences**, v. 06, n. 02, p. 351–366, 2016. 7, 8

CHAVES, R. R.; NOBRE, P. Interactions between sea surface temperature over the South Atlantic Ocean and the South Atlantic Convergence Zone. **Geophysical Research Letters**, v. 31, n. 3, 2004. 5

CHOU, S. C.; LYRA, A.; MOURÃO, C.; DERECZYNSKI, C.; PILOTTO, I.; GOMES, J.; BUSTAMANTE, J.; TAVARES, P.; SILVA, A.; RODRIGUES, D.; CAMPOS, D.; CHAGAS, D.; SUEIRO, G.; SIQUEIRA, G.; NOBRE, P.; MARENGO, J. Evaluation of the ETA simulations nested in three global climate models. **American Journal of Climate Change**, v. 03, n. 05, p. 438–454, 2014. 1, 9

EMORI, S.; TAYLOR, K.; HEWITSON, B.; ZERMOGLIO, F.; JUCKES, M.; LAUTENSCHLAGER, M.; STOCKHAUSE, M. **CMIP5 data provided at the IPCC Data Distribution Centre**. 2016. Disponível em: <http://www.ipcc-data.org/docs/factsheets/TGICA_Fact_Sheet_CMIP5_data_provided_at_the_IPCC_DDC_Ver_1_2016.pdf>. 8

FIALHO, W. M. B. **A importância do acoplamento oceano-atmosfera para a previsão da Zona de Convergência do Atlântico Sul**. 90 p. Dissertação (Mestrado em Meteorologia) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2017. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m21b/2016/08.10.18.22>. Acesso em: 29 abr. 2018. 9

GIAROLLA, E.; SIQUEIRA, L. S. P.; BOTTINO, M. J.; MALAGUTTI, M.; CAPISTRANO, V. B.; NOBRE, P. Equatorial atlantic ocean dynamics in a coupled ocean–atmosphere model simulation. **Ocean Dynamics**, v. 65, n. 6, p. 831–843, apr 2015. 9

GRIFFIES, S.; ZADEH, N. **The MOM user guide**. 2019. Disponível em: <https://github.com/mom-ocean/MOM5/blob/master/doc/web/user_guide.md>. 19

HAMMOZ. **ECHAM-HAMMOZ - aerosol & atmospheric chemistry modules**. 2019. Disponível em: <https://redmine.hammoz.ethz.ch/projects/hammoz>. 8

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS - INPE. **Tupã - Supercomputação do INPE**. 2012. Disponível em: <http://supercomputacao.inpe.br/tupa>. 15

INTEL. **Vectorization: a key tool to improve performance on modern CPUs**. 2018. Disponível em: <https://software.intel.com/en-us/articles/vectorization-a-key-tool-to-improve-performance-on-modern-cpus>. 12

_____. **Spin time**. 2019. Disponível em: <https://software.intel.com/en-us/vtune-amplifier-help-spin-time>. 25

KUBOTA, P. Y. Visão geral e estado do desenvolvimento MCGA-CPTEC/INPE. In: WORKSHOP ANUAL DO PROJETO DO MODELO BRASILEIRO DO SISTEMA TERRESTRE BESM/FAPESP/INCT-MC/REDE CLIMA, 2015. **Anais...** [S.l.], 2015. 16

LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA - LNCC. **SDumont - Sistema de Computação Petaflópica do SINAPAD**. 2019. Disponível em: <http://sdumont.lncc.br/>. 2, 15, 20

MIGUEL, J. C. H.; MONTEIRO, M. Mudanças climáticas, tecnociência e geopolítica: Um modelo do sistema terrestre brasileiro e a soberania na produção de futuros climáticos. In: REUNIÃO DE ANTROPOLOGIA DA CIÊNCIA E DA TECNOLOGIA, 5. **Anais...** Porto Alegre, 2015. v. 2, p. 30. 6, 8, 9

MPI FORUM. **MPI Forum**. 2019. Disponível em: <http://mpi-forum.org/>. 10

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH - NCAR. **CESM1.0.4 user's guide - chapter 1 - introduction**. 2018. Disponível em: <http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc_1_0_4/x42.html>. 2

_____. **CESM1.0.4 user's guide - chapter 3 - configuring a case**. 2018. Disponível em: <http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc_1_0_4/x939.html>. 2

NOBRE, P.; ALMEIDA, R. A. D.; MALAGUTTI, M.; GIAROLLA, E. Coupled ocean–atmosphere variations over the South Atlantic Ocean. **Journal of Climate**, v. 25, n. 18, p. 6349–6358, sep 2012. 5

NOBRE, P.; CAPISTRANO, V. B.; BAPTISTA JUNIOR, M.; BOTTINO, M. J.; GIAROLLA, E.; PESQUERO, J. F.; FIGUEROA, S. N.; KUBOTA, P. Y.; BONATTI, J. P.; OLIVEIRA, G. S. d.; CARDOSO, M. F. Modelo brasileiro do sistema terrestre (BESM) para cenários de mudanças climáticas globais. In: MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO (Ed.). **Modelagem climática e vulnerabilidades setoriais à mudança do clima no Brasil**. Brasilia: MCTI, 2016. v. 1, p. 33–48. ISBN 9788588063303. Acesso em: 29 abr. 2018. 10

NOBRE, P.; MALAGUTTI, M.; URBANO, D. F.; ALMEIDA, R. A. F.; GIAROLLA, E. Amazon deforestation and climate change in a coupled model simulation. **Journal of Climate**, v. 22, n. 21, p. 5686–5697, nov 2009. 5

NOBRE, P.; SIQUEIRA, L. S. P.; ALMEIDA, R. A. F.; MALAGUTTI, M.; GIAROLLA, E.; CASTELÃO, G. P.; BOTTINO, M. J.; KUBOTA, P.; FIGUEROA, S. N.; COSTA, M. C.; BAPTISTA, M.; IRBER, L.; MARCONDES, G. G. Climate simulation and change in the brazilian climate model. **Journal of Climate**, v. 26, n. 17, p. 6716–6732, sep 2013. 1, 5, 6, 7, 8

OPENMP. **OpenMP - Enabling HPC since 1997. The OpenMP API specification for parallel programming**. 2019. Disponível em: <http://www.openmp.org/>. 10

PILOTTO, I. L.; CHOU, S. C.; NOBRE, P. Seasonal climate hindcasts with ETA model nested in CPTEC coupled ocean–atmosphere general circulation model. **Theoretical and Applied Climatology**, v. 110, n. 3, p. 437–456, apr 2012. 5

RODRIGUES, E. R.; NAVAUX, P. O. A.; PANETTA, J.; FAZENDA, A.; MENDES, C. L.; KALE, L. V. A comparative analysis of load balancing algorithms applied to a weather forecast model. In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 22., 2010. **Proceedings...** [S.l.], 2010. p. 71–78. ISSN 1550-6533. 16, 33

SEVERANCE, C.; DOWD, K. Loop optimizations - blocking to ease memory access patterns. In: **High Performance Computing**. OpenStax CNX, 2010. Disponível em: <http://cnx.org/content/m33756/1.3/>. 14

SIQUEIRA, L.; NOBRE, P. Tropical atlantic sea surface temperature and heat flux simulations in a coupled GCM. **Geophysical Research Letters**, v. 33, n. 15, 2006. 5

SOUZA, J. G. **Paralelização e otimização do desempenho computacional do modelo BRASIL-SR**. 128 p. Dissertação (Mestrado em Computação Aplicada) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2019. Disponível em: <http://urlib.net/rep/8JMKD3MGP3W34R/3T295ES>. Acesso em: 19 jan. 2020. 17

UNIVERSITY OF WISCONSIN-MADISON. **IBIS - Integrated Biosphere Simulator**: center for sustainability and the global environment. 2019. Disponível em: <https://nelson.wisc.edu/sage/data-and-models/lba/ibis.php>. 8

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.