SENAI CIMATEC

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL E TECNOLOGIA INDUSTRIAL

Mestrado em Modelagem Computacional e Tecnologia Industrial

Dissertação de mestrado

MCAP: MODELO COMPUTACIONAL DE AUTO-PARALELISMO

Apresentada por: André Luiz Lima da Costa
Orientador: Josemar Rodrigues de Souza, Ph.D.

Outubro de 2013

André Luiz Lima da Costa

MCAP: MODELO COMPUTACIONAL DE AUTO-PARALELISMO

Dissertação de mestrado apresentada ao Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial, Curso de Mestrado em Modelagem Computacional e Tecnologia Industrial do SENAI CIMATEC, como requisito parcial para a obtenção do título de **Mestre em Modelagem Computacional e Tecnologia Industrial**.

Área de conhecimento: Interdisciplinar

Orientador: Professor Josemar Rodrigues de Souza, Ph.D.
SENAI CIMATEC

Salvador
SENAI CIMATEC
2013

Nota sobre o estilo do PPGMCTI

Esta dissertação de mestrado foi elaborada considerando as normas de estilo (i.e. estéticas e estruturais) propostas aprovadas pelo colegiado do Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial e estão disponíveis em formato eletrônico (*download* na Página Web http://portais.fieb.org.br/portal_faculdades/dissertacoes-e-teses-mcti.html ou solicitação via e-mail à secretaria do programa) e em formato impresso somente para consulta.

Ressalta-se que o formato proposto considera diversos itens das normas da Associação Brasileira de Normas Técnicas (ABNT), entretanto opta-se, em alguns aspectos, seguir um estilo próprio elaborado e amadurecido pelos professores do programa de pós-graduação supracitado.

SENAI CIMATEC

Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial

Mestrado em Modelagem Computacional e Tecnologia Industrial

A Banca Examinadora, constituída pelos professores abaixo listados, leram e recomendam a aprovação [com distinção] da Dissertação de mestrado, intitulada "MCAP: MODELO COMPUTACIONAL DE AUTO-PARALELISMO", apresentada no dia 04 de outubro de 2013, como requisito parcial para a obtenção do título de **Mestre em Modelagem Computacional e Tecnologia Industrial**.

Orientador:

Professor Josemar Rodrigues de Souza, Ph.D.
SENAI CIMATEC

Membro interno da Banca:

Professor Dr. Renelson Ribeiro Sampaio
SENAI CIMATEC

Membro externo da Banca:

Professor Dr. Esbel Tomás Valero Orellana
Universidade Estadual de Santa Cruz - UESC

Dedico este trabalho aos meus pais, Epifânio e Arlinda, aos meus irmãos Mariane, Cauã, Cíntia e Márcia, à minha tia Nazaré, e à minha noiva, Lívia.

Agradecimentos

Agradeço primeiramente aos meus familiares, principalmente aos meus pais pelo apoio e aos meus irmãos por compreenderem minhas ausências. À minha noiva, que esteve sempre ao meu lado, me incentivando e ajudando a fazer uma pesquisa de qualidade. Aos meus amigos que também souberam compreender as minhas faltas nos finais de semana em que tive que me dedicar ao mestrado. Ao meu orientador Josemar Rodrigues de Souza, pela orientação que dedicou a este trabalho, pelas suas contribuições sempre pertinentes e pela forma que conduziu e contribuiu com a construção deste trabalho. À Felipe Zacarias pelo apoio na execução dos testes realizados no cluster do Laboratório de Modelagem Computacional. Aos meus colegas de mestrado, por estes dois anos de constante dedicação e aprendizado. E por fim, aos professores do Programa de Pós-Graduação Stricto Sensu da Faculdade de Tecnologia SENAI CIMATEC.

Salvador, Brasil
19 de Outubro de 2013

André Luiz Lima da Costa

Resumo

Com a chegada dos *clusters multi-core*, o conceito de programação híbrida tem se consolidado como um importante aliado dos usuários de Computação de Alto Desempenho, pois permite explorar do *hardware* um poder de processamento até então inutilizado. Como alternativa de adequação à esse conceito, o presente trabalho apresenta o MCAP, um Modelo Computacional de Auto-Paralelismo, cujo objetivo é gerar aplicações paralelas híbridas (MPI+OpenMP), a partir de aplicações MPI, de forma simples, automatizada e transparente ao usuário, que possibilite a redução do tempo de processamento destas aplicações paralelas. O modelo proposto foi implementado, como um *testbet*, através de uma aplicação web, disponível na internet. Ao final, foram realizadas comparações de aplicações MPI com suas versões híbridas, geradas pelo MCAP, onde foi constatado um aumento de performance médio de até 73,18%.

Palavras-Chave: modelo híbrido; auto-paralelismo; programação paralela; MPI; OpenMP.

Abstract

With the multi-cores clusters arrival , the concept of hybrid programming has been consolidated as an important allied for users of High Performance Computing, since it permits to explore from the hardware a processing capability never used before. This work presents the APCM, a Auto-Parallellism Computational Model, and its goal is to generate hybrid parallel applications (MPI + OpenMP) by MPI applications in a simple, automated and transparent way to the user, in order to reduce the processing time of these parallel apps. This model was implemented as a testbet by a web application available on the internet. At the end, MPI apps and its hybrid versions were compared by the APCM, in wich was observed 76,18% as a medium increase in performance.

Keywords: hybrid model; auto-parallelism; parallel programming; MPI; OpenMP.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Lista de Siglas

AMD	Advanced Micro Devices
API	Application Program Interface
CNPq	Concelho Nacional de Desenvolvimento Científico e Tecnológico
CUDA	Compute Unified Device Architecture
FAOCG	Framework for Automatic OpenMP Code Generation
GCC	GNU Compiler Collection
GPU	Graphics Processing Unit
LLVM	Low Level Virtual Machine
LNCC	Laboratório Nacional de Computação Científica
MCAP	Modelo Computacional de Auto-Paralelismo
MPI	Message Passing Interface
MVC	Model View Controller
NASA	National Aeronautics and Space Administration
OLAM	Ocean-Land Atmosphere Model
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
SaaS	Software as a Service
SIMD	Single Instruction, Multiple Data
TACC	Texas Advanced Computing Center
UESC	Universidade Estadual de Santa Cruz

Capítulo Um

Introdução

Enquanto os novos processadores *multi-core* oferecem um maior poder de processamento para melhorar o desempenho de aplicações paralelas, é do desenvolvedor a responsabilidade de empregá-los de forma eficiente em suas aplicações (OSTHOFF C.; SCHEPKE, 2010). A utilização do modelo de programação em memória compartilhada juntamente com o modelo de memória distribuída é uma prática antiga, e o objetivo dessa combinação, chamada de **programação híbrida**, é explorar o ponto forte dos dois modelos: a facilidade de programação e economia de memória do modelo de memória compartilhada e a escalabilidade do modelo em memória distribuída (LUSK E., 2008). Até pouco tempo, estes dois modelos evoluíam de forma independente, com o MPI (*Message Passing Interface*) firmando-se como a abordagem dominante na programação de memória distribuída, e o OpenMP (*Open Multi-Processing*) emergindo como a biblioteca dominante na utilização do modelo de memória compartilhada.

O modelo de programação híbrida (MPI + OpenMP) tem atraído a atenção dos programadores pelos seguintes motivos: a parceria do MPI com o OpenMP, que já se tornou um produto comercial sólido, sendo disponibilizado por diversos fornecedores de compilador no mercado, facilitando a integração destas duas plataformas e aumentando a performance de sua execução em conjunto; além do fato de os supercomputadores estarem preparados para executar aplicações híbridas por serem equipados por um conjunto de máquinas *multi-core*, aumentando o seu poder de processamento (LUSK E., 2008).

Assim, estudos têm evidenciado os benefícios de se optar pelo modelo de programação híbrida, por se utilizar as vantagens de dois modelos de programação para permitir o tão desejado aumento de performance das aplicações, porém adequar uma aplicação utilizando um novo conceito de programação paralela pode se tornar uma tarefa árdua, devido à grande quantidade de aplicações paralelas existentes que utilizam MPI.

1.1 Definição do Problema

Como aumentar a performance de uma aplicação paralela? Assim como a programação híbrida surge como uma solução para aumentar o desempenho destas aplicações, ela também pode se tornar uma barreira para os desenvolvedores, visto que eles precisam aprender um novo conceito de programação, utilizando memória distribuída, através do OpenMP, que apesar de ser simples, é totalmente diferente do conceito que normalmente

utiliza em seus algoritmos escritos em MPI, além da quantidade de códigos paralelos que precisam ser adequados para este novo paradigma de programação híbrida. Estas barreiras dificultam, e em alguns casos inviabilizam, a conversão destes algoritmos para uma nova versão híbrida, tornando-se assim um problema para um possível aumento de performance dessas aplicações paralelas.

1.2 Objetivo

Desenvolver um modelo computacional que possibilite o aumento de performance em aplicações paralelas, inserindo paralelismo a nível de memória compartilhada, utilizando o OpenMP, em algoritmos escritos na linguagem C com paralelismo em memória distribuída, através do MPI, gerando assim um novo algoritmo híbrido (MPI + OpenMP). Este processo é realizado de forma simples e transparente para o usuário, através de uma interface web, onde o desenvolvedor MPI não precisará conhecer o OpenMP para tornar sua aplicação híbrida. Os objetivos específicos desta dissertação são:

- Desenvolver um modelo capaz de gerar um algoritmo híbrido (MPI + OpenMP) a partir de um algoritmo MPI escrito em C;
- Desenvolver uma interface web, onde qualquer usuário possa utilizar o modelo proposto para gerar seus algoritmos híbridos de forma simples e intuitiva;
- Adicionar às aplicações MPI, através da conversão automática para a versão híbrida, a capacidade de explorar os recursos disponíveis nos nós com arquitetura de memória compartilhada;
- Testar e Avaliar o desempenho dos códigos híbridos produzidos com o modelo proposto.

1.3 Importância da Pesquisa

A Computação Paralela vem se consolidando nos últimos anos como uma alternativa viável de realizar o processamento de uma grande massa de dados num tempo considerado razoável para a quantidade de informação analisada. Este processamento é realizado em poderosos *clusters* computacionais, que hoje chegam a realizar trilhões de cálculos em um único segundo. Exemplos de áreas que demandam esse alto poder de processamento são a biotecnologia, com pesquisas aplicadas na decifração dos genomas, e a meteorologia, com suas constantes, e cada vez mais apuradas, previsões climáticas.

Na computação de alto desempenho, um único segundo pode fazer diferença, por este motivo, os desenvolvedores estão sempre pesquisando formas de melhorar a performance de suas aplicações paralelas, seja na utilização de novos modelos matemáticos, na otimização do algoritmo ou na utilização de novos conceitos de paralelismo, como a programação híbrida. Um dos modelos híbridos que vem sendo utilizado pela comunidade científica é a junção do MPI com o OpenMP, ideal para serem executados em *clusters multi-core* , extraindo um maior poder de processamento de seu *hardware* , pois, além de dividir o processamento entre os diversos nós do cluster, em cada nó é possível dividir o processamento entre os seus respectivos núcleos de processamento. O ganho de performance de aplicações na versão híbrida sobre a sua versão que utiliza puramente o MPI já é comprovado, conforme (WU X.; TAYLOR, 2013), (OSTHOFF C.; GRUNMANN, 2011), (JIN H.; JESPERSEN, 2011), (LI S. B. R.; SCHULZ, 2010), (RABENSEIFNER R.; HAGER, 2009) e (LUSK E., 2008).

Diante da quantidade de códigos paralelos passíveis de serem adequados às modernas arquiteturas paralelas, desenvolver um modelo computacional que permita gerar um algoritmo híbrido a partir de um algoritmo MPI, de forma transparente e automatizada, a fim de aumentar a performance destes algoritmos, é relevante para o cenário da computação de alto desempenho, pois diminui a necessidade de o desenvolvedor trabalhar com distintas metodologias de paralelismo, permitindo concentrar os seus esforços numa única metodologia. Dessa forma, a contribuição dessa pesquisa é disponibilizar o modelo proposto, através de uma interface multiplataforma, disponível na web, podendo ser utilizado por qualquer desenvolvedor conectado à internet.

1.4 Organização da Dissertação de Mestrado

Esta dissertação está organizada em 5 capítulos, são eles:

- **Capítulo 1: Introdução -** Este é o capítulo atual, no qual o trabalho é brevemente apresentado, descrevendo o seu objetivo e a sua importância para a comunidade científica;
- **Capítulo 2: Estado da Arte** Neste capítulo, são apresentadas algumas pesquisas que abordam a geração automática de paralelismo em algoritmos e, ainda, trabalhos que utilizam a programação híbrida;
- **Capítulo 3: Modelo Proposto** Neste capítulo, é apresentado o MCAP, abordando a arquitetura do modelo, detalhando o seu processo de desenvolvimento e apresentando a sua aplicação web;

- **Capítulo 4: Apresentação dos Resultados** Neste capítulo são apresentados os resultados obtidos nos testes realizados para a validação do modelo, além da análise desses resultados;
- **Capítulo 5: Considerações Finais** Por fim, neste capítulo, são apresentadas as conclusões obtidas através da análise dos resultados dos testes realizado com o MCAP.

Capítulo Dois

Estado da Arte

Para o desenvolvimento dessa pesquisa, foi necessário o aprofundamento de dois temas-chave, são eles: Geração de Código-Fonte Paralelo Automatizado e Programação Híbrida. No primeiro, para entender como outros trabalhos semelhantes transformavam algoritmos numa versão paralela, de forma automatizada. No segundo, para elencar e entender as vantagens do modelo de programação híbrida, utilizando MPI e OpenMP, em *clusters multi-core*.

2.1 Geração de Código-Fonte Paralelo Automatizado

Existem, no mercado, projetos que realizam a geração de código-fonte em linguagens de programação paralela de forma automatizada, partindo de algoritmos seriais para novos algoritmos paralelos. Na tabela 2.1, observa-se a comparação entre as linguagens de programação e o tipo de paralelismo gerado por alguns destes trabalhos. Dos quatro projetos abordados, todos aceitam aplicações escritas em C/C++ e geram paralelismo utilizando o OpenMP. A seguir, mais detalhes de cada um deles.

Projeto	Linguagem		Modelo de Programação Paralela Utilizada			
	C/C++	FORTAN	MPI	OpenMP	OpenCL	CUDA
Par4All	x	x		x	x	x
Polly	x			x		
FAOCG	x			x		
CETUS	x		x	x		x

Tabela 2.1: Comparação de ferramentas que geram código paralelo a partir de código sequencial. Fonte: Autor, 2013 .

2.1.1 Par4All

O Par4All (TORQUATI M.; VANNESCHI, 2012) é um projeto *open-source* cujo objetivo é adaptar de forma automatizada algoritmos sequenciais para diversos tipos de *hardwares* como GPU, sistemas *multicores* , computadores de alto desempenho e sistemas embarcados, ou seja, ele transforma uma aplicação sequencial em uma nova aplicação paralela e

otimizada, mantendo a aplicação original intacta, além de possibilitar que o desenvolvedor explore a execução de seu algoritmo de forma mais eficiente em ambientes *multi-core* .

Através do Par4All é possível transformar um algoritmo sequencial escrito na linguagem C numa versão paralela, utilizando OpenMP, CUDA ou OpenCL, ou da linguagem FORTRAN 77 para o OpenMP. O Par4All utiliza como motor de paralelização o PIPS (AMINI M.; ANCOURT, 2011), que é um *framework* dedicado a realizar a tradução de códigos-fonte baseado no modelo de compilação poliedral, porém com foco na manipulação de aplicações completas e não apenas em parte delas.

2.1.2 Polly

Polly (GROSSER T.; ZHENG, 2011) é um projeto que permite otimização poliedral em *Low Level Virtual Machine* - LLVM (ZHAO J.; NAGARAKATTE, 2012), criado para detectar e transformar automaticamente partes relevantes de um programa em uma linguagem independente. Este projeto suporta algoritmos escritos nas principais linguagens de programação, como o C/C++. Construído utilizando avançadas bibliotecas poliédricas, o Polly tem uma interface simples, onde é possível aplicar transformações manuais ou utilizar otimizações externas.

Este projeto usa uma representação matemática abstrata, que analisa e otimiza o padrão de acesso de memória de um programa, que inclui os dados locais, otimizados para o *cache* local, bem como de paralelização automática em *thread-level* (utilizando o OpenMP) e paralelismo em SIMD (*Single Instruction, Multiple Data*). Seu principal objetivo é ser um otimizador integrado para acesso aos dados locais e para gerar paralelismo, sendo melhor aproveitado por ambientes *multi-core* , hierarquias de cache, instruções curtas de vetor, bem como aceleradores dedicados.

2.1.3 Framework for Automatic OpenMP Code Generation - FAOCG

O *Framework for Automatic OpenMP Code Generation* (RAGHESHI, 2011) é um trabalho fruto de uma dissertação de mestrado, cujo objetivo foi desenvolver um *framework* capaz de paralelizar de forma automatizada programas sequenciais, utilizando as diretivas do OpenMP na geração da versão paralela. Este *framework* é baseado no modelo poliédrico, mas especificamente no modelo de Polly (GROSSER T.; ZHENG, 2011).

O que diferencia este trabalho dos demais modelos especializados na geração de código-fonte, de forma automatizada e que utilizam o modelo poliédrico, é que ele não se restringe

a paralelizar programas escritos em apenas uma ou duas linguagens, que normalmente é o C/C++, ele abrange uma maior variedade de linguagens de programação, dando uma maior possibilidade aos desenvolvedores de tornar sua aplicação paralela e pronta para extrair mais processamento dos *clusters multi-core*.

2.1.4 CETUS

CETUS (DAVE C.; BAE, 2009) é uma ferramenta de geração de código-fonte que enfatiza a paralelização automatizada, provendo aos pesquisadores uma infraestrutura para compilação em ambientes *multi-core*, com uma interface de fácil utilização pelos usuários. O projeto embasa-se na pesquisa Polaris (BLUMEETAL, 1996), que, segundo Dave, é a pesquisa de infraestrutura mais avançada para otimização de compiladores para ambientes paralelos.

Enquanto o Polaris concentra-se na transformação de programas que utilizavam a linguagem FORTRAN, o CETUS trabalha com programas escritos em C. A ferramenta surgiu através de uma pesquisa de vários alunos de pós-graduação e foi escrita na linguagem JAVA. O CETUS permite a transformação de programas de memória compartilhada (OpenMP) para outros modelos, como o de troca de mensagem (MPI) ou de unidade de processamento gráfico (CUDA).

2.2 Programação Híbrida

Na computação de alto desempenho, a programação híbrida reflete a junção de dois ou mais modelos de programação paralela em um mesmo algoritmo, e o exemplo mais comum é a junção do MPI com o OpenMP. A principal motivação para utilizar programação híbrida é extrair as melhores características de ambos os modelos de programação, onde um programa está hierarquicamente estruturado como uma série de tarefas MPI, cujo código sequencial está enriquecido com diretivas OpenMP para adicionar *multithreading* e aproveitar as características da presença de memória compartilhada e multiprocessadores dos nós (RIBEIRO N. S.; FERNANDES,). A programação híbrida surge, também, como uma solução viável de adequação da grande quantidade de aplicações MPI existentes a estarem aptos a explorar, de forma mais eficaz, os *clusters multi-core* , utilizando o processamento em memória compartilhada.

O MPI é uma interface para troca de mensagem em aplicações paralelas, que utiliza ambientes de memória distribuída (MPI, 2013), onde uma aplicação é dividida e processada separadamente em diferentes máquinas de um *cluster*, mantendo sua comunicação através

da troca de mensagens entre essas máquinas. Definida como um padrão pelo Fórum MPI, ela suporta as linguagens C, C++ e FORTRAN, tendo como principais vantagens a portabilidade e o padrão para a troca de mensagens. Atualmente, existem versões do MPI para os mais diversos sistemas operacionais (SILLA P. R.; BRUNETTO, 2008).

Já o OpenMP é uma API *(Application Program Interface)* para desenvolvimento de aplicações paralelas utilizando ambientes de memória compartilhada, em que o processamento de uma aplicação é dividido pelos processadores da máquina na qual ela está sendo executada. Adota um modelo de programação portável e escalável, permitindo sua utilização em diversas plataformas, variando de um simples computador pessoal até um supercomputador (OPENMP, 2013).

Nas figuras 2.1 e 2.2, é possível observar como funciona a execução de uma aplicação híbrida que utiliza o MPI e o OpenMP. O exemplo contido nas figuras simula um algoritmo que realiza a multiplicação de uma matriz quadrada [4x4], desenvolvida no modelo *master/worker*, sendo executada em um *cluster multi-core* com 5 nós *dual-core*, onde 1 nó é o *master* e os 4 restantes são os *workers*.

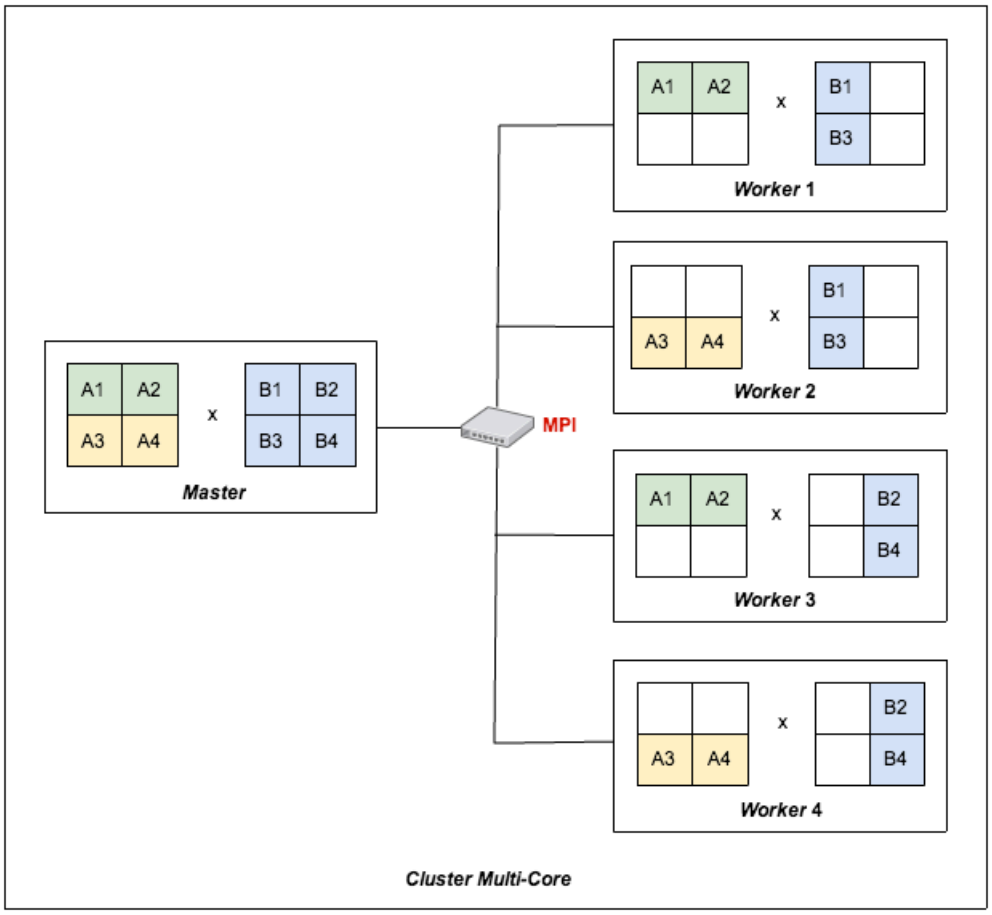

Figura 2.1: Multiplicação de Matrizes com MPI. Fonte: O Autor, 2013.

Na figura 2.1, o master divide o processamento fruto da multiplicação de matrizes entre os *workers* e envia para cada um deles uma linha da matriz A e a matriz B inteira, onde cada nó realiza a multiplicação de parte das matrizes A (linha) e da matriz B (coluna), e depois devolve os resultados obtidos para o *master* poder juntá-los e formar uma nova matriz C.

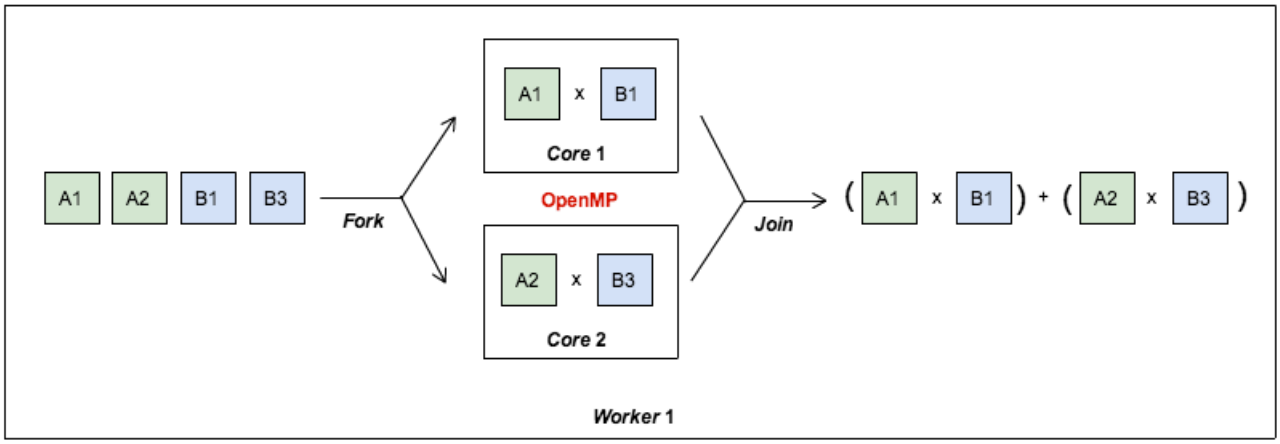

Figura 2.2: Divisão de tarefas em um nó com o OpenMP. Fonte: O Autor, 2013.

Já na figura 2.2 é detalhado como o OpenMP age em cada nó do *cluster*, neste caso, no *worker* 1. O nó obtém do *master*, via MPI, os blocos das matrizes A e B (coluna e linha respectivamente) e inicia o paralelismo em memória compartilhada, ativando duas *threads*, uma para cada *core* no nó, já que o processador é *dual-core*. Em cada *thread* é realizada parte do processamento total do nó e, após terminarem as *threads*, são finalizadas, voltando para a programação sequencial do nó, onde os resultados de cada *core* são juntados e devolvidos para o *master*.

Utilizar aplicações paralelas híbridas (MPI + OpenMP) em *clusters multi-core* é vantajoso quando sua performance é comparada com aplicações puramente MPI (JIN H.; JESPERSEN, 2011), (OSTHOFF C.; GRUNMANN, 2011), (WU X.; TAYLOR, 2013), (LI S. B. R.; SCHULZ, 2010), (RABENSEIFNER R.; HAGER, 2009) e (LUSK E., 2008), porém tranformar uma aplicação paralela em uma versão híbrida não garante que a aplicação aumente o seu desempenho, em alguns casos a aplicação pode não diminuir o seu tempo de processamento ou, até mesmo, apresentar uma performance inferior à sua versão puramente MPI, conforme explorado em (OSTHOFF C.; GRUNMANN, 2011), onde esses resultados negativos podem ser motivados por um mau balanceamento de carga ou utilização de diretivas inadequadas para a aplicação. A seguir, serão apresentados dois dos trabalhos pesquisados durante esta dissertação, que exploraram os benefícios em se utilizar o modelo de programação híbrida.

2.2.1 Programação Híbrida em Aplicação Meteorológica

A pesquisa de (OSTHOFF C.; GRUNMANN, 2011) faz parte do projeto Atmosfera Massiva, financiado pelo CNPq, cujo objetivo é o estudo do impacto de arquiteturas *multi-core* e paralelismo de múltiplos níveis em modelos meteorológicos e ambientais. O artigo mostra como um modelo de programação híbrida, utilizando MPI e OpenMP, pode aumentar a performance do *Ocean-Land Atmosphere Model* OLAM, que é um modelo computacional desenvolvido para realizar previsões meteorológicas em escala global, que exige um alto poder de processamento em suas previsões. No trabalho, foi comparada a execução do OLAM, tanto na versão MPI quanto na versão híbrida, para três diferentes tipos de configurações da aplicação. A comparação foi realizada em um *cluster* Altix composto por 30 *dual node Intel Xeon E5520 2.27GHz Quad Core* com 24Gb de memória e rede *infiniband*, localizado no Laboratório Nacional de Computação Científica LNCC.

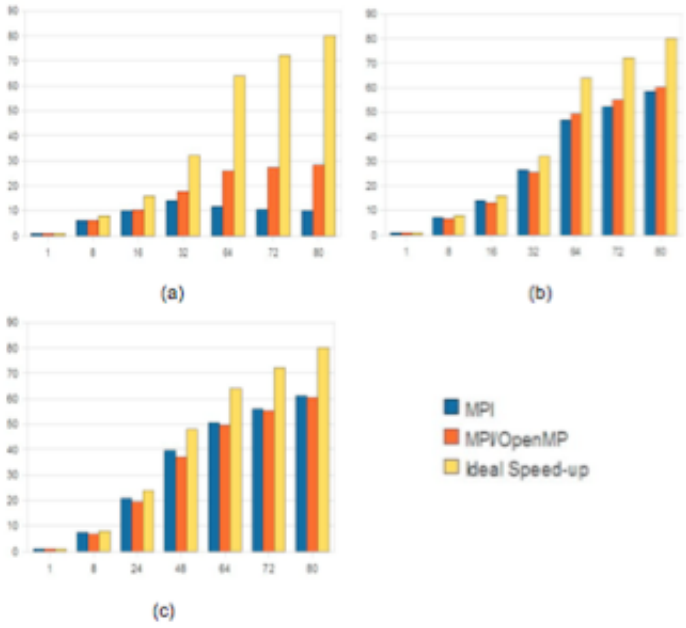

Figura 2.3: Resultados em diferentes configurações. Fonte: (OSTHOFF C.; GRUNMANN, 2011).

Na Figura 2.3, observa-se o resultado comparativo do OLAM na versão MPI, na versão híbrida e com o seu *speed-up* ideal, para três diferentes configurações de balanceamento de carga, utilizando de 1 a 80 *cores*. No gráfico (a), utilizando uma configuração com resolução de 200 km de distância, a versão híbrida apresentou melhores resultados do que a versão MPI quando processado a partir de 32 *cores*. No gráfico (b), utilizando uma resolução de 40 km, o processamento em ambas as versões ficou mais próximo de seu *speed-up* ideal, onde a versão híbrida apresentou melhores resultados apenas a partir de 64 *cores*. Já no gráfico (c), mantendo uma resolução de 40 km, porém com otimizações físicas em seu modelo, a versão híbrida não apresentou melhores resultados em relação

à versão MPI, provavelmente isso só aconteceria com uma quantidade maior de *cores* no *cluster*, conforme escalabilidade do modelo.

Em (OSTHOFF C.; GRUNMANN, 2011), a customização de um algoritmo híbrido para execução em *clusters multi-core* foi viável, porém estes resultados só apareceram à medida que foi aumentada a quantidade de *cores* utilizados no processamento, dependendo do pacote enviado pelo MPI para cada máquina. Estes resultados demonstram a importância do balanceamento de carga na performance da aplicação e da escalabilidade do cluster para a execução de algoritmos híbridos. Desta forma, apenas desenvolver uma versão híbrida de uma aplicação não garante um ganho de performance na sua execução, esta ação tem que ser conjunta com um balanceamento de carga ideal para cada aplicação que utiliza MPI e OpenMP.

2.2.2 Programação Híbrida em Clusters MultiCore

O artigo de (RABENSEIFNER R.; HAGER, 2009) apresenta os benefícios do modelo de programação híbrida utilizando MPI e OpenMP, em *clusters multicores*. Para os autores, a topografia do supercomputador, onde as aplicações paralelas são executadas, tem impacto significativo no desempenho da aplicação, independente da estratégia utilizada na paralelização do algoritmo. Os testes realizados nesta pesquisa utilizam dois algoritmos da *NAS Parallel Benchmark* (NPB), pertencente à NASA *Advanced Supercomputing Division*, são eles: SP-MZ e o MT-MZ. Os testes comparam a execução de ambos os algoritmos com suas versões híbridas.

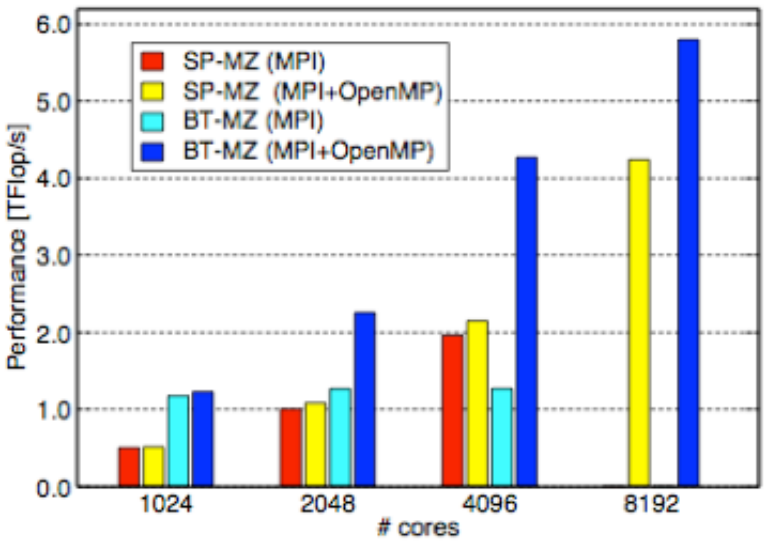

Figura 2.4: Resultados obtidos nos testes. Fonte: (RABENSEIFNER R.; HAGER, 2009).

Na figura 2.4, pode-se observar que, a partir de 1024 *cores*, já é possível notar uma discreta vantagem da versão híbrida para a versão MPI, tanto para o SP-MZ quanto para BT-MZ; a partir de 8192 *cores*, já é destacada a diferença de desempenho da versão híbrida (chegando perto de 6 *teraflops*), a ponto da versão MPI não ser observada no gráfico. Para a execução dos testes, foi utilizado o cluster da *Texas Advanced Computing Center* (TACC), chamado de *Ranger*, composto por 3936 *blades AMD quad-core de 2.3 GHz* com 32 GB de memória, ligados em uma rede *infiniband*.

Capítulo Três

Modelo Computacional de Auto-Paralelismo - MCAP

O Modelo Computacional de AutoParalelismo (MCAP) tem como objetivo aumentar a performance de aplicações paralelas que utilizam MPI em *clusters multi-core*. Isso ocorre através da geração automatizada de paralelismo em memória compartilhada, utilizando a biblioteca OpenMP, onde o MCAP transforma algoritmos MPI em uma versão híbrida (MPI + OpenMP), assim, o novo algoritmo, além de explorar o paralelismo com a divisão do processamento entre as máquinas do *cluster*, também irá usufruir do paralelismo gerado pelo OpenMP, dividindo o processamento enviado para cada nó entre os seus respectivos núcleos.

Apesar de ter como foco o autoparalelismo de aplicações já paralelas, que utilizam MPI, o MCAP também funciona com o paralelismo de aplicações seriais, neste caso, a aplicação serial é convertida em uma versão paralela utilizando o OpenMP, também de forma automatizada e transparente para o usuário. Assim, os programadores que utilizam a linguagem C poderão explorar melhor o poder de processamento dos *clusters multi-core*, sem a necessidade de realizar modificações em seu ambiente de produção.

Como o processo de gerar paralelismo em memória compartilhada não requer um alto poder de processamento, o MCAP foi implementado para ser utilizado na plataforma WEB, disponibilizando assim o seu serviço em nuvem, onde pesquisadores de todo o mundo poderão utilizá-los sem a necessidade de baixar, instalar e configurar em seu *cluster*, como é feito pelos trabalhos citados no estado da arte aqui apresentado.

3.1 O Modelo

O MCAP é um modelo computacional, cuja finalidade é gerar paralelismo de forma automatizada e transparente ao usuário final. A figura 3.1 representa a interação entre o usuário e o modelo, através de uma aplicação web, demonstrando, assim, o fluxo desde o envio do código MPI até o recebimento do novo código híbrido gerado.

Figura 3.1: Fluxo de utilização do MCAP pelo usuário. Fonte: O Autor, 2013.

Durante o processo de transformação de um algoritmo MPI em um algoritmo híbrido, o MCAP realiza algumas etapas claramente definidas e já validadas por projetos similares. Na Figura 3.2, é apresentado o fluxo das 5 macroetapas realizadas pelo MCAP, no qual cada etapa só inicia quando a etapa anterior finaliza.

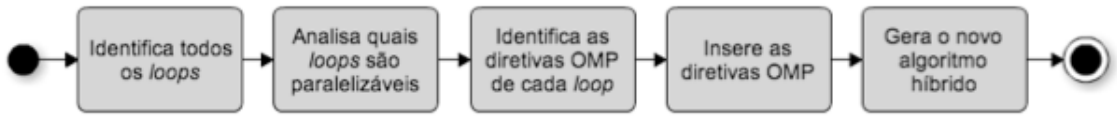

Figura 3.2: Passos realizados para gerar paralelismo. Fonte: O Autor, 2013.

A seguir, o detalhamento de cada etapa utilizada para gerar o paralelismo abordado na figura 3.2:

- **Identifica todos os *loops*:** Esta é a primeira etapa, na qual a aplicação varre todo o algoritmo enviado, mapeando todos *loops* do tipo FOR. Esse passo é muito importante, pois o paralelismo será embutido justamente nos laços de repetição, por se tratar de trechos do algoritmo que consomem a maior parte do tempo de processamento das aplicações (LI J.; SHU, 2005);
- **Analisa quais *loops* são paralelizáveis:** Após a identificação dos *loops*, a aplicação verifica quais daqueles *loops* podem ser paralelizáveis, para isso, é observado se o *loop* contém alguma variável que dependa de outro *loop*, se ele executa algum comando MPI (ex.: *MPI_Bcast(), MPI_Send(), MPI_Recv()*, etc.) ou, até mesmo, se o *loop* está contido dentro de outro *loop*;
- **Identifica as diretivas OMP de cada *loop*:** Nesta etapa, a aplicação identifica qual a melhor configuração do OpenMP para cada loop paralelizável o algoritmo , para isso, é observado quais as cláusulas do OpenMP (*reduction, private, shared,*

schedule, etc.) melhor se adequam a cada *loop*, para que se possa extrair o máximo de performance de cada um deles;

- **Insere as diretivas OMP:** Após identificar as diretivas de cada *loop*, a aplicação insere o código OpenMP no algoritmo enviado pelo usuário, sem modificar nenhum trecho do código original;
- **Gerar o novo algoritmo híbrido:** Por fim, a aplicação irá gerar a versão híbrida do algoritmo em um novo arquivo e a disponibilizará para o usuário fazer o *download*.

Os cinco macroprocessos apresentados na figura 3.2, representados detalhados no diagrama de sequência ilustrado na figura 3.3, que apresenta passo a passo o processo de autoparalelismo realizado pelo MCAP na visão de camadas da engenharia de software.

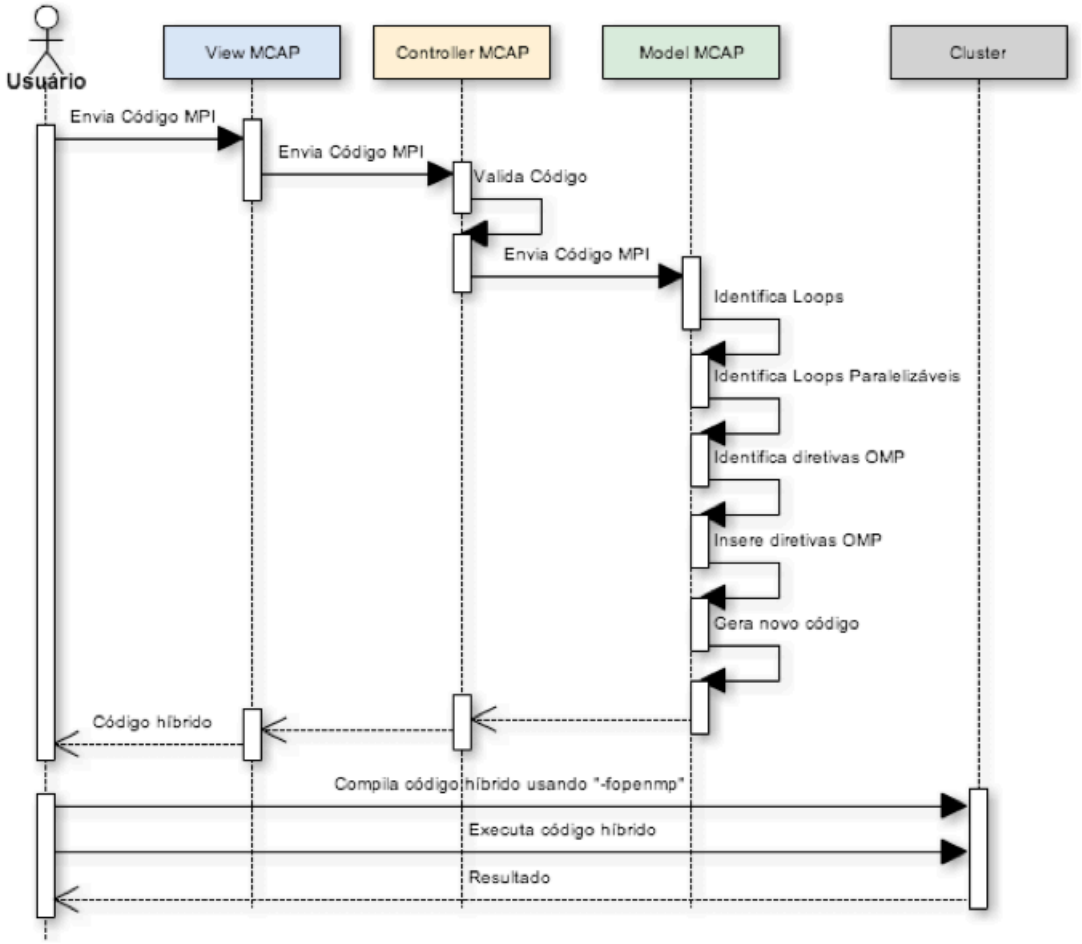

Figura 3.3: Diagrama de sequência do MCAP. Fonte: O Autor, 2013.

Por se tratar de um modelo e não de um *framework* ou compilador, o MCAP tem como premissa a necessidade de obedecer alguns pré-requisitos para que a conversão do algo-

ritmo seja feita da forma correta e a versão híbrida tenha potencial para aumentar a performance de sua aplicação. Os pré-requisitos exigidos pelo modelo são:

- Estar escrito na linguagem de programação C e, consequentemente, estar salvo na extensão *.c. Isto é necessário, pois as diretivas OpenMP que serão utilizadas no MCAP são compatíveis apenas com a linguagem C;
- Utilizar apenas MPI nos algoritmos paralelos, pois outras bibliotecas de paralelismo, como o OpenMP, não foram devidamente testadas no modelo;
- Utilizar a técnica de multiplicação de matrizes juntamente o modelo *Master/Worker* em sua composição, pois o MCAP foi validado apenas em aplicações com estas configurações e, por isso, não garante sua eficiência e eficácia em algoritmos que não sigam esse padrão;
- Colocar o abre chaves ({) na mesma linha da função a qual ela pertence e nunca na linha de baixo;
- Colocar o fecha chaves (}) em uma linha separada e nunca na linha onde fica a função ou seu conteúdo;
- Transformar todos os laços de repetição ou pelo menos os que queira que sejam paralelizados no tipo FOR, pois o MCAP não trabalha com os demais tipos de laços;
- Colocar todos os *loops* do tipo FOR com "abre chaves"e "fecha chaves", mesmo que a função contenha uma única instrução em seu conteúdo, caso contrário, o MCAP não contemplará este *loop* para ser paralelizável;
- Não poderá conter erros de sintaxe, principalmente relacionados ao "abrir chaves"e "fechar chaves", que delimitam o conteúdo de cada função, pois o MCAP poderá delimitar de forma errada os loops identificados;
- Enviar um único algoritmo por vez para gerar o autoparalelismo, pois o MCAP não aceita projetos ou múltiplos arquivos.

Essas premissas são necessárias para a conversão correta do algoritmo original em sua versão híbrida (ou paralela, caso o algoritmo original seja serial). Caso alguma dessas premissas não seja seguida, o MCAP poderá não completar a conversão, gerar um novo algoritmo incompleto, gerar um algoritmo com erro de sintaxe ou gerar uma versão híbrida que não incremente a performance ou, até mesmo, que aumente o tempo de execução da aplicação.

```
2   int main()
3   {
4
5       int num;
6
7       printf("Digite um numero: ");
8       scanf("%d", &num);
9
10      if ( num > 10 ) { printf("Maior que 10!"); }
11
12      if ( num > 20 )
13          printf("Maior que 20!");
14
15      return 0;
16
17  }
```

Figura 3.4: Código inadequado para MCAP. Fonte: O Autor, 2013.

```
21  int main() {
22
23      int num;
24
25      printf("Digite um numero: ");
26      scanf("%d", &num);
27
28      if ( num > 10 ) {
29          printf("Maior que 10!");
30      }
31
32      if ( num > 20 ) {
33          printf("Maior que 20!");
34      }
35
36      return 0;
37
38  }
```

Figura 3.5: Versão do código adequado para o MCAP. Fonte: O Autor, 2013.

Na figura 3.4, pode-se verificar um exemplo de código-fonte que não segue os pré-requisitos do MCAP e, consequentemente, está inadequado para o modelo. As caixas em vermelho identificam os problemas do algoritmo, que são: abre chave na linha abaixo do comando IF (o correto seria abrir na mesma linha), o conteúdo do IF e o fecha chave estão na mesma linha do IF (o correto seria o conteúdo do IF iniciando na linha de baixo, e o fecha chave numa nova linha) e a função FOR está sem as chaves (o correto é sempre utilizar o abre chaves e o fecha chaves). Na figura 3.5, observa-se o mesmo algoritmo em uma versão adequada para a utilização do MCAP.

O MCAP é um modelo que pode ser utilizado por usuários de Computação de Alto Desempenho com diferentes níveis de instrução, abrangendo desde usuários iniciantes, que

estão aprendendo a criar aplicações paralelas, até usuários experientes, que querem agilidade em incrementar suas aplicações. Por este motivo, a sua aplicação web disponibiliza aos seus usuários três opções de configurações, são elas:

- **Básico:** é a opção indicada para usuários que querem utilizar os recursos mínimos do OpenMP, seja por não saber qual as diretivas mais indicadas para a sua aplicação ou pelo fato de seu algoritmo apresentar melhores resultados nessa opção. Neste caso, o MCAP insere apenas a diretiva *#pragma omp for* nos laços;
- **Padrão:** é a opção indicada para usuários que não conhecem o OpenMP ou têm pouca experiência com esta biblioteca. Esta configuração tem ativada as diretivas de compilação que apresentam, na média, um melhor desempenho em algoritmos de multiplicação de matrizes no modelo *Master/Worker*, são elas: *Schedule* do tipo *Dynamic* e a cláusula *Reduction*;
- **Customizado:** é a opção indicada para usuários avançados no OpenMP, pois lhes permite escolher qual configuração irão utilizar para gerar o paralelismo. As opções de customização disponíveis para o usuário na aplicação do MCAP estão listadas a seguir, juntamente com suas respectivas definições:
  - ***Schedule(Static)*:** as iterações do loop são divididas igualmente e distribuídas estaticamente para cada *thread*;
  - ***Schedule(Dynamic)*:** as iterações do loop são divididas em partes menores e distribuídas dinamicamente para cada *thread*. Quando uma *thread* termina o processamento, volta para pegar um novo pedaço;
  - ***Schedule(Guided)*:** assim como no *Dynamic*, as interações são divididas e distribuídas dinamicamente. A diferença é que inicia com pedaços maiores e vai decrementando para pedaços menores ao decorrer das iterações;
  - ***Private*:** a cláusula private define quais os parâmetros terão propriedades privadas no laço de repetição, ou seja, os dados dos parâmetros são individuais para cada *thread*;
  - ***Reduction*:** reduz os valores de variáveis locais e transforma em variáveis globais na região parelala ;
  - ***Default(Shared)*:** transforma todas as variáveis da região paralela visível e acessível simultaneamente por todas as *threads*;
  - **Número de *Threads*:** define o número de *threads* que a aplicação irá criar nas regiões paralelas; por padrão, o OpenMP cria uma *thread* para cada núcleo do processador.

3.2 A Arquitetura

O MCAP foi implementado na plataforma web e projetado para utilizar a arquitetura de camadas *Model-View-Controller* (MVC), que é um modelo de desenvolvimento atualmente considerado um padrão de projeto, utilizado na Engenharia de Software e desenvolvido por (REENSKAUG,). A arquitetura do MCAP está representada na figura 3.6 e é composta por uma Aplicação WEB que contém as 3 camadas, *Model, View e Controller*, sendo a primeira onde fica a regra de negócio . O MCAP relaciona-se indiretamente com o *cluster multi-core*, que é formado pela Aplicação Paralela (transformado pelo MCAP), MPI, Compilador GCC com a biblioteca do OpenMP e o Sistema Operacional.

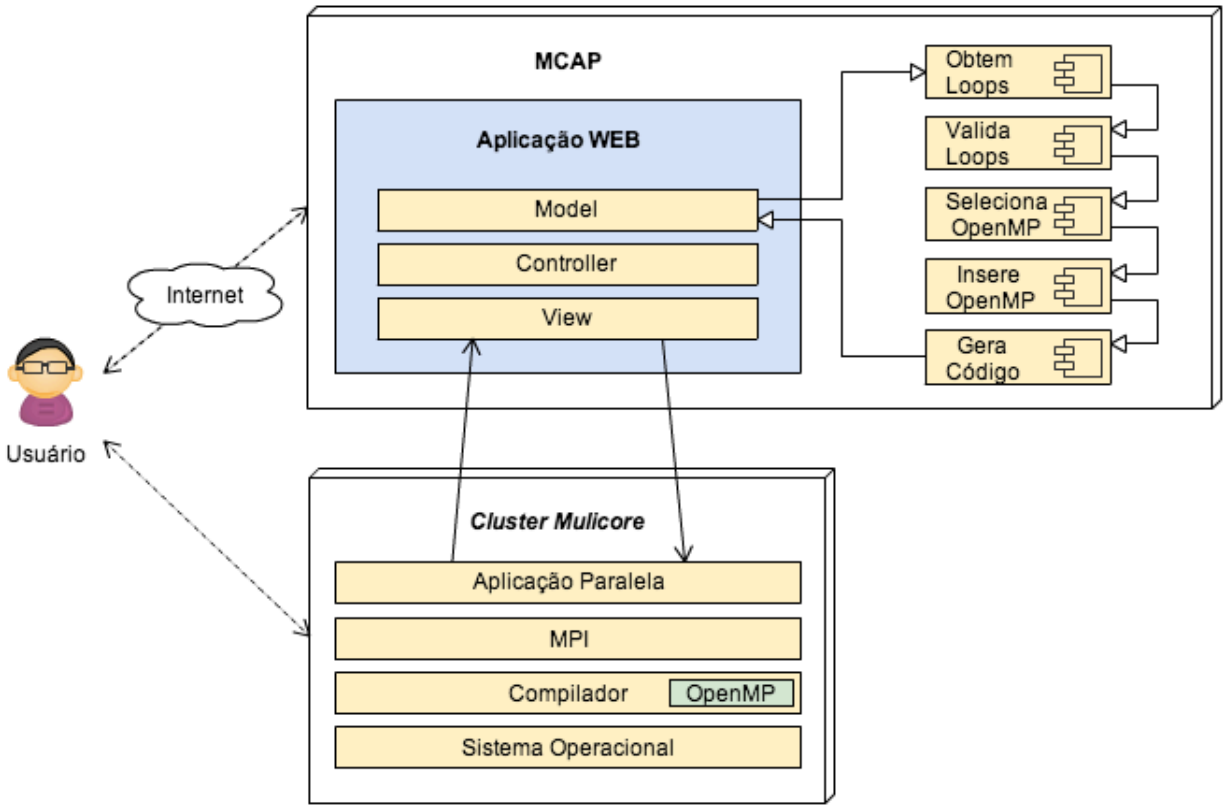

Figura 3.6: Arquitetura do MCAP. Fonte: O Autor, 2013.

A Aplicação web do MCAP, por utilizar a estrutura do MVC, também está dividida em 3 camadas, representadas da seguinte forma: a) *Model* - onde estão implementadas as regras de negócio do MCAP, através de métodos contidos na classe MCAP; b) *View* - onde fica a interface de apresentação da aplicação, que é apresentada para o usuário . Esta interface foi desenvolvida com os padrões da WEB 2.0 e utiliza o componente *Twitter Bootstrap*; c) *Controller* - responsável por obter o código enviado pelo usuário e disponibilizar para a camada Model realizar o paralelismo e, depois, devolver a nova versão para o usuário.

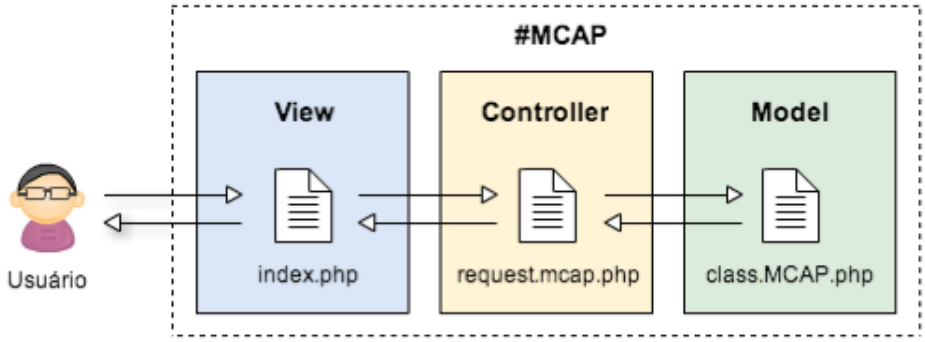

Figura 3.7: Interação do usuário com a arquitetura MVC do MCAP. Fonte: O Autor, 2013.

Na figura 3.7, observa-se a interação do usuário com a aplicação do MCAP, onde o usuário tem acesso apenas à camada de apresentação, tornando o processo de conversão do algoritmo totalmente transparente. Um dos benefícios da utilização do MVC é que as camadas podem ser desenvolvidas e/ou atualizadas separadamente, para isso, basta definir previamente um padrão de comunicação entre as camadas que ficarão no *Controller*.

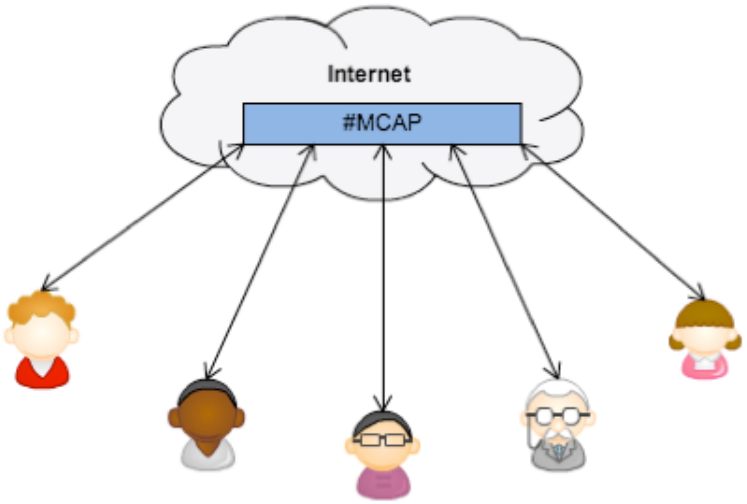

Figura 3.8: MCAP em nuvem. Fonte: O Autor, 2013.

Na figura 3.8, verifica-se o MCAP disponível na internet como um serviço de *cloud computing*, utilizando o conceito de *Software as a Service* (SaaS), no qual um ou mais usuários podem acessar a aplicação simultaneamente, de qualquer lugar e a qualquer momento, desde que estejam on-line.

3.3 A Aplicação

A aplicação WEB, testbed que valida o MCAP, foi desenvolvida utilizando HTML (*HyperText Markup Language*) para a organização dos dados, CSS (*Cascading Style Sheets*) para a apresentação dos dados, JavaScript para programação no cliente (*front-end*) e PHP (acrônimo recursivo para *PHP: Hypertext Preprocessor*) para programação no servidor (*back-end*). Utilizou o conceito de Orientação a Objetos, MVC, e na camada de visualização adotou o *framework Twitter Bootstrap*. A aplicação está disponível na internet para qualquer usuário através do endereço ***http://andrecosta.info/mcap***. Ela está dividida em cinco seções, são elas: Gerar Código OpenMP, O que é, Como Funciona, Autores e Publicações. Na figura 3.9 é apresentada a tela da seção Gerar Código OpenMP.

Figura 3.9: Aplicação WEB do MCAP. Fonte: O Autor, 2013.

Na aplicação, a opção de configuração sugerida ao usuário é o Padrão, por ser a opção com parâmetros testados e que apresentam as melhores médias de resultados. Ao mudar para a opção Customizado, a aplicação dispõe de uma interface que permite ao usuário personalizar as propriedades do paralelismo. A figura 3.10 representa essa interface com a opção Customizado selecionado, na qual, para gerar um código híbrido, o usuário precisa selecionar o código MPI, informar a configuração do OpenMP (ela já vem com os parâmetros da configuração Padrão preenchida) e, por fim, clicar em Gerar Paralelismo, onde a aplicação faz a conversão e disponibiliza o novo arquivo para *download*.

Figura 3.10: Aplicação WEB do MCAP com a opção Customizado. Fonte: O Autor, 2013.

3.4 Resultados dos Testes para Obter a Configuração Padrão do MCAP

Para se obter a configuração padrão, disponibilizada pelo MCAP na aplicação web, foi necessária a realização de testes comparativos entres as cláusulas disponíveis para configuração na aplicação. Desta forma, foram preparados 8 algoritmos com configurações OpenMP diferentes: configuração básica sem cláusulas e sem definições de quantidade de *threads* (chamado de OMP Puro), com apenas a cláusula *schedule* do tipo *static*, com apenas a cláusula *schedule* do tipo *dynamic*, com apenas a cláusula *schedule* do tipo *guided*, com apenas a cláusula private, com apenas a cláusula reduction, com apenas a cláusula *shared* definida como *default* e por fim utilizando um número diferente de *treads*, no caso 4 . Todos os algoritmos tiveram a mesma base, uma aplicação de multiplicação de matrizes (10.000 x 10.000) utilizando MPI (executando com 4 processos, 1 processo em cada nó) no modelo *master/worker*, tendo como diferença apenas as suas respectivas diretivas do OpenMP.

Resultados dos Testes Comparativos (em segundos)			
dynamic	***static***	***guided***	***OMP***
2053,46	4172,94	4416,04	4088,88
private	***reduction***	***shared***	***4 threads***
7358,31	3249,39	4495,49	7671,66

Tabela 3.1: Média da execução de cada algoritmo sem o maior e menor tempo. Fonte: O Autor, 2013.

Os testes comparativos foram realizados cinco vezes, excluindo-se o menor e o maior tempo de cada execução e, depois, extraiu-se a média das três execuções restantes (Tabela 3.1), obtendo assim um resultado satisfatório. Estes tempos foram obtidos através da função *MPI_Wtime()*, calculado apenas o tempo de processamento e desprezando o tempo de inicialização da aplicação e do MPI. Para os testes, foi utilizado o *cluster multi-core* do laboratório de Modelagem Computacional do SENAI CIMATEC, contendo a seguinte configuração: 8 *blades HP ProLiant DL120 G6* com Processador *Intel Xeon QuadCore X3440 2.53 GHZ com HyperThreading*, 8 GB de memória RAM, 2 TB de armazenamento e sistema operacional *Linux 2.6.35-32-generic X86 64 GNU Linux Ubuntu 10.10*, interligados em um *switch D-Link Des-1024D 10/100 FastEthernet*. Para compilação (mpicc) e execução (mpirun) dos testes, foi utilizado o Open MPI, na versão 1.7.2, que é uma implementação *open source* do MPI-2, além do OpenMP 4.0 contido no compilador GCC 4.8.1. Como os testes tiveram fins comparativos entre os algoritmos executados, a deficiência na comunicação entre os nós, devido à baixa velocidade do *switch*, não prejudicou ou influenciou nos resultados obtidos.

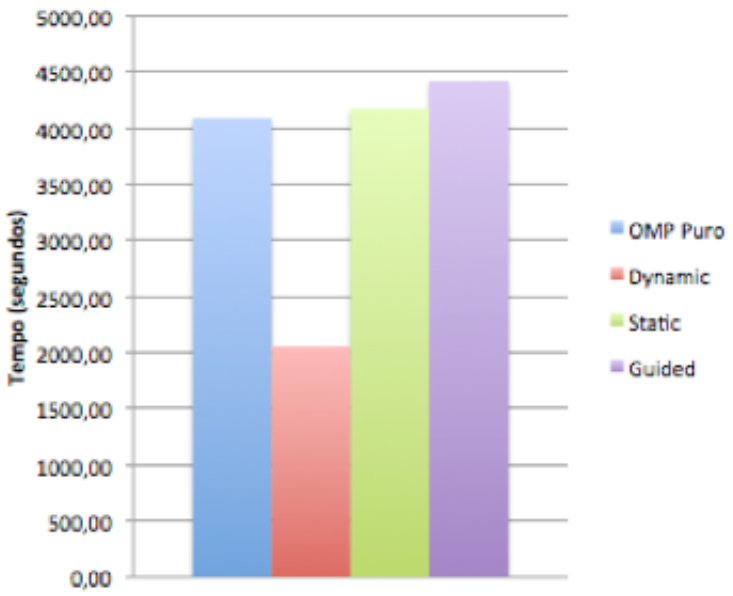

Figura 3.11: OpenMP Puro x Schedule (Dynamic, Static e Guided). Fonte: O Autor, 2013.

Na figura 3.11, observam-se os resultados da comparação de desempenho do algoritmo OMP Puro com as versões, utilizando a cláusula *schedule static*, *schedule dynamic* e *schedule guided*. Das três opções de schedule, a única que apresentou um melhor resultado que a versão pura do OpenMP foi a cláusula *schedule dynamic*.

Figura 3.12: OpenMP Puro x Private. Fonte: O Autor, 2013.

Ao utilizar a cláusula *private* no algoritmo testado, o tempo de processamento aumentou quando comparado à versão pura do OpenMP (Figura 3.12).

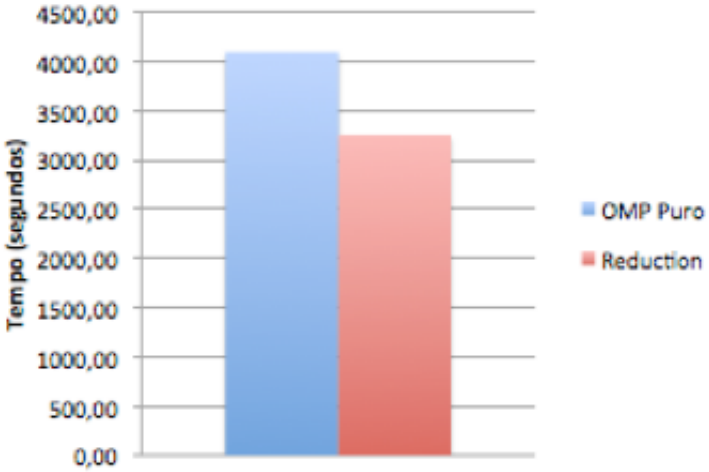

Figura 3.13: OpenMP Puro x Reduction. Fonte: O Autor, 2013.

Na figura 3.13, pode-se observar a comparação do OMP Puro com a versão do algoritmo utilizando a cláusula *reduction* do OpenMP. A utilização da cláusula diminuiu o tempo de execução do algoritmo e, consequentemente, melhorou a performance da aplicação.

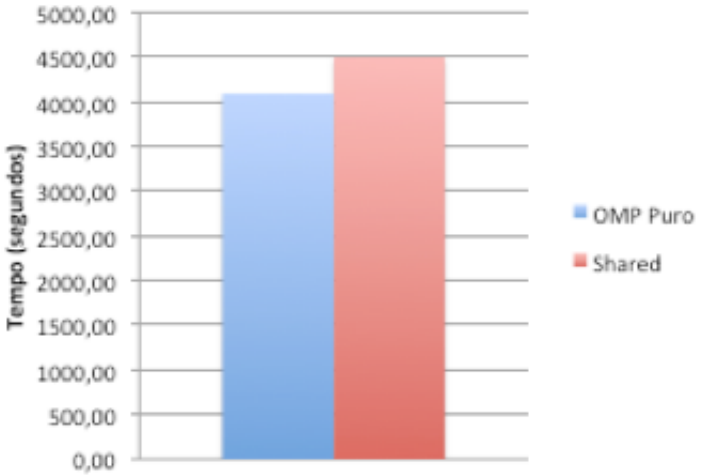

Figura 3.14: OpenMP Puro x Shared. Fonte: O Autor, 2013.

Na figura 3.14, observa-se a perda de performance do algoritmo utilizando a cláusula *shared* como padrão dos variáveis nos *loops*. O OMP Puro apresentou um melhor tempo de execução.

Figura 3.15: OpenMP Puro x 4 Threads. Fonte: O Autor, 2013.

E, por fim, na figura 3.15, é verificada a comparação do OMP Puro, que utiliza 8 *threads*, com a versão do algoritmo configurada para utilizar apenas 4 *threads*, onde mais uma vez a versão básica do OpenMP obtém um melhor desempenho. A opção de 4 *threads* foi utilizada pelo o fado do processador de cada nó do *cluster* ser *quad-core*, ou seja, tem 4 núcleos físicos, porém, o processador conta com a tecnologia *HyperThreading*, que simula o dobro de núcleos, criando mais 4 núcleos virtuais, totalizando 8 núcleos e, por este motivo, o OMP Puro utilizou por padrão 8 *threads*.

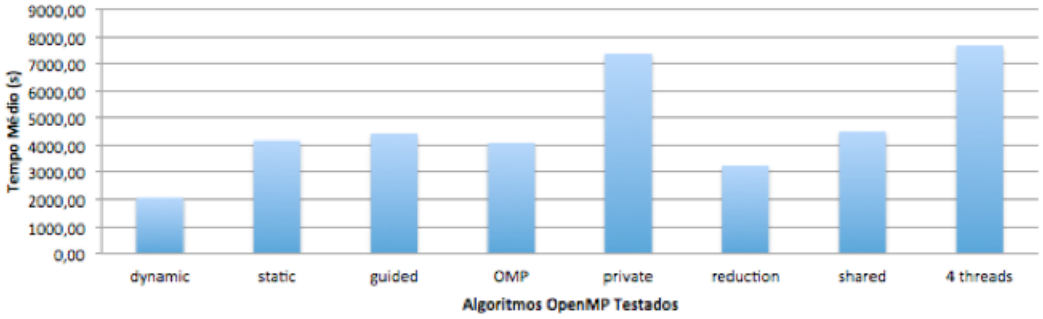

Figura 3.16: Comparativo de todos os algoritmos testados. Fonte: O Autor, 2013.

Após a apresentação desses resultados, é possível concluir que, para o algoritmo de multiplicação de matrizes testado, utilizando MPI no modelo *master/worker*, a configuração do OpenMP que apresentou um melhor desempenho é a que utiliza as cláusulas *reduction* e *schedule* do tipo *dynamic* em sua diretiva de compilação, conforme figura 3.16, resultados similares aos encontrados em (LI J.; SHU, 2005). Dessa forma, a opção Padrão do MCAP foi configurada com bases nesses resultados, acreditando ser a melhor opção para usuários que não têm conhecimento no OpenMP.

Capítulo Quatro

Resultados Obtidos

Para analisar a eficiência do MCAP, foram realizados testes comparativos com os algoritmos gerados pelo modelo com sua versão original. Estes testes também foram realizados no *cluster multi-core* do SENAI CIMATEC, com o mesmo algoritmo de multiplicação de matrizes, com uma matriz 10.000 x 10.000, mas, desta vez, realizando um escalonamento de 1 a 8 máquinas. Os dados foram obtidos através de 5 execuções para cada algoritmo, onde excluiu-se o menor e o maior tempo de cada algoritmo e obteve-se a média dos 3 resultados restantes. Os testes foram organizados da seguinte forma: Comparando um algoritmo MPI com sua versão híbrida, utilizando 1 processo com 8 *threads* por nó, comparando um algoritmo MPI com sua versão híbrida, utilizando 2 processos com 4 *threads* por nó e comparando um algoritmo MPI com sua versão híbrida utilizando 4 processos com 2 *threads* por nó. Além da versão híbrida, também foi testada uma versão paralela gerada pelo MCAP, utilizando 2, 4, 6 e 8 *threads*, e comparando com sua versão serial. Para garantir a quantidade de processos MPI específicos de cada teste, foi utilizado um arquivo *hostfile* informando os nós utilizados na execução, juntamente com o parâmetro *slots* que força a quantidade de processos por cada nó.

Para cada teste, serão apresentados uma tabela com o tempo médio obtido, um gráfico desse tempo médio, um gráfico com o ganho de performance, um gráfico com *SpeedUp* e um gráfico com o *Efficiency*. Os tempos dos algoritmos MPI/Híbridos foram calculados com a função *MPI_Wtime()*; já nos algoritmos Serial/Paralelo foi utilizada a função *time()*, e em ambos os casos foram medidas apenas as funções que realizam a multiplicação das matrizes. As fórmulas utilizadas para obter alguns dos resultados que serão apresentados são:

Ganho de Performance em Porcentagem:

$$GP = 100 - \frac{Tm * 100}{To}$$

Onde "Tm"é o tempo do algoritmo gerado pelo o MCAP e o "To"é o tempo do algoritmo original.

SpeedUp:

$$Sp = \frac{T1}{Tp}$$

Onde "T1"é o tempo do algoritmo serial e o "Tp"é o tempo do algoritmo paralelo rorando com "p"processos.

Efficiency:

$$Ep = \frac{S}{p}$$

Onde "Sp" textitSppedUp para "p"processos e o "p"é o número de processos utilizados na execução paralela.

4.1 MPI vs. Híbrido (MCAP) com 1 Processo e 8 Threads por nó

	Tempo Médio (segundos)		
Processo(s)	MPI Puro	Híbrido	Ganho (%)
1	21202,132832	4517,385393	78,69
2	10747,495021	2537,892949	76,39
3	7234,514239	1765,511283	75,60
4	5480,870150	1435,881681	73,80
5	4427,975790	1215,467759	72,55
6	3727,100404	1025,368321	72,49
7	3243,208471	922,792780	71,55
8	2868,257028	859,623461	70,03
			73,18

Tabela 4.1: Resultado médio e ganho de performance entre o algoritmo híbrido e MPI. Fonte: O Autor, 2013.

Na Tabela 4.1, é apresentada a média dos resultados com as duas versões dos algoritmos comparados por número de processos utilizados, apresentando em porcentagem o ganho de performance da versão híbrida em relação à versão original. Este teste foi realizado com 1 proceso MPI e 8 *threads* OpenMP por nó, ou seja, utilizou uma thread exclusiva

para cada um dos 8 *cores* do nó e sem concorrênca de processo MPI. Com esses dados, é possível constatar o aumento de performance com o uso do MCAP. Através da média entre o ganho de performance para os diferentes níveis de escalonamento, chegou-se a um ganho médio de **73,18%**, um número expressivo para aplicações de computação de alto desempenho.

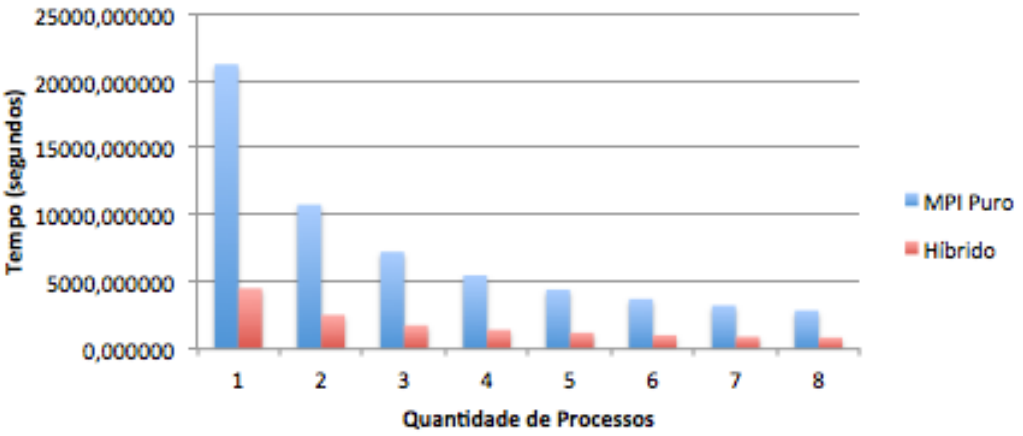

Figura 4.1: Comparação de Resultados do MPI Puro x Híbrido. Fonte: O Autor, 2013.

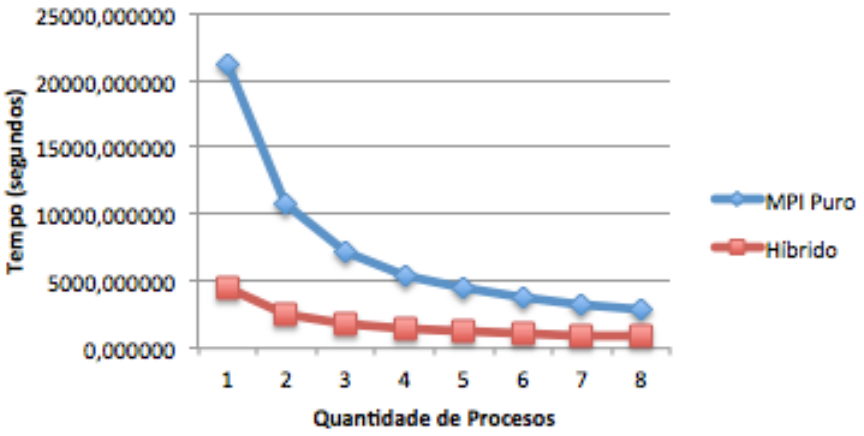

Figura 4.2: Comparação de Resultados do MPI Puro x Híbrido. Fonte: O Autor, 2013.

Observa-se na figura 4.1 e na figura 4.2 a vantagem na escolha de um modelo de programação em memória compartilhada (OpenMP), juntamente com o modelo de programação distribuída (MPI) em *cluster multi-core*, onde o tempo de processamento do algoritmo híbrido é menor que o da versão MPI, ou seja, apresenta um melhor desempenho.

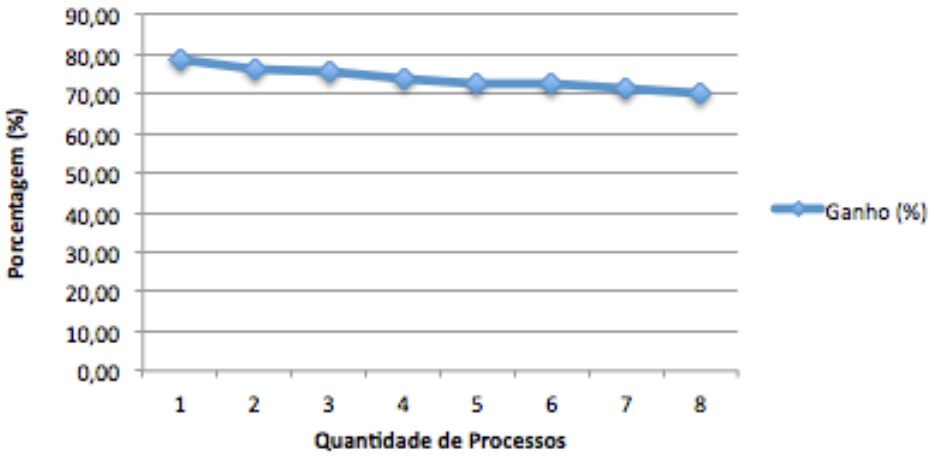

Figura 4.3: Ganho de Performance utilizando o MCAP. Fonte: O Autor, 2013.

O ganho de performance obtido com o MCAP apresenta uma variação 78,69% a 70,03%, utilizando de 1 a 8 processos respectivamente (Figura 4.3), que são resultados satisfatórios. À medida que foi aumentando o número de processos na execução, o ganho de performance tendeu a diminuir.

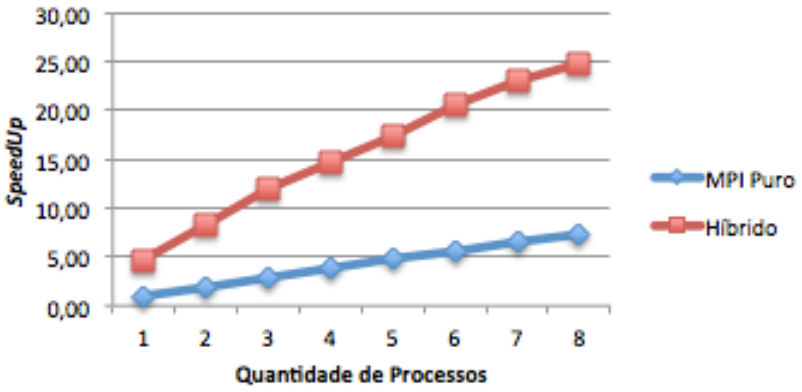

Figura 4.4: SpeedUp das versões MPI Puro e Híbrido. Fonte: O Autor, 2013.

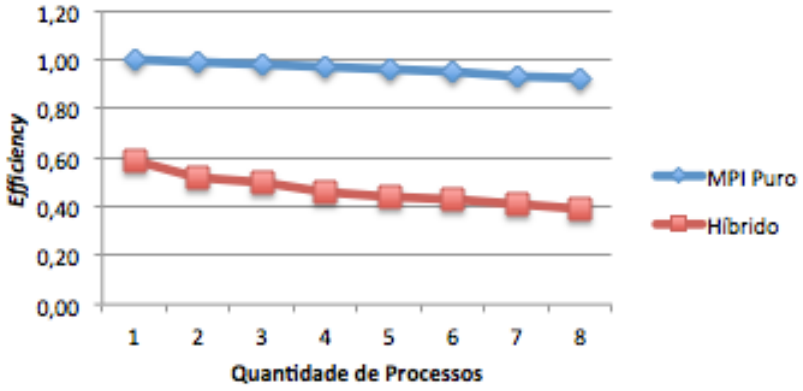

Figura 4.5: Efficiency das versões MPI Puro e Híbrido. Fonte: O Autor, 2013.

Nas figuras 4.4 e 4.5 são apresentados o *SpeedUp* e o *Efficiency* respectivamente, sendo possível constatar um melhor *SpeedUp* da versão híbrida gerada pelo MCAP em relação à versão MPI pura, devido ao seu ganho de performance, porém a versão MPI apresentou uma melhor *Efficiency*, por se tratar de uma linguagem que apresenta um melhor escalonamento.

4.2 MPI vs. Híbrido (MCAP) com 2 Processos e 4 Threads por nó

	Tempo Médio (segundos)		
Processo(s)	MPI Puro	Híbrido	Ganho (%)
2	21202,132832	4701,139156	77,83
4	10747,495021	2800,712097	73,94
6	7234,514239	1836,546634	74,61
8	5480,870150	1470,391001	73,17
10	4427,975790	1219,438669	72,46
12	3727,100404	1014,656857	72,78
14	3243,208471	916,706785	71,73
16	2868,257028	837,523204	70,80
			72,97

Tabela 4.2: Resultado médio e ganho de performance entre o algoritmo híbrido e MPI. Fonte: O Autor, 2013.

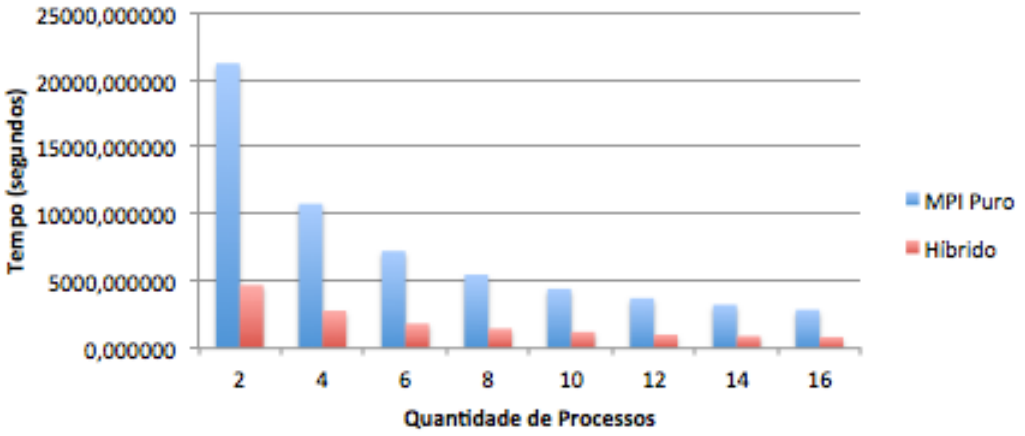

Figura 4.6: Comparação de Resultados do MPI Puro x Híbrido. Fonte: O Autor, 2013.

O resultado dos testes utilizando a versão híbrida, executando com 2 processos MPI e 4 Threads por nó do *cluster multi-core* (Tabela 4.2 e Figura 4.6), foi similar ao realizado utilizando apenas 1 processo com 8 threads por nó, evidenciando, também, a vantagem da versão híbrida gerada pelo MCAP, dessa vez com um ganho médio de performance em 72,97%.

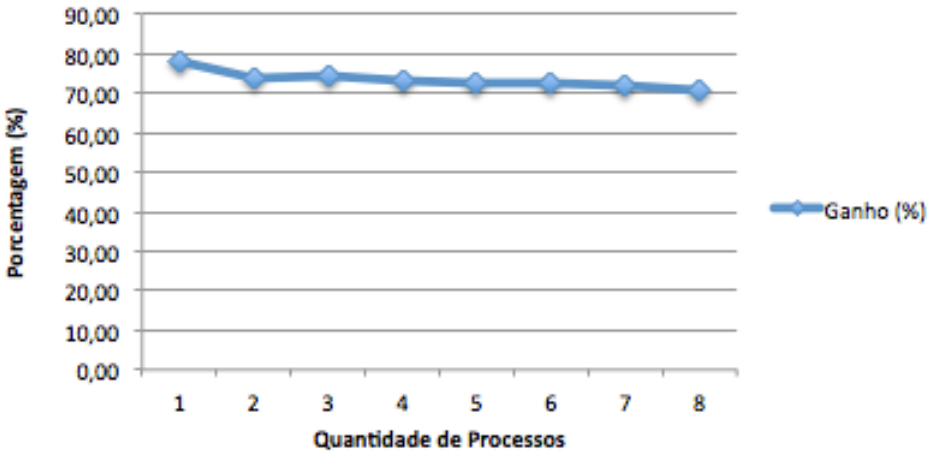

Figura 4.7: Ganho de Performance utilizando o MCAP. Fonte: O Autor, 2013.

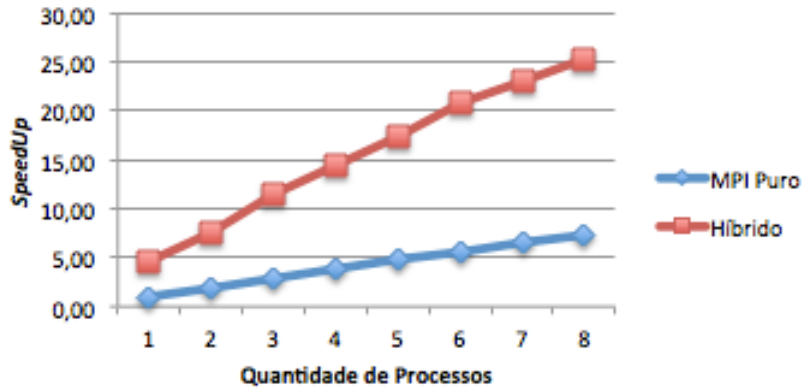

Figura 4.8: SpeedUp das versões MPI Puro e Híbrido. Fonte: O Autor, 2013.

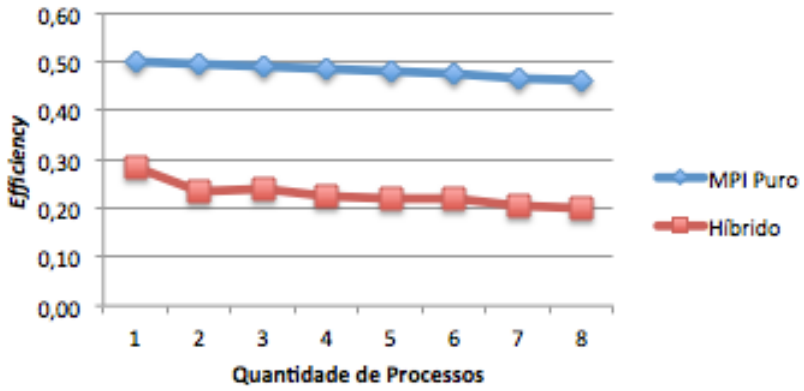

Figura 4.9: Efficiency das versões MPI Puro e Híbrido. Fonte: O Autor, 2013.

Nas figuras 4.7, 4.8 e 4.9, são apresentados, respectivamente, o Ganho de Performance, *SpeedUp* e o *Efficiency* para os 8 nós; é apresentado, ainda, um resultado similar ao teste anterior, mesmo modificando o número de processos e threads.

4.3 MPI vs. Híbrido (MCAP) com 4 Processos e 2 Threads por nó

	Tempo Médio (segundos)		
Processo(s)	MPI Puro	Híbrido	Ganho (%)
4	21202,132832	6622,081293	68,77
8	10747,495021	3761,158467	65,00
12	7234,514239	2557,392539	64,65
16	5480,870150	2133,317833	61,08
20	4427,975790	1662,627439	62,45
24	3727,100404	1427,013312	61,71
28	3243,208471	1348,194485	58,43
32	2868,257028	1173,083367	59,10
			62,08

Tabela 4.3: Resultado médio e ganho de performance entre o algoritmo híbrido e MPI. Fonte: O Autor, 2013.

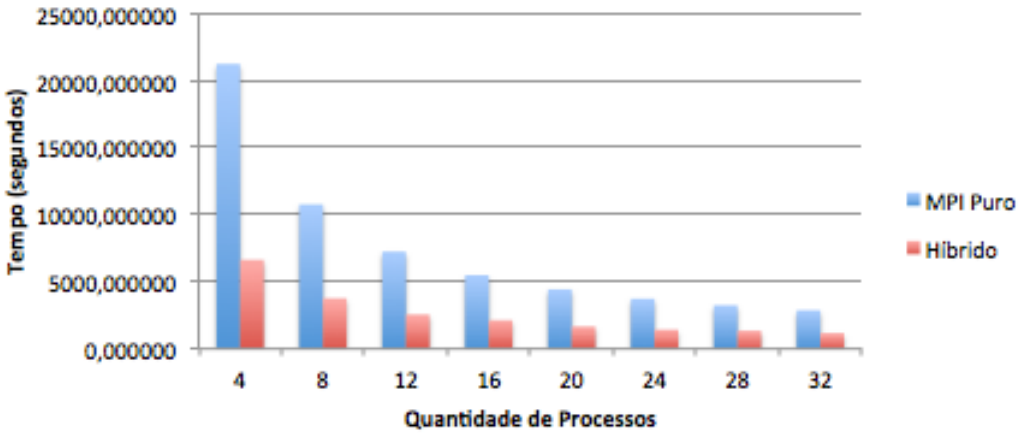

Figura 4.10: Comparação de Resultados do MPI Puro x Híbrido. Fonte: O Autor, 2013.

O resultado dos testes, utilizando a versão híbrida executando com 4 processos MPI e 2 Threads por nó do *cluster multi-core* (Tabela 4.3 e Figura 4.10), já apresentou um resultado inferior ao realizado utilizando apenas 1 processo com 8 threads por nó. Essa queda se justifica pela quantidade de *threads* utilizados pelo OpenMP, já que foi explorado menos da metade dos cores disponíveis pelo nó. Mesmo com a queda de desempenho, a versão híbrida apresentou um ganho médio de performance em 62,08%.

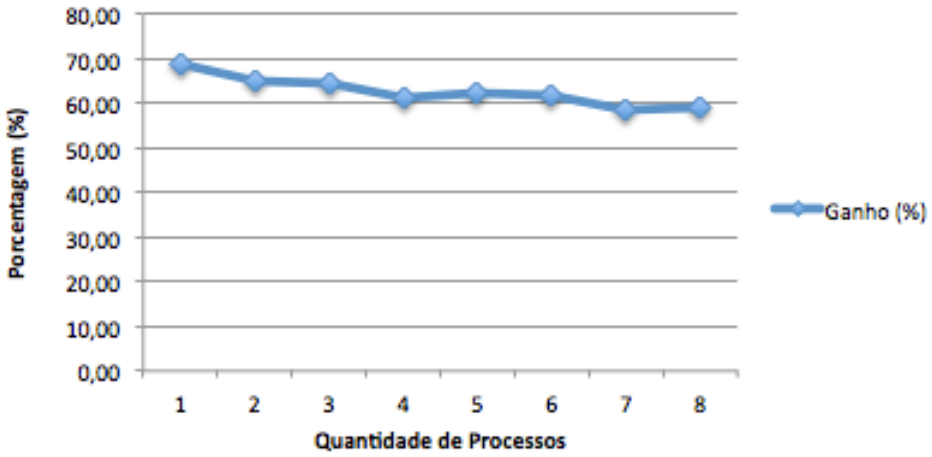

Figura 4.11: Ganho de Performance utilizando o MCAP. Fonte: O Autor, 2013.

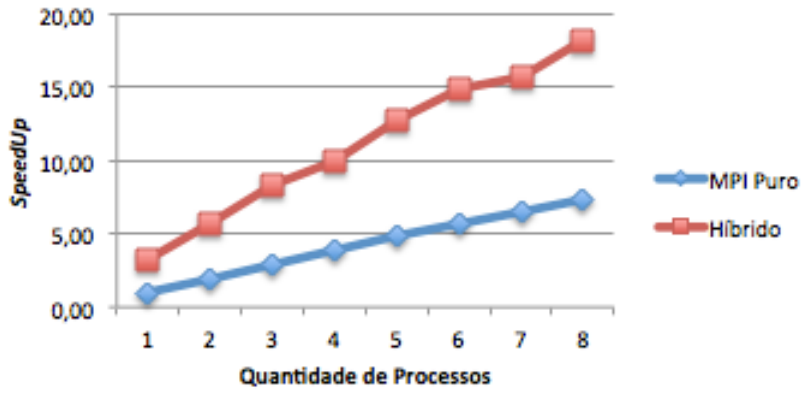

Figura 4.12: SpeedUp das versões MPI Puro e Híbrido. Fonte: O Autor, 2013.

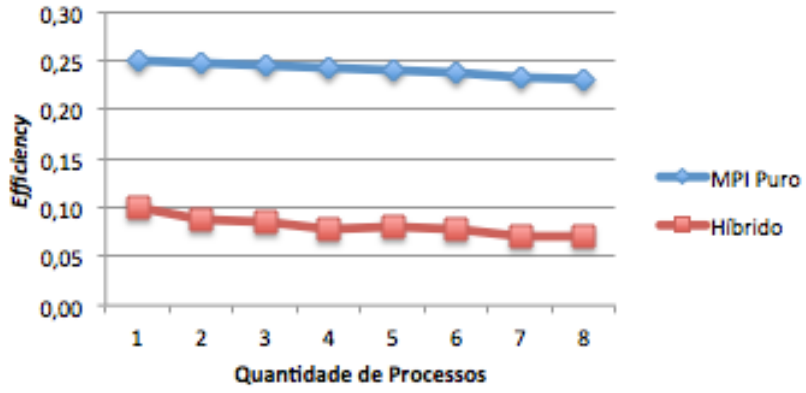

Figura 4.13: Efficiency das versões MPI Puro e Híbrido. Fonte: O Autor, 2013.

Nas figuras 4.11, 4.12 e 4.13, são apresentados o Ganho de Performance, *SpeedUp* e o *Efficiency* respectivamente para os 8 nós, onde, apesar do aumento no tempo de proces-

samento da versão híbrida, os gráficos apresentaram um comportamento similar ao do teste com 1 processo e 8 *threads* por nó.

4.4 Serial vs. Paralelo OpenMP (MCAP)

	Serial	2 Threads	4 Threads	6 Threads	8 Threads
Tempo Médio (s)	33997,33	21114,33	10372,67	6417,67	5775,00
Ganho (%)		37,89	69,49	81,12	83,01

Tabela 4.4: Resultado médio e ganho de performance entre o algoritmo serial e paralelo. Fonte: O Autor, 2013.

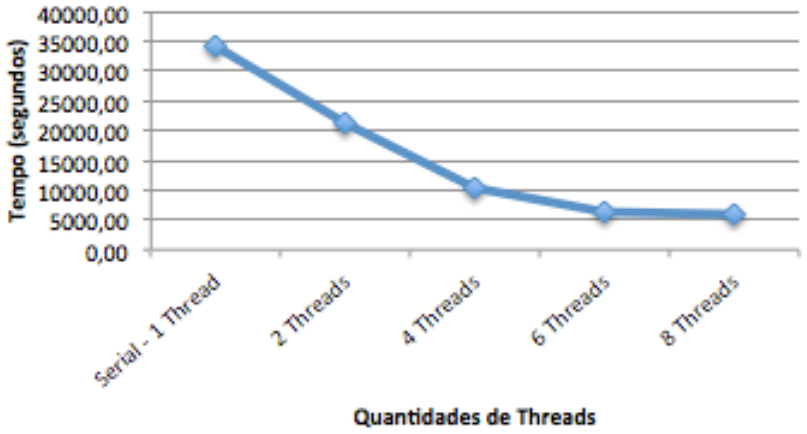

Figura 4.14: Comparação de Resultados Serial e Paralelo. Fonte: O Autor, 2013.

Esse teste comparou uma versão serial com sua versão paralela OpenMP, gerada pelo MCAP, utilizando 2, 4, 6 e 8 threads. Assim como ocorreu na versão híbrida em relação à versão MPI, a versão parelela com OpenMP apresentou tempos de processamento menores do que a versão serial, independente do número de *threads* utilizados na configuração da versão paralela. Nesse teste, o ganho de performance chegou a 83,01% quando utilizando 8 *threads*.

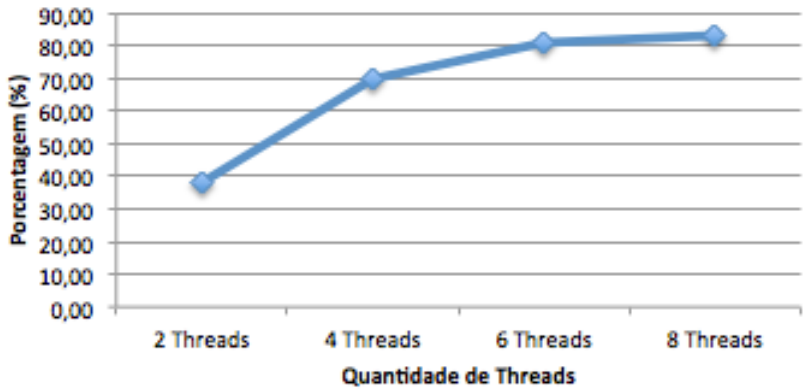

Figura 4.15: Ganho de Performance utilizando o MCAP. Fonte: O Autor, 2013.

Na figura 4.15, é apresentado o gráfico com o ganho de performance da versão paralela sobre a serial, no qual podemos notar que, à medida que se inserem novas *threads* na execução, o tempo de processamento diminui e, consequentemente, a performance da aplicação aumenta, isso porque cada nova *thread* criada ocupa um dos *cores* do nó que estava ocioso, aumentando assim o seu processamento paralelo.

Capítulo Cinco

Considerações Finais

Foi constatada uma redução de 73,18% no tempo de processamento de aplicações híbridas (MPI + OpenMP) geradas pelo MCAP em comparação com a aplicação MPI Pura original, que foi submetida à conversão. Este resultado foi obtido utilizando a configuração Padrão do MCAP, contendo as diretivas *reduction* e *schedule* do tipo *dynamic* do OpenMP. Desta forma, observa-se que o MCAP conseguiu atingir o objetivo inicialmente proposto de aumentar a performance de algoritmos paralelos, promovendo o paralelismo a nível de memória compartilhada, usando o OpenMP, em algoritmos escritos na linguagem C, que utilizam o paralelismo em memória distribuída, através do MPI, gerando assim um novo algoritmo híbrido (MPI + OpenMP), de forma automatizada e transparente para o usuário final.

Com o modelo proposto também foi possível gerar paralelismo em aplicações seriais, escritas na linguagem de programação C, utilizando o OpenMP, que também apresentou um ganho médio de performance em até 83,01% utilizando a configuração *schedule* do tipo *dynamic* e com o *private*. Tanto na versão híbrida quanto na versão paralela, o MCAP não modificou o conteúdo existente no código-fonte original enviado pelo usuário, preservando a integridade da aplicação. O ganho médio de performance obtido neste trabalho poderá ser incrementado pelo desenvolvedor através da otimização do balanceamento de carga enviado pelo MPI e na customização das diretivas de compilação do OpenMP, disponíveis na interface web.

Os métodos utilizados pelo MCAP para gerar paralelismo foram inspirados nos trabalhos apresentados no capítulo Estado da Arte. O que difere este trabalho daqueles pesquisados durante essa dissertação é o fato de gerar paralelismo em aplicações já paralelas e permitir que esse procedimento esteja disponível para qualquer usuário através da internet.

Os resultados analisados foram obtidos utilizando aplicações de multiplicação de matrizes, de tamanho 10.000 x 10.000, no modelo *master/worker*, em um *cluster multi-core*, seguindo todos os pré-requisitos estabelecidos pelo modelo. Não é garantido o ganho de performance em aplicações MPI com a utilização do MCAP num cenário diferente do que foi apresentado neste trabalho. Mesmo seguindo todas as orientações e pré-requisitos do MCAP, seu uso não garante ao usuário o aumento de performance de sua aplicação devido a fatores externos, como balanceamento de carga e configuração do *cluster*. Diferente dos trabalhos abordados no capítulo 2.1, o modelo proposto não foi implementado sob forma de compilador, dessa forma, o programador tem liberdade para escolher o seu compilador, independente da platadorma de desenvolvimento utilizada.

5.1 Conclusão

O modelo computacional proposto nesta pesquisa foi capaz de transformar uma aplicação MPI numa aplicação híbrida, com a inclusão automatizada do OpenMP, e, consequentemente, aumentar a performance dessa aplicação, demonstrando destarte a sua relevância no cenário científico da Computação de Alto Desempenho. O MCAP está disponível como um serviço na internet, através do endereço *http://andrecosta.info/mcap*, para qualquer usuário, a fim de que possam usufruir de seus benefícios e contribuições.

5.2 Trabalhos Futuros

Como trabalhos futuros, sugerimos as seguintes ações: a) Relizar os testes dos tópicos 4.2 e 4.2 na versão MPI Pura, com a mesma quantidade de processos da versão Híbrida; b) Inserir a opção "chucksize"para o parâmetro "schedule"do MCAP; c) Só permitir a opção "reduction"em parâmetros locais (privado); d) Customização do MCAP para funcionamento em outros tipos de aplicações, além de multiplicação de matrizes; e) Customização do MCAP para funcionamento em outros modelos de programação paralela, além do *master/worker*; f) Customização do MCAP para outras linguagens, tanto de origem (Ex.: JAVA e Pyton) quanto de destino (Ex.: OpenCL); g) Testes do modelo com diferentes tipos de algoritmos; h) Testes de escalonamento em *clusters* com um maior poder de processamento.

Apêndice A

Aplicação do MCAP

Figura A.1: Aplicação do MCAP. Fonte: O Autor, 2013.

Apêndice B

Resultados dos Testes Comparativos para Compor a Configuração "Padrão"do MCAP

Resultados dos Testes Comparativos (em segundos)			
dynamic	***static***	***guided***	***OMP***
1963,66	3987,50	3784,29	4049,45
2005,27	4057,23	3939,70	4076,77
2053,46	4172,94	4416,04	4088,88
2102,45	4610,05	4442,45	4142,99
2148,85	4675,68	4445,58	4594,76
private	***reduction***	***shared***	***4 threads***
7330,70	3194,72	4130,78	7306,87
7349,01	3243,67	4407,56	7660,42
7358,31	3249,39	4495,49	7671,66
7530,36	3352,44	4552,04	7736,08
7783,15	3499,34	4868,76	8465,45

Tabela B.1: Resultados das 5 execuções para cara algoritmo. Fonte: O Autor, 2013.

Resultados dos Testes Comparativos (em segundos)			
dynamic	***static***	***guided***	***OMP***
2005,27	4057,23	3939,70	4076,77
2053,46	4172,94	4416,04	4088,88
2102,45	4610,05	4442,45	4142,99
private	***reduction***	***shared***	***4 threads***
7349,01	3243,67	4407,56	7660,42
7358,31	3249,39	4495,49	7671,66
7530,36	3352,44	4552,04	7736,08

Tabela B.2: Resultados sem o maior e menor tempo de cada algoritmo. Fonte: O Autor, 2013.

Apêndice C

Resultados de Performance do MCAP

Processos	Resultados MPI Puro (Ordenado pelo tempo)				
1	21188,354752	21198,262521	21202,132832	21231,721800	21244,498502
2	10735,572870	10736,309567	10747,495021	10748,033570	10767,073143
3	7224,537529	7229,661970	7234,514239	7235,319949	7245,796452
4	5474,141969	5474,526734	5480,870150	5482,413359	5482,585126
5	4424,861094	4425,873781	4427,975790	4428,405800	4430,388468
6	3722,668459	3725,886310	3727,100404	3730,308841	4439,096146
7	3238,337676	3240,659562	3243,208471	3243,592523	3844,422800
8	2862,395236	2864,629304	2868,257028	2869,658165	3380,502095
Processos	**Resultados Híbrido (Ordenado pelo tempo)**				
1	4429,895265	4486,136579	4517,385393	4635,439885	5021,723779
2	2384,934886	2433,914897	2537,892949	2625,161629	2714,123745
3	1669,346319	1690,011425	1765,511283	1832,675064	1894,133228
4	1265,684123	1391,869081	1435,881681	1462,101220	1468,529899
5	1188,166109	1194,246584	1215,467759	1219,936089	1236,701057
6	964,267993	1021,178587	1025,368321	1049,179271	1371,806197
7	917,204720	920,045677	922,792780	958,025230	1358,088147
8	798,229715	829,580081	859,623461	891,608824	1108,686810

Tabela C.1: Resultado das 5 execuções entre o algoritmo híbrido e MPI. Fonte: O Autor, 2013.

Processos	Resultados MPI Puro (Ordenado pelo tempo)		
1	21198,262521	21202,132832	21231,721800
2	10736,309567	10747,495021	10748,033570
3	7229,661970	7234,514239	7235,319949
4	5474,526734	5480,870150	5482,413359
5	4425,873781	4427,975790	4428,405800
6	3725,886310	3727,100404	3730,308841
7	3240,659562	3243,208471	3243,592523
8	2864,629304	2868,257028	2869,658165
Processos	**Resultados Híbrido (Ordenado pelo tempo)**		
1	4486,136579	4517,385393	4635,439885
2	2433,914897	2537,892949	2625,161629
3	1690,011425	1765,511283	1832,675064
4	1391,869081	1435,881681	1462,101220
5	1194,246584	1215,467759	1219,936089
6	1021,178587	1025,368321	1049,179271
7	920,045677	922,792780	958,025230
8	829,580081	859,623461	891,608824

Tabela C.2: Resultado entre o algoritmo híbrido e MPI sem o menor e maior tempo. Fonte: O Autor, 2013.

Apêndice D

Gráfico com resultados comparativos MPI x Híbrido (MCAP)

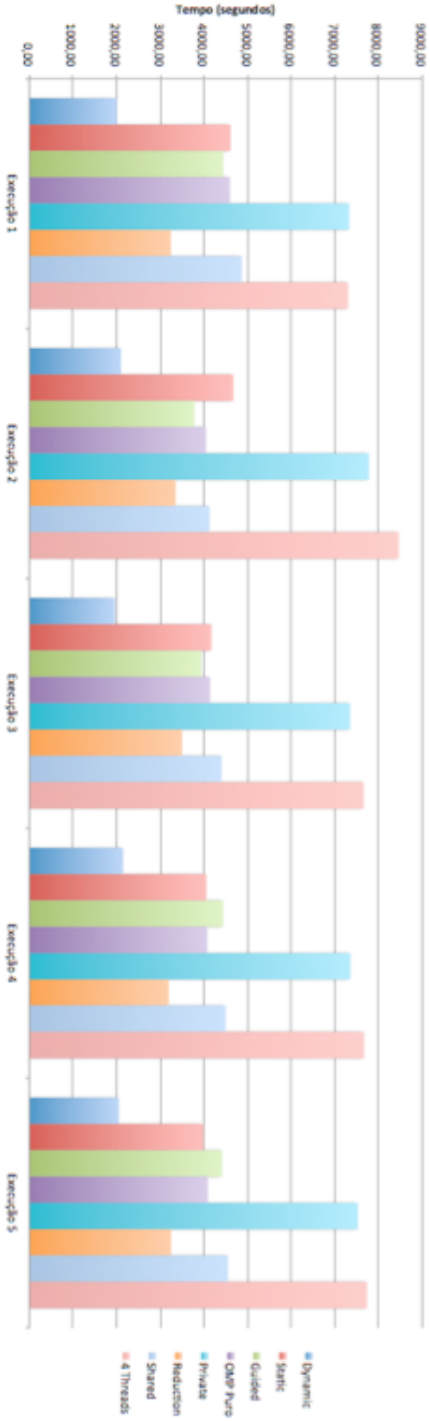

Figura D.1: Gráfico com resultados comparativos MPI x Híbrido (MCAP). Fonte: O Autor, 2013.

Apêndice E

Código de Multiplicação de Matrizes em MPI Puro

```
/*****************************************************************
* Francisco Almeida, Domingo Giménez, José Miguel Mantas, Antonio M. Vidal
* Introducción a la programación paralela,
* Paraninfo Cengage Learning, 2008
*
* Capítulo 6,
* Sección 6.3 Particionado de datos: Código 6.10
* Multiplicación de matrices por particionado de datos
*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <mpi.h>

// multiplicación de matrices secuencial
// por cada matriz aparece la zona de datos (a, b y c)
// y el número de filas, de columnas y el leading dimension
void mms(double *a, int fa, int ca, int lda, double *b, int fb, int cb, int ldb,
    double *c, int fc, int cc, int ldc) {
    int i, j, k;
    double s;
    for (i = 0; i < fa; i++)
        for (j = 0; j < cb; j++) {
            s = 0.;
            for (k = 0; k < ca; k++)
                s += a[i * lda + k] * b[k * ldb + j];
            c[i * ldc + j] = s;
        }
}

// nodo es un identificador del proceso
// y np el número total de procesos
void mm(double *a, int fa, int ca, int lda, double *b, int fb, int cb, int ldb,
    double *c, int fc, int cc, int ldc, int nodo, int np) {
    int i, j, k;
    double s;
    if (nodo == 0) {
        for (i = 1; i < np; i++)
            MPI_Send(&a[i * lda * fa / np], fa / np * ca, MPI_DOUBLE, i, 20,
                MPI_COMM_WORLD);
        MPI_Bcast(b, fb * cb, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    } else {
        MPI_Recv(a, fa / np * ca, MPI_DOUBLE, 0, 20, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
        MPI_Bcast(b, fb * cb, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    }
```

```
44      mms(a, fa / np, ca, lda, b, fb, cb, ldb, c, fc / np, cc, ldc);
45      if (nodo == 0)
46          for (i = 1; i < np; i++)
47              MPI_Recv(&c[i * ldc * fc / np],fc / np * cc, MPI_DOUBLE, i, 30,
                    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
48      else
49          MPI_Send(c, fc / np * cc, MPI_DOUBLE, 0, 30, MPI_COMM_WORLD);
50  }
51
52  /*
53  c
54  c initialize - random initialization for array
55  c
56  */
57
58  void initialize(double *m, int f, int c, int ld) {
59    int i, j;
60
61    for (i = 0; i < f; i++) {
62      for (j = 0; j < c; j++) {
63        m[i * ld + j] = (double)(i + j);
64      }
65    }
66  }
67
68  void initializealea(double *m, int f, int c, int ld) {
69    int i, j;
70
71    for (i = 0; i < f; i++) {
72      for (j = 0; j < c; j++) {
73        m[i * ld + j] = (double)rand() / RAND_MAX;
74      }
75    }
76  }
77
78  void escribir(double *m, int f, int c, int ld) {
79    int i, j;
80
81    for (i = 0; i < f; i++) {
82      for (j = 0; j < c; j++) {
83        printf("%.4lf ",m[i * ld + j]);
84      }
85      printf("\n");
86    }
87  }
88
89  void comparar(double *m1, int fm1, int cm1, int ldm1, double *m2, int fm2, int cm2
        , int ldm2)
90  {
91    int i, j;
92
93    for(i = 0; i < fm1; i++)
```

```
94      for(j = 0; j < cm1; j++) {
95        if(m1[i * ldm1 + j] != m2[i * ldm2 + j]) {
96          printf("Diferencia en %d,%d: %.8lf , %.8lf\n", i, j, m1[i * ldm1 + j], m2[
                i * ldm2 + j]);
97          return;
98        }
99      }
100 }
101
102 int main(int argc, char *argv[]) {
103   int nodo, np, i, j, fa, fal, ca, lda, fb, cb, ldb, fc, fcl, cc, ldc, N;
104   int long_name;
105   double ti, tf;
106   double *a, *b, *c, *c0;
107   char     nombre_procesador[MPI_MAX_PROCESSOR_NAME];
108   MPI_Status estado;
109
110   MPI_Init(&argc, &argv);
111   MPI_Comm_size(MPI_COMM_WORLD, &np);
112   MPI_Comm_rank(MPI_COMM_WORLD, &nodo);
113   MPI_Get_processor_name(nombre_procesador, &long_name);
114
115 // Se ejecuta con mpirun -np numeroprocesos ejecutable tamañomatriz
116
117   if (nodo == 0) {
118     N = atoi(argv[1]);
119   }
120   MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
121
122   fa = ca = lda = fb = cb = ldb = fc = cc = ldc = N;
123   fal = N / np;
124   fcl = N / np;
125   if (nodo == 0) {
126     a = (double *) malloc(sizeof(double) * fa * ca);
127     b = (double *) malloc(sizeof(double) * fb * cb);
128     c = (double *) malloc(sizeof(double) * fc * cc);
129   } else {
130     a = (double *) malloc(sizeof(double) * fal * ca);
131     b = (double *) malloc(sizeof(double) * fb * cb);
132     c = (double *) malloc(sizeof(double) * fcl * cc);
133   }
134
135   if (nodo == 0) {
136     c0 = (double *) malloc(sizeof(double) * fc * cc);
137     initialize(a, fa, ca, lda);
138     initialize(b, fb, cb, ldb);
139
140     mms(a, fa, ca, lda, b, fb, cb, ldb, c0, fc, cc, ldc);
141   }
142
143   MPI_Barrier(MPI_COMM_WORLD);
144
```

```
145     ti = MPI_Wtime();
146
147     mm(a, fa, ca, lda, b, fb, cb, ldb, c, fc, cc, ldc, nodo, np);
148
149     MPI_Barrier(MPI_COMM_WORLD);
150     tf = MPI_Wtime();
151     if (nodo == 0) {
152       printf("Proceso %d, %s, Tiempo %.6lf\n", nodo, nombre_procesador, tf - ti);
153       comparar(c, fc, cc, ldc, c0, fc, cc, ldc);
154     }
155
156     free(a);
157     free(b);
158     free(c);
159     if (nodo == 0)
160       free(c0);
161     MPI_Finalize();
162 }
```

Codigos/algoritmo_mpi.c

Apêndice F

Código de Multiplicação de Matrizes Híbrido (MPI + OpenMP) Gerado pelo MCAP

```
/**#************************************
 **#**
 **#** Código modificado pelo #MCAP
 **#** Link: http://andrecosta.info/mcap
 **#** Versão: 1
 **#** Modificado em: 11/08/2013 às 04:33:34
 **#** Configuração: Padrão (Schedule: dynamic + Reduction)
 **#**
 **#************************************/

#include <omp.h>

/*****************************************************************
* Francisco Almeida, Domingo Giménez, José Miguel Mantas, Antonio M. Vidal
* Introducción a la programación paralela,
* Paraninfo Cengage Learning, 2008
*
* Capítulo 6,
* Sección 6.3 Particionado de datos: Código 6.10
* Multiplicación de matrices por particionado de datos
*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <mpi.h>

// multiplicación de matrices secuencial
// por cada matriz aparece la zona de datos (a, b y c)
// y el número de filas, de columnas y el leading dimension
void mms(double *a, int fa, int ca, int lda, double *b, int fb, int cb, int ldb,
    double *c, int fc, int cc, int ldc) {
    int i, j, k;
    double s;
    #pragma omp for schedule( dynamic ) reduction( +:s )
    for (i = 0; i < fa; i++) {
        for (j = 0; j < cb; j++) {
            s = 0.;
            for (k = 0; k < ca; k++)
                s += a[i * lda + k] * b[k * ldb + j];
            c[i * ldc + j] = s;
        }
    }
}

```

```
45 // nodo es un identificador del proceso
46 // y np el número total de procesos
47 void mm(double *a, int fa, int ca, int lda, double *b, int fb, int cb, int ldb,
       double *c, int fc, int cc, int ldc, int nodo, int np) {
48     int i, j, k;
49     double s;
50     if (nodo == 0) {
51         for (i = 1; i < np; i++)
52             MPI_Send(&a[i * lda * fa / np], fa / np * ca, MPI_DOUBLE, i, 20,
                   MPI_COMM_WORLD);
53         MPI_Bcast(b, fb * cb, MPI_DOUBLE, 0, MPI_COMM_WORLD);
54     } else {
55         MPI_Recv(a, fa / np * ca, MPI_DOUBLE, 0, 20, MPI_COMM_WORLD,
               MPI_STATUS_IGNORE);
56         MPI_Bcast(b, fb * cb, MPI_DOUBLE, 0, MPI_COMM_WORLD);
57     }
58     mms(a, fa / np, ca, lda, b, fb, cb, ldb, c, fc / np, cc, ldc);
59     if (nodo == 0)
60         for (i = 1; i < np; i++)
61             MPI_Recv(&c[i * ldc * fc / np],fc / np * cc, MPI_DOUBLE, i, 30,
                   MPI_COMM_WORLD, MPI_STATUS_IGNORE);
62     else
63         MPI_Send(c, fc / np * cc, MPI_DOUBLE, 0, 30, MPI_COMM_WORLD);
64 }
65
66 /*
67 c
68 c initialize - random initialization for array
69 c
70 */
71
72 void initialize(double *m, int f, int c, int ld) {
73   int i, j;
74
75   #pragma omp for schedule( dynamic )
76   for (i = 0; i < f; i++) {
77     for (j = 0; j < c; j++) {
78       m[i * ld + j] = (double)(i + j);
79     }
80   }
81 }
82
83 void initializealea(double *m, int f, int c, int ld) {
84   int i, j;
85
86   #pragma omp for schedule( dynamic )
87   for (i = 0; i < f; i++) {
88     for (j = 0; j < c; j++) {
89       m[i * ld + j] = (double)rand() / RAND_MAX;
90     }
91   }
92 }
```

```
93
94  void escribir(double *m, int f, int c, int ld) {
95    int i, j;
96
97    for (i = 0; i < f; i++) {
98      for (j = 0; j < c; j++) {
99        printf("%.4lf ",m[i * ld + j]);
100     }
101     printf("\n");
102   }
103 }
104
105 void comparar(double *m1, int fm1, int cm1, int ldm1, double *m2, int fm2, int cm2
        , int ldm2)
106 {
107   int i, j;
108
109   for(i = 0; i < fm1; i++)
110     for(j = 0; j < cm1; j++) {
111       if(m1[i * ldm1 + j] != m2[i * ldm2 + j]) {
112         printf("Diferencia en %d,%d: %.8lf , %.8lf\n", i, j, m1[i * ldm1 + j], m2[
              i * ldm2 + j]);
113         return;
114       }
115     }
116 }
117
118 int main(int argc, char *argv[]) {
119   int nodo, np, i, j, fa, fal, ca, lda, fb, cb, ldb, fc, fcl, cc, ldc, N;
120   int long_name;
121   double ti, tf;
122   double *a, *b, *c, *c0;
123   char     nombre_procesador[MPI_MAX_PROCESSOR_NAME];
124   MPI_Status estado;
125
126   MPI_Init(&argc, &argv);
127   MPI_Comm_size(MPI_COMM_WORLD, &np);
128   MPI_Comm_rank(MPI_COMM_WORLD, &nodo);
129   MPI_Get_processor_name(nombre_procesador, &long_name);
130
131 // Se ejecuta con mpirun -np numeroprocesos ejecutable tamañomatriz
132
133   if (nodo == 0) {
134     N = atoi(argv[1]);
135   }
136   MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
137
138   fa = ca = lda = fb = cb = ldb = fc = cc = ldc = N;
139   fal = N / np;
140   fcl = N / np;
141   if (nodo == 0) {
142     a = (double *) malloc(sizeof(double) * fa * ca);
```

```
143     b = (double *) malloc(sizeof(double) * fb * cb);
144     c = (double *) malloc(sizeof(double) * fc * cc);
145   } else {
146     a = (double *) malloc(sizeof(double) * fal * ca);
147     b = (double *) malloc(sizeof(double) * fb * cb);
148     c = (double *) malloc(sizeof(double) * fcl * cc);
149   }
150
151   if (nodo == 0) {
152     c0 = (double *) malloc(sizeof(double) * fc * cc);
153     initialize(a, fa, ca, lda);
154     initialize(b, fb, cb, ldb);
155
156     mms(a, fa, ca, lda, b, fb, cb, ldb, c0, fc, cc, ldc);
157   }
158
159   MPI_Barrier(MPI_COMM_WORLD);
160
161   ti = MPI_Wtime();
162
163   mm(a, fa, ca, lda, b, fb, cb, ldb, c, fc, cc, ldc, nodo, np);
164
165   MPI_Barrier(MPI_COMM_WORLD);
166   tf = MPI_Wtime();
167   if (nodo == 0) {
168     printf("Proceso %d, %s, Tiempo %.6lf\n", nodo, nombre_procesador, tf - ti);
169     comparar(c, fc, cc, ldc, c0, fc, cc, ldc);
170   }
171
172   free(a);
173   free(b);
174   free(c);
175   if (nodo == 0)
176     free(c0);
177   MPI_Finalize();
178 }
```

Codigos/algoritmo_hibrido_by_MCAP.c

Apêndice G

Código do MCAP: MODEL

```php
<?php

class MCAP {

    // inicializa variáveis
    protected $aAbreChaves = array();
    protected $chaves = array();
    protected $arquivo = array( "" );

    protected $private = "";
    protected $reduction = "";

    function __construct() {

        // Evita erro de exceder o tempo padrao de 300 segundos processando
  ini_set('max_execution_time','300');

        $this->getCodigoEnviado( $_FILES['codigo_mpi']['tmp_name'] );

    }

    function getCodigoEnviado( $nomeArquivo ) {

        // salva arquivo original
        move_uploaded_file( $nomeArquivo, "codigo_hibrido.c" );

    }

    function checkIsNotParallelOMP() {

        // Testa se é uma linha com instrução do MCAP ou OMP
        if (( count( explode( "**#**", $conteudoLinha )) <= 1 ) &&
            ( count( explode( "<omp.h>", $conteudoLinha )) <= 1 ) &&
            ( count( explode( "#pragma", $conteudoLinha )) <= 1 ) &&
            ( count( explode( "omp_set_num_threads", $conteudoLinha )) <= 1 )) {

            return true;

        }

        return false;

    }

    function getComando( $conteudoLinha, $linha ) {

```

```
47          // Indentifica o comando (função) que ativa o abre chave, se for um "FOR"
                deixa como paralelizável
48          //
49          if ( count( explode( "for", strtolower( $conteudoLinha ))) > 1 ) {
50
51              $this->chaves[ $linha ]["comando"] = "for";
52              $this->chaves[ $linha ]["identacao"] = strpos( $conteudoLinha, "for" )
                    ;
53              $this->chaves[ $linha ][ "paralelizavel" ] = 1;
54
55          } else if ( count( explode( "while", strtolower( $conteudoLinha ))) > 1 )
                {
56
57              $this->chaves[ $linha ]["comando"] = "while";
58              $this->chaves[ $linha ][ "paralelizavel" ] = 0;
59
60          } else if ( count( explode( "do", strtolower( $conteudoLinha ))) > 1 ) {
61
62              $this->chaves[ $linha ]["comando"] = "do";
63              $this->chaves[ $linha ][ "paralelizavel" ] = 0;
64
65          } else if ( count( explode( "if", strtolower( $conteudoLinha ))) > 1 ) {
66
67              $this->chaves[ $linha ]["comando"] = "if";
68              $this->chaves[ $linha ][ "paralelizavel" ] = 0;
69
70          } else if ( count( explode( "else", strtolower( $conteudoLinha ))) > 1 ) {
71
72              $this->chaves[ $linha ]["comando"] = "else";
73              $this->chaves[ $linha ][ "paralelizavel" ] = 0;
74
75          } else if ( count( explode( "main", strtolower( $conteudoLinha ))) > 1 ) {
76
77              $this->chaves[ $linha ]["comando"] = "main";
78              $this->chaves[ $linha ][ "paralelizavel" ] = 0;
79
80          } else {
81
82              $this->chaves[ $linha ]["comando"] = "outro";
83              $this->chaves[ $linha ][ "paralelizavel" ] = 0;
84
85
86          }
87
88      }
89
90      function obterLoops() {
91
92          // abre arquivo
93          $codigo = fopen ("codigo_hibrido.c", "r");
94          $linha = 0;
95
```

```
96          // percorre linhas do arquivo enviado
97          while (!feof ($codigo)) {
98
99              // obtem o código da linha
100             $conteudoLinha = fgets( $codigo );
101
102             if ( $this->checkIsNotParallelOMP() ) {
103
104
105                 // cola o conteúdo da linha no array $arquivo
106                 array_push($this->arquivo, $conteudoLinha );
107
108                 // obtem número da linha atual
109                 $linha++;
110
111             }
112
113             // Obtem Fecha Chave, Associando ao seu Abre Chave
114             // IMPORTANTE: tem que vir antes de obter abre chave por causa do '} else {'
115             if ( count( explode( "}", $conteudoLinha )) > 1 ) {
116
117                 // guarda a linha onde é fechada a chave da do último abre chave em aberto
118                 $this->chaves[ array_pop( $this->aAbreChaves ) ]["fecha"] = $linha;
119
120             }
121
122             // Obtem Abre Chave
123             if ( count( explode( "{", $conteudoLinha )) > 1 ) {
124
125                 // iniciar propriedades para o abre chave encontrado
126                 $this->chaves[ $linha ] = array(
127                                                 "abre" => $linha,
128                                                 "fecha" => 0,
129                                                 "comando" => "",
130                                                 "identacao" => 0,
131                                                 "paralelizavel" => 0
132                                             );
133
134                 // informa a linha onde foi encontrado o abre chaves
135                 array_push( $this->aAbreChaves, $linha );
136
137                 $this->getComando( $conteudoLinha, $linha );
138
139             }
140
141         }
142
143         // fecha algoritmo original sem modificá-lo
144         fclose( $codigo );
```

```
145
146     }
147
148     function checarLoopsParalelizaveis() {
149
150         // Percorre todas as chaves abertas
151         foreach ( $this->chaves as $item ) {
152
153             // identifica os as chaves abertas pelo comando "FOR"
154             if ( $item["comando"] == "for" ) {
155
156                 // percorre o códido do comando (do bre chave até fecha chave)
157                 for ( $i = $item["abre"]; $i < $item["fecha"]; $i++ ) {
158
159                     // Se encontrar comando MPI no trecho de código do comando "
                            FOR" coloca como não paralelizável
160                     if ( strpos( $arquivo[ $i ], "MPI_" )) {
161
162                         $this->chaves[ $item["abre"] ]["paralelizavel"] = 0;
163                         break;
164
165                     }
166
167                     // Se encontrar comando PRINT ou PINRTF no trecho de código do
                            comando "FOR" coloca como não paralelizável
168                     if ( strpos( $arquivo[ $i ], strtolower( "print" )) || strpos(
                            $arquivo[ $i ], strtolower( "printf" )) ) {
169
170                         $this->chaves[ $item["abre"] ]["paralelizavel"] = 0;
171                         break;
172
173                     }
174
175                 }
176
177                 // se o "FOR" estiver dentro de outro "FOR", coloca o filho como
                        não paralelizável
178                 foreach ( $this->chaves as $item_pai ) {
179
180                     if ( $item_pai["abre"] < $item["abre"] && $item_pai["fecha"] >
                            $item["fecha"] ) {
181
182                         if ( $item_pai["comando"] == "for" && $item_pai["
                                paralelizavel"] == 1 ) {
183
184                             $this->chaves[ $item["abre"] ]["paralelizavel"] = 0;
185                             break;
186
187                         }
188
189                     }
190
```

```
191                 }
192
193             }
194
195         }
196
197     }
198
199     function checarDiretivasOMP( $item ) {
200
201         // função main?
202         if ( $item["comando"] == "main" ) {
203
204             // usuário definiu quantidade de threads?
205             if ( $_POST["num_threads"] != "" ) {
206
207                 $linha = $item["abre"];
208                 $arquivo[ $linha . ".1" ] = "\n";
209                 $arquivo[ $linha . ".2" ] = "omp_set_num_threads( ". $_POST["
                        num_threads"] ." ); \n";
210                 $arquivo[ $linha . ".3" ] = "\n";
211
212             }
213
214         }
215
216         // identifica os as chaves abertas pelo comando "FOR"
217         if ( $item["comando"] == "for" && $item["paralelizavel"] == 1 ) {
218
219             // percorre o códido do comando (do bre chave até fecha chave)
220             for ( $i = $item["abre"]; $i < $item["fecha"]; $i++ ) {
221
222                 // verifica se já foi criado a diretiva "private"
223                 if ( $this->private == "" ) {
224
225                     // verifica se é a linha do comando "FOR"
226                     if ( strpos( $arquivo[ $i ], strtolower( "for" ) )) {
227
228                         // verifica se utiliza incremento no for
229                         if ( strpos( $arquivo[ $i ], strtolower( "++" ) )) {
230
231                             // coloca a variável do incremento como privada
232                             $this->private = substr( $arquivo[ $i ], 0, strpos(
                                    $arquivo[ $i ], strtolower( "++" )) );
233                             $this->private = substr( $this->private, strrpos(
                                    $arquivo[ $i ], ";" ) +1 );
234                             $this->private = trim( $this->private );
235
236                         }
237
238                         // verifica se utiliza decremento no for
239                         if ( strpos( $arquivo[ $i ], strtolower( "--" ) )) {
```

```
240
241                         // coloca a variável do decremento como privada
242                         $this->private = substr( $arquivo[ $i ], 0, strpos(
                                $arquivo[ $i ], strtolower( "--" )) );
243                         $this->private = substr( $this->private, strrpos(
                                $arquivo[ $i ], ";" ) +1 );
244                         $this->private = trim( $this->private );
245
246                     }
247
248                 }
249
250             }
251
252             // verifica se já foi criado a diretiva "reduction"
253             if ( $this->reduction == "" ) {
254
255                 // identifica confição para o reduction
256                 if ( strpos( $arquivo[ $i ], strtolower( "+=" ) )) {
257
258                     $this->reduction = substr( $arquivo[ $i ], 0, strpos(
                            $arquivo[ $i ], strtolower( "+=" )) );
259                     $this->reduction = "+:" . trim( $this->reduction );
260
261                 }
262
263                 // identifica confição para o reduction
264                 if ( strpos( $arquivo[ $i ], strtolower( "-=" ) )) {
265
266                     $this->reduction = substr( $arquivo[ $i ], 0, strpos(
                            $arquivo[ $i ], strtolower( "-=" )) );
267                     $this->reduction = "-:" . trim( $this->reduction );
268
269                 }
270
271                 // identifica confição para o reduction
272                 if ( strpos( $arquivo[ $i ], strtolower( "*=" ) )) {
273
274                     $this->reduction = substr( $arquivo[ $i ], 0, strpos(
                            $arquivo[ $i ], strtolower( "*=" )) );
275                     $this->reduction = "*:" . trim( $this->reduction );
276
277                 }
278
279                 // identifica confição para o reduction
280                 if ( strpos( $arquivo[ $i ], strtolower( "/=" ) )) {
281
282                     $this->reduction = substr( $arquivo[ $i ], 0, strpos(
                            $arquivo[ $i ], strtolower( "/=" )) );
283                     $this->reduction = "/:" . trim( $this->reduction );
284
285                 }
```

```
286
287                 }
288
289             }
290
291         }
292
293     }
294
295     function inserirDiretivasOMP( $item ) {
296
297         // se o conteúdo entre as chaves for paralelizável, adiciona o código
                OpenMP
298         if ( $this->chaves[ $item["abre"] ]["paralelizavel"] ) {
299
300             $identacao = "";
301             for ( $i = 0; $i < $chaves[ $item["abre"] ]["identacao"]; $i++ )
302                 $identacao .= " ";
303
304             $diretiva = $identacao . "#pragma omp parallel for ";
305
306             if ( $_POST["configuracao"] == "padrao" ) {
307
308                 $diretiva .= "schedule( dynamic ) ";
309
310                 if ( $this->reduction != "" )
311                     $diretiva .= "reduction( ". $reduction . " ) ";
312
313             } else if ( $_POST["configuracao"] == "customizado" ) {
314
315                 if ( $_POST["schedule"] != "nenhum" )
316                     $diretiva .= "schedule( ". $_POST["schedule"] ." ) ";
317
318                 if ( $this->private != "" )
319                     $diretiva .= "private( ". $private . " ) ";
320
321                 if ( $this->reduction != "" )
322                     $diretiva .= "reduction( ". $reduction . " ) ";
323
324                 $diretiva .= "default( shared ) ";
325
326             }
327
328             // adiciona a diretiva logo acima do comando do abre chave
329             $linha = $item["abre"] - 1;
330             $this->arquivo[ $linha . ".1" ] = $diretiva . "\n";
331
332             $this->private = "";
333             $this->reduction = "";
334
335         }
336
```

```
337     }
338
339     function gerarNovoCodigo() {
340
341         if ( $_POST["configuracao"] == "basico" ) {
342
343             $config = "Básico";
344
345
346         } else if ( $_POST["configuracao"] == "padrao" ) {
347
348             $config = "Padrão (Schedule: dynamic + Reduction)";
349
350         } else if ( $_POST["configuracao"] == "customizado" ) {
351
352             $param = "";
353
354             if ( $_POST["schedule"] ) $param .= " Schedule: " . $_POST["schedule"
                    ];
355             if ( $_POST["private"] ) $param .= " + Private";
356             if ( $_POST["reduction"] ) $param .= " + Reduction";
357             if ( $_POST["shared"] ) $param .= " + Shared";
358             if ( $_POST["num_thread"] != "" ) $param .= " + Threads: " . $_POST["
                    num_thread"];
359
360             $config = "Customizado ($param)";
361
362
363         }
364
365         // insere cabeçalho do #MCAP
366         $this->arquivo[ "0.1" ] = "/**#************************************ \n";
367         $this->arquivo[ "0.2" ] = " **#** \n";
368         $this->arquivo[ "0.3" ] = " **#** Código modificado pelo #MCAP \n";
369         $this->arquivo[ "0.4" ] = " **#** Link: http://andrecosta.info/mcap \n";
370         $this->arquivo[ "0.5" ] = " **#** Versão: 1 \n";
371         $this->arquivo[ "0.6" ] = " **#** Modificado em: " . date("d/m/Y") . " às
                " . date("H:i:s") . " \n";
372         $this->arquivo[ "0.7" ] = " **#** Configuração: $config \n";
373         $this->arquivo[ "0.8" ] = " **#** \n";
374         $this->arquivo[ "0.9" ] = " **#************************************/ \n\n
                ";
375
376         // insere biblioteca do OpenMP na primeira linha
377         $this->arquivo[ "0.99" ] = "#include <omp.h> \n\n";
378
379         // ordena arquivo com as novas linhas de código
380         ksort( $this->arquivo );
381
382         // abre o algoritmo original
383         $codigo = fopen ("codigo_hibrido.c", "w");
384
```

```
385         // subistitui seu conteúdo pela nova versão
386         foreach ( $this->arquivo as $linhaArquivo )
387             fwrite( $codigo, $linhaArquivo );
388 
389         // fecha algoritmo
390         fclose( $codigo );
391 
392 
393         // insere metadados do arquivo e gera arquivo para download
394         $this->arquivo = "codigo_hibrido.c";
395         header("Content-Type: ");
396         header("Content-Length: ".filesize($arquivo));
397         header("Content-Disposition: attachment; filename=".basename($this->
                arquivo));
398         readfile($this->arquivo);
399         exit;
400 
401     }
402 
403     function gerarParalelismoOMP() {
404 
405         // Itendifica todos os loops
406         $this->obterLoops();
407 
408         // Analisa quais loops são paralelizáveis
409         $this->checarLoopsParalelizaveis();
410 
411         // Percorre todas os loops paralelizáveis
412         foreach ( $this->chaves as $item ) {
413 
414             // Identifica as diretivas OMP de cada loop
415             $this->checarDiretivasOMP( $item );
416 
417             // Insere as diretivas OMP
418             $this->inserirDiretivasOMP( $item );
419 
420         }
421 
422         // Gera o novo algorítimo híbrido
423         $this->gerarNovoCodigo();
424 
425     }
426 
427 }
428 
429 ?>
```

Codigos/class.MCAP.php

Apêndice H

Código do MCAP: VIEW

```
<?php

// CONTROLLER
include_once 'request.MCAP.php';

?>

<!-- VIEW -->
<html>
  <head>
    <meta charset="utf-8">
    <title>MCAP :: Modelo Computacional de Auto-Paralelismo</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="">
    <meta name="author" content="">

    <!-- Le styles -->
    <link href="css/bootstrap.css" rel="stylesheet">
    <style type="text/css">
      body {
          padding-top: 41px;
          padding-bottom: 40px;
      }
      .hero-unit h1 {
          font-size: 36px;
      }
      .hero-unit {
          padding:40px;
          margin-top:20px;
          text-align: justify;
      }
      .span10 {
          text-align: justify;
      }
      #div_customizado {
          width:430px;
          background-image:url(img/bg_customizado.png);
          padding:10px;
          padding-top:18px;
          display:none;
      }
    </style>
    <link href="css/bootstrap-responsive.css" rel="stylesheet">
    <link href="css/dropzone.css" rel="stylesheet">

    <link rel="shortcut icon" href="img/favicon.png">

```

```
48  </head>
49
50  <body>
51
52    <div class="navbar navbar-inverse navbar-fixed-top">
53      <div class="navbar-inner">
54        <div class="container">
55          <button type="button" class="btn btn-navbar" data-toggle="collapse" data
                -target=".nav-collapse">
56            <span class="icon-bar"></span>
57            <span class="icon-bar"></span>
58            <span class="icon-bar"></span>
59          </button>
60          <a class="brand" href="#">#MCAP <small><i>beta</i></small></a>
61          <div class="nav-collapse collapse">
62            <ul class="nav">
63              <li class="active"><a href="#upload">Gerar Código OpenMP</a></li>
64              <li><a href="#sobre">O que é?</a></li>
65              <li><a href="#tutorial">Como Funciona?</a></li>
66              <li><a href="#autores">Autores</a></li>
67              <li><a href="#publicacoes">Publicações</a></li>
68            </ul>
69          </div>
70        </div>
71      </div>
72    </div>
73
74
75  <div class="jumbotron subhead" id="overview" style="background-image:url(img/
        bj_upload.jpeg);">
76  <div class="container">
77        <div class="hero-unit">
78        <center><h1>#MCAP - Modelo Computacional de Auto-Paralelismo</h1></center>
79        <br />
80        <p>Quer melhorar a performance de sua aplicação? Então, envie seu
              algoritmo serial para transformá-lo numa versão paralela, utilizando o
               OpenMP, ou envie seu algoritmo MPI para gerar uma versão híbrida (MPI
               + OpenMP).</p>
81        <br />
82        <center>
83
84        <form method="POST" enctype="multipart/form-data" >
85
86        <input type="file" name="codigo_mpi" title="Selecione Seu Código: *.c"
              onclick="$('.file-input-name').html('');">
87        <br /><br />
88        <small><strong>Configuração do OpenMP: </strong></small><br />
89
90
91                <label class="radio inline">
92                  <input type="radio" name="configuracao" id="configuracao_basico"
```

```
93                    onClick="document.getElementById('div_customizado').style.
                          display = 'none';" value="basico">
94                  Básico
95                </label>
96
97                <label class="radio inline">
98                  <input type="radio" name="configuracao" id="configuracao_padrao"
99                    onClick="document.getElementById('div_customizado').style.
                          display = 'none';" value="padrao" checked >
100                 Padrão (recomendado)
101               </label>
102
103               <label class="radio inline">
104                 <input type="radio" name="configuracao" id="
                        configuracao_customizado"
105                   onClick="document.getElementById('div_customizado').style.
                          display = 'block';" value="customizado">
106                 Customizado
107               </label>
108
109               <div id="div_customizado">
110                 <label class="radio inline"><i>Schedule</i>:</label>
111
112                   <label class="radio inline">
113                     <input type="radio" name="schedule" value="nenhum">
114                     Nenhum
115                   </label>
116
117                   <label class="radio inline">
118                     <input type="radio" name="schedule" value="static">
119                     <i>Static</i>
120                   </label>
121
122                   <label class="radio inline">
123                     <input type="radio" name="schedule" value="dynamic" checked
                            >
124                     <i>Dynamic</i>
125                   </label>
126
127                   <label class="radio inline">
128                     <input type="radio" name="schedule" value="guided">
129                     <i>Guided</i>
130                   </label>
131
132                   <br />
133
134                   <label class="inline checkbox">
135                     <input type="checkbox" name="private"> <i>Private</i>
136                   </label>
137
138                   <label class="inline checkbox">
```

```
139                       <input type="checkbox" name="reduction" checked > <i>
                              Reduction</i>
140                     </label>
141
142                     <label class="inline checkbox">
143                       <input type="checkbox" name="shared"> <i>Default Shared</i>
144                     </label>
145
146                     <br />
147
148                     <label class="radio inline">* Número de <i>Threads</i>:</label
                            >
149                     <input type="text" name="num_threads" style="width:20px;">
150                     <br />
151                     <small><i>* Deixar em branco para ativar uma therad em cada
                            núcleo do(s) processador(es).</i></small>
152
153                   </div>
154
155                   <div style="margin-top:20px;">
156                           <input type="hidden" name="acao" value="enviar" />
157                           <input type="submit" class="btn-large btn-primary" value="
                                Gerar Paralelismo">
158                   </div>
159
160         </form>
161
162         <small><b>Importante:</b> Após gerar a versão híbrida, inclua o parâmetro
                "<font color="red">-fopenmp</font>" <br />quando for compilar sua
                aplicação, para utilizar os benefícios do OpenMP.</small>
163
164         </center>
165         </div>
166   </div>
167   </div>
168
169
170 <div class="container" id="sobre">
171       <div class="row">
172         <div class="span10">
173           <h2>O que é?</h2>
174           <p>
175               O #MCAP é um Modelo Computacional de Auto-Paralelismo, cujo objetivo
                      é aumentar a performance de aplicações paralelas que utilizam
                    MPI em ambientes multi-cores. Isso ocorre através da geração
                    automatizada de paralelismo em memória compartilhada, utilizando
                     a biblioteca OpenMP. Dessa forma, o MCAP transforma algoritmos
                    MPI em uma versão híbrida, utilizando o MPI e OpenMP, assim, o
                    novo algoritmo, além de explorar o paralelismo com a divisão do
                    processamento entre as máquinas do cluster, também irá usufruir
                    do paralelismo gerado pelo OpenMP, dividindo o processamento
                    enviado para cada nó entre os seus respectivos núcleos.
```

```
176                 <br />
177                 <center>
178                     <img src="img/mcap.png" alt="Fluxo do MCAP" />
179                     <br />
180                     <small>Fonte: COSTA e SOUZA, 2013.</small>
181                 </center>
182             </p>
183           </div>
184         </div>
185 </div>
186
187   <div class="jumbotron subhead" id="overview" style="background-color:rgb(231,
        231, 231);">
188     <div class="container" id="tutorial">
189       <div class="row">
190         <div class="span10">
191           <h2>Como Funciona?</h2>
192           <p>
193               Para gerar paralelismo em seu algoritmo, basta selecionar seu código
                    -fonte e clicar no botão "Gerar Paralelismo", mas para que o #
                    MCAP funcione de forma correta, é necessário que seu algoritmo
                    siga alguns pré-requisitos, são eles:
194               <br />
195               <ul>
196                   <li>O algoritmo precisa estar escrito na linguagem de
                        programação C e consequentemente estar salvo na extensão *.c
                        , além de não conter erros de sintaxe;</li>
197                   <li>A aplicação paralela deverá ser serial ou utilizar apenas
                        MPI;</li>
198                   <li>Se utilizar MPI, o algoritmo deverá utilizar o conceito de
                        multiplicação de matrizes, juntamente com o modelo Master/
                        Worker. O MCAP foi validado apenas em aplicações com estas
                        configurações, e por isso não garante sua eficiência e
                        algoritmos que não sigam este padrão;</li>
199                   <li>É restritamente necessário que o abre chaves ({) seja feita
                        sempre na mesma linha da função a qual ela pertence, e nunca
                         na linha de baixo;</li>
200               </ul>
201               <br />
202               Veja no exemplo abaixo uma estrutura de código aceito pelo #MCAP:
203               <br />
204               <pre>
205 #include < stdio.h>
206
207 int main() {
208
209     int i = 0, max = 100;
210
211     for ( i = 0; i < max; i++ ) {
212
213         pinrtln( "%d", i );
214
```

```
215     }
216
217     return 0;
218
219 }
220                 </pre>
221                 <br />
222                 <b>Importante:</b>
223                 <br />
224                 O #MCAP só paraleliza <i>loops</i> criados a partir do comando <b>
                        for</b>, dessa forma, verifique se seu algoritimo utiliza outros
                         comandos nos laços de repetição (Ex.: while, do while, etc.) e
                        transforme-o para "for", dessa forma, você conseguirá extrair
                        uma melhor performance do OpenMP gerado no seu algoritmo; Esta
                        aplicação trabalha apenas com 1 único algoritmo por vez.
225             </p>
226           </div>
227         </div>
228         </div>
229 </div>
230
231     <div class="container" id="autores">
232       <div class="row">
233         <div class="span10">
234           <h2>Autores</h2>
235           <p>
236           <table width="100%">
237               <tr>
238                   <td valign="top"><img src="img/andre.png" alt="Foto de André"
                        /></td>
239                   <td width="10px"> </td>
240                   <td width="300px" valign="top" align="left">
241                       <b>André Luiz Lima da Costa</b>
242                       <br />
243                       Mestrando em Modelagem Computacional e Tecnologia Industrial
                             pelo SENAI CIMATEC e MBA em Gestão de TI e Business
                            Intelligence pela UNIFACS.
244                       <br />
245                       <a href="http://www.andrecosta.info/">http://www.andrecosta.
                            info/</a>
246                   </td>
247                   <td width="100px"> </td>
248                   <td valign="top"><img src="img/josemar.png" alt="Foto de Josemar
                        " /></td>
249                   <td width="10px"> </td>
250                   <td width="300px" valign="top" align="left">
251                       <b>Josemar Rodrigues de Souza</b>
252                       <br />
253                       Ph.D. em Informática e Mestre em Arquitetura de Computadores
                             e Processamento Paralelo pela Universidad Autónoma de
                            Barcelona - UAB.
254                       <br />
```

```
255                         <a href="http://www.josemar.org/">http://www.josemar.org/</a
                                >
256                     </td>
257                 </tr>
258             </table>
259             </p>
260           </div>
261         </div>
262         </div>
263
264 <div class="jumbotron subhead" id="overview" style="background-color:rgb(231, 231,
      231);">
265 <div class="container"  id="publicacoes">
266         <div class="row" id="contato">
267           <div class="span10">
268             <h2>Publicações</h2>
269             <p>
270                 COSTA, A. L. L.; SOUZA, J. R. <b>MCAP: Modelo Computacional de Auto-
                        Paralelismo</b>. I Jornadas de Cloud Computing. La Plata, 2013.
                         666-6038 Journal of Computer Science and Technology.
271                 <br /><br />
272                 <i>* Caso utilize nosso trabalho como referencial em sua pesquisa,
                        entre em contato para que possamos avaliar e disponibilizar sua
                        publicação aqui.</i>
273             </p>
274           </div>
275         </div>
276         </div>
277         </div>
278
279         <hr>
280
281         <footer>
282           <center><p>#MCAP &copy; Todos os Direitos Reservados - 2013</p></center>
283         </footer>
284
285
286       <script src="js/jquery.js"></script>
287       <script src="js/bootstrap.js"></script>
288       <script src="js/bootstrap.file-input.js"></script>
289
290 <script>
291   (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
292   (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
293   m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
294   })(window,document,'script','//www.google-analytics.com/analytics.js','ga');
295
296   ga('create', 'UA-8754294-3', 'andrecosta.info');
297   ga('send', 'pageview');
298
299 </script>
300
```

```
301     </body>
302 </html>
```

Codigos/index.php

Apêndice I

Código do MCAP: CONTROLLER

```php
<?php

// MODEL
include_once 'class.MCAP.php';

// Gerar Parelelismo?
if ( $_POST["acao"] == "enviar" ) {


    // Verifica se arquivo foi enviado
    if ( is_uploaded_file( $_FILES['codigo_mpi']['tmp_name'] )) {



        $extenssao = substr( $_FILES['codigo_mpi']['name'], strpos( $_FILES['
            codigo_mpi']['name'], "." ));
  $ext = strtolower( $extenssao );

        if ( $ext == ".c" ) {

            $mcap = new MCAP();
            $mcap->gerarParalelismoOMP();

        } else {

            echo "Arquivo com extensão inválida!";

        }

    } else {

        echo "Arquivo não enviado!";

    }

}

?>
```

Codigos/request.MCAP.php

Referências Bibliográficas

AMINI M.; ANCOURT, C. C. F. C. B. G. S. I. F. J. P. K. R. V. P. Pips is not (just) polyhedral software. In: *First International Workshop on Polyhedral Compilation Techniques (IMPACT 2011).* Chamonix, France: [s.n.], 2011.

BLUMEETAL, W. Parallel programming with polaris. *Computer Magazine*, p. 78–82, 1996.

DAVE C.; BAE, H. M. S. J. L. S. E. R. M. S. Cetus: A source-to-source compiler infrastructure for multicores. *IEEE Computer*, v. 42, p. 36–42, 2009.

GROSSER T.; ZHENG, H. A. R. S. A. G. A. P. L. N. Polly polyhedral optimization in llvm. In: *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11).* [S.l.]: INRIA Grenoble Rhône-Alpes, 2011.

JIN H.; JESPERSEN, D. M. P. B. R. H. L. C. B. High performance computing using {MPI} and openmp on multi-core parallel systems. *Parallel Computing*, v. 37, n. 9, p. 562 – 575, 2011. ISSN 0167-8191.

LI J.; SHU, J. C. Y. W. D. Z. W. Analysis of factors affecting execution performance of openmp programs. *Tsinghua Science & Technology*, v. 10, n. 3, p. 304 – 308, 2005. ISSN 1007-0214.

LI S. B. R.; SCHULZ, M. C. K. N. D. D. Hybrid mpi/openmp power-aware computing. parallel & distributed processing (ipdps). In: *IEEE International Symposium.* [S.l.: s.n.], 2010.

LUSK E., C. A. Early experiments with the openmp/mpi hybrid programming model. In: *4th international conference on OpenMP in a new era of parallelism (IWOMP'08).* [S.l.: s.n.], 2008. p. 37–47.

MPI, F. Message passing interface forum. Disponível em: `<www.mpi-forum.org`. Acesso em: 12 out 2012. 2013.

OPENMP. The openmp® api specification for parallel programming. Disponível em: `http://openmp.org/`. Acesso em: 12 out 2012. 2013.

OSTHOFF C.; GRUNMANN, P. B. F. K. R. P. L. N. P. S. C. P. J. M. N. D. P. L. S. W. R. Improving performance on atmospheric models through a hybrid openmp/mpi implementation. In: *IEEE 9th International SymposiumParallel and Distributed Processing with Applications (ISPA'11).* [S.l.: s.n.], 2011. p. 69–74.

OSTHOFF C.; SCHEPKE, C. P. J. G. P. D. P. L. S. K. B. F. N. P. Improving core selection on a multicore cluster to increase the scalability of an atmospheric model simulation. *Mecánica Computacional*, XXIX, p. 3143–3153, 2010.

RABENSEIFNER R.; HAGER, G. J. G. Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In: *17th Euromicro International Conference*. [S.l.: s.n.], 2009. p. 427–436.

RAGHESHI, A. A framework for automatic openmp code generation. In: *Dissertação de Mestrado*. Madras, Índia: Department of Computer Science and Engineering, Indian Institute of Technology, 2011.

REENSKAUG, T. M. H. Mvc - xerox parc, 1979. Disponível em: `http://heim.ifi.uio.no/ trygver/themes/mvc/mvc-index.html`. Acesso em: 02 Jul 2013.

RIBEIRO N. S.; FERNANDES, L. G. L. Programação híbrida: Mpi e openmp. Disponível em: `http://www.inf.pucrs.br/gmap/pdfs/Neumar/Artigo-IP2%20-%20Neumar%20Ribeiro.pdf`. Acesso em: 13 jun 2013.

SILLA P. R.; BRUNETTO, M. A. O. C. Análise de um algoritmo de multiplicação de matrizes em versão paralela utilizando pvm e mpi. In: *Anais do III W2C*. Londrina, Paraná: Departamento de Computação, Universidade Estadual de Londrina, 2008.

TORQUATI M.; VANNESCHI, M. A. M. G. S. K. R. L. V. P. F. X. B. M. B. R. P. C. T. L. E. C. C. S. F. An innovative compilation tool-chain for embedded multi-core architectures. In: *Embedded World Conference*. Nuremberg, Germany: [s.n.], 2012.

WU X.; TAYLOR, V. Performance modeling of hybrid mpi/openmp scientific applications on large-scale multicore supercomputers. *Journal of Computer and System Sciences*, v. 79, n. 8, p. 1256 – 1268, 2013. ISSN 0022-0000.

ZHAO J.; NAGARAKATTE, S. M. M. M. K. Z. S. Formalizing the llvm intermediate representation for verified program transformation. In: *SIGACT Symposium on Principles of Programming Languages (POPL'12)*. [S.l.: s.n.], 2012.

PUBLICAÇÕES

COSTA, A. L. L; SOUZA, J. R. Aumentando a Escalabilidade de um Cluster com a versão paralela do Ocean-Land-Atmosphere Model. In: ERAD-NE 2011 - FPG. [s.n.], 2011. Disponível em: http://www.infojr.com.br/ERAD/view/ERAD_NE_2011_ForumPG.pdf.

COSTA, A. L. L; SOUZA, J. R. APCM: An Auto-Parallellism Computational Model. Increasing the performance of MPI applications in multi-core environments. I Jornadas de Cloud Computing. Universidad Nacional de La Plata. Buenos Aires, 2013. Disponível em: http://jcc2013.info.unlp.edu.ar/ponencias1.php.

COSTA, A. L. L; SOUZA, J. R. APCM: An Auto-Parallellism Computational Model. Increasing the performance of MPI applications in multi-core environments. Journal of Computer Science and Technology (La Plata. En línea) - 1666-6038. Qualis B2 Em Ciência da Computação.

MCAP: MODELO COMPUTACIONAL DE AUTO-PARALELISMO
André Luiz Lima da Costa
Salvador, Outubro de 2013.