Uma Estratégia Paralela e Autônoma para a Projeção de Modelos de Nicho Ecológico em Ambientes Computacionais Heterogêneos

Fernanda G. O. Passos, Vinod E. F. Rebello

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói, RJ – Brasil.

{fernanda,vinod}@ic.uff.br

Abstract. *Ecological Niche Modelling (ENM) is an important process to help ecologists understand and predict the potential geographic distribution of species. In addition to creating correlative models for each species, the projection of the model onto a geographical environment is an essential step in the process. Given the demand for improved precision and the need to address wider geospatial domains, using larger data sets means that the adopted methods incur relatively higher processing, memory and I/O demands. This paper proposes a new parallel and autonomic algorithm for the projection stage of an existing ENM tool widely used in large heterogeneous computing environments. The proposal is compared to the current default sequential implementation and a parallel MPI version both distributed with the software tool. An empirical analysis reveals gains of over 170% in terms of performance against the original MPI version, while the experiments also indicate improved scalability with efficiencies above 80%, in evaluations with up to 128 processors.*

Resumo. *A modelagem de nichos ecológicos (Ecological Niche Modelling – ENM) é um processo importante para ajudar ecologistas a entender e prever a distribuição geográfica potencial de espécies. Junto à criação de modelos correlativos para cada espécie, a projeção do modelo em um ambiente geográfico é um passo essencial neste processo. Devido à necessidade para melhorar a precisão e para tratar grandes domínios geoespaciais, tal método pode demandar níveis de processamento, memória e E/S, relativamente altos. Este artigo propõe um novo algoritmo paralelo e autônomo para a etapa de projeção de uma ferramenta ENM existente bastante utilizada em ambientes computacionais heterogêneos de larga escala. A proposta é comparada com a implementação sequencial padrão e uma versão paralela em MPI, ambas disponibilizadas com a ferramenta. Uma análise empírica indica ganhos em termos de desempenho acima de 170% em relação a versão MPI assim como uma melhor escalabilidade com eficiências acima de 80%, em testes com até 128 processadores.*

1. Introdução

A modelagem de nichos ecológicos – ENM (*Ecological Niche Modelling*) – é um procedimento comum para determinar a extensão da distribuição geográfica das espécies, provendo um procedimento poderoso para prever a potencial distribuição de espécies em diferentes contextos geográficos e temporais, assim como para estudar outros aspectos da

evolução e ecologia. ENM tem sido bastante usado em várias situações como por exemplo: busca de espécies raras ou ameaçadas de extinção; identificação de regiões adequadas para a (re-)introdução de espécies; previsão do impacto das mudanças climáticas sobre a biodiversidade; definição e avaliação de áreas protegidas; prevenção da propagação de espécies invasoras, entre outras aplicações importantes.

Aplicações de ENM combinam informações sobre a ocorrência de espécies (bióticas) com bases de dados ambientais (abióticas) na forma de camadas rasterizadas geo-referenciadas (tais como temperatura, precipitação e salinidade) para gerar potenciais modelos de distribuição. Este processo inclui uma combinação de etapas dependentes (em *workflow*) como: a criação, teste e projeção de modelos. Os modelos são gerados por algoritmos geralmente estatísticos como máxima entropia ou de inteligência artificial como redes neurais artificiais [Muñoz et al. 2011].

Este artigo irá tratar a paralelização da etapa de projeção em domínio geoespacial de modelos de nichos ecológicos, uma vez que o processamento deste tipo de dados é custoso. Tanto um algoritmo sequencial quanto um paralelo e distribuído é disponibilizado em uma ferramenta existente chamada OpenModeller [Muñoz et al. 2011]. Além disso, atualmente, existem projetos como [Amaral et al. 2015, Leidenberger et al. 2015, EUBrazil Cloud Connect 2015], que se baseiam no openModeller para prover serviços em nuvem computacional de ENM. No entanto, tais implementações não levam em consideração as características de recursos heterogêneos e compartilhados de ambientes como nuvens. O objetivo deste trabalho é propor um novo algoritmo paralelo e distribuído para a projeção ENM de modo que ele seja capaz de autogerenciar sua execução para torná-la mais eficiente. Para isto, este algoritmo é executado junto a um sistema gerenciador de aplicações chamada EasyGrid AMS.

Inicialmente, na Seção 2 são relatados trabalhos que fazem uso da ferramenta OpenModeller e trabalhos que exploram a paralelização de problemas que tratam dados geográficos. A Seção 3 descreve o caso de estudo deste artigo: a projeção do modelo de nicho ecológico, assim como sua implementação sequencial na ferramenta OpenModeller. A sua versão paralela MPI é explicada na Seção 4. Na Seção 5, o EasyGrid AMS e seu modelo de programação são apresentados para que, na Seção 6, a proposta do novo algoritmo paralelo e autônomo da projeção ENM seja exposta. A Seção 7 defende a proposta através de experimentos e resultados que demonstram sua eficiência de desempenho em relação a outras propostas. Por último, algumas conclusões sobre o trabalho são apresentadas.

2. Trabalhos Relacionados e openModeller

Existem dois pontos importantes que podem ser abordados como trabalhos relacionados neste artigo: 1) o tratamento de dados geoespaciais em paralelo e 2) o uso do *framework* ENM chamado OpenModeller, que, por sua vez, trata dados geográficos.

O estudo de aplicações que tratam dados geoespaciais ganharam força nos últimos anos devido à propagação de ferramentas como *Sistemas de Informação Geográfico* (GIS – *Geographic Information System*) [Bonham-Carter 2014] capazes de permitir a visualização e análise simples dos dados espacial e suas informações através de mapas, facilitando o trabalho dos especialistas. Por muitas vezes as aplicações usarem intensivamente uma grande quantidade de dados, existem trabalhos que exploram a paralelização

delas. Em [Abrahart and See 2014, Aji et al. 2013], diversos trabalhos são descritos considerando a implementação da distribuição de blocos estáticos de dados entre processadores, mas questionando a falta de modelos de programação para ambientes mais complexos. Os trabalhos em [Zhao et al. 2013, Aji et al. 2013] apresentam implementações para GPU seguindo mesma distribuição de dados dos anteriores.

2.1. O framework openModeller

O openModeller (OM) é uma ferramenta para ENM desenvolvida pelo Centro de Referência em Informação Ambiental (CRIA), junto com outros parceiros nacionais e internacionais [Sutton et al. 2007, Muñoz et al. 2011], bastante difundida entre a comunidade biológica e ecológica [Geller and Melton 2008, Ramachandra et al. 2010, Zarco-González et al. 2013]. Ele trabalha com modelos de distribuição potencial e inclui mecanismos para a leitura de dados ambientais e de ocorrência de espécies, seleção de camadas ambientais sobre as quais o modelo deve se basear, criação de um modelo de nicho fundamental, projeção de modelo em um cenário ambiental e escrita de dados em vários tipos de arquivo relativos a grandes imagens.

Vários algoritmos são providos na forma de *plugins* para a criação de modelos [Muñoz et al. 2011], incluindo:

- BIOCLIM (*Bioclimatic envelopes*): envoltórios bioclimáticos;
- CSMBS (*Climate Space Model*): modelo de clima espacial;
- GARP (*Genetic Algorithm for Rule-set Production*): algoritmo genético para produção de conjunto de regras;
- GARP_BS (*Best Subsets*): GARP com melhores subconjuntos;
- ENFA (*Ecological-Niche Factor Analysis*): análise de fator do nicho ecológico;
- ENVSCORE (*Envelope Score*): pontuação de envoltório;
- ENVDIST (*Environmental Distance*): distância ambiental;
- MAXENT (*Maximum Entropy*): entropia máxima;
- NICHE_MOSAIC (*Niche Mosaic*): mosaico de nicho;
- ANN (*Artificial Neural Network*): rede neural artificial;
- RF (*Random Forests*): florestas aleatórias;
- SVM (*Support Vector Machines*): máquina de vetor de suporte, e;
- AQUAMAPS: algoritmo para organismos aquáticos.

3. Estudo de Caso: Projeção do Modelo de Nicho Ecológico

Em uma modelagem de nicho ecológico, uma etapa muito importante e que requer grande processamento de dados geoespaciais é a projeção do modelo. Uma vez que o modelo de distribuição seja gerado, a projeção no mapa pode ser realizada. Tais modelos podem ser projetados em diferentes regiões geográficas, em diferentes cenários ambientais, tornando-se possível prever o impacto das mudanças climáticas sobre a biodiversidade, evitar a propagação de espécies invasoras, identificar aspectos geográficos e ecológicos da transmissão de doença, ajudar no planejamento de conservação, guiar campos de pesquisas, entre muitos outros usos [Peterson et al. 2011].

OM project – openModeller *project* – é a ferramenta do openModeller capaz de fazer a projeção dos modelos de distribuição. Ele irá gerar uma imagem com coordenadas (x, y) em uma determinada região geográfica cujos valores associados a cada ponto representam a probabilidade de se encontrar a espécie de acordo com os dados ambientais.

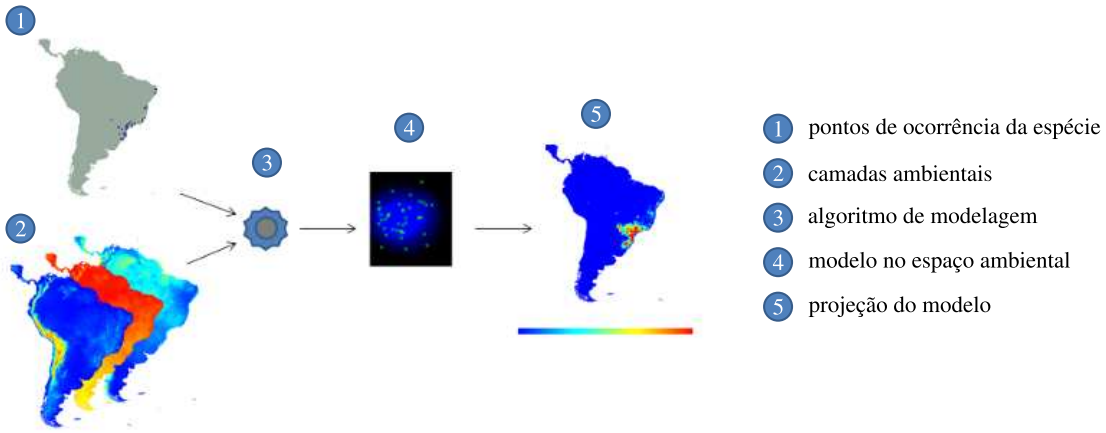

Figura 1. Exemplo de modelagem e projeção de um nicho ecológico [OpenModeller Team 2015].

A Figura 1 representa a modelagem e projeção de um nicho ecológico, onde o item 5 mostra o resultado da projeção de um modelo. A escala de cores que varia de azul a vermelho indica a adequação da espécie na região (vermelho é alta e azul é baixa). O usuário seleciona camadas de dados ambientais para a projeção e pode, também, selecionar uma máscara geoespacial, para projetar o modelo em uma área diferente de onde foi gerado. A imagem gerada geralmente é de alta resolução de *pixels* que deve ser visualizada por programas específicos como os sistemas de informação geográfica Quantum GIS ou QGIS, de código aberto [QGIS Project 2015].

Resumidamente, o Algoritmo 1 apresenta a implementação sequencial da projeção de um modelo de nicho ecológico e executa os seguintes passos para cada ponto (x, y):

1. adquire todos os dados das camadas ambientais no ponto (x, y) (corresponde à Linha 3 do algoritmo);
2. aplica o modelo no ponto (x, y) considerando tais dados ambientais e, então, gera uma probabilidade P (corresponde à Linha 4 do algoritmo);
3. converte P em valor da imagem e o escreve no arquivo na posição (x, y) (corresponde à Linha 5 do algoritmo).

As entradas do algoritmo são o modelo e os dados ambientais e, a saída, um arquivo contendo o mapa com as probabilidades. O modelo é previamente gerado pela etapa de modelagem através de um algoritmo descrito na Seção 2.1. Assim, a ferramenta do OpenModeller *OM project* tipicamente faz a leitura, processamento e escrita de uma grande quantidade de dados geográficos. Isto faz com que haja a iniciativa da criação de versões paralelas. Atualmente, incluída na ferramenta, existe um algoritmo paralelo desenvolvido para *clusters* de computadores homogêneos.

4. Versão MPI Original

A paralelização original do *OM project* usa um modelo semelhante ao mestre-trabalhador. Além dos processos trabalhadores que computam a projeção, existem dois processos mestres (ao invés de um) responsáveis por distribuir e receber as soluções, respectivamente, o que difere do modelo mestre-trabalhador tradicional, mas que tem efeito semelhante. Denomina-se *mestreD* o processo que distribui os intervalos de dados para cada processo

Algoritmo 1: Algoritmo sequencial de projeção do modelo de um nicho.

Entrada: $modelo$ - modelo previamente gerado pela ferramenta OM.
$dados\_ambientais$ - dados ambientais.
Saída: $mapa$ - arquivo que contém a projeção.

```
1  para x ← 0 a Dim_x − 1 faça
2      para y ← 0 a Dim_y − 1 faça
3          amb ← dados_ambientais na posição (x, y)
4          aplica modelo a amb e coloca o valor em Pr
5          escreve Pr em mapa na posição (x, y)
```

trabalhador e *mestreR* o processo que recebe as soluções de cada processo trabalhador e as escreve no arquivo imagem de saída (que contém a projeção).

O algoritmo de *mestreD* divide o número total de pontos a serem projetados (pontos da imagem) em blocos (valor padrão é 30.000) e os distribui aos processos trabalhadores sob demanda. Nesta aplicação, um bloco corresponde a um conjunto de linhas do mapa a ser projetado e, por isto, cada bloco é associado a (l_i, l_j), onde l_i é a linha inicial e l_j é a linha final. Para isso, um pedido de requisição deve ser feito pelos processos trabalhadores ao processo *mestreD* para que ele envie os dados informando qual o bloco a ser computado. Uma vez calculado, este bloco é enviado para o processo *mestreR* que irá receber os dados e escrevê-los no arquivo de saída. Quando todos os blocos forem calculados e escritos no arquivo pelo processo *mestreR*, o algoritmo termina. A separação de atividades do mestre entre 2 processos, *mestreD* e *mestreR*, é feita para que a distribuição de tarefas não seja sobrecarregada pela recepção dos dados, que é mais custosa.

Algoritmo 2: Algoritmo usado pelos trabalhadores da versão MPI original.

Entrada: $modelo$ - modelo previamente gerado por um algoritmo.
$dados\_ambientais$ - dados ambientais.
Dim_y - dimensão Y do mapa completo.

```
1  fim ← FALSO
2  enquanto fim = FALSO faça
3      Envia requisição a mestreD
4      Recebe (l_i, l_j) de mestreD
5      se l_i = −1 então
6          fim ← VERDADEIRO
7      senão
8          pacote ← ∅
9          para x ← l_i a l_j − 1 faça
10             para y ← 0 a Dim_y − 1 faça
11                 amb ← dados_ambientais na posição (x, y)
12                 Aplicar modelo em amb e colocar o valor em Pr
13                 pacote ← pacote ∪ (x, y, Pr)
14         Envia pacote para mestreR
```

As etapas de execução dos processos trabalhadores da versão original são apresentadas pelo Algoritmo 2. Neste caso, as entradas são o modelo, previamente gerado por um algoritmo na fase de modelagem, os dados ambientais e a dimensão Dim_y que denota o tamanho de uma linha. A variável fim controla o laço de repetição iniciado na linha 2. O término da repetição é feito nas linhas 5 e 6, onde é verificado se o l_i possui o valor -1 (bloco nulo). Caso positivo, a variável fim é alterada para verdadeiro e, então o laço de repetição termina. No início do laço, nas linhas 3 e 4, ocorre o envio da requisição para o processo $mestreD$ e o recebimento do bloco requerido em (l_i, l_j). Se forem valores válidos, o cálculo da projeção é iniciado. Para tal, um conjunto chamado $pacote$ é inicializado por $\varnothing$, na linha 8, e nele serão armazenados os valores já projetados. As linhas 9 e 10 indicam dois laços aninhados onde acontece o percorrimento do bloco atribuído à tarefa. Para cada ponto (x, y) deste bloco, o modelo é aplicado nos dados ambientais, a probabilidade é gerada e este resultado é guardado no conjunto $pacote$ (linha 13). Na linha 14, o envio do pacote completo para o $mestreR$ é realizado.

5. EasyGrid AMS e seu Modelo de Programação

O EasyGrid AMS [Boeres and Rebello 2004] é um sistema gerenciador de aplicações (AMS) que funciona como um meio entre a aplicação e o sistema computacional, de forma a facilitar o uso dos recursos considerando características da própria aplicação. O *middleware* EasyGrid AMS possui um gerenciamento hierárquico de três níveis e sua filosofia [Oliveira and Rebello 2011] indica que as aplicações devem ser gerenciadas por um sistema simples agregado a seu próprio *middleware* autônomo.

Como características autônomas, o EasyGrid AMS possui um escalonador de tarefas estático e dinâmico [Nascimento et al. 2007a] que fornecem auto-otimização (*self-optimization*) às tarefas da aplicação; um mecanismo tolerante a falhas por reexecução de tarefas e *checkpoint* no nível do gerenciador global que é capaz de detectar e recuperar as falhas da aplicação (autorrecuperação ou *self-healing*) [Silva and Rebello 2007]. A autoconfiguração (*self-configuring*) pode ser implementada de acordo com características da aplicação e, portanto, é específica por aplicação ou classe de aplicações. Trabalhos da literatura mostram bons resultados utilizando o EasyGrid AMS [Nascimento et al. 2007b, Silva and Rebello 2007, Ribeiro et al. 2010, Oliveira and Rebello 2010].

O seu modelo de programação é baseado no 1Ptask [Nascimento et al. 2007b], um modelo diferente do tradicional aqui chamado 1Pproc. O modelo 1Pproc considera um processo por processador e cada processo adquire toda a carga de trabalho atribuída a ele. Já o modelo 1Ptask indica um processo por tarefa. Assim, cada processo representa uma tarefa que geralmente é uma parte pequena de todo o trabalho da aplicação, tendo, portanto, um quantidade grande de processos por processador.

6. Versão Autônoma com EasyGrid AMS

A versão MPI original apresenta basicamente 2 problemas: 1) o algoritmo é centralizado e a troca de mensagens pelos mestres torna um gargalo de execução, prejudicando o desempenho. 2) o modelo de programação 1Pproc não é propício para ambientes distribuídos de larga escala. O modelo 1Ptask (ver Seção 5) é mais indicado para ambientes computacionais de larga escala, tipicamente sistemas heterogêneos e compartilhados com servidores *multicores*, comuns nas *grades* e *nuvens* computacionais, pois permite uma maior flexibilidade de execução as tarefas para fazer o balanceamento de carga da aplicação. A versão

autônoma indica colocar o algoritmo da aplicação no modelo 1Ptask, realizar um processo de autoconfiguração da aplicação e incluir a utilização do *middleware* EasyGrid AMS para o gerenciamento autônomo das tarefas. Como ocorre com a maioria das aplicação projetadas com o EasyGrid AMS, os benefícios dessa abordagem incluem adotar políticas (de escalonamento, comunicação, tolerância a falhas) adaptadas às necessidades específicas de cada aplicação, gerando assim um desempenho ainda melhor.

Uma outra questão relacionada ao modelo de programação é a divisão do domínio da aplicação em blocos de tamanhos fixos realizada na versão MPI original. Através da Figura 2, é possível visualizar este problema. A execução sequencial da projeção usando o algoritmo BIOCLIM foi conduzida em uma máquina dedicada, pertencente ao *cluster* (descrito em Seção 7), e, a cada contagem de blocos (1 linha do mapa), o tempo de execução da projeção dele foi exibido. Pelo gráfico da figura, pode-se verificar que os tempos variam bastante, com média de 0,28 segundos e desvio padrão de 0,15, indicando que a quantidade de trabalho de cada bloco é diferente. Por esta razão, a escolha de um tamanho fixo de bloco prejudica o balanceamento de carga visto que o cálculo, para alguns blocos, pode ser rápido e, para outros, demorado.

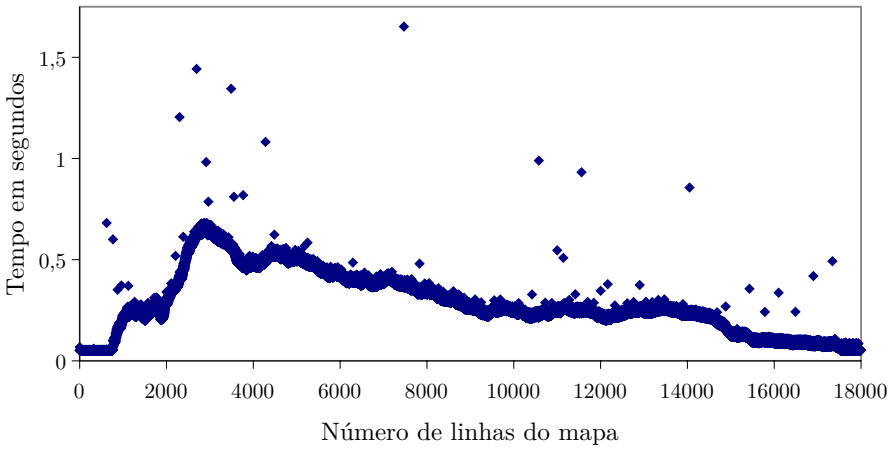

Figura 2. Tempo de execução de cada bloco fixo do domínio da projeção ENM.

A versão EasyGrid AMS usa tamanhos de blocos variáveis, já que a granularidade das tarefas é desconhecida. Este é um processo de reconfiguração dinâmica. Cada tarefa (lembrando que no modelo 1Ptask cada tarefa é representada por um processo) recebe um bloco inicial fixo com o valor de, por exemplo, o número total de pontos no mapa dividido pelo número de processadores. A partir de então, a tarefa calcula a projeção do bloco da mesma maneira que no algoritmo sequencial, mas limita a execução deste cálculo por um temporizador $timeout$. Caso o tempo de execução da projeção do bloco ultrapasse $timeout$, uma ou mais novas tarefas serão criadas para que continuam a projeção deste bloco. O particionamento do mapa é feito dinamicamente e, a cada nível de criação (tarefas iniciais estão no nível 0, seus filhos estão no nível 1 e assim sucessivamente) o valor do temporizador $timeout$ é alterado para um valor inferior. Considerando que $timeout$ representa a granularidade de tempo da tarefa, a razão para a alteração do $timeout$ é tornar a granularidade das tarefas que vão ficando para o final mais fina. Deste modo, as tarefas criadas que vão para a fila de escalonamento do EasyGrid AMS serão distribuídas entre os processadores mais eficientemente [Nascimento et al. 2007a].

O Algoritmo 3 corresponde aos passos de execução de cada tarefa da aplicação de projeção do OpenModeller, chamada de *partitioner*. Como entrada, ele recebe o modelo produzido na fase de modelagem, os dados ambientais, a dimensão Y do mapa original, o bloco e o valor inicial do temporizador $timeout$. Cada tarefa terá uma saída representada por $mapa_{l_i}$, onde l_i é a linha inicial do bloco e é única (identifica a posição do bloco no mapa). As repetições encontradas nas linhas 1 e 2, onde a segunda é aninhada à primeira, representam o percorrimento de cada ponto no bloco de dados do mapa. O valor x varia de l_i a l_j, passados como argumento do algoritmo, que são as linhas inicial e final que delimitam o bloco. O valor y varia de 0 a Dim_y. Assim, os dados ambientais são capturados na posição (x, y) e colocados na variável amb (linha 3). O modelo é, então, aplicado a amb e o valor da probabilidade é colocado em P, na linha 4. Em seguida (linha 5), o valor P convertido para valor de imagem é escrito em $mapa_{l_i}$ na posição $(x - l_i, y)$. Como é uma imagem nova, ela deve começar na coordenada $(0, 0)$ e a expressão para a linha do mapa $x - l_i$ fornece esta configuração.

O trecho do Algoritmo 3 compreendido entre as linhas 1 e 5 apresenta o mesmo comportamento existente no trecho entre as linhas 9 e 13 do Algoritmo 2. Este trecho corresponde ao processamento do problema de projeção ENM. A diferença está na forma como é feito o particionamento (autoconfiguração na versão EasyGrid AMS) e no gerenciamento das tarefas.

Algoritmo 3: Algoritmo das tarefas *partitioner* de projeção com o EasyGrid AMS.

Entrada: $modelo$ - modelo previamente gerado por um algoritmo.
$dados\_ambientais$ - dados ambientais.
Dim_y - dimensão Y do mapa completo.
(l_i, l_j) - bloco a ser calculado.
$timeout$ - temporizador inicial.
Saída: $mapa_{l_i}$ - mapas parciais.

```
1  para x ← l_i a l_j faça
2      para cada y ← 0 a Dim_y − 1 faça
3          amb ← dados_ambientais na posição (x, y)
4          Aplica modelo a amb e coloca o valor em p
5          Escreve p em mapa_{l_i} na posição (x − l_i, y)
6      se temporizar por timeout então
7          Calcula ntarefas
8          Calcula novo timeout
9          Divide bloco restante (x + 1, l_j) por ntarefas e coloca cada sub-bloco em
           S
10         para cada k ∈ S faça
11             Cria tarefa com argumentos modelo, dados_ambientais, Dim_y, k e
               timeout
```

Na linha 6 do Algoritmo 3, após a repetição em relação ao eixo Y (isto é, uma linha completa), é verificado se o temporizador esgotou pelo valor de $timeout$. Caso negativo, o cálculo segue para a próxima linha. Em caso positivo, na linha 6, o número de novas tarefas é gerado de acordo com os dados que foram processados neste tempo $timeout$.

Por exemplo, num total de 100 blocos, se 10 blocos foram calculados em 5 segundos por 1 tarefa, para calcular os restantes 90 blocos é preciso de 9 tarefas ($ntarefas = 9$). Neste caso, o valor de $ntarefas$ é estimado. Na linha 8, o valor de $timeout$, que será passado para as novas tarefas, é decrementado de acordo com o nível de criação das tarefas (tarefas iniciais estão no nível 0, seus filhos estão no nível 1 e assim sucessivamente). A razão para isto é tornar a granularidade das tarefas que vão ficando para o final mais fina e permitir, então, um melhor balanceamento de carga feito pelo EasyGrid AMS. Na linha 9, o restante do bloco é dividido entre as $ntarefas$ tarefas e cada sub-bloco é adicionado ao conjunto S de $ntarefas$ elementos. Por fim, para cada $k \in S$ (linha 10), uma nova tarefa é criada com os argumentos $modelo$, $dados\_ambientais$, Dim_y, k e $timeout$ (linha 11).

No fim de todo esse processo de mapeamento, toda a imagem estará calculada mas dividida entre diversos arquivos de saída. Uma tarefa do tipo *collector* é, então, usada para agrupar os dados em cada arquivo e gerar uma imagem final contendo todo o mapa de projeção.

7. Avaliação Experimental

Dois tipos de experimentos foram realizados: 1) para comparar a versão MPI original com a versão autônoma com EasyGrid AMS e 2) para avaliar a escalabilidade da projeção do openModeller com o EasyGrid AMS. No primeiro tipo de experimento, foi utilizado um *cluster* com 2 máquinas Intel Xeon X5650 com 2 processadores de 6 núcleos, cada, totalizando 24 CPUs, e com 24 GB de memória principal. O segundo tipo de experimento usa um *cluster* que possui 16 máquinas de 8 núcleos cada, totalizando 128 CPUs. Todos estes ambientes de execução estavam dedicados aos experimentos e as entradas usadas para cada algoritmo executado foram as mesmas.

Para os valores apresentados nesta seção de experimentos, a eficiência é calculado considerando a equação: $eficiencia = \frac{speedup}{P} \times 100\%$, onde P é o número de processadores ou CPU e $speedup = \frac{\textit{tempo sequencial}}{\textit{tempo paralelo}}$ é o *speed-up* da execução paralela em relação à versão sequencial disponível na ferramenta OM. A métrica eficiência indica o quanto um algoritmo paralelo utiliza bem os recursos disponibilizados.

7.1. Experimento 1

No experimento 1, P processos trabalhadores são usados para ambas as implementações – original com MPI e autônoma com EasyGrid AMS – em P CPUs. Portanto, na verdade, $P + 2$ processos são usados para a versão original (existem os 2 processos extras). Os algoritmos BIOCLIM, ENVSCORE, GARP, DG_GARP, DG_GARP_BS, MAXENT, GARP_BS, SVM, RF e NICHE_MOSAIC foram escolhidos para serem usados neste experimento, de acordo com seus tempos de execução (os mais altos foram preferidos, exceto o ENVDIST, por durar cerca de 20 dias) e a disponibilidade de suas entradas (por exemplo, não havia a nossa disposição as entradas para o algoritmo AQUAMAPS).

A Tabela 1 apresenta os resultados comparativos de *speed-ups*, em relação ao algoritmo sequencial, obtidos para a versão MPI original e a versão EasyGrid AMS com o uso de vários algoritmos diferentes. Variando o número de processadores (de 4 a 24), a versão autônoma com o EasyGrid AMS apresenta valores significativamente melhores de *speed-ups* e, para a maioria dos algoritmos de modelagem, o valor obtido foi próximo do

Tabela 1. *Speed-ups* obtidos usando as versões MPI original e a autônoma com o EasyGrid AMS.

Algoritmo	Versão	*Speed-up*				
		4	8	12	16	24
BIOCLIM	MPI Original	2,58	4,29	5,54	5,45	5,70
	EasyGrid AMS	3,67	7,15	9,96	13,03	19,21
ENVSCORE	MPI Original	2,51	3,86	5,82	6,82	6,49
	EasyGrid AMS	3,32	6,48	9,35	12,67	18,28
GARP	MPI Original	3,12	4,90	6,85	8,25	8,03
	EasyGrid AMS	3,85	7,57	11,21	14,59	21,58
DG_GARP	MPI Original	2,90	4,74	6,17	7,59	7,49
	EasyGrid AMS	3,67	7,11	10,01	13,10	19,73
DG_GARP_BS	MPI Original	3,11	5,80	7,26	9,68	10,49
	EasyGrid AMS	3,78	7,65	10,87	14,32	21,48
MAXENT	MPI Original	3,46	6,07	7,84	10,38	11,45
	EasyGrid AMS	3,88	7,62	10,99	14,61	21,70
GARP_BS	MPI Original	2,98	5,42	7,50	9,46	10,33
	EasyGrid AMS	3,85	7,60	10,96	14,52	21,77
SVM	MPI Original	3,32	5,63	8,01	10,54	12,59
	EasyGrid AMS	3,90	7,71	11,17	14,85	22,24
RF	MPI Original	3,28	4,99	7,97	10,15	11,30
	EasyGrid AMS	3,59	7,14	10,35	13,78	20,58
NICHE_MOSAIC	MPI Original	3,28	5,68	7,70	7,49	4,63
	EasyGrid AMS	3,94	7,81	10,10	11,48	12,74

número de processadores usados. Para 24 processadores, isto resulta em um valor médio de eficiência maior que 83%, e com eficiências de 93% e 91%, no caso do algoritmo SVM e GARP_BS, respectivamente.

O gráfico da Figura 3 provê uma melhor visualização da comparação dos resultados produzidos por ambas as versões paralelas. Cada grupo de barras com rótulos de 4, 8, 12, 16 e 24 processos em paralelo representam o ganho de desempenho e da eficiência da versão EasyGrid AMS em relação a versão original MPI, isto é, o quanto o *speed-up* da versão em questão foi superior ao *speed-up* da versão original MPI. Cada barra em cada grupo é um algoritmo usado na projeção do openModeller. A versão paralela da projeção com o EasyGrid AMS sempre obtém um *speed-up* maior em relação à versão original MPI e, na maior parte das vezes, é próximo a P. Para a maioria dos algoritmos, conforme o número de processos aumenta, o ganho do EasyGrid AMS em relação ao MPI aumenta também. Para BIOCLIM, com $P = 24$, o *speed-up* da versão da projeção com o EasyGrid AMS chega a ser 3,4 vezes maior que o valor do *speed-up* da versão original MPI.

A escalabilidade é um ponto fraco da versão MPI. Para alguns algoritmos, os *speed-ups* crescem levemente até os 24 processos e, para outros algoritmos (como DG_GARP e NICHE_MOSAIC), o valor diminui com 24 processos comparado a 16 processos. Para uma boa escalabilidade, *speed-up* linear é esperado conforme o número P aumenta. Para a versão EasyGrid AMS, esta propriedade é melhor alcançada embora não seja perfeito. No caso do algoritmo NICHE_MOSAIC, a escalabilidade é mais fraca (o

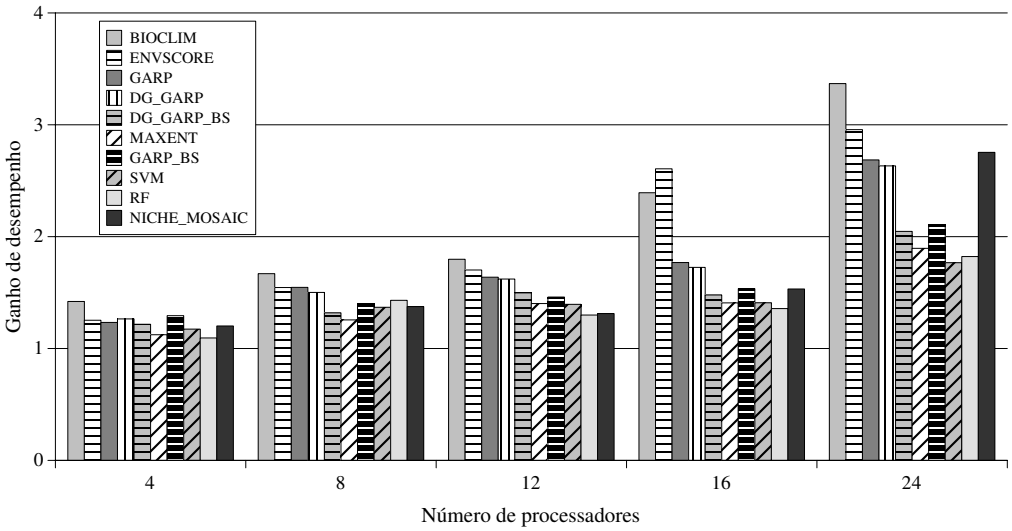

Figura 3. Comparação dos ganhos de desempenho e eficiência da versão EasyGrid AMS em relação a versão paralelo original em MPI.

speed-up é aproximadamente 13 enquanto poderia ser próximo de 24, como acontece para os outros algoritmos) mesmo tendo o *speed-up* quase três vezes melhor que a versão original MPI. Alguma característica do algoritmo não permite uma escalabilidade melhor. Mesmo assim, a eficiência, em termos de utilização dos recursos, é claramente melhor para o EasyGrid AMS uma vez que tem a vantagem devido a sua capacidade de atingir níveis de desempenho mais aceitáveis.

7.2. Experimento 2

No experimento 1, já foi possível concluir que a versão MPI original apresenta um resultado de desempenho e escalabilidade ruins, com até 24 processadores, e projeta-se a queda de desempenho para uma quantidade maior de processadores. Isto se deve ao gargalo de gerenciamento de tarefas que é menos frequente na versão proposta neste artigo. Neste segundo experimento, a escalabilidade da versão autônoma com o EasyGrid AMS é avaliada. Os algoritmos utilizados são ENVDIST e SVM, sendo que ENVDIST é o mais demorado de todos, chegando a quase 20 dias de execução sequencial. O SVM foi escolhido por apresentar o melhor *speed-up* para $P = 24$. Os números de CPUs P utilizados são 24, 48, 96 e 128.

Primeiramente, para se ter uma ideia do número de tarefas criadas durante a execução da projeção ENM, a quantidade total de tarefas para a instância SVM é de aproximadamente 5400, para todos os números de CPUs utilizados, apresentando baixa variância. Para a instância ENVDIST, o quantidade total de tarefas é de aproximadamente 16000, para todos os números de CPUs utilizados, apresentando também baixa variância. O número de tarefas não aumenta conforme o aumento de CPUs porque o objetivo do algoritmo é manter a maior quantidade de tarefas possível, independente do número de CPUs, de modo que elas tenham granularidade fina (mas que sobreponha a sobrecarga de gerenciamento do *middleware*).

A Tabela 2 mostra os tempos em segundos de execução da parte *partitioner*, a mais custosa, e a parte *collector* (concatenação das imagens geradas paralelamente) para o a

projeção usando os algoritmos ENVDIST e SVM. Os tempos de *collector* são próximos entre si para cada algoritmo e parecem não ter relação com o aumento do número P de CPUs, os valores crescem ligeiramente. O tempo total de execução é a soma dos tempos do *partitioner* e do *collector*. Como a implementação do *collector* não é paralelo, o tempo de execução dele pode representar uma parte significante do tempo total de execução. Isto ocorre no caso do algoritmo SVM, onde o tempo de execução total é de 285 segundos e a parte referente ao *collector* corresponde a 16% deste valor.

Tabela 2. Tempos em segundos (*partitioner* e *collector*) obtidos para ENVDIST e SVM.

	ENVDIST		SVM	
P	*partitioner*	*collector*	*partitioner*	*collector*
24	70758,88	75,93	1186,28	41,18
48	35316,90	85,26	614,53	59,50
96	17758,27	88,44	313,29	47,33
128	13390,62	98,59	238,36	46,59

O gráfico representado na Figura 4 mostra a eficiência em relação à execução sequencial de toda a projeção paralela com o EasyGrid AMS para cada um dos dois algoritmos. Nestes resultados, inclui-se o tempo de execução da tarefa *collector* que é executada sequencialmente. Para cada P (eixo horizontal indica o número de processadores), existem duas barras, uma para o ENVDIST e outra para o SVM. A barra cinza escura representa a eficiência de execução para o algoritmo ENVDIST e a barra cinza clara é referente ao algoritmo SVM. Para o algoritmo ENVDIST, a eficiência é alta tanto contando com a parte de *collector* quanto sem ela (ver também a Tabela 2). Neste caso, o *collector* pouco prejudica o desempenho. Ao contrário, para o algoritmo SVM, a eficiência é ligeiramente prejudicada pela tarefa não paralela *collector*, havendo uma diferença representativa entre a eficiência com e sem ela. Uma melhoria seria paralelizar o algoritmo do *collector* também, usando uma estratégia de redução hierárquica, por exemplo. Mesmo assim, as execuções para ambos os algoritmos mostraram-se bastante eficientes.

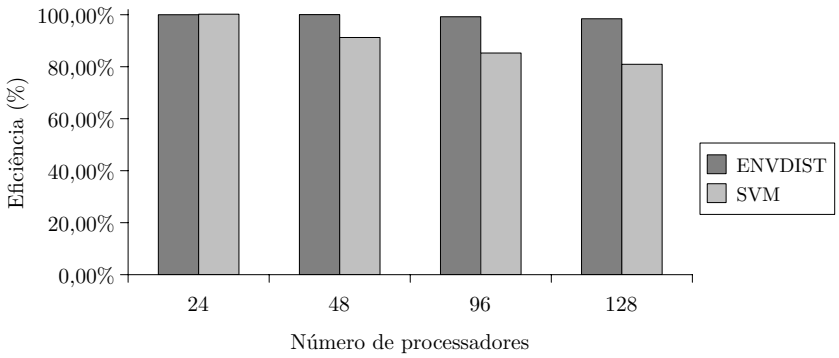

Figura 4. Eficiência da projeção para os algoritmos ENVDIST e SVM usando 24, 48, 96 e 128 CPUs.

8. Conclusão

Neste artigo foi proposto um novo algoritmo paralelo e autônomo para um problema de projeção em domínio geoespacial: a projeção do modelo de nicho ecológico. Este tipo

de aplicação apresenta as características de granularidade desconhecida e independência entre as tarefas, já que as operações sobre o domínio geoespacial são independentes.

A abordagem usada, pelo algoritmo paralelo original da ferramenta openModeller, é do tipo mestre-trabalhador e realiza um particionamento em blocos fixos possibilitando o balanceamento do trabalho entre tarefas no modelo 1Pproc. Já a nova abordagem do algoritmo paralelo e autônomo coloca a aplicação no modelo 1Ptask através de um particionamento de tarefas em blocos de tamanho variável autoconfigurado e utiliza o *middleware* EasyGrid AMS para fazer o gerenciamento das tarefas provendo autootimização e autorrecuperação à aplicação. Além disso, o algoritmo original centraliza os resultados no mestre, o que gera uma grande sobrecarga, já que os resultados são referentes a pedaços de mapas e apresentam uma grande quantidade de dados. No caso do algoritmo proposto, cada tarefa gera seu próprio sub-mapa em arquivo e não apresenta este gargalo durante a execução. A junção dos sub-mapas é feita ao término da projeção. Esta mesma abordagem autônoma pode ser utilizada em outros problemas como, por exemplo, um problema da descoberta de estruturas de proteínas da área de bioquímica [Oliveira and Rebello 2010].

Os resultados mostraram que o algoritmo autônomo com o EasyGrid AMS apresentou melhor desempenho em todos os experimentos comparado ao algoritmo original. A versão original MPI apresenta baixa escalabilidade enquanto a execução do algoritmo proposto é melhor escalável na grande maioria dos casos. Com 24 processadores, os *speed-ups* obtidos com a versão EasyGrid AMS chegam a ser de 177% a 337% maior do que os *speed-ups* da versão original MPI. Além disso, a versão paralela com o *middleware* escala bem nos testes com até 128 processadores, obtendo valores de eficiências perto de 100%, em uma média de 89% entre todos os resultados. Para melhorar os resultados ainda mais, a junção dos sub-mapas deve ser realizada em paralelo futuramente.

Referências

Abrahart, R. and See, L. (2014). *GeoComputation, Second Edition*. CRC Press.

Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., and Saltz, J. (2013). Hadoop GIS: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, 6(11):1009–1020.

Amaral, R., Badia, R. M., Blanquer, I., Braga-Neto, R., Candela, L., Castelli, D., Flann, C., De Giovanni, R., Gray, W. A., Jones, A., et al. (2015). Supporting biodiversity studies with the EUBrazilOpenBio hybrid data infrastructure. *Concurrency and Computation: Practice and Experience*, 27(2):376–394.

Boeres, C. and Rebello, V. E. F. (2004). EasyGrid: Towards a framework for the automatic grid enabling of legacy MPI applications. *Concurrency and Computation: Practice and Experience*, 16(5):425–432.

Bonham-Carter, G. F. (2014). *Geographic information systems for geoscientists: modelling with GIS*, volume 13. Elsevier.

EUBrazil Cloud Connect (2015). EUBrazil Cloud Connect. `http://www.eubrazilcloudconnect.eu`.

Geller, G. N. and Melton, F. (2008). Looking forward: Applying an ecological model web to assess impacts of climate change. *Biodiversity*, 9(3-4):79–83.

Leidenberger, S., De Giovanni, R., Kulawik, R., Williams, A. R., and Bourlat, S. J. (2015). Mapping present and future potential distribution patterns for a meso-grazer guild in the baltic sea. *Journal of biogeography*, 42(2):241–254.

Muñoz, M. E. S., De Giovanni, R., Siqueira, M. F., Sutton, T., Brewer, P., Pereira, R. S., Canhos, D. A. L., and Canhos, V. P. (2011). openModeller: a generic approach to species' potential distribution modelling. *GeoInformatica*, 15(1):111–135.

Nascimento, A., Sena, A., Boeres, C., and Rebello, V. E. F. (2007a). Distributed and dynamic self-scheduling of parallel MPI grid applications. *Concurrency and Computation: Practice and Experience*, 19(14):1955–1974.

Nascimento, A., Sena, A., da Silva, J., Vianna, D. Q. C., Boeres, C., and Rebello, V. E. F. (2007b). On the advantages of an alternative MPI execution model for grids. In *CCGRID '07*, pages 575–582, Rio de Janeiro, Brazil. IEEE Computer Society.

Oliveira, F. G. and Rebello, V. E. F. (2010). Algoritmos branch-and-prune autônomos. In *SBRC'10*, pages 335–348, Gramado, Rio Grande do Sul, Brazil.

Oliveira, F. G. and Rebello, V. E. F. (2011). Aplicações Autônomas + Sistemas Simples = Futuro Feliz? In *WoSida, SBRC'11*, Campo Grande, Mato Grosso do Sul, Brazil.

OpenModeller Team (2015). Openmodeller webpage. `http://openmodeller.sourceforge.net/`.

Peterson, A. T., Soberón, J., Pearson, R. G., Anderson, R. P., Martínez-Meyer, E., Nakamura, M., and Araújo, M. B. (2011). *Ecological niches and geographic distributions (MPB-49)*. Princeton University Press.

QGIS Project (2015). QGIS - A Free and Open Source Geographic Information System. `http://www.qgis.org/`.

Ramachandra, T., Kumar, U., Aithal, B. H., Diwakar, P., and Joshi, N. (2010). Landslide susceptible locations in western ghats: Prediction through OpenModeller. In *Proc. of the 26th Annual In-House Symposium on Space Science and Technology, ISRO-IISc*, pages 65–74, Bangalore, India.

Ribeiro, F., Sena, A., Nascimento, A., Boeres, C., and Rebello, V. E. F. (2010). A self-configuring N-body application. In *CIS, SBAC-PAD'10*, Petrópolis, Rio de Janeiro, Brazil.

Silva, J. and Rebello, V. E. F. (2007). Low Cost Self-healing in MPI Applications. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 14th European PVM/MPI User's Group Meeting*, pages 144–152, Paris, France. Springer.

Sutton, T., De Giovanni, R., and Siqueira, M. F. (2007). Introducing open modeller – a fundamental niche modelling framework. *OSGeo Journal*, 1(1).

Zarco-González, M. M., Monroy-Vilchis, O., and Alaníz, J. (2013). Spatial model of livestock predation by jaguar and puma in Mexico: Conservation planning. *Biological Conservation*, 159:80 – 87.

Zhao, Y., Padmanabhan, A., and Wang, S. (2013). A parallel computing approach to viewshed analysis of large terrain data using graphics processing units. *International Journal of Geographical Information Science*, 27(2):363–384.