Escalabilidade de aplicação operacional em ambiente massivamente paralelo[1]

Álvaro L. Fazenda, Jairo Panetta,
Luiz F. Rodrigues, Daniel M. Katsurayama,
Luis F. Motta

Centro de Previsão de Tempo e E. Climáticos
Instituto Nacional de Pesquisas Espaciais

Philippe O. A. Navaux

Instituto de Informatica
Universidade Federal do Rio Grande do Sul

Resumo

A demanda constante para melhorar a qualidade de previsões numéricas do tempo obriga o uso de computadores progressivamente mais potentes. Com a popularização de processadores "multicore", o uso de sistemas com muitas centenas de processadores tornou-se economicamente viável. Este trabalho investiga como escalar o paralelismo de uma aplicação operacional para a previsão do tempo que executa eficientemente em muitas dezenas de processadores para máquinas com muitas centenas de processadores. A investigação determinou as limitações da aplicação, suas surpreendentes causas e permitiu desenvolver uma solução eficiente que atinge a escala de paralelismo desejado.

1. Introdução

Ao longo dos últimos 30 anos, fabricantes de processadores canalizaram sucessivos progressos na miniaturização de componentes ("Moore's Law") para aumentar a velocidade dos processadores. Essa tendência terminou. Embora o grau de miniaturização continue aumentando, não é mais canalizado para aumentar a velocidade de um processador pela existência de barreiras aparentemente intransponíveis no consumo de energia e na dissipação térmica [1].

Gerações recentes de processadores mantiveram o poderio computacional individual das gerações anteriores e utilizaram o aumento de miniaturização para replicar processadores no mesmo "chip", gerando arquiteturas "multicore" [1, 2].

Esta mudança de direção tem impacto significativo em aplicações de Processamento de Alto Desempenho. Na tendência anterior, a conversão do aumento de velocidade de um processador em execuções mais rápidas da aplicação era obtida praticamente sem alterar a aplicação. Na nova tendência é necessário gerar grau de paralelismo suficientemente alto para usufruir do aumento no número de processadores.

Este trabalho relata as modificações efetuadas em aplicação operacional de previsão numérica de tempo, rotineiramente utilizada em máquinas com muitas dezenas de processadores, para usufruir de máquina com muitas centenas de processadores. Demonstra que o aumento do número de processadores altera a importância relativa das fases da computação, obrigando alterações substanciais em fases outrora inexpressivas.

Este trabalho está estruturado na forma a seguir. A Seção 2 apresenta a aplicação e o arcabouço experimental. A Seção 3 avalia o desempenho paralelo da versão original da aplicação. A Seção 4 descreve as modificações realizadas em trechos da aplicação e os ganhos em cada trecho. A Seção 5 avalia os resultados atingidos com a nova versão e a Seção 6 apresenta as conclusões e direções futuras de trabalho.

2. Aplicação e Arcabouço Experimental

A aplicação utilizada é o BRAMS [4, 9], um modelo de previsão de tempo de área limitada (regional) mantido pelo INPE/CPTEC. O BRAMS é utilizado para pesquisas e para operação diária em múltiplos locais do mundo. O escopo de disseminação do BRAMS é representado na Figura 1.

O BRAMS é utilizado em vasta gama de sistemas computacionais, desde desktops mono processados até clusters com muitas dezenas de processadores. Esta gama de uso requer código fonte robusto, portável, flexível e eficiente – um grade desafio para a equipe do CPTEC que mantém, desenvolve e dissemina mundialmente o produto. A esse desafio acrescenta-se nova necessidade: ampliar a

1 Este trabalho foi parcialmente financiado pelo Projeto Atmosfera Massiva do CNPq [3], processo número 551033/2007-0, contemplado pelo edital MCT/CNPq/CT-INFO 07/2007, "Grandes Desafios da Computação no Brasil: 2006-2016"

escalabilidade do BRAMS em uma ordem de grandeza no número de processadores.

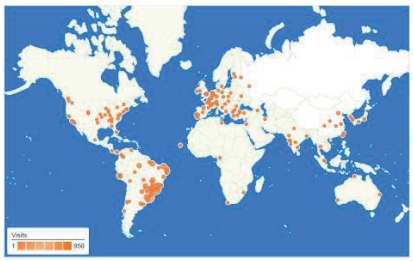

Figura 1. Acessos à página do BRAMS[4]

Este trabalho relata experimentos que utilizaram até 700 processadores em grades com resolução horizontal de 20km e 10km sobre a totalidade da América Latina. Trata-se de estudo pioneiro – o comportamento do BRAMS tanto em muitas centenas de processadores quanto em resolução fina sobre grandes domínios geográficos é desconhecido, pois seus usuários tipicamente utilizam dezenas de processadores, quer em previsões com resolução mais grosseira (80km ou 40km) nesse grande domínio geográfico, quer em previsões com resoluções mais finas (poucos km) em domínios geográficos muito menores. Como a complexidade computacional do BRAMS é cúbica no número de pontos em uma dimensão da grade horizontal para um período de previsão fixo, refinar a resolução horizontal de 20km para 10km aumenta o tempo de execução em aproximadamente oito vezes, justificando o aumento no número de processadores.

Para medições iniciais de desempenho utilizou-se a versão 4.1 do BRAMS em previsões de 24 horas sobre a América Latina em grades com 340 por 370 pontos horizontais (resolução de 20km) e de 680 por 740 pontos horizontais (resolução de 10km), sempre com 38 níveis verticais. A grade horizontal é particionada (bidimensionalmente) pelos processadores, que comunicam-se por troca de mensagens MPI [5]. Como o menor domínio possível para um processador é de 3 pontos em cada direção horizontal, o número máximo de processadores é da ordem de 14000 e 56000 processadores para as grades de 20km e 10km, respectivamente. Logo, centenas de processadores não esgotam o paralelismo.

O paralelismo do BRAMS é do tipo mestre-escravo. Grosseiramente, apenas os escravos avançam o estado da atmosfera no tempo, delegando ao mestre as outras funções como a divisão de domínio e I/O. O estado da atmosfera é composto por centenas de campos meteorológicos, cada campo representando uma grandeza física em todos os pontos da grade.

O sistema computacional utilizado é o cluster "UNA" do INPE/CPTEC, composto por 275 nós, cada qual com 8GB de memória principal e com dois processadores AMD Opteron "dual core" (Opteron 2218), totalizando 1100 "cores". Os nós são interligados por rede Infiniband que também conecta o sistema de arquivos paralelo (LUSTRE) de 70TB. Todas as execuções utilizaram quatro "cores" por nó, com um único processo MPI por "core".

3. Avaliação de Desempenho Paralelo

A Figura 2 contém os tempos de execução ("wall clock" expresso em segundos) do BRAMS medidos na UNA para um dia de previsão nas resoluções de 20Km e 10Km, variando o número de processadores escravos.

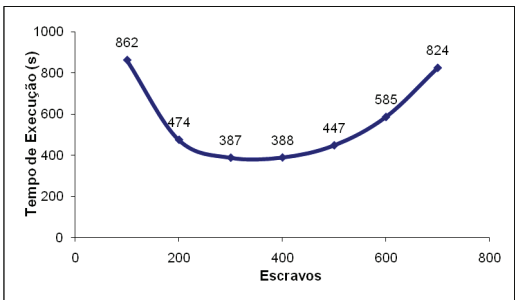

Figura 2a. Escalabilidade original (20Km)

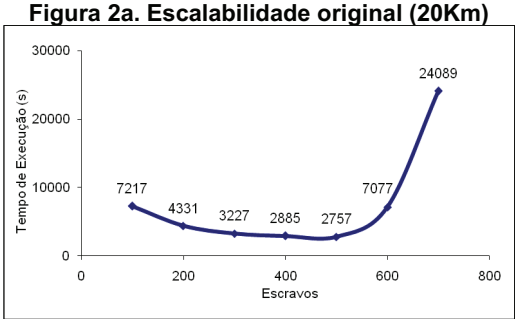

Figura 2b. Escalabilidade original (10Km)

Os limites da escalabilidade são evidentes. Há número ótimo de processadores em cada resolução, muito inferiores aos máximos anteriormente citados.

A Tabela 1 apresenta outra característica indesejável do BRAMS: o processo mestre usa memória em excesso. O excesso acarreta paginação na resolução de 10Km, por exceder os 8GB disponíveis por nó. Não há paginação nas execuções de 20Km, quer no mestre quer nos escravos.

O mestre requer mais memória que um escravo pois armazena os campos meteorológicos no domínio completo, enquanto cada escravo armazena os campos apenas na sua partição de domínio. Computações estruturadas dessa forma são adequadas para execuções pouco demandantes (domínio e resolução) em máquinas com poucos processadores, mas totalmente inadequadas para esta computação em máquina deste porte.

Tabela 1. Memória (GB) original em 20/10Km

Escravos		100	200	300	400	500	600	700
20km	Mestre	3,1	3,2	3,3	3,4	3,5	3,7	3,8
	Escravo	1,1	1,2	1,3	1,4	1,6	1,7	1,8
10km	Mestre	9,0	9,2	9,3	9,4	9,5	9,6	9,7
	Escravo	1,4	1,4	1,5	1,6	1,7	1,8	1,9

Para identificar as fases da computação responsáveis pelos limites de escalabilidade, a Tabela 2 apresenta pseudo-código do BRAMS com as ações correspondentes no mestre e nos escravos. Distingue cinco fases denotadas da seguinte forma:

- inicialização (sublinhado);
- condições de contorno (*itálico*);
- timestep (fundo escuro);
- CFL (texto comum);
- saída (duplo sublinhado).

Tabela 2. Pseudo-código do BRAMS

Mestre	Escravos
inicializa (envia p/escravos) faz até tempo máximo *se (ler c.contorno), então* *lê c.contorno* *particiona* *envia(escravo) sub-cont.* *fim se* recebe(escravos) CFL se (tempo de saída), então recebe(esc.) sub-domínio concatena sub-domínios escreve saída no disco fim se fim faz	inicializa (recebe do mestre) faz até tempo máximo *se (ler c.contorno), então* *recebe(mestre) sub-cont.* *fim-se* avança Timestep calcula CFL envia(mestre) CFL se (tempo de saída), então envia(mestre) sub-domín. fim se fim faz

A fase denominada "timestep" avança o estado da atmosfera ao resolver numericamente um sistema de equações diferenciais parciais que representa o transporte de fluídos pela atmosfera ("dinâmica"). Para tanto, também computa as forçantes das equações, oriundas de processos físicos como a precipitação, a radiação, as interações com a superfície e outros, que na maioria dos casos requerem solução de outras equações diferenciais. As demais fases demandam poucas operações ponto flutuante.

Conseqüentemente, a fase "timestep" é a mais demandante da computação, pela quantidade de operações ponto flutuante por passo de tempo. Além disso, essa fase é executada em cada iteração do laço central, enquanto as fases "saída" e "condições de contorno", guardadas por condicionais, são raramente computadas. Por exemplo, as previsões de 24 horas na grade de 20km foram executadas com passo no tempo de 30 segundos, com uma saída a cada 3 horas e com leitura de nova condição de contorno a cada 6 horas. Logo, o laço central tem 2880 iterações, todas processando as fases "timestep" e "CFL" (estabilidade numérica de Courant, Fridrichs e Lewy), das quais apenas 8 iterações processam a fase "saída" e 4 iterações processam a fase "condição de contorno".

Historicamente, o desenvolvimento do paralelismo no BRAMS foi concentrado no trecho mais demandante ("timestep"). É natural supor ser este trecho o maior responsável pela limitada escalabilidade observada. A suposição é falsa para o caso em estudos, como será justificado a seguir.

Para atribuir responsabilidades da limitada escalabilidade a cada fase da computação, foi inserida instrumentação detalhada que mede os tempos de execução de cada fase. Para evitar que atrasos em uma fase sejam atribuídos a fases subseqüentes, foram inseridos pontos de sincronização artificiais (barreiras) na fronteira entre fases. A sincronização também permite que os tempos de execução de cada fase sejam idênticos no mestre e nos escravos. Claramente, a inserção das barreiras altera os tempos totais de execução, mas atribui corretamente à cada fase seu tempo de execução e eventual atraso.

A Tabela 3 contém os tempos de execução na resolução de 20km utilizando o código assim instrumentado. Optou-se por não utilizar a resolução de 10km para evitar a contaminação das medidas pela paginação.

Tabela 3. Tempo de execução (s) por fase da computação original

Escravos	100	200	300	400	500	600
Inicialização	39	44	49	66	91	135
Cond.Contorno	7	14	33	73	141	262
Timestep	724	354	235	187	157	131
CFL	3	8	8	16	16	25
Saída	183	189	176	180	188	197
Total	956	609	501	522	593	750

A Tabela 3 mostra três comportamentos distintos do tempo de execução de uma fase com o aumento do

número de processadores. Há três fases com tempo de execução crescente (“inicialização”, “condição de contorno” e “CFL”), uma com tempo de execução aproximadamente constante (“saída”) e outra com tempo de execução decrescente (“timestep”). Logo, basta aumentar o número de processadores para que as fases com tempo de execução não decrescente dominem a computação. A Tabela 3 mostra que esse fato ocorre a partir de 300 processadores.

Inspeção detalhada do código fonte apontou as razões para o comportamento indesejado de cada uma das quatro fases não decrescentes. De uma forma geral, os algoritmos utilizados nessas fases tem componentes seqüenciais. Em particular, nas fases “inicialização”, “condições de contorno” e “saída”, há troca de grandes volumes de dados entre o mestre e os escravos visando I/O. Para minimizar o número de mensagens e aumentar o tamanho de cada mensagem, o processo mestre agrega todos os dados a enviar (ou recebidos) em uma única mensagem para (de) cada escravo. Logo, o mestre precisa decompor (ou recompor) cada campo meteorológico do (no) domínio completo, o que é obrigatoriamente seqüencial no número de escravos. Além disso, decompor (ou recompor) todos os campos antes de enviar (ou após receber) as mensagens requer memória em excesso.

4. Solução desenvolvida

Apresentamos as falhas detectadas pela inspeção e as soluções desenvolvidas, organizadas nas fases acima definidas.

4.1. Fase CFL

A sigla CFL representa um critério de estabilidade numérica para a integração de equações diferenciais parciais definido por Courant, Fridrichs e Lewy. Caso o critério de estabilidade seja ultrapassado em algum ponto do domínio, o passo de tempo da integração deve ser reduzido. Logo, basta obter o valor máximo do critério de estabilidade sobre todos os pontos do domínio.

Na versão original do BRAMS, o algoritmo para o cálculo de CFL acumula seqüencialmente, no mestre, os máximos CFL locais computados por cada escravo, no domínio sob sua responsabilidade. Logo, a parte paralela é realizada pelos escravos e a parte seqüencial é realizada pelo mestre ao recolher, seqüencialmente, o CFL computado por cada processo escravo e gerar, seqüencialmente, o CFL máximo sobre todo o domínio.

O novo algoritmo colapsa a troca de mensagens e posterior cálculo do máximo em uma operação coletiva (MPI_REDUCE). Com esta modificação os tempos de execução do novo algoritmo tornaram-se desprezíveis, sendo esta fase incorporada à fase “timestep”.

4.2 Fase Condições de Contorno

Um modelo que prevê o tempo em área geográfica limitada (modelo regional) deve incorporar variações do estado da atmosfera na fronteira da área limitada, que afetam diretamente o estado da atmosfera no interior do domínio. Logo, modelos regionais são alimentados por previsões previamente feitas por outros modelos que possuem domínio geográfico com maior abrangência, como os modelos globais. A cada passo de tempo do modelo regional o estado da atmosfera deve ser conhecido no contorno do seu domínio, o que é obtido por interpolação entre dois estados externos previamente lidos (geralmente com espaçamento da ordem de horas).

O processo mestre tem a função de ler todos os dados relativos às condições de contorno, decompor os dados nos sub-domínios de cada processo escravo e enviar a partição devida a cada escravo. O algoritmo é seqüencial, pois apenas o processo mestre computa (decomposição). O tempo de execução aumenta com o número de escravos porque o número de partições (e de mensagens) aumenta com o número de escravos. O mestre utiliza memória em excesso pois os dados lidos contém múltiplos campos no domínio completo. Para minimizar o número de mensagens, o mestre envia todos os campos decompostos em uma única mensagem para cada escravo, obrigando o mestre a armazenar todos os campos no domínio completo.

A primeira tentativa de solução foi delegar a leitura a todos os processos escravos. Cada processo lê todos os campos no domínio completo, um campo por vez, e armazena apenas o trecho do campo no domínio sob sua responsabilidade. Essa solução foi abandonada por gerar competição por I/O, aumentando o tempo de execução.

Para evitar a competição por I/O, mantivemos a leitura de cada campo completo em um único processo, que propaga (“broadcast”) a informação lida para todos os outros processos. Cada processo que recebe a informação extrai e armazena o trecho do campo na sua partição do domínio. A leitura e a propagação de campos completos sucessivos reutiliza a mesma área de memória, reduzindo o uso de memória.

A figura 3 contém o tempo de execução desta fase em função do número de processos, tanto na versão original quanto na nova versão. Claramente, a seqüencialidade do algoritmo anterior foi substituída pela propagação e pelo trabalho simultâneo dos processos, apresentando tempo de execução praticamente inalterado com o número de processos.

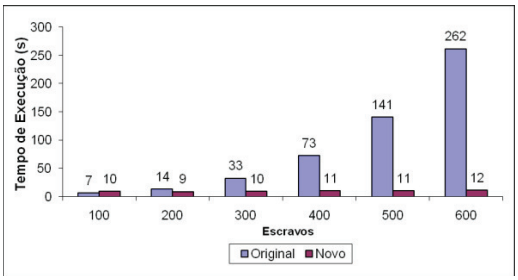

Figura 3. Ganho de desempenho da fase Condição de Contorno (20Km)

4.3 Fase Saída

O tempo de execução desta fase é relativamente alto, embora praticamente invariante com o número de processadores. Originalmente o processo mestre recebe de cada escravo todos os campos da partição do domínio daquele escravo, recompõe os campos completos para em seguida escrever todos os campos em disco, acarretando uso excessivo de memória.

Uma possível solução é utilizar MPI-IO [11], na qual cada processo escreve, individualmente, sua partição do domínio em um arquivo contendo a totalidade da informação. Esta solução foi descartada pois obriga todos os usuários a instalar bibliotecas MPI-2, modificando sua infra-estrutura computacional, o que nem sempre é possível.

A solução proposta consiste em trabalhar um campo por vez, compondo-o por operações coletivas que agreguem o campo (MPI_GATHER) e escrevendo-o em disco antes de passar ao próximo campo. Novamente, há maior número de mensagens (uma por campo) mas utiliza-se operações paralelas e reduz-se os requisitos de memória.

Entretanto, os tempos de execução continuaram altos. Estudo pormenorizado concluiu que a composição do campo completo a partir de suas partições requeria tempo de execução excessivamente alto. Assim, optou-se por uma nova estratégia que evita compor o campo completo: cada processo escreve sua porção do domínio em seu próprio arquivo. O tempo de execução resultante foi extremamente satisfatório, como demonstrado pela Figura 4, onde "Versão 1" representa a solução que produz um único arquivo (compondo os campos) e "Versão 2" representa a solução que produz um arquivo por processo.

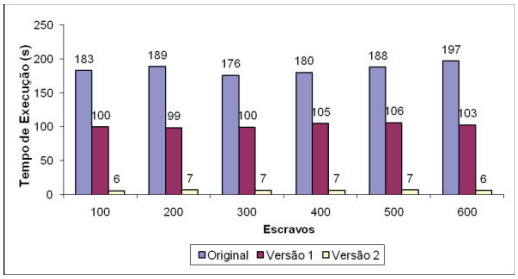

Figura 4. Ganho de desempenho da fase Saída (20Km) – versões 1 e 2

4.4 Fase Inicialização

O algoritmo de inicialização é similar ao algoritmo das condições de contorno. A decomposição de domínio seqüencial é responsável pelo aumento do tempo de execução com o número de processos. Por outro lado, minimizar o número de mensagens a cada processador e aumentar o tamanho de cada mensagem acarreta uso excessivo de memória.

A solução adotada é similar: a cada campo lido, o processo responsável pela leitura envia um campo completo para os outros processos ("broadcast"), que por sua vez extrai e armazena apenas a parte do domínio sob sua responsabilidade. Novamente, um algoritmo seqüencial foi substituído por um algoritmo paralelo. O aumento do número de mensagens permitiu redução substancial de memória.

Deve-se ressaltar que o volume de informações a propagar durante esta fase é substancialmente maior que na fase "condições de contorno", pois propaga-se campos que não são alterados durante a execução, como a topografia, por exemplo. A Figura 5 contrasta os tempos de execução desta fase nas versões original e nova.

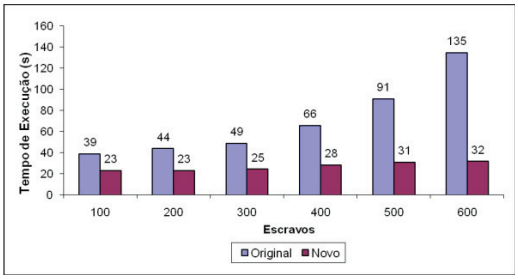

Figura 5. Ganho de desempenho da fase Inicialização (20Km)

Os tempos de execução da nova versão pouco variam com o número de processadores e são substancialmente inferiores aos da versão original, seqüencial. Embora o tempo de execução resultante ainda seja significativo, seu custo é amortizado em integrações mais longas, típicas de casos operacionais.

4.5 Reestruturando a computação

A transformação dos gargalos seqüenciais em algoritmos paralelos reduziu substancialmente o trabalho do processo mestre, que apenas lê e propaga as informações na inicialização, lê as condições de contorno e obtém o CFL máximo. Ao mesmo tempo, o uso de memória desse processo foi substancialmente reduzido. Dessa forma, não há mais motivos para distinguir o mestre dos escravos: todos os processos efetuam a mesma computação (todos computam a fase "timestep"), entretanto apenas um deles (o antigo mestre) lê as informações de disco.

Outra modificação necessária é a remoção de procedimentos específicos para execuções em um único processador. Na versão original o papel de mestre e o papel de escravo requerem no mínimo dois processadores. Para acomodar execuções em um único processador, havia uma terceira codificação que unia os procedimentos similares ao mestre e ao escravo em um único procedimento, sem troca de mensagens. Como na nova codificação os processos são autônomos exceto pela leitura de dados e CFL, bastou tratar esses casos especificamente.

O código atual do modelo BRAMS, com as modificações citadas, tem ao todo, aproximadamente, 160.000 linhas de codificação Fortran-90 (além de algumas centenas de linhas em linguagem C). O volume de linhas alteradas em relação à versão original chega a 30% desse total.

5. Apresentação e avaliação dos resultados

As Figuras 6a e 6b contrastam os tempos de execução da versão original e da nova versão nas resoluções de 20km e de 10km, removida a instrumentação detalhada e o sincronismo artificial. Os tempos de execução ("wall clock") reportados são as médias de cinco execuções.

O aumento da escalabilidade do BRAMS é notável. Claramente a maior parte do ganho advém da redução do tempo de execução obtido pela remoção de algoritmos seqüenciais nas fases com pouca demanda de operações ponto flutuante. Destaque-se a redução do tempo de execução com 700 processadores na resolução de 20km – a nova versão é aproximadamente seis vezes mais rápida que a versão anterior. Mas a remoção de algoritmos seqüenciais não é o único fator na redução do tempo de execução.

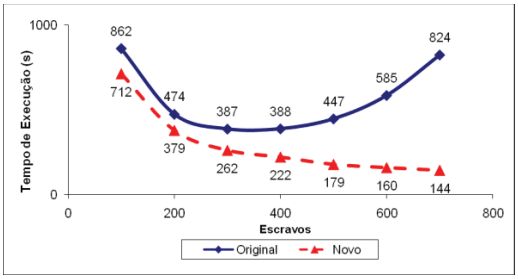

Figura 6a: Escalabilidade 20Km-original/novo

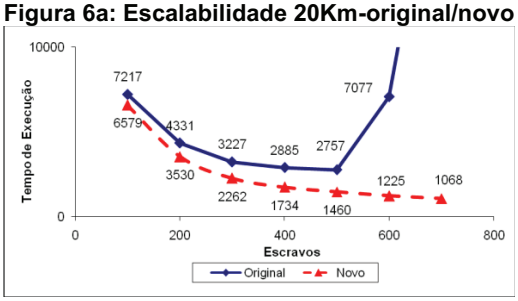

Figura 6b:Escalabilidade 10Km-original/novo

Outro fator com grande impacto na redução do tempo de execução foi a redução do uso de memória e conseqüente eliminação do mestre. A Tabela 4 contém o uso de memória da nova versão, utilizando a notação "Lê disco" para o único processo que lê dados de entrada (antigo mestre) e "Não lê" para os demais processos (antigos escravos). Confronto entre as Tabelas 1 e 4 mostra a redução de memória.

Tabela 4. Memória (GB) da nova versão

Escravos		100	200	300	400	500	600	700
20km	Lê disco	0,5	0,6	0,7	0,8	0,9	1,1	1,2
	Não lê	0,5	0,6	0,7	0,8	0,9	1,1	1,2
10km	Lê disco	1,0	1,0	1,1	1,2	1,3	1,5	1,6
	Não lê	0,8	0,9	0,9	1,1	1,1	1,3	1,4

O uso excessivo de memória pelo mestre na formulação original retardava a computação e tornava inviável aumentar ainda mais a resolução. Era inviável executar o BRAMS na resolução de 5Km por excesso de paginação no mestre. Após 24 horas de execução com 500 processadores, a versão original ainda estava na fase de inicialização. Em contraste, a Figura 7 contém os tempos de execução da nova versão na resolução de 5Km.

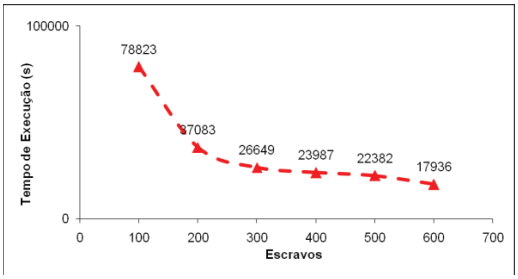

Figura 7: Escalabilidade em 5Km

É notável que a redução do tempo de execução da nova versão com o número de processadores não se restrinja à resolução utilizada durante o desenvolvimento. As curvas de tempo de execução da nova versão nas Figuras 6 e 7 tem formatos similares, mas os tempos de execução compreendem três ordens de magnitude (milhar, dezena de milhar e centena de milhar de segundos), pelo aumento da resolução.

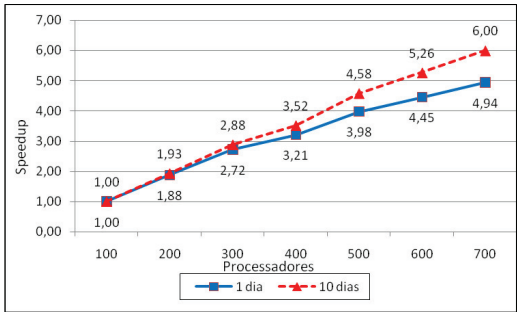

Figura 8a: Ganho paralelo da nova versão para um dia e dez dias de previsão (20Km)

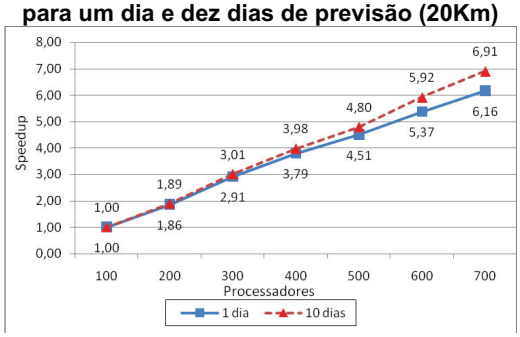

Figura 8b: Ganho paralelo da nova versão para um dia e dez dias de previsão (10Km)

Para investigar os limites da escalabilidade da nova versão é adequado observar as curvas de ganho ("speed-up"). Entretanto, o ganho varia com o número de dias de previsão, pois o custo da inicialização é amortizado pelo aumento do período de previsão. As Figuras 8a e 8b apresentam o ganho da nova versão em função do número de processadores (base 100 processadores) para um dia e para dez dias de previsão, nas resoluções de 20km e de 10km.

Os ganhos para 10 dias de previsão são notáveis nas duas resoluções. Entretanto, são melhores na resolução de 10km do que na resolução de 20km. Esse é um fato conhecido em paralelismo – fixo o número de processadores, o aumento do tamanho do problema (e da quantidade de computação por processador) mascara imperfeições. É provável que se utilizássemos dez vezes mais processadores (1000 a 7000) a curva de ganho na resolução de 10km apresentasse novas imperfeições.

Para buscar a origem das imperfeições na resolução de 20Km, avaliou-se a única fase restante – "timestep" – nessa resolução. A Tabela 5 contém os ganhos da execução com sincronismo forçado, computada sobre os tempos de execução da Tabela 3.

Tabela 5: Ganho na fase "timestep" (20Km)

Escravos	100	200	300	400	500	600
Ganho	1,0	2,05	3,08	3,87	4,61	5,53

Nota-se um ganho de desempenho com 200 e 300 escravos, seguido de queda de eficiência de 400 a 600 escravos, tomando como base o tempo de processamento com 100 escravos. Para análise mais detalhada, é necessário observar como a carga é distribuída pelos escravos. Para tanto, os tempos de execução dessa fase foram divididos em dois tempos: o tempo de computação e o tempo de espera na barreira, escravo a escravo. A Figura 9 contém os tempos de execução (área clara) e de espera na barreira (área escura) nesta fase para execuções de 100 e 600 escravos, escravo a escravo.

Análise visual mostra que a carga não é precisamente distribuída entre os escravos nos dois casos. Aparentemente há crescimento da importância do sincronismo com o número de escravos, fruto do desequilíbrio de carga. Parte do desequilíbrio de carga provêm da divisão de domínio 2D. Observa-se que os gráficos das Figuras 9a e 9b apresentam formas que se repetem. Há 10 repetições no gráfico com 100 escravos e há 26 repetições no gráfico com 600 escravos. Estes números correspondem exatamente à quantidade de trechos em que o algoritmo de divisão de domínio do BRAMS divide um dos eixos.

Entretanto, o desequilíbrio de carga não pode ser atribuído apenas à divisão de domínio (desequilíbrio estático). É sabido que o comportamento da atmosfera produz desequilíbrio dinâmico de carga.

Por exemplo, a chuva se desloca pela grade ao longo da integração, aumentando a quantidade de computação nos trechos e instantes em que ocorre. Balanceamento de carga em modelos meteorológicos é um problema pouco explorado. Limitar a escalabilidade do BRAMS por este fator, após a remoção dos outros fatores, demonstra o sucesso da atividade.

Figura 9a: Desbalanceamento da fase "timestep" com 100 processos

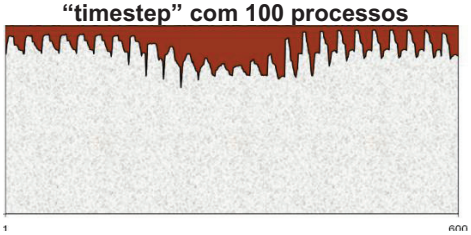

Figura 9b: Desbalanceamento da fase "timestep" com 600 processos

6. Conclusão e Trabalhos Futuros

Obteve-se sucesso em ampliar a escalabilidade ("strong scalling") de uma aplicação real de dezenas para centenas de processadores. Demonstrou-se que o sucesso requer atenção a fases outrora inexpressivas da computação, visando o uso adequado de diversos componentes da arquitetura, incluindo a memória, o I/O e a rede de comunicações. Para determinar os gargalos do paralelismo, foi essencial sincronizar a execução de cada fase, evitando que atrasos em fases sem sincronismo sejam atribuídos a fases posteriores.

O resultado não se restringe a um problema de tamanho fixo. Aplica-se a uma gama de computações cujos tempos de execução abrangem três ordens de magnitude ao variar o tamanho do problema. Esta característica é essencial para o sucesso de códigos altamente disseminados como o BRAMS.

Para o futuro, aguarda-se a popularização de máquinas com milhares e dezenas de milhares de processadores. O comportamento de modelos atmosféricos nessa classe de máquinas começa a ser investigado [6,7]. Este trabalho aponta o desbalanceamento de carga como possível gargalo do paralelismo do BRAMS nas máquinas futuras, visto ser a fonte de pequenas imperfeições no paralelismo de máquinas atuais. Esse fator já havia sido identificado em máquinas menos demandantes [8] em outros modelos meteorológicos. Dentre as possíveis soluções, aponta-se o balanceamento automático por virtualização [10]. Entretanto, o real impacto desse fator na escalabilidade do BRAMS para máquinas com milhares de processadores requer experimentação.

7. Referências

[1] K. Asanovic et all, "The Landscape of Parallel Computing Research: A View from Berkeley." *Technical Report No. UCB/EECS-2006-183*. December 18, 2006.

[2] J. Dongarra et all, "The Impact of Multicore on Computational Science Software". *CTWatch Quarterly online journal* (ISSN 1555-9874), 3(1), feb 2007.

[3] Projeto Atmosfera Massiva, edital CNPq Grandes Desafios, http://gppd.inf.ufrgs.br/atmosferamassiva/

[4] BRAMS Website, http://brams.cptec.inpe.br

[5] W. Gropp et all. "A high-performance, portable implementation of the MPI message passing interface standard". *Journal of Parallel Computing,* 22(6), p. 789-828, sept 1996.

[6] J. Michalakes et al, "WRF Nature Run", *Supercomputing 2007*, ACM 2007.

[7] M. Wehner, L. Oliker, J. Shalf, "Towards Ultra-High Resolution Models of Climaet and Weather", *International Journal of High Performance Computing Applications,* 22(2), p. 149-165, summer 2008.

[8] J. Drake et all, "Design and Performance of a scalable parallel community climate model", *Parallel Computing,* 21, p. 1571-1591, 1995.

[9] S. Freitas et all: "The Coupled Aerosol and Tracer Transport model to the Brazilian developments on the Regional Atmospheric Modeling System (CATT-BRAMS). Part 1: Model description and evaluation", *Atmos. Chem. Phys. Discuss.*, 7, p. 8525-8569, 2007

[10] A. Gusroy, L. V. Kale, "Performance and Modularity Benefits of Message-Driven Execution", *Journal of Parallel and Distributed Computing,* 64, p. 461-480, 2004.

[11] MPI: A Message-Passing Interface Standard - Version 2.1 . *Message Passing Interface Forum* . University of Tennessee, Knoxville, Tennessee . June 23, 2008 .